Technical Report

Rapid Signalling of Queue Dynamics

Bob Briscoe[*]

28 Nov 2021

Abstract

This paper focuses on reducing the delay before an active queue management (AQM) algorithm emits congestion signals, e.g. explicit congestion notification (ECN) or packet drop. These algorithmic delays can be greater than the delay that the data itself experiences within the queue. Once the congestion signals are delayed, regulation of the load becomes more sloppy, and the queue tends to overshoot and undershoot more as a result, leading the data itself to experience greater peaks in queuing delay as well as intermittent under-utilization. Also, even if an AQM algorithm only slightly delays its congestion signals, it tends to shift the signals from more bursty flows onto other smoother flows.

The importance of immediate congestion signalling has been recognized in approaches such as Data Center TCP (DCTCP) and Low Latency Low Loss Scalable throughput (L4S). However, this paper points out that the sojourn time metric that is increasingly used in these approaches introduces unnecessary internal measurement delay, which is worst during bursts. Expected service time is proposed as an alternative metric. Three potential implementations are proposed, which keep some or all of the benefits of sojourn time, but without the internal measurement delay. The paper also briefly surveys ways to remove other delays within AQMs, such as the delay due to randomness in the signal encoding.

CCS Concepts

•Networks → Cross-layer protocols; Network algorithms; Network dynamics;

Keywords

Data Communication, Networks, Internet, Control, Congestion Control, Quality of Service, Performance, Latency, Responsiveness, Dynamics, Algorithm, Active Queue Management, AQM, Congestion Signalling, Sojourn time, Queue delay, Service time, Wait time, Expectation, Estimation, Blame, Fair marking, Burstiness, Cost-fairness, Explicit Congestion Notification, ECN, Packet Drop, Discard

[*]research@bobbriscoe.net,

1 Introduction and Scope

Much attention has been paid to reducing the delay experienced on the data path through packet networks. For instance, see sections II and IV of the extensive survey of latency reducing techniques in [BBP+16], which aim to reduce propagation delay, queuing delay, serialization delay, switching delay, medium acquisition delay and link error recovery delay.

This memo focuses on reducing unnecessary delays in the system that attempts to match load to capacity. Reducing queuing delay has also been the focus of much other recent work. But the focus here is on cutting delays in the control path rather than the data path. That is, delays in measuring queue dynamics and in communicating the resulting control signals.

Once congestion signals are delayed, regulation of the load becomes more sloppy, and the queue tends to overshoot and undershoot more as a result, leading the data itself to experience greater peaks in queuing delay as well as intermittent under-utilization. And, perhaps most importantly, if the congestion signals due to bursts of data are delayed, even slightly, they will be applied to the packets just after each burst. Then bursty traffic could shift much of the blame for congestion onto other traffic running more smoothly in the background.

To be concrete, this memo assumes congestion signals that are transmitted from an active queue management (AQM) algorithm [Ada13] using either drop or explicit congestion notification (ECN) [Flo94], which are the only standardized signalling protocols [RFB01] for end-to-end use over one of the two Internet protocols, IPv4 and IPv6. Nonetheless, many of the ideas concern algorithmic improvements, which could be applied in other settings with different congestion signalling.

Control path delay consists of the following elements:

- propagation delay (in common with the data);
- queuing delay (in common with the data);

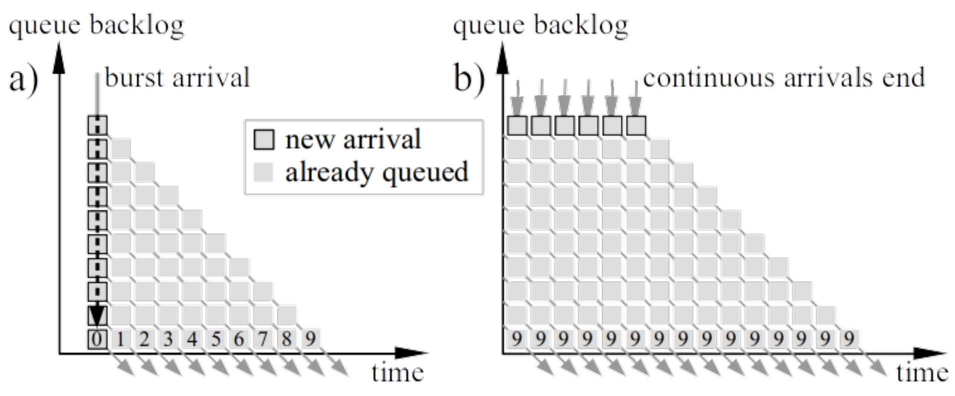

Figure 1: Schematic Illustrating Two Problems with the Sojourn Time Metric. a) It does not measure the full size of a burst until the end (left); b) It does not measure a draining queue (right). Draining is visualized at one equisized packet per timeslot. Sojourn time is represented just before each packet is dequeued as the number of timeslots along its diagonal path.

- measurement delay: measuring the queue, as well as arrival and/or departure rates;
- smoothing delay: filtering out fluctuations in measurements;
- signal encoding delay: a number representing the signal is produced within an AQM algorithm, which is then compressed into a unary 'encoding' in each packet, and 'decoded' by the congestion control algorithm's response to the unary-encoded signals. A unary encoding is used so that the AQM does not have to recognize flows or hold per-flow state. But it constrains the bandwidth of the signalling channel, which introduces encoding delay;
- randomization delay: randomness is introduced to break up oscillations, but it requires longer to detect the underlying signal.

This memo pays most attention to two of these: measurement and queuing delay. The other four are briefly surveyed in § 7.

2 The Problem

2.1 The Problem in Brief

The signal from an AQM can be subject to unnecessary **queuing delay** if it is applied to a packet during the enqueue process, so that it has to work its way through the queue before being transmitted to the line. In classical AQMs, queuing delay is configured to be of the same order of magnitude as typical propagation delays. Therefore subjecting the congestion signal to the delay of the queue will add unnecessary sloppiness to the control loop.

Even if a signal is applied to a packet as it is dequeued, it is often based on a **measurement** that has taken some time to collect. For instance, the sojourn time technique, which is becoming common for measuring the queue in modern AQMs, gives a queue delay metric that is always out of date by the amount of time the packet spent in the queue. So even if the signal is applied at dequeue, it is delayed by the time spent measuring it in the queue.

The sojourn time measured when a packet reaches the head of the queue takes no account of any change in the queue while that packet is working towards the head. Despite the queuing system holding all the information about those recent changes. So if a burst arrives, the sojourn time of packets in front of the burst will show no evidence of the queue building behind them. For instance, in Figure 1a), the sojourn of the first packet of the burst takes '0' timeslots. And the sojourn time metric only fully measures the burst when the last packet of the burst reaches the head of the queue, where '9' is shown. Even though, in this case, all the packets of the burst had arrived before the packet tagged '1' was dequeued. In § 2.3 we will consider a mix of smooth and bursty traffic, then we will see how this delay attributing blame for a burst tends to shift the blame from bursty to smooth traffic.

