

Improving the Fairness of Alternative Backoff with ECN (ABE)

Naeem Khademi, Michael Welzl
Department of Informatics
University of Oslo
Norway
{naeemk,michawe}@ifi.uio.no

Grenville Armitage
Faculty of Science, Engineering and Technology
Swinburne University of Technology
Melbourne, Australia
garmitage@swin.edu.au

Stein Gjessing
Department of Informatics
University of Oslo
Norway
steing@ifi.uio.no

Abstract—Explicit Congestion Notification (ECN) lets a bottleneck’s Active Queue Management (AQM) mechanism inform an endpoint about congestion without having to drop a packet. A recently proposed sender-side modification called Alternative Backoff with ECN (ABE) enables reduced latency while maintaining good utilization with ECN. However, under certain circumstances ABE can produce a degree of unfair behavior between ABE-enabled TCP senders and conventional TCP senders. We propose specific guidance for configuring bottleneck AQMs to assist in fairness between ABE-enabled and conventional TCP flows. We evaluate our proposal using RED, then describe how it can be applied to other AQM mechanisms and incrementally introduced into the Internet.

I. INTRODUCTION

The last decade has seen a phenomenal growth of bandwidth in both core and edges of the Internet. Yet latency remains a key challenge for the end-user experience provided by applications as diverse as interactive websites (e.g. online banking and shopping), Voice over IP (IP Telephony), online gaming and online automated share trading.

Beginning with Random Early Detection (RED) [1] back in the late 1990s, the Internet Engineering Task Force (IETF) has considered a number of Active Queue Management (AQM) techniques to limit the potentially-excessive queueing delays induced at bottlenecks by loss-based TCP flows. AQM schemes lower queueing delays using early congestion signals to TCP senders – either by dropping packets or using Explicit Congestion Notification (ECN [2]). Yet despite being over 15 years old, ECN is still not widely enabled in end-hosts and congestion signalling is still largely performed by packet drop.

In recent work called “Alternative Backoff with ECN” (ABE) [3] we showed how latency-constraining AQM schemes lead to a loss of performance for TCP flows traversing paths with moderate to high base round trip time (RTT). We further demonstrated mitigation of this performance loss by modifying how TCP senders back-off in response to ECN signals. Attractions of ABE include that it can be incrementally deployed, it is a TCP sender-side only modification that provides immediate local benefits, and motivates more widespread adoption of ECN as an alternative to loss-based congestion signalling.

However, under certain circumstances ABE alone produces unfairness between ABE-enabled TCP senders and conventional TCP senders. Our novel contribution in this paper is

to propose and evaluate an AQM modification that provides backward compatibility and improved fairness between TCP flows from traditional senders and TCP flows from ABE-enabled senders. At its core, we propose that AQM bottlenecks apply earlier (more aggressive) marking to ECN-enabled flows, but only when no non-ECN flows have recently coexisted in the bottleneck for a configurable period of time. We also describe some simple changes to ECN to enable more efficient configuration and deployment of ECN while remaining backwards compatible with regular ECN.

The rest of this paper is structured as follows: Section II provides background on ABE and potential unfairness issues, Section III describes and evaluates our proposed AQM enhancements, possible deployment issues are summarised in Section IV and we conclude in Section V.

II. BACKGROUND – ABE AND UNFAIRNESS

ABE [3] consists of (a) enabling ECN in hosts and AQM-enabled bottlenecks and (b) letting individual TCP senders use a larger *multiplicative decrease factor* (β) in response to ECN-marks (β_{ecn}).¹ Here we summarise the interactions between ECN, ABE and fairness.

A. Establishing ECN flows

First, we briefly summarize how RFC 3168 [2] enables a TCP connection to become an “ECN flow”:

- The sender starts with a SYN carrying the “ECN Echo” (ECE) and “Congestion Window Reduced” (CWR) flags set – an “ECN-setup SYN packet” [2]
- If the receiver supports ECN, it returns a SYN/ACK packet with the ECE flag set and CWR flag cleared – an “ECN-setup SYN/ACK packet” [2]

If this handshake completes successfully, both sides use ECN for the remainder of this TCP connection. This is a necessary precondition before a sender can utilize ABE. If the receiver does not support ECN, or some middlebox interferes with the ECE and CWR flags, the connection falls back to regular loss-based congestion signalling.

¹For example, NewReno’s current $\beta = 0.5$ might be split into $\beta_{ecn} = 0.8$ for responding to ECN-marks and $\beta_{loss} = 0.5$ for responding to loss [4].

B. Thresholds for packet drop versus ECN mark should differ

Non-ECN (legacy) TCP flows and ABE-enabled flows can coexist in an AQM-enabled bottleneck without starvation of either types of traffic [3]. However, some degree of unfairness in favor of ECN flows emerges when (per RFC 3168 [2]) packets are marked under the same conditions as they would otherwise be dropped. This unfairness is evident for regular ECN flows ($\beta_{ecn} = 0.5$) and moreso for ABE-enabled flows ($\beta_{ecn} > 0.5$). Such behaviour is not necessarily favored in cases where strict fairness is important.

When a bottleneck AQM admits and marks packets (rather than drops them), local queue length and latency (queuing delay) increases. If the AQM mechanism triggers even more marking events in response to the additional queue length, we see more sender-side backoffs, lower utilization and reduced latency. Actual results depend on which of these two forces win on a case-by-case basis.

These observations hold whether an AQM mechanism bases its decisions on the average queue length (as in RED [1]) or ARED [5]), queuing delay (CoDel’s $target_delay$ [6], [7]) or estimated queuing delay (PIE’s T_{target} [8]). Figures 1 and 2 illustrate the inconsistent results that can occur with and without ECN using either CoDel, PIE or ARED AQM schemes (using the testbed described in Appendix A).