Whenever the drain rate abruptly reduces, use of sojourn time is similarly problematic. As soon as the reduction occurs, the time that packets will take to work through the queue increases. But, the sojourn time of those packets that have already worked their way towards the head of the queue does not reflect the delay that will be experienced by the packets behind them.

Conversely, consider the queue in Figure 1b) that has been stable then the data flow ends abruptly, so that no further packets arrive. Then, even when the last packet to arrive is about to be dequeued from the head of the queue, its sojourn time still measures the stable queue delay when it arrived, because that's how long it took to drain the queue. If sojourn alone were used for marking and dropping, there would be no externally visible evidence of the now empty queue behind the last packet until traffic started again.

Implementations of AQMs that use sojourn time sometimes include code to deal with certain exceptional cases (such as an empty queue). But this memo proposes simple techniques to cut out the root cause of that measurement delay in all cases; by using all the information available in the queue at the point that a packet is dequeued. At dequeue there is very little time for additional processing, so considerable attention has had to be paid to minimizing execution time as well.

2.2 The Time to Measure the Service Time of a Queue

In around 2012, it became recognized that one of the main problems with AQMs was the sensitivy of their configuration to changing environments. For example:

- access links often change their rate when modems retrain in response to interference.
- a queue can be part of a scheduling hierarchy and traffic in higher priority queues varies the capacity left for a lower priority queue, rapidly varying the drain rate that the AQM experiences.
- the capacity of radio links varies rapidly over time [MS10].

The CoDel algorithm [NJ12] proposed to solve this problem by measuring the queue in units of time, rather than bytes. This made the configuration of the thresholds in the algorithm independent of the drain rate.

Actually, as far back as 2002, Kwon and Fahmy [KF02] had advised that the queue should be measured in units of time. Also, in 2003, Sågfors *et al* had modified the Packet Discard Prevention Counter (PDPC+ [SLMP03]) algorithm by converting queue length to queuing delay to cope with the varying link rate of 3G wireless networks. PDCP still measured the queue in bytes, but then converted the result to time by dividing by the link rate, which it measured over a brief interval.

CoDel proposed an elegant way to measure the service time of the queue by adding a timestamp to each packet's internal metadata on enqueue. Then at dequeue, it subtracted this timestamp from the system time. The authors called the result the sojourn time of the packet. It was also pointed out that this sojourn time could be measured over an arbitrarily complex structure of queues, even across distributed input and output processors.

Because PIE [PPP+13] was initially designed for implementation using existing hardware, it did not measure the service time of the queue directly using the time-stamping approach of CoDel. Instead, like PDPC, it converted queue length to queuing delay using a regularly updated estimate of the link rate, measured over a set amount of packets. When there were insufficient packets in the queue to measure the link rate or the rate was varying rapidly, PIE's estimate of the link rate became stale. So in later specifications of PIE [PNB+17], it recommended the sojourn approach of CoDel that had originally been designed for software implementation.

The queue length (in bytes or an equivalent unit), also called the backlog, can be measured instantaneously when a packet is enqueued or when it is dequeued. In contrast, it takes a sojourn time to measure sojourn time (which can only be measured as a packet is dequeued). So measuring sojourn time inherently introduces delay into the control path.

To minimize delay, the signal should be applied at dequeue. However, in some hardware pipelines the process of preparing link layer frames, including potential encryption, compression and framing, is already in progress by the time a packet is dequeued. So it is too late to mark or drop a packet. This is one reason that PIE initially applied the congestion signal when it enqueued a packet. That is, it probabilistically dropped (or ECN-marked) the packet when it enqueued it. This signal then worked its way through the queue before being transmitted, adding a sojourn time of delay to the signal. This is still the case for DOCSIS PIE, but software variants of PIE now apply marking or dropping at dequeue.

The matrix in Table 1 shows the delay added to the signal by various techniques for measuring queue delay that will be introduced later (horizontal) and the two choices for where to apply the signal (vertical). It uses the following terminology: t_r is the duration used to sample the drain rate and t_s is the sojourn time.

The centre column shows the effective delay added by the simple 'Time-Based Backlog' technique proposed in § 4.1. It also applies to the variant of that technique called 'Size-Adjusted Threshold' in § 4.1.2. The right hand column shows the delay of a technique called 'Scaled Sojourn Time' introduced in § 4.2, which can be used where the ability of sojourn time to measure delay across a complex set of

Where signal is applied	Technique to measure queue delay		
	Sojourn Time	Expected Service Time	
		Time-Based Backlog	Scaled Sojourn Time
at enq	$2t_s$	$t_r + t_s$	$3t_s/2$
at deq	t_s	t_r	$t_s/2$

Table 1: Delay added to congestion signal by three different measurement techniques

queues is required. Nonetheless, for a simple software queue, the time-based backlog is preferable, because it always adds minimal measurement delay.

It can be clearly seen that applying a signal at enqueue adds t_s to the signal delay. So applying the signal at enqueue would only be appropriate if it were not possible to mark (or drop) a packet at the head of the queue, e.g. due to implementation or timing constraints.

2.3 The Blame Shifting Problem

When the Random Early Detection (RED) AQM algorithm was first proposed, fairness was one evaluation factor [FJ93, § 8]. Fairness in the context of marking was defined as "the fraction of marked packets for each connection is roughly proportional to that connection's share of the bandwidth".

Such fairness would be sufficient if all flows were long lived and smooth, but they are not. Wischik [Wis99a] contrives a simple two-flow scenario to demonstrate how RED can shift nearly all the marking from a burst in one flow onto a smoother flow that continues after the burst.

Classical AQMs like RED or more recent designs such as CoDel or PIE are designed to filter out variations in the queue over a likely maximum round trip. So they inherently introduce smoothing delay of about 100–200 ms prior to signalling congestion, which is far too long to be able to mark packet bursts correctly. More recently, the importance of immediate congestion signalling has been recognized in approaches such as Data Center TCP (DCTCP [AGM+10]) and Low Latency Low Loss Scalable throughput (L4S [BEDSB17]), where the job of smoothing out variations is shifted from the AQM to the sender.

Marking fairness has been termed 'shadow pricing' [KMT98] or 'cost fairness' [Bri07], because it ensures the cost or harm to others of each user's behaviour can be measured. Note that marking fairness is an important mechanism requirement for all AQMs and should not be confused with flow-rate 'fairness', which is an arbitrary policy choice concerning allocation of benefits at any instant (which is why 'fairness' is placed in quotes in this latter case)[1].

This section introduces fairness problems when sojourn time is used to mark flows with different degrees of burstiness. Then it uses a worked example to give better intuition for how to make marking fairer. The question of what marking would actually be fair for different degrees of burstiness is deferred to a later discussion section (§ 5).