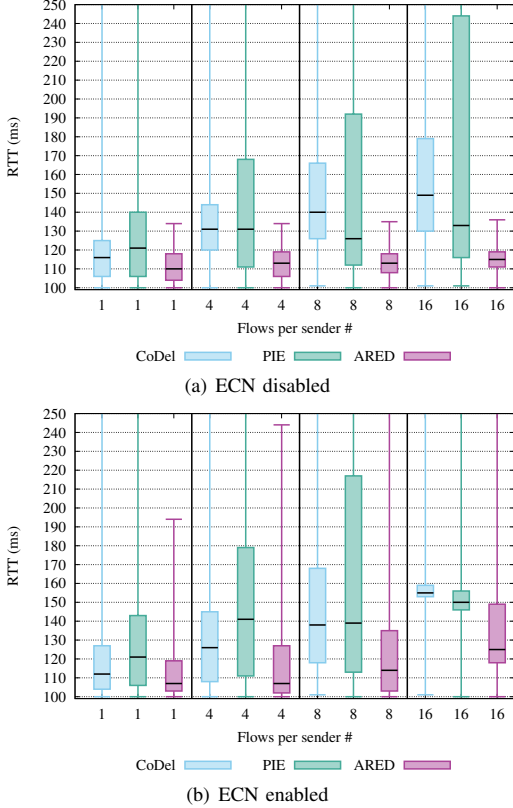


Fig. 1. TCP NewReno’s per-packet RTT for ECN-capable and non-ECN flows. Four senders, $target_delay=5$ ms and $RTT_{base}=100$ ms (no ABE).

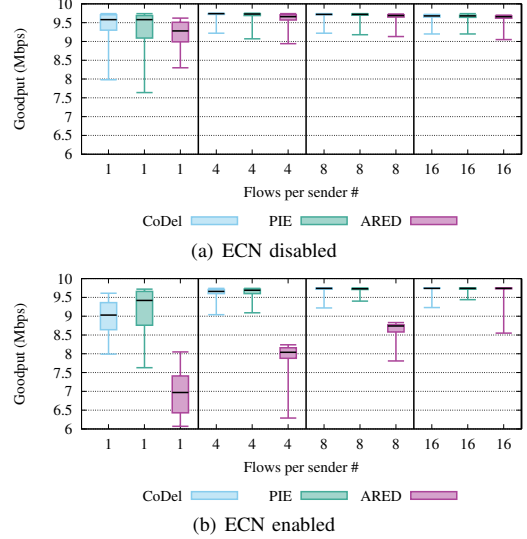


Fig. 2. TCP NewReno’s throughput at the bottleneck link (per 5-sec intervals) for ECN-capable and non-ECN flows. Four senders, $target_delay=5$ ms and $RTT_{base}=100$ ms (no ABE).

In contrast with RFC 3168’s recommendation, we conclude: *An AQM should allow the ECN marking threshold(s) to differ from dropping threshold(s).*

To the best of our knowledge, there is no guidance available, in either RFCs or academic literature, on precisely how an alternative ECN marking threshold should be set. We make such a recommendation in Section III-A.

C. Configuring AQM: trading between latency and utilization

The difficulty of parameter choice is often mentioned as the reason for the sparse deployment of RED, a situation that later AQM mechanisms try to simplify [9]. Most newer AQM mechanisms have at least one parameter controlling how aggressively they mark or drop packets based on the queue length or sojourn time (e.g. CoDel’s default $target_delay = 5$ ms and PIE’s $T_{target} = 20$ ms). The trade-off is that reacting more aggressively produces less queuing delay at the cost of throughput.

The throughput cost comes about because early packet marking or dropping reduces a queue’s effective capacity. Standard NewReno halves TCP’s congestion window ($cwnd$) in the face of congestion, so to maintain full path utilization a single NewReno flow needs a bottleneck queue equal to the path’s $bandwidth \times delay$ product (BDP).² Dropping or marking packets from such a flow at a $queue_size < BDP$ can reduce this flow’s throughput.

The detrimental impact of $queue_size < BDP$ diminishes as the number of TCP flows sharing the bottleneck queue increases [10]. However, the latency reduction benefits of AQMs are most desirable in consumer-access gateways³ where

²“delay” in this context is the path’s base RTT, and the total RTT experienced by the flow doubles as the BDP-sized queue becomes full.

³The speed imbalance between home LAN and “last mile” ISP access makes this point a significant bottleneck and contributor to experienced RTT.

the number of concurrent flows can be quite small. So we shall focus on the impact of AQM in lightly-multiplexed scenarios (a handful of ABE-enabled and standard flows).

ABE-enabled TCP flows compensate for the small effective AQM queue by utilizing $\beta_{ecn} > \beta_{loss}$. But this does nothing for legacy TCP traffic. Therefore, to strike a good balance (a form of *fairness*) between delay and throughput for both legacy (non-ECN) TCP flows and ABE-enabled TCP flows, AQM mechanisms should give more queue space to non-ECN flows – i.e. different thresholds for dropping versus ECN marking.

III. AQM CONFIGURATION FOR FAIRNESS WITH ABE

Here we propose a new approach to configuring AQM bottlenecks that provides improved fairness between loss-based and ABE-enabled TCP flows wherever fairness might be desired, and most specifically for when very low marking thresholds (hence very low latency) are favored along with the fairness. We demonstrate our approach using ABE in conjunction with a dual-threshold RED AQM mechanism, although the technique generalizes to other AQM techniques.

A. Incremental ABE deployment with improved fairness

To promote incremental deployment it should be possible for early adopters to gain benefit without detriment to parts of the Internet that have not yet adopted the new protocol. For ABE, this means focusing on *coexistence* – sharing of an AQM bottleneck with standard loss-based TCP flows. ABE flows should (a) not highly penalize or starve the standard loss-based TCP flows and (b) preferably achieve their own *relative* fair share of bandwidth. These goals are met in [3], although fairness among two types of traffic might be additionally favored in certain scenarios. Consequently we propose a set of ideas to enhance fairness with loss-based flows.

One way to achieve fairness is by physically separating the flows by using different sub-queues. Examples include stochastic fair queuing (SFQ) [11] schedulers and the more recent “FlowQueue-CoDel” (FQ-CoDel) AQM [12]. However, the resulting fair rate allocation imposes per-flow fairness that is not favored in many scenarios. Hence we focus on improving fairness for single-queue AQM solutions.

There is no universal solution which performs well on every AQM design. However, we do illustrate optimal parameter tuning for a candid AQM mechanism and, using typical access link scenarios and demonstrate that: (1) it is possible to configure an AQM mechanism to achieve fairness among ABE and standard loss-based TCP flows; (2) it is possible to determine regions in the parameter space where *full* fairness among such flows is achieved and none of the flows (either loss-based or ABE) face starvation.