Nonetheless, we will now demonstrate that even the delay spent measuring sojourn time is enough to cause immediate marking to miss packet bursts, and hit smoother flows instead. Although we cannot yet say exactly what is fair, we can recognize this shift of blame, when one marking approach is significantly less fair than another.

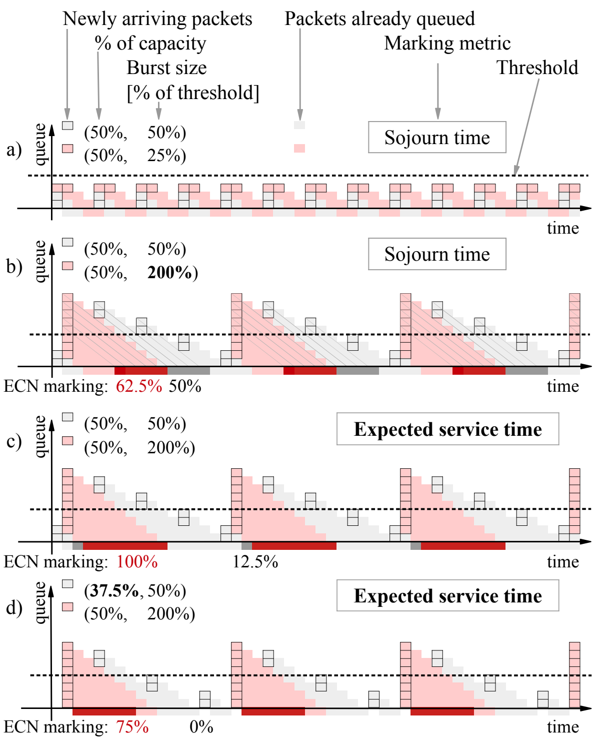

Figure 2: The Marking Fairness Problem with the Sojourn metric (b) and the ECN marking outcomes of a Solution (c–d). The dark red and dark grey packets below each x-axis indicate congestion-marked packets (see text for explanation and commentary).

Figure 2 shows how a packet queue behaves in a

[1] The RED paper went on to explain that "RED gateways do not attempt to ensure that each connection receives the same fraction of the total throughput".

progression of four scenarios reduced to their essentials by using equal-sized packets; one timeslot per delivered packet; a constant rate link; a step ECN marking threshold (the dashed horizontal line); and high link utilization but without modelling the interaction with the sender's congestion control.

In the first three scenarios (a–c), two flows (grey and pink) fully utilize the link, consuming 80% and 20% each. Newly arriving packets have a border while packets already queued do not, as shown in the legend above the top scenario. In all the scenarios, the grey packets arrive in bursts, each the size of half the queue delay threshold in the buffer. Other aspects of the traffic and marking algorithm change over the progression of scenarios, as highlighted in bold in each row and described below:

a) This is the baseline case to show that the pink flow can fill the capacity left by the grey flow without exceeding the threshold, as long as it keeps its bursts small enough (in this case just 25% of the threshold, making at most 75% if a pink burst were to coincide with a grey one).
b) In this case, each burst from the pink flow occupies 200% of the threshold in the buffer and happens to arrive when one packet of a grey two-packet burst is left in the queue. The average rate of both flows is unchanged, so the link is still 100% utilized, and the grey flow still arrives in the same pattern of small bursts. So, in the time the pink burst takes to drain, some grey bursts back up behind it and, while they are draining, more small grey bursts accumulate. So the queue only finally empties just as the cycle starts to repeat with the arrival of the next pink burst.
c) The arrivals in this scenario are identical to (b), but sojourn-based queue measurement is replaced with expected service time (EST—described below).
d) In this last case, the pink flow stays unchanged at 20% of the link and still in large bursts twice the depth of the threshold. However, the flow rate of the grey flow reduces from 80% to 70% of the link by halving every fourth burst. This is an attempt to represent the grey flow starting to respond to ECN marking, while the pink flow remains unresponsive.

The ECN marking algorithm is identified in the box above each scenario:

In scenario b) in Figure 2, each diagonal darker line traces the sojourn through the queue of those packets that arrive when the queue is above the threshold. The **sojourn time** of these packets will have exceeded the threshold so they will be marked on departure (shown under the axis as darker coloured packets). The proportion of marked packets of each colour is written under that.

In scenarios c) or d) there are no diagonal lines, because the marking algorithm uses the **expected service time** of the queue based on its instantaneous depth at departure. So departing packets are marked (darker coloured) if the vertical height of the queue in the timeslot just before departure exceeds the threshold.

In this second pass, we describe the ECN marking outcome from each scenario as follows:

a) There is no ECN marking in this baseline case (whatever the metric).
b) The sojourn times of 5/8 of the packets at the tail of the pink burst exceed the threshold. So the sojourn-based AQM marks 62.5% of the pink packets at dequeue. Because grey packets back up behind the pink burst, then behind themselves, their sojourn time exceeds the threshold for the first 62.5% (5/8) of the grey packets between each pair of pink bursts. Thus, the marking probability of the two flows is equal, even though the pink flow is a lot more bursty.
c) Here the evolution of the queue is identical to scenario (b), but marking is based on a metric we call 'expected service time' rather than sojourn time. This will be fully explained in § 4, but for now it is enough to say that it marks a packet just before it is dequeued based on whether the backlog *at that instant* is expected to need longer than the threshold to drain. This increases the pink marking probability from 62.5% to 100%. In contrast, marking shifts away from grey packets, as grey marking reduces from 62.5% to 50%.
d) As the load from grey traffic falls a little, the queue falls below the threshold considerably sooner, considerably reducing the grey marking probability. Indeed a reduction of the grey traffic by 1/8 reduces grey marking probability by more than 3/8 (from 50% to below 29%). Nonetheless, the AQM still marks 100% of the pink packets, because the backed up grey packets still exceed the threshold for the whole period while the pink burst is draining.

The wider space of scenarios like this has been investigated by varying the relative shares and relative burst sizes between flows (see § 5). Although the difference between the sojourn and expected service time metrics is sometimes less dramatic and sometimes more, the following intuition is generally true for all scenarios.

A smoother flow has smaller but more frequent arrival events compared to a bursty flow. So, although the bursty flow might happen to back up behind one of the smaller bursts of the smoother flow, multiple arrival events of the smoother flow will back up behind the larger burst. Therefore proportionately more of a bursty flow will be near the head of the buffer, and the packets of a smoother flow will be more likely to be occupying the tail.

Therefore, marking the head packet when there is an excessive backlog behind it is likely to lead to fairer marking than marking the head packet because it had an excessive backlog in front of it (during its sojourn through the queue).