B. Tuning dual-threshold RED – an illustrative example

Our fairness study uses RED [13] – the earliest IETF-documented AQM mechanism, used as a baseline in many studies and widely available for use (although not commonly employed) in the Internet’s routers. Our approach may also be

applied to different state-of-the-art AQM mechanisms that are considered easier to tune than RED.

We utilize dual-threshold RED as illustrated in Figure 3. Here RED is extended to use two different th_min values – one for loss-based TCP flows and another, lower threshold value th_min_{ecn} , for ECN-enabled (ABE) flows. Based on the recommendations in [5], th_max and th_max_{ecn} are set to $3 * th_min$ and $3 * th_min_{ecn}$ respectively.⁴

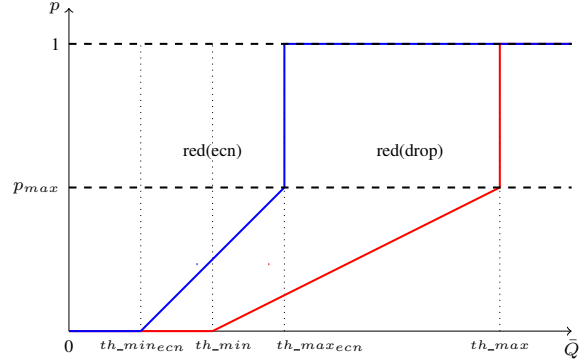


Fig. 3. Drop ($red(drop)$) and marking ($red(ecn)$) probability curves for a dual-threshold RED AQM bottleneck.

Our approach is similar to recent work [14], [15] that substantially redefines ECN semantics for Data Center TCP (DCTCP) and differentiates ECN-enabled and non-ECN traffic using separate mark/drop thresholds [14] or dual-queues [15]. However, we use conventional ECN with a single queue and instead introduce dynamic mode-switching to control which mark/drop threshold is used at a given time.

1) *Don’t use both thresholds in parallel:* Concurrent use of different mark and drop thresholds works out badly during coexistence situations (when the single queue is carrying both ABE-enabled and non-ECN traffic). The results tends towards extreme unfairness against (and starvation of) ECN flows.

We conducted ns-2 simulations to investigate this for different values of the *ECN Threshold Ratio* ($\frac{th_min_{ecn}}{th_min_{drop}}$). Each simulation run is 300sec of two concurrent long-lived TCP flows (one ABE-enabled and one standard loss-based) over a dumbbell network topology having $RTT_{base}=100$ ms, a dual-threshold RED bottleneck on the forward path and a 10 Mbps bottleneck link. All other simulated links are 100Mbps. TCP flows are transporting FTP application data and the packet size is set to 1500 bytes. The presented results are averages over 10 runs of simulation for each parameter combination.

Figure 4 shows how the ratio of the ABE flow’s throughput to loss-based flow’s throughput varies with $\frac{th_min_{ecn}}{th_min_{drop}}$ (for $th_min_{drop} = 0.5 * BDP$ and $th_min_{drop} = 0.1 * BDP$) and β_{ecn} (for the ABE flow’s sender). Figure 5 shows the link utilization over the same parameter ranges.

Our results suggest there is no simple and safe way to use different $red(ecn)$ and the $red(drop)$ curves simultaneously.

⁴We shall explicitly use th_min_{drop} and th_max_{drop} for the drop thresholds in the remainder of this paper.

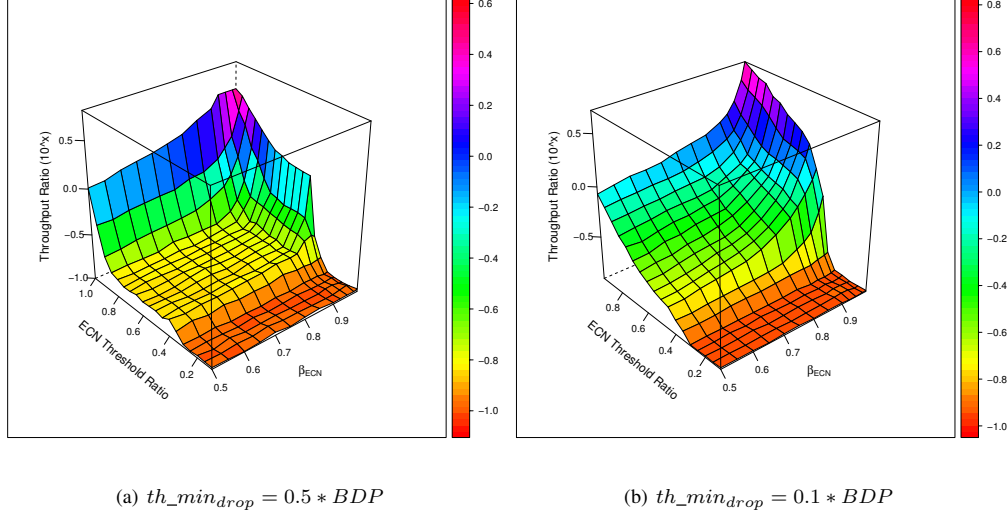


Fig. 4. Throughput ratio of one ABE to one standard loss-based TCP flow for different β_{ecn} and ECN Threshold Ratios (ns-2 simulation).

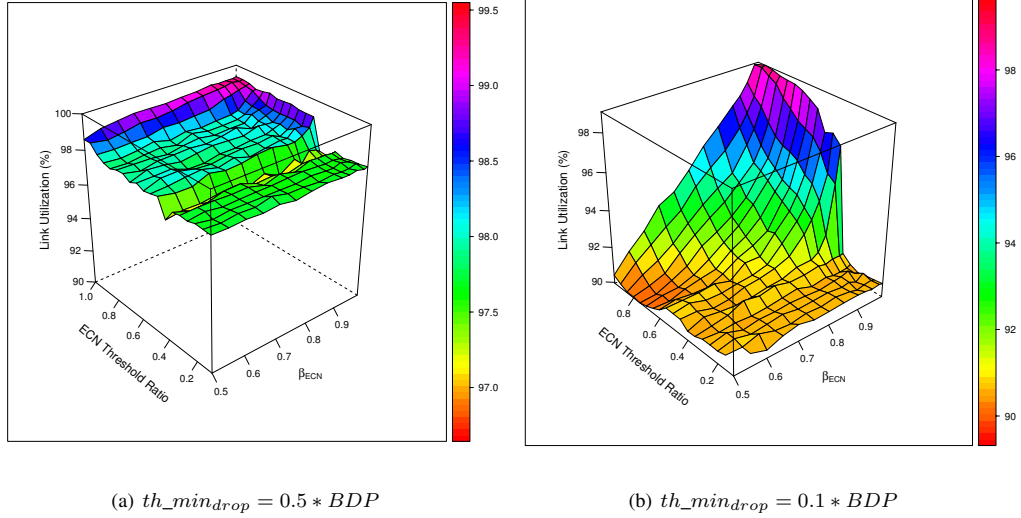


Fig. 5. Link utilization of one ABE and one standard loss-based TCP flow for different β_{ecn} and ECN Threshold Ratios (ns-2 simulation).

Loss-based flow(s) using the $red(drop)$ curve will frequently raise the average queue size above the point where $red(ecn)$ reaches 100%. The resulting high marking rates for ECN-enabled packets risks starvation of coexisting ECN flows.

To compensate, an ABE flow's β_{ecn} has to be very high (e.g. $0.95 < \beta_{ecn} < 0.975$), $\frac{th_{min_ecn}}{th_{min_drop}}$ should be higher than 0.5, and th_{min_drop} should preferably be small (e.g. $0.1 * BDP$). The first two constraints are not favorable for the aim of reducing latency, and the latter is not favorable for good link utilization (see Figure 5(b)). Moreover such high values of β_{ecn} were found to be disadvantageous in [3].

2) *Switch modes during ECN/non-ECN coexistence:* Our proposed solution is for the AQM to operate in two distinct

modes depending on whether recent traffic is made up solely from ECN-enabled flows or also contains non-ECN flows. Expressed in terms of dual-threshold RED: Use the $red(ecn)$ curve when *only* ABE flows are present at the bottleneck queue otherwise fall back to using higher threshold $red(drop)$ curve for *both* marking *and* dropping when non-ECN packets are also observed passing through the bottleneck.

Using the same $red(drop)$ curve for both marking and dropping during periods of coexistence avoids starvation of ABE flows. The impact of ABE on flow latency reduction is moot, as during periods of coexistence the queuing delay for all flows would be dominated by the competing loss-based flows being managed by the $red(drop)$ curve.

Data: *count_noecn*: number of non-ECT-enabled packets in the queue
Result: *p*: early drop probability
header* hf = header(pkt);
if hf $\rightarrow$ ect() **then**
 if *count_noecn* > 0 **then**
 | *p* = red(drop);
 else
 | *p* = red(ecn);
 end
else
 | *p* = red(drop);
end

Algorithm 1: Choosing a *red()* probability curve for ECN-marked (ECT) packets based on estimated coexistence.

Algorithm 1 shows how such mode switching may be achieved. A counter (*count_noecn*) tracks the number of non-ECT-marked⁵ packets in the queue at any given time. When a non-ECT-marked packet arrives, the queue uses an early drop probability based on *red(drop)* (standard RED). When an ECT-marked packet arrives, the queue marks according to the *red(ecn)* curve if *count_ecn* = 0 (no co-existing traffic) or according to the *red(drop)* curve if *count_noecn* > 0 (there is coexisting loss-based traffic in the queue).

Preliminary experiments with a dual-threshold RED AQM modified per Algorithm 1 revealed a significant weakness. Back-to-back packet trains from loss-based flows would often enter and depart the queue in bursts. In response, Algorithm 1 incorrectly perceives the queue to be briefly devoid of loss-based competing traffic, switches out of coexistence mode and starts marking ECN-enabled traffic on the low threshold (*red(ecn)*) curve (despite the fact they are still competing against a loss-based flow at larger time scales), leading to increased unfairness in favour of ECN-enabled flows.

3) *Delayed switching out of coexistence mode:* Algorithm 2 addresses the problem by introducing hysteresis – a time *interval* imposing a lower bound on how soon RED can come out of coexistence mode after observing *count_noecn* = 0.

When an ECT-marked packet arrives the coexistence mode flag (*coex*) is set if *count_ecn* > 0. If *count_noecn* subsequently becomes 0 while *coex* is set (i.e., we just left the coexistence mode) a timestamp for exiting coexistence mode is recorded in *t_nocoex* and the *coex* flag is unset (reflecting the new, non-coexistence state of the queue).

When the difference between current packet arrival time and the time of entering non-coexistence mode (*now* – *t_nocoex*) exceeds *interval*, we conclude (a) the queue has been without loss-based traffic for a *sufficient* amount of time to infer there is no competing loss-based traffic and (b) we are free to use the *red(ecn)* curve for marking ECN-enabled packets.

We now evaluate Algorithm 2 with combinations of ABE-enabled and loss-based TCP CUBIC⁶ flows. For the case where one ABE flow coexists with one loss-based flow,

⁵ECT-marked indicates a packet of a TCP flow where ECN was successfully negotiated. Non-ECT-marked means no ECN was negotiated.

⁶Chosen because, in addition to being the default TCP in Linux for many years with $\beta = 0.7$, it has been shown to not starve coexisting loss-based TCP flows while being more detrimental to coexisting ECN flows than NewReno.

Figure 6 shows Jain’s fairness index (JFI), the Throughput Ratio (ABE flow versus loss-based flow), Median Queuing Delay and Link Utilization for a range of ECN Threshold Ratios (with *th_min* = $0.5 * BDP$) and Algorithm 2 *intervals* (labelled “ECN Coexistence Interval”). We explored the impact of using other *th_min* values in Figure 7, although due to space constraints we only provide Throughput Ratio results.

From Figure 6 we can clearly see that fairness can be achieved for all ranges of ECN Threshold Ratios when *interval* > 100 ms (or more generally, *interval* > *RTT*). The specific choice of interval is not critical to fairness so long as it is sufficient, but it should not be too large, to ensure ABE flows achieve latency gains within a reasonable time.