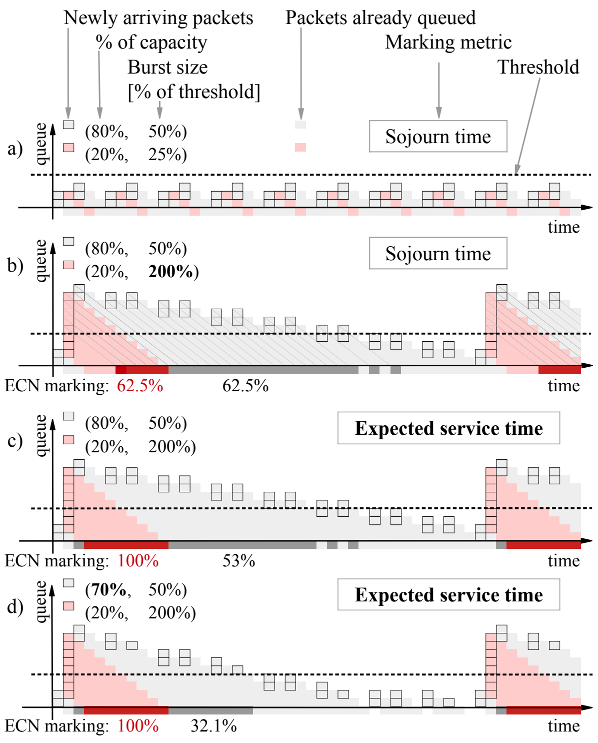
Newly arriving packets
% of capacity
Burst size
[% of threshold]
Packets already queued
Marking metric
Threshold
a) queue (80%, 50%) (20%, 25%) Sojourn time time
b) queue (80%, 50%) (20%, 200%) Sojourn time
ECN marking: 62.5% 62.5% time
c) queue (80%, 50%) (20%, 200%) Expected service time
ECN marking: 100% 53% time
d) queue (70%, 50%) (20%, 200%) Expected service time
ECN marking: 100% 32.1% time

Figure 3: TBA

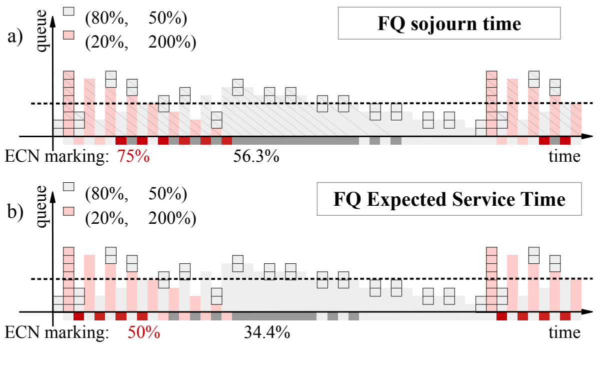
a) queue (80%, 50%) (20%, 200%) FQ sojourn time
ECN marking: 75% 56.3% time
b) queue (80%, 50%) (20%, 200%) FQ Expected Service Time
ECN marking: 50% 34.4% time

Figure 4: TBA

3 Implications of Unfair Marking

The section “Underutilization with Bursty Traffic” in [HM21] injects large bursts of unresponsive traffic into a queue that applies immediate ECN marking above a shallow threshold based on the sojourn metric. When there is also smoothly paced traffic in the same queue, the bursts cause the AQM to focus ECN markings onto the smooth traffic, not the bursts (even though the bursts would not respond to them). This is a good example of a serious implication of unfair marking.

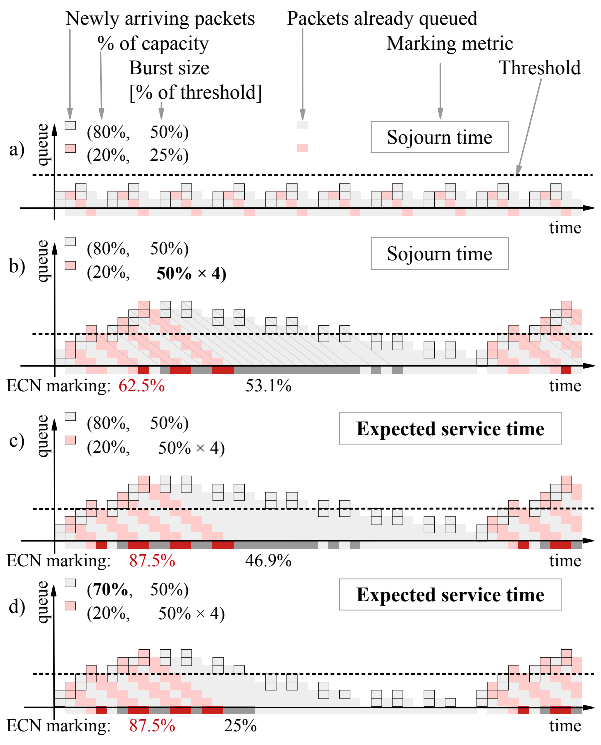
Newly arriving packets
% of capacity
Burst size
[% of threshold]
Packets already queued
Marking metric
Threshold
a) queue (80%, 50%) (20%, 25%) Sojourn time time
b) queue (80%, 50%) (20%, 50% × 4) Sojourn time
ECN marking: 62.5% 53.1% time
c) queue (80%, 50%) (20%, 50% × 4) Expected service time
ECN marking: 87.5% 46.9% time
d) queue (70%, 50%) (20%, 50% × 4) Expected service time
ECN marking: 87.5% 25% time

Figure 5: TBA

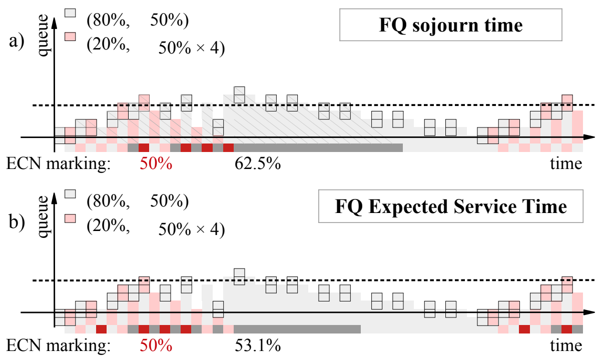
a) queue (80%, 50%) (20%, 50% × 4) FQ sojourn time
ECN marking: 50% 62.5% time
b) queue (80%, 50%) (20%, 50% × 4) FQ Expected Service Time
ECN marking: 50% 53.1% time

Figure 6: TBA

The section “Underutilization with Bursty Links” in the same online collection of tests [HM21] shows a similar effect. When the traffic transmitted by a smooth link (e.g. fixed Ethernet) is mixed with traffic that has traversed a bursty link (e.g. WiFi), the bursty traffic causes a shallow immediate AQM to focus marking on the smooth traffic.

TCP Segmentation Offload (TSO) in Linux (and possibly other OSs) groups a set of packet into a back-to-back burst and calculates the maximum packets in the burst from the current pacing rate of the flow:

```
    max\_burst = pacing\_rate * MAX\_BURST\_DELAY / MTU\_BITS.
```

The flow's pacing rate is upper bounded by the link rate, so this ensures that the burst will not cause more than MAX_BURST_DELAY of queuing at the bottleneck. However, when a new flow starts, its pacing rate is typically well below the link rate and well below the pacing rate of any flows already established over the bottleneck link. Therefore new Linux flows consist of smaller bursts while established Linux flows consist of larger bursts.