Data: *count_noecn*: number of non-ECT-enabled packets in the queue
coex: coexistence mode (set to *TRUE* if there is a non-ECT packet in the queue)
t_nocoex: duration of non-coexistence mode (when *coex* = *FALSE*)
interval: ECN coexistence interval
Result: *p*: early drop probability
header* hf = header(pkt);
if hf $\rightarrow$ ect() **then**
 if *count_noecn* > 0 **then**
 | *coex* = *TRUE*;
 | *p* = red(drop);
 else
 if *coex* == *TRUE* **then**
 | *t_nocoex* = now;
 end
 coex = *FALSE*;
 if now – *t_nocoex* > *interval* **then**
 | *p* = red(ecn);
 else
 | *p* = red(drop);
 end
 end
else
 | *p* = red(drop);
end

Algorithm 2: Choosing a *red()* probability curve based on estimated coexistence and hysteresis interval.

Figure 6 reveals that Algorithm 2 allows the use of very low ECN Threshold Ratios, providing the best latency gain without jeopardizing fairness and with no starvation for any type of traffic. Figure 7 shows a similar pattern for when RED’s *th_min* values is decreased to $1/4 * BDP$ and $1/8 * BDP$ (which we know brings down the entire queuing region for both types of traffic).

It can also be observed that fairness is achievable though it is favorable to set the interval to a value slightly larger than an RTT to achieve a better fairness level for low standard RED’s *th_min* values. The effect of this slight increase in interval will be negligible on long-lived flows that contribute most to the latency, as the interval only controls how soon the queue begins to mark ECN flows using the *red(ecn)* curve after the the queue no longer contains non-ECN traffic.

A useful scheme should achieve fairness with more than two flows, so we also investigated the performance of Algorithm 2 for three different combinations of {1,9}, {5,5} and {9,1} competing ABE versus loss-based flows (always ten flows

in total). Figure 8 shows the Throughput Ratio for these three scenarios, for a range of ECN Threshold Ratios (with $th_{min} = 0.5 * BDP$) and *intervals*.

Figure 8 highlights the fact that Algorithm 2 achieves reasonable fairness (Throughput Ratio close to 1.0) for all three combinations of flows when $interval > RTT$. Figure 8(c) in particular reassures us that $interval > RTT$ will protect multiple ABE flows from being penalized by a single TCP flow (which is an important consideration if ABE flows are to become more common across the Internet).

Finally, we highlight the significant latency benefits of deploying Algorithm 2 on access links. Figure 9 captures the median queuing delays experienced by one (light congestion) and ten (heavy congestion) ABE-enabled CUBIC flows for varying ECN Threshold Ratios and $th_{min} = \{0.125, 0.25, 0.5\} * BDP$. For the light congestion scenario (Figure 9(a)) we can bring down the queuing delay to *zero* for threshold ratios below 0.55. For the heavy congestion scenario (Figure 9(b)), latency can be decreased by 48%~95% depending on the choice of RED's th_{min} value. This degree of latency reduction will be a perceivable benefit in many use cases ranging from web access and real-time multimedia to online gaming competing with the uplink TCP traffic.

To provide a better understanding of the Algorithm 2's operation in simpler terms, let us assume an example of a common home scenario (a last-hop access link with the modem's uplink interface being the bottleneck) where two persons share the Internet access together – with one person using a non-ECN-enabled device while the other using an ECN-enabled device (i.e. with ABE). When two persons use the Internet concurrently (e.g. uploading some files at the same time) and therefore sharing the bottleneck, Algorithm 2 will indeed almost always perform in non-ECN-mode (using $red(drop)$). But whenever only person(s) using ECN-enabled devices are using the Internet alone (e.g. when the user of non-ECN-enabled device is away or using the Internet passively), Algorithm 2 will also achieve low latency for all of their traffic (by using (using $red(ecn)$)).

C. Augmenting other AQM schemes

Our proposal is not specific to dual-threshold RED. The essence of Algorithm 2 is applicable to any AQM scheme that can be augmented to mark packets from ECN-enabled flows using lower thresholds than they use to drop packets from non-ECN flows. One might envisage, for example, a CoDel implementation having separate $target_delay_{ecn}$ and $target_delay_{drop}$ thresholds, and switching its drop/mark decision making between them according to Algorithm 2.

IV. A WAY AHEAD FOR ECN

Defined by RFC 3168 [2] in 2001, ECN capabilities exist today in most mainstream routers and host OS network stacks, yet ECN-enabled flows are distressingly rare [16]. This has partly been due to early experiences showing faulty behaviour of some middleboxes resulting in failure of ECN-enabled TCP connections. However, this was reduced to $< 1\%$ of the tested

web servers in 2004 [17] from 8% in 2000 [18], although there has been a non-trivial number of paths mangling the ECN field in the IP header in 2011 [19]. Extensive active measurements in 2014 showed that the majority of the top million web servers provided ECN negotiation when requested by the client [16].

ABE and our Algorithm 2 enhancements to AQMs offer new benefits to enabling ECN, but at least one detail about ECN negotiation (Section II-A) is currently a stumbling block to wider adoption. In this section we briefly reflect on changes to [2] that could encourage more incremental growth in ECN use across the Internet.

A. Enabling Server-initiated ECN

In a typical client-server application scenario it is the client (usually not sending much traffic) who is responsible for initiating ECN negotiation. According to [2] the recipient of the client's initial TCP SYN (the application layer server, usually sending lots of traffic) is simply *not allowed* to unilaterally enable ECN if the client (connection initiator) has not requested it. Unless the client is predisposed to requesting ECN on all new connections (and most clients across common OSes currently are not), server-to-client traffic flow never reaps the benefits of ECN being enabled. This is even more unfortunate if the server's TCP stack implements ABE, as the client has no way to know this a priori. Simply encouraging client OSes to request ECN on all connections has not worked well to date, so we are left with changing the RFC 3168 handshake.

One solution is to allow a connection recipient to send an ECN-setup SYN/ACK packet even without having received an ECN-setup SYN packet, and use the initiator's subsequent ACK to confirm the client will answer CE-marks with ECE.

Another approach is to update the encoding of ECN-setup in the connection initiator's SYN packet such that only “new” hosts understand it. These hosts would then be assumed to be used in a configuration where ECN is expected to be beneficial, i.e. they would use ABE and likely exist behind AQM bottlenecks who embrace the coexistence-dependent, lower-threshold ECN marking of Algorithm 2.