If an AQM at the bottleneck is based on sojourn time and therefore marks larger bursts less than smaller bursts, new flows will attract more of the congestion marking. Then as a new flow tries to displace an established flow, it will tend to reach a point of local equilibrium before it has reached the same rate as the established flow. .

ToDo: Got to here - add figures from JM's paper

4 Solution: Expected Service Time

Two approaches will be proposed to minimize signalling delay. Both start with the backlog measured instantaneously at dequeue, then translate it into the expected time needed to drain this backlog—thus both measure the 'expected service time':

- Time-based backlog;
- Scaled sojourn time.

4.1 Time-Based Backlog

In this approach, as each packet is about to be dequeued from the head of the queue, the. expected service time to clear the backlog behind it is calculated as

$$\mathbb{E}(\text{svc\_time}) = (\text{backlog\_deq}) \times \frac{\text{avg\_serializn\_time}}{(\text{avg\_pkt\_size})},$$

or in English, the expected service time, t_b^*, to clear the backlog is the backlog at dequeue, b (e.g. in bytes), multiplied by the recent average serialization time of each packet, t_s^* and divided by the recent average packet size (in bytes), s^*.

Multiplying by the quotient on the right is the same as dividing by the average drain rate. As with averaging any rate, the quotient should be calculated as a quotient of averages, not an average of quotients. This is particularly important if the drain rate varies considerably. Also the averages should be exponentially weighted moving averages (EWMAs) with high gain, e.g. $g = 1/2$, so that they respond rapidly to changing delivery rate. The gain should be an integer power of 2 so that it can be implemented as a bit-shift.

For instance, assuming the times when dequeue of the previous packet started and ended were stored as t_1 & t_2 and the packet size was s, the EWMAs would be updated as

$$t_s^* = g\left((t_2 - t_1) - t_s^*\right)$$
$$s^* = g(s - s^*)$$

The expected time to clear the backlog is then,

$$t_b^* = b * t_s^* / s^*. \tag{1}$$

While the buffer is non-empty, the time that dequeue ends, t_2 will typically become the time that serialization of the next packet starts. Reusing the same time value for both will ensure that any error introduced by coarse clock precision is averaged out.

On the other hand, if the buffer was empty when the previous packet finished dequeuing, the time that serialization took has to be used, without including the idle time waiting for the next dequeue to start.

Note that any media acquisition delay should not be counted, and the backlog that builds while waiting to acquire the medium should not be counted for AQM marking, because it is not caused by the load from the sender, and therefore cannot be reduced by getting the sender to slow down.

4.1.1 Rationale for Time-Based Backlog

The time-based backlog approach assumes that the drain rate to clear the backlog will be similar to that averaged over the last few packets. There is no reason to believe that the recent drain rate is the best estimate of the rate at which the backlog will drain. For instance, a radio link is continually testing different rates to find which is the best and if a queue is continually yielding to a higher priority queue, it will proceed in fits and starts. However, for the purpose of signalling congestion, the recent drain rate gives the best available estimate of the time to drain the backlog, which itself was a result of the recent drain rate.

It should be pointed out that sojourn time also measures the drain rate over the last few packets. But it is indisputable that it is better to use a recent drain rate to estimate how fast the current backlog will drain, rather than just measuring how long it took an earlier backlog to drain that happened to be in front of the head packet when it arrived—while ignoring the current backlog.

One might consider estimating the recent drain rate from the size of just the single most recent head packet and the time to dequeue it. However, such an approach is prone to errors due to coarse clock precision or interruptions affecting access to the clock. By using EWMAs, any such errors should average out, while using a high gain keeps the measurement lag down to about two packets.

The time-based backlog approach is similar to that used in PIE, except with PIE the backlog is divided by the average drain rate over sixteen contiguous packets, and whenever the queue is not that long, the last available average rate is used.

4.1.2 Size-Adjusted Threshold(s)

The following technique is really just an optimization of the time-based backlog. It avoids the per-packet division in Equation 1 above.

It is easiest to explain with an example AQM algorithm. Say, for instance that the AQM marks packets if queuing delay exceeds a simple step threshold, T. Then, as each packet is dequeued, instead of comparing the time-based backlog with the threshold queuing delay, the AQM marks a packet if

$$b * t_s^* \geq s^* * T. \tag{2}$$

In other words, instead of using the average packet size to scale down the backlog, it is used to scale up the threshold.

A similar approach would be used for other AQM functions. For instance, if the likelihood of marking increases by a linear ramp function, both the min and max thresholds of the ramp would be scaled up by s^*.

An alternative optimization would be to ammortize the per packet calculation over a certain number of packets by accumulating the combined dequeue times of a certain number of packets and accumulating the sum of their sizes. Then every so often updating the two EWMAs, and calculating $s^* * T/t_s^*$ to give a new threshold in bytes to compare the backlog against. However, the per-packet processing cost of the original size-adjustment is already fairly minimal (two adds, a bit-shift and two integer-multiplies).

4.2 Scaled Sojourn Time

Another approach would be to scale the sojourn time by the ratio of the backlogs at dequeue and enqueue. That is, the expected service time at any instant would be:

$$\mathbb{E}(\text{svc\_time}) = \text{sojourn\_time} \times \frac{\text{backlog\_deq}}{\text{backlog\_enq}},$$

where backlog_enq can be written into the packet's metadata at enqueue, along with the arrival time, which is how sojourn time is already implemented.

4.2.1 Rationale for Scaling Sojourn Time

Scaled sojourn time is comparable to the above 'time-based backlog' approach, but with the drain rate measured over the sojourn time of each head packet, which is equivalent to backlog_enq/sojourn_time. There are two disadvantages, but one advantage, to measuring over a sojourn time.

Disadvantage 1: The measurement delay depends on the queuing delay, which makes it problematic to signal bursts quickly;

Disadvantage 2: Each sojourn measurement is sensitive to errors reading the clock and, when the ratio of the backlogs is large, as it is during a burst, any error will be greatly magnified;

Advantage 1: Like sojourn time, scaled sojourn time can be measured over an arbitrarily complex set of queues, by only measuring the time of first enqueue and last dequeue and the backlog at enqueue and dequeue times.

Scaled sojourn time would not normally be of interest because of the two disadvantages. However, where an existing deployment or a complex queue structure makes the other approaches infeasible, the advantage of scaled sojourn time might outweigh these disadvantages (but see § 4.3).

Scaling the sojourn time might also make more sense if an implementation is already measuring the sojourn time for another reason. Then it will not need any additional measurement code, because it might already need to maintain the backlogs to do basic queue handling.

Figure 7 visualizes a geometric interpretation of the rationale for scaling the sojourn time. The two plots in the chart at the top of the figure show cumulative arrivals and departures of data in packets. Between times t_0 and t_1 a burst of packets arrives and between t_1 and t_2 a few packets arrive at first, then none. Over the whole time the departure rate is varying independently as, for example, a radio link would. At any time, for instance t_1, the sojourn time (t_{s1}) can be visualized as the horizontal

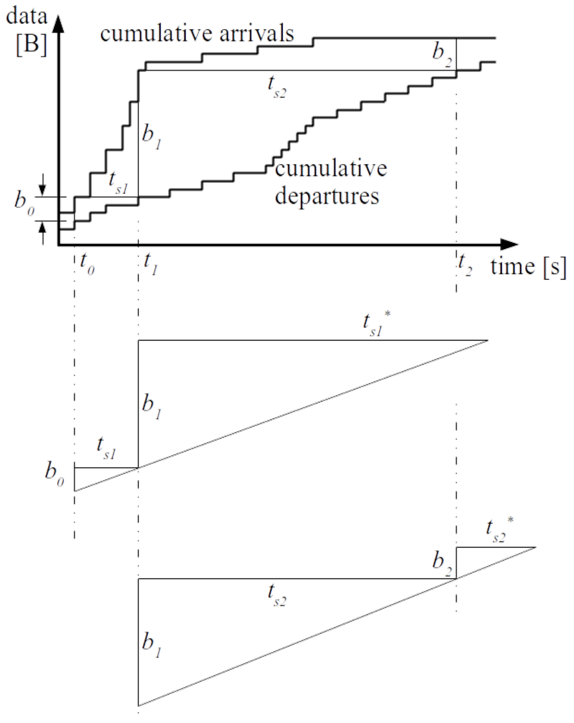

Figure 7: Rationale for Scaling Sojourn Time

distance back from the departures plot to the arrivals plot. And the backlog is shown as the vertical distance between the plots (b_1).

It can be seen that the sojourn time (t_{s1}) between t_0 and t_1 takes account of the departure rate, but not the arrival rate (the burst), during that time. It is proposed to scale the sojourn time by the ratio of the backlogs at departure and arrival of the packet. That is $t^*_{s1} = t_{s1}b_1/b_0$. This scaled sojourn time uses all the latest information available at time t_1.

The schematic in the middle of the figure shows, using similar triangles, how scaled sojourn time is constructed. The departure rate during the sojourn is represented by the slope of the smaller of the middle triangle. The larger triangle extrapolates that departure rate to predict the time (t^*_{s1}) that it will take for the most recent backlog to drain.

The lower schematic shows the situation at time t_2. The actual sojourn time of the new head packet t_{s2} is slightly shorter than the prediction t^*_{s1}. From this new actual sojourn time, a new prediction can now be constructed from the slightly steeper rate slope. This time the backlog b_2 has reduced during the sojourn of the head packet, because there has been a lull in arrivals since t_1. Therefore the formula predicts that the sojourn time will be scaled down relative to its measured value.

We will now return to time t_1 and derive the scaled sojourn time algebraically, rather than geometrically. The departure rate during the sojourn of the head packet is

$$r_{d1} = \frac{b_0}{t_{s1}}. \tag{3}$$

The expected service time to drain the backlog at t_1 is

$$t^*_{s1} = \frac{b_1}{r_{d1}}.$$

Substituting from Equation 3:

$$= t_{s1}\frac{b_1}{b_0}. \tag{4}$$

The expected service time can also be expressed in terms of a ratio of average arrival and departure rates during the sojourn, r_{a1} and r_{d1}. The backlog at t_1 can be expressed in terms of the arrival rate over the sojourn time:

$$b_1 = t_{s1}r_{a1}. \tag{5}$$

Substituting this into Equation 4, the scaled sojourn time at t_1,

$$t^*_{s1} = t_{s1}\frac{r_{a1}}{r_{d1}} \tag{6}$$

That is, scaling the sojourn time by the ratio between the backlogs at dequeue and enqueue is equivalent to scaling it by the ratio between the average arrival and departure rates between enqueue and dequeue.

4.2.2 Implementing Scaled Sojourn Time

To implement scaling of the sojourn time, it is probably easiest to store `backlog_enq` in the packet's metadata when the packet is enqueued. Then at dequeue it can be divided into `backlog_deq`.

But some implementations choose not to do too much at dequeue, because there is limited time between the packet reaching the head of the queue and starting to be forwarded. Therefore, it could be challenging to measure the system time, subtract the stored timestamp then also scale the result by a ratio.

The division in the ratio can be avoided by in at least two ways:

- Multiply the threshold(s) of the AQM by `backlog_enq` rather than dividing it into `backlog_deq` (as in § 4.1.2);
- Use the techniques below to optimize execution, although efficiency will be machine-architecture-dependent, and precision is only to the nearest binary order of magnitude:

`qdelay <<= (lg(backlog_deq) - lg(backlog_enq) + 1/2)`
It is roughly equivalent to multiplying by the ratio between the backlogs, to the nearest integer power of 2.

The `<<=` operator bit-shifts `qdelay` to the left by the expression on the right. `lg()` is the logarithm function base 2. The expression bit-shifts `qdelay` to the left by the difference between the logs of the backlogs at enqueue and dequeue. The addition of 1/2 is necessary so that integer truncation of the result will round to the nearest integer, rather than always rounding down.

The `clz()` function to count leading zeros could be used as a cheaper but more approximate base-2 log function, as follows:
`qdelay <<= (clz(backlog_enq) - clz(backlog_deq))`
This also avoids the need for any boundary checking code.

For example, if the `backlog_*` variables are 32-bit unsigned integers and

`backlog_enq = 3000`, so `clz(3000)=20`
`backlog_deq = 30000`, so `clz(30000)=17`

Then

```
qdelay <<= 20 - 17
```

is the same as

`qdelay *= 2^3`,

which scales qdelay by 8, which approximates to `30,000/3,000 = 10` but is an integer power of 2. This is sufficient to scale the sojourn time to the correct binary order of magnitude, while still taking account of all the latest information in the queue.

However, `clz()` introduces truncation bias because it always rounds down, which could lead the result to be persistently out by up to $\times 2$ or $/2$ for a particular target sojourn time. Using the `lg()`-based expression could be out by from $\sqrt{2}$ to $1/\sqrt{2}$, but with no bias—it is equally likely to be out either way.

A high performance implementation will maintain the backlog of a queue by maintaining two variables (much like the two plots at the top of Figure 7):

`count_enq` written solely by the enqueue routine;
`count_deq` written solely by the dequeue routine

Then the backlog can be measured as `count_enq - count_deq`. These two shared variables can be read from any routine, but they are only incremented by the routine that owns them, which avoids the performance hit of a mutual exclusion lock. The two counters monotonically increase like the system clock for the sojourn measurement, but at the rate of data transfer in and out respectively, not the rate of time passing.

4.3 Distributed Queues

Using sojourn time leverages the advantage that it can be measured across a complex set of queues, including the case where the initial enqueue and the final dequeue routines are distributed across different machines or processors, as already mentioned (separate clocks would need to be synchronized).