B. ECN backwards compatibility

Our recommendation for AQMs to early-mark ECN flows during periods of non-coexistence is based on the assumption that future ECN flows will be initiated by TCP stacks implementing ABE. However, there remains the possibility of a legacy TCP stack enabling ECN (i.e. leaving $\beta_{ecn} = \beta_{loss}$ and not implementing ABE or something ABE-like). Such flows will experience lower throughput when coexisting in bottlenecks that implement Algorithm 2.

We could simply ignore such cases – as ECN is *hardly used* to date anyway the harm done would be minimal. Alternatively, we could re-purpose the ECT(1) codepoint so that ABE-enabled senders can signal bottlenecks to “early-mark this packet since it comes from a sender with an aggressive congestion control”. ECT(1) was originally introduced in [2] in order to implement the ECN nonce. The nonce is only

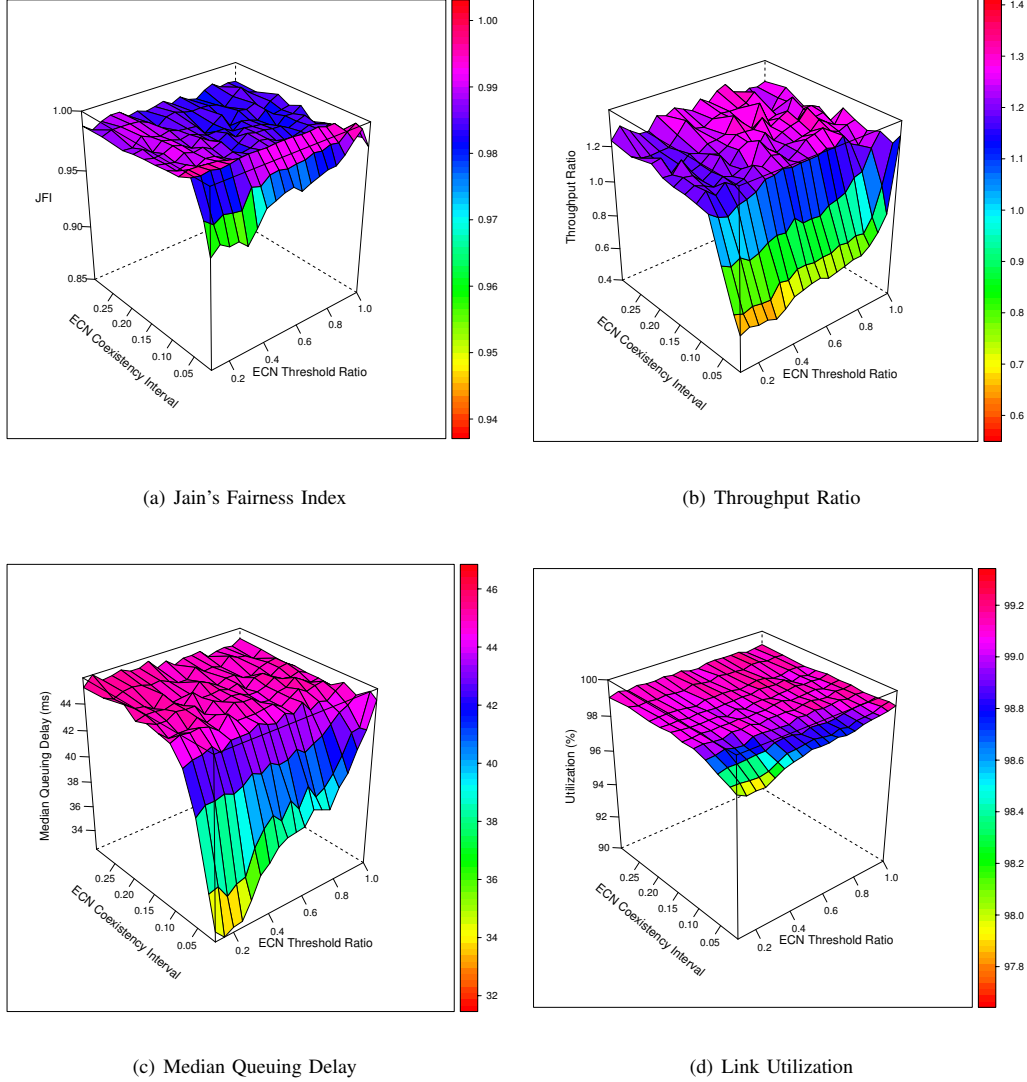


Fig. 6. One ABE (CUBIC) flow and one standard loss-based TCP flow with different ECN threshold ratios and $th_{min_drop} = 0.5 * BDP$ when mode switching with “interval” was used at AQM mechanism (ns-2 simulation).

useful if deployed at both ECN sender and receiver, and has not seen widespread deployment. With this enhancement, an AQM bottleneck would provide its “traditional” ECN marking for ECT(0) (regular ECN) packets.

Admittedly, using ECT(1) in this way is at odds with the ECN nonce idea of randomly choosing between ECT(0) and ECT(1). However, we can maintain a weaker variant of the nonce by *mostly* marking packets with ECT(1), at the risk of seeing larger queue growth from only a small fraction of ECT(0) packets. If, for example, 1 out of 10 packets is sent with ECT(0), a malicious receiver still has only a $1/10$ chance of guessing the correct nonce in this sequence. As a sender need only detect a lying receiver once during a connection’s lifetime, we might alter the $ECT(0)/ECT(1)$ ratio to initially send more ECT(0) packets early then send more

ECT(1) packets later in the connection’s lifetime. The nonce’s role would be fulfilled early, while the benefits of an ECT(1) marked ABE-enabled flow are realized over the longer term.

V. CONCLUSION

We have proposed and evaluated a modification to bottleneck-based AQMs to be used in conjunction with the end-host modification ABE [3]. In short, AQM bottlenecks (regardless of their specific AQM strategy, such as triggering congestion signals based on local queue length in bytes or time) should apply earlier (more aggressive) marking to ECN-enabled flows, but only when no non-ECN flows have recently coexisted in the bottleneck for a configurable period of time.