This could include the case where the inputs are located on multiple client machines (e.g. mobile user equipment, WiFi stations, cable modems or passive optical network modems) while the output is a located at an aggregation node (e.g. a cellular base station (eNodeB) [TST10], a WiFi access point (AP), a centralized controller for multiple WiFi APs, a cable modem terminal server (CMTS) or optical line termination (OLT) equipment), with a multiplexed access network between the clients and the aggregation node.

In complex cases like these, if minimization of measurement delay is important, the best approach to use will depend on which of metrics are most feasible to measure and communicate to the dequeue process, which will depend on the architecture of the distributed queues. Table 2 tabulates which metrics are needed by which approach.

Note that the backlog at enqueue time and the backlog at dequeue time need to include all the data buffered between the two, not just that in the ingress and egress buffers. Therefore the backlog metric is probably the critical factor for feasibility. And given both approaches need the backlog at dequeue time, and scaled sojourn also needs the backlog at enqueue time, scaled sojourn might end up being the more complex approach.‘

For instance, on a single machine, the `count_enq` variable would be available in a memory shared between ingress and egress. But if the ingress and egress are separate, the ingress machine would have to communicate the `count_enq` variable to the egress over a (preferably non-blocking) control channel, so that the egress could calculate `backlog_deq` by subtracting `count_deq`.

Certain access network technologies, e.g. those for cellular radio access networks, already include such a control channel, over which buffer status report (BSR) control messages are sent from the user equipment at the ingress to the radio network controller. The delay to access a control variable at the

		Time-based backlog	Scaled sojourn time
enqueue time	arrival time	-	✓
	backlog	-	✓
dequeue time	departure time	✓	✓
	backlog	✓	✓
	serialization time	✓	-
	packet size	✓	-

Table 2: Metrics needed by each approach

input machine from the output machine would be larger than that in a non-distributed system, but it would at least be a known, constant delay. So the control system could still provide robust metrics to control queuing in the data channel.

As well as the aggregation node using expected service time (EST) to apply congestion signalling within the final dequeue routine (effectively on behalf of the input queue), the aggregation node could also use EST to govern the scheduling algorithm for controlling each client's inward (upstream) access rate into the shared medium, by altering the rate at which it granted medium access slots to each client.

4.4 Applicability of Expected Service Time

Using any of the approaches in § 4 to calculate EST would improve timeliness relative to using sojourn time or other techniques to measure queue delay. However, it would not be worth modifying an existing deployment unless all other delays were going to be removed at the same time.

It might be thought that an algorithm like the proportional integral (PI) controller[2] already takes account of the change in queuing delay between samples, so changing the queuing delay measurement itself seems redundant. However, using EST actually ensures that a PI algorithm takes account of the change between the latest queue delay measurements at each sample time, not between two outdated measurements.

It might also be thought that PI controllers do not need to care so much about instantaneous measurements, because they are maintaining the fairly large queue that is needed by classic TCP algorithms like Reno, Cubic, Compound or BBR. However, even though a PI algorithm only samples the queue fairly infrequently (relative to packet serialization time), using an out of date queue metric makes it necessary to introduce extra heuristic code to deal with the resulting sloppiness.

For instance, in the case of PIE [PNB+17], some heuristic code suppresses any drop once the last sample of queue delay falls below half the target delay.[3] This is an attempt to suppress drop when the queue is draining after the load has gone idle. However, it is ineffective if sojourn time is used to measure the queue, because the sojourn time does not reduce until after the last packet (as explained on the right of **??**). Using EST whenever it is sampled should eliminate the need for this heuristic because it takes account of the reducing backlog as the queue drains.[4]

EST is also applicable to the CoDel algorithm for the same reasons—sojourn time fails to take account of the evolution of the queue after the head packet was enqueued (again, as in **??**). This can tend to cause the AQM to continue dropping or marking packets at the end of a flow, because the sojourn time does not recognize that the queue has gone below the target delay. The CoDel code in Linux includes a heuristic to exit dropping mode when the backlog goes below 1 MTU, but that still continues dropping mode until the last packets of a flow, and tail loss is particularly problematic for a flow to detect.

EST is particularly applicable to a simple AQM algorithm like the time-based shallow threshold recommended for DCTCP in [BCCW16] or the native AQM for L4S traffic in DualPI2 [DSBEBT17, Appx. A] or Low Latency DOCSIS [DOC19]. It would be applicable whether the threshold is a simple step, or a probabilistic ramp like the RED function (but based on instantaneous queue delay, not smoothed queue length), or a deterministic ramp or convex function of instantaneous queueing delay, as in Curvy RED [DSBEBT17, Appx. B]. The queue in these cases is intended to be very shallow,

[2] Used in QCN [FE10], PIE [PNB+17], PI2 [DSBTB16] or the base AQM of DualPI2 [DSBEBT17].

[3] As long as some other conditions hold that are not important here.

[4] Indeed, this was the original motivation for this work.

so it might seem that the extra measurement delay would be minimal. However, the sensitivity of these very low delay schemes to burstiness makes it particularly important to ensure that bursts are measured rapidly and correctly.

EST could apply to many types of queue, not just packet queues, as long as the size of each job is quantified in common units that are additive. Examples include, but are not limited to, queues of datagrams, frames or packets, as well as message queues, call-server queues, computer process scheduling queues, storage queues (e.g. SSD or disk), workflow queues for mechanical or human-operated stages of tasks.

As well as dropping or ECN-marking, different sanctions could be applied using the same basic ideas. Examples include, but are not limited to: truncating or otherwise damaging the data or checksum of a message or packet but preserving the information necessary for delivery; rerouting; delaying; downgrading the class of service; and tagging.

5 Marking Fairness

5.1 Marking Fairness Experiments

5.1.1 EST-based Marking Fairness

Figure 8 shows the degree to which EST-based ECN marking is or is not focused onto the more bursty of a pair of flows indexed as a & b. Both flows were modelled as unresponsive in a time-slotted model similar to the example in Figure 2b). So the results are not necessarily an accurate reflection of a real AQM, but they should at least be indicative of the likely outcome.

The charts show the ECN marking level that results from scanning the scenario parameters across two dimensions:

1. The capacity share of flow a, λ_a is varied from $1/32$ to $1/2$ and the share the other flow b is also varied so that it fills the remainder of the link ($\lambda_a + \lambda_b = 100\%$)). The left-hand chart of each pair is identical to the right-hand chart except, to help pick out the plots, the last two capacity-share scenarios are omitted.
2. The normalized burst size of flow a, β_a is increased from $1/16$ in steps of $1/16$, while the burst size of flow b is reduced such that the sum of both burst sizes is constant. The sum is 1.25 in the top plots, and 1.0625 in the bottom. Normalization is relative to the ECN marking threshold, so a sum of 1.25 means that if bursts from both flows coincide, a previously empty queue would exceed the marking threshold by 25%.