Our proposal provides improved fairness between TCP flows from traditional senders and TCP flows from ABE-

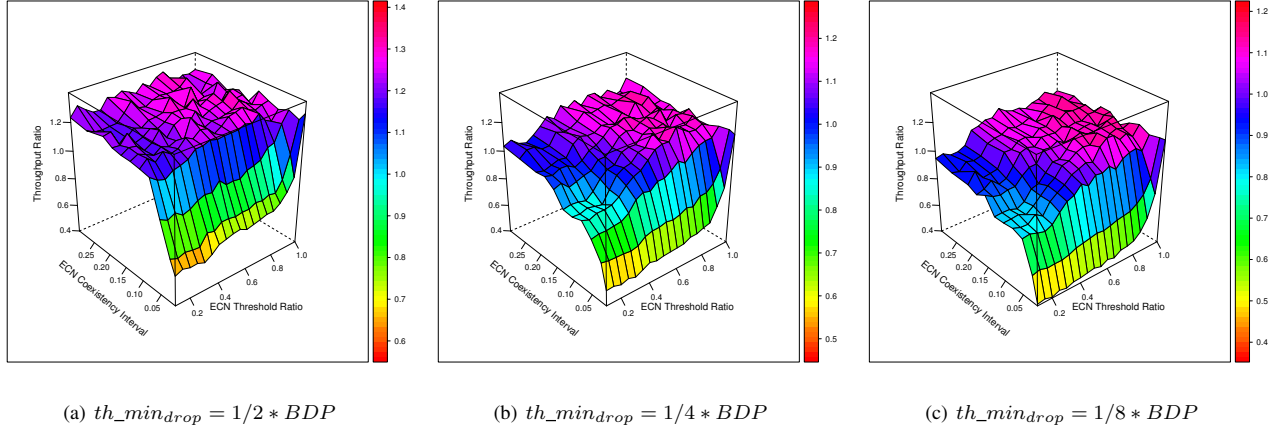


Fig. 7. Throughput ratio between one ABE (CUBIC) flow and one standard loss-based TCP flow with different th_{min_drop} values when mode switching and “interval” was used at AQM mechanism (ns-2 simulation).

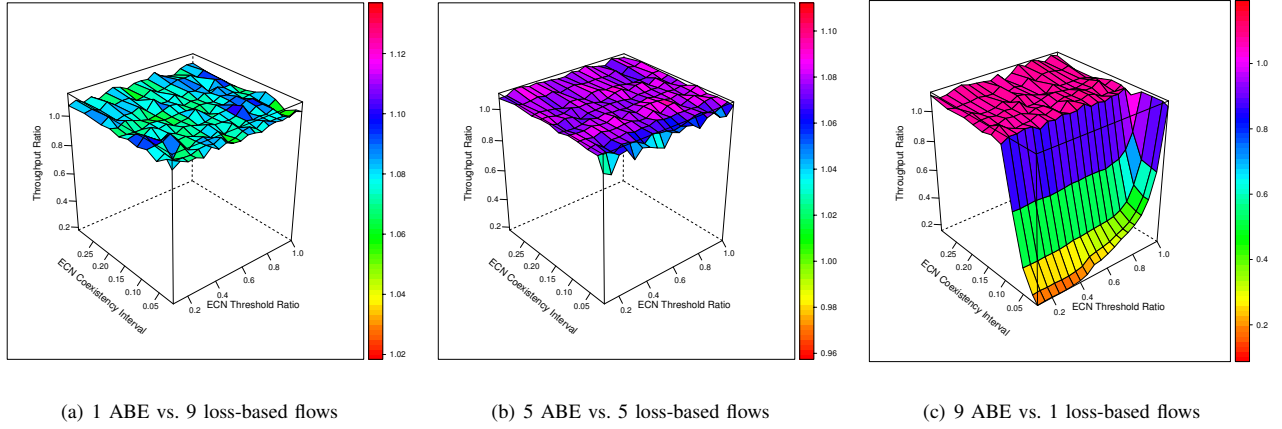
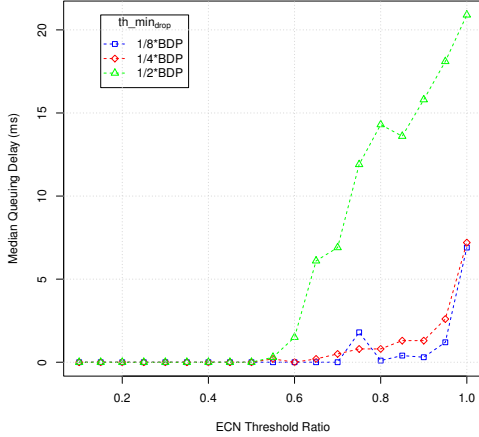


Fig. 8. Throughput ratio between ABE (CUBIC) and standard loss-based TCP flows with different ECN threshold ratios and $th_{min_drop} = 0.5 * BDP$ when mode switching and “interval” was used at AQM mechanism (ns-2 simulation).

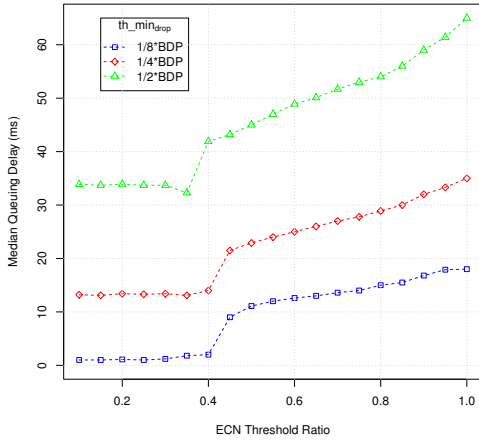
enabled senders. Both ABE and our proposal can be incrementally rolled Internet-wide without flag-days, providing incentives to both deploy AQMs in bottlenecks and enable ECN far more widely. Together AQMs and ECN are technologies that will significantly reduce the latencies experienced by all Internet applications.

REFERENCES

- [1] 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), Internet Engineering Task Force, Apr 1998. [Online]. Available: <http://www.ietf.org/rfc/rfc2309.txt>
- [2] K. Ramakrishnan, S. Floyd, and D. Black, “The Addition of Explicit Congestion Notification (ECN) to IP,” RFC 3168 (Proposed Standard), Internet Engineering Task Force, Sep 2001, updated by RFCs 4301, 6040. [Online]. Available: <http://www.ietf.org/rfc/rfc3168.txt>
- [3] N. Khademi, M. Welzl, G. Armitage, C. Kulatunga, D. Ros, G. Fairhurst, S. Gjessing, and S. Zander, “Alternative Backoff: Achieving Low Latency and High Throughput with ECN and AQM,” Centre for Advanced Internet Architectures, Swinburne University of Technology, Australia, Tech. Rep. 150710A, 10 July 2015. [Online]. Available: <http://caia.swin.edu.au/reports/150710A/CAIA-TR-150710A.pdf>
- [4] N. Khademi, M. Welzl, G. Armitage, and G. Fairhurst, “TCP Alternative Backoff with ECN (ABE),” IETF Draft, May 2016. [Online]. Available: <https://tools.ietf.org/html/draft-khademi-tcpm-alternativebackoff-ecn-00>
- [5] S. Floyd, R. Gummadi, and S. Shenker, “Adaptive RED: An Algorithm for Increasing the Robustness of RED’s Active Queue Management,” ICIR, Tech. Rep., 2001. [Online]. Available: <http://www.icir.org/floyd/papers/adaptiveRed.pdf>
- [6] K. Nichols and V. Jacobson, “Controlling Queue Delay,” *ACM Queue*, vol. 10, no. 5, pp. 20:20–20:34, May 2012. [Online]. Available: <http://doi.acm.org/10.1145/2208917.2209336>
- [7] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, “Controlled Delay Active Queue Management,” IETF Draft, Jun 2016. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-aqm-codel-04>
- [8] R. Pan, P. Natarajan, F. Baker, G. White, B. VerSteeg, M. Prabhu, C. Piglion, and V. Subramanian, “PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem,” Aug 2016. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-aqm-pie-09>
- [9] R. Adams, “Active Queue Management: A Survey,” *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1425–1476, Jul 2013.
- [10] G. Appenzeller, I. Keslassy, and N. McKeown, “Sizing Router Buffers,” in *ACM SIGCOMM*, Portland, Oregon, USA, Sep 2004.
- [11] P. McKenney, “Stochastic Fairness Queueing,” in *IEEE INFOCOM*, San Francisco, California, USA, Jun 1990.



(a) One ABE flow (light congestion)



(b) Ten ABE flows (heavy congestion)

Fig. 9. Median queuing delay experienced when one or ten ABE-enabled CUBIC flow(s) enter a RED bottleneck using Algorithm 2 with $interval=100$ ms, versus ECN Threshold Ratios and th_min (ns-2 simulation).

- Communication Review, vol. 35, no. 2, pp. 37–52, Apr 2005. [Online]. Available: <http://doi.acm.org/10.1145/1064413.1064418>
- [18] J. Padhye and S. Floyd, “Identifying the TCP Behavior of Web Servers,” in *ACM SIGCOMM*, Stockholm, Sweden, Aug 2000.
- [19] S. Bauer, R. Beverly, and A. Berger, “Measuring the State of ECN Readiness in Servers, Clients, and Routers,” in *ACM SIGCOMM Conference on Internet Measurement (IMC)*, Berlin, Germany, Nov 2011. [Online]. Available: <http://doi.acm.org/10.1145/2068816.2068833>
- [20] M. Carbone and L. Rizzo, “Dummynet revisited,” *SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, Apr 2010. [Online]. Available: <http://doi.acm.org/10.1145/1764873.1764876>
- [21] PIE Linux Code (Cisco). ftp://ftpeng.cisco.com/pie/linux_code.
- [22] S. Zander and G. Armitage, “Minimally-Intrusive Frequent Round Trip Time Measurements Using Synthetic Packet-Pairs,” in *IEEE Conference on Local Computer Networks (LCN 2013)*, Sydney, Australia, Oct 2013. [Online]. Available: <http://dx.doi.org/10.1109/LCN.2013.6761245>

APPENDIX

The experimental network is two software routers (one an AQM bottleneck feeding 100Mbps traffic to a 10 Mbps link, the other using *ipfw*’s dummynet [20] in each direction to provide an artificial $RTT_{base} = 100$ ms) and multiple hosts connected to the routers via 100 Mbps links to act as TCP flow senders and receivers. Machines were Dell OptiPlex GX620 (Pentium 4 @ 3 GHz, 1 GB RAM). The AQM router ran Linux kernel 3.10.4 (FC16), using CoDel and ARED present in the 3.10.4 kernel and PIE code added from [21].

Experiments consisted of 180 sec of *iperf* traffic between each sender/receiver pair, repeated 10 times and averaged across all runs. TCP-Segmentation-Offload (TSO) was disabled on all NICs, the bottleneck’s AQM *qdisc* and maximum queue size (*txqueuelen*) were both set to 1000 packets and Byte-Queue-Limit (BQL) was 1514 bytes. Hosts ran NewReno with appropriate-byte-counting (ABC), Selective ACK (SACK) and Duplicate-SACK options enabled. TCP send and receive buffers were larger than the path’s bandwidth-delay product (BDP).

Per-packet RTTs between hosts were estimated using Synthetic Packet Pairs (SPP) [22] on traffic captured by *tcpdump* on all NICs. TCP throughput was calculated using *tcptrace*. All AQMs used default settings unless otherwise noted (with ARED parameters based on the recommendations in [5]).

- [12] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, “FlowQueue-Codel,” IETF Draft, March 2016. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-aqm-fq-codel-06>
- [13] S. Floyd and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance,” *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug 1993.
- [14] M. Kühlewind, D. P. Wagner, J. M. R. Espinosa, and B. Briscoe, “Using data center TCP (DCTCP) in the Internet,” in *IEEE Globecom Workshops (GC Wkshps)*, Austin, Texas, USA, Dec 2014.
- [15] K. D. Schepper, B. Briscoe, O. Bondarenko, and I. Tsang, “DualQ Coupled AQM for Low Latency, Low Loss and Scalable Throughput,” IETF Draft, March 2016. [Online]. Available: <https://tools.ietf.org/html/draft-briscoe-aqm-dualq-coupled-01>
- [16] B. Trammell, M. Kühlewind, D. Boppart, I. Learmonth, G. Fairhurst, and R. Scheffneger, “Enabling Internet-wide Deployment of Explicit Congestion Notification,” in *Passive and Active Measurement Conference (PAM)*, New York, New York, USA, Mar 2015.
- [17] A. Medina, M. Allman, and S. Floyd, “Measuring the Evolution of Transport Protocols in the Internet,” *SIGCOMM Computer*