It can be seen from the right-hand side of any of the plots that, if flow a utilizes a small fraction of the capacity (up to roughly $1/8$) and if it is even slightly more bursty than the flow b (which is using more of the capacity), EST-based marking focuses a high fraction of the marking onto the more bursty flow.

It is also noticeable that wobbles increasingly appear; some so great that at certain points, even when sharing capacity 50:50, the more bursty flow attracts *less* of the marking. It is possible that these wobbles are an artefact of the time-slotting in the model, so a more precise simulation will be necessary.

Working down from top to bottom of Figure 8, it can also be seen that, as the combined level of burstiness increases, it pushes the marking level of the more bursty flow up to 100% over a wider range of capacity shares, ultimately remaining at 100% across any share of burstiness.

However, it must be admitted that the region where the plots of flow a and b cross (for any capacity share, λ_a) is always to the left of centre. The capacity share of a is never more than half, so this means that there is a range to the left of centre where a's marking probability is higher even though both its burstiness and capacity share are lower than b's, although this inversion corrects itself as b's relative burstiness increases.

Focusing again on the right hand side of the charts (where flow a is more bursty than b), it may seem counter-intuitive that those flows with a smaller share of the capacity attract a higher marking probability. Even though a's marking probability is still generally higher than b's, the difference continually narrows, even where a is considerably more bursty than b.

The explanation is that, when the size of flow a's bursts relative to b's remains the same (on the same vertical), but its share of capacity increases, its bursts will become more closely spaced. Then, they are more likely to arrive behind some of b's traffic. So flow a's marking probability will decrease as its capacity share increases.

Also it should be kept in mind that two flows with the same burstiness (on the vertical line down the middle of the plots) would be expected to have the same marking *probability* even if they have different shares of capacity. But a flow with a lesser share of the capacity will attract a proportionately

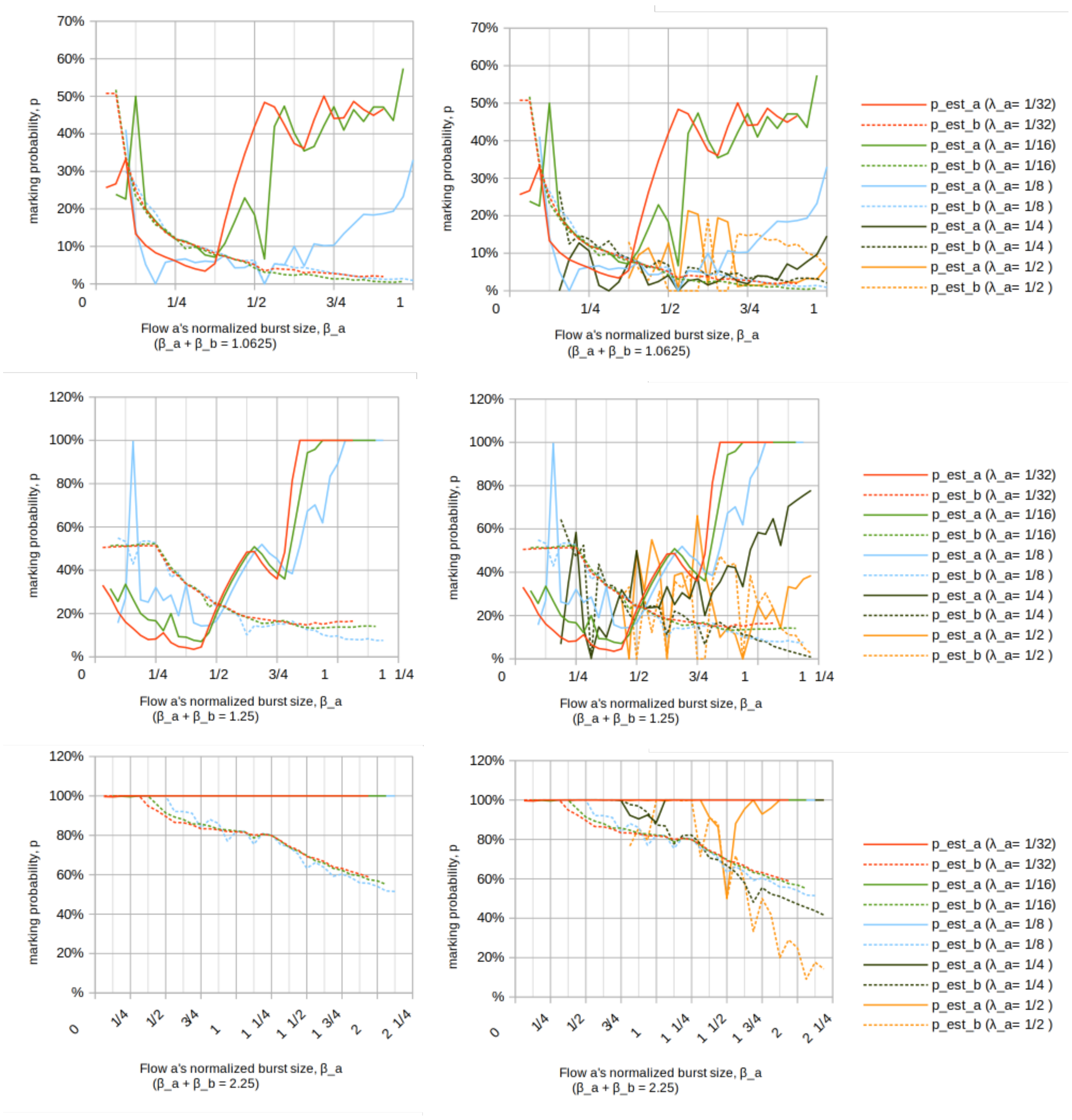

Figure 8: EST-based marking fairness of two flows wrt capacity share, λ, and relative burstiness, β. $\lambda_a + \lambda_b = 100\%$; top :$\beta_a + \beta_b = 1.0625$; middle :$\beta_a + \beta_b = 1.25$ bottom :$\beta_a + \beta_b = 2.25$. The left-hand charts are the same as the right, except they exclude two scenarios that otherwise obscure the other plots

lesser share of the *volume* of marks. This is explained in Figure 9, which shows the same traffic scenario as the middle of Figure 8, but the metric on the y-axis is normalized congestion-rate, which represents the proportion of link throughput that is marked, rather than just the proportion of flow a. Or formally, the normalized congestion rate of flow a, $\nu_a = \lambda_a p_a$. With this metric it can be seen that, for the same relative burstiness (same vertical), the proportion of marks increases with capacity share, as would be expected intuitively.

5.1.2 Sojourn-based Marking Fairness

For comparison, Figure 10 shows the ECN marking that results from sojourn-based marking under the same traffic scenarios as the EST-based marking in Figure 8 (note that the lower middle row is the same scenario as the upper middle, but the metric is normalized congestion-rate, which can be compared with Figure 9).

The most obvious failing can be seen on the left-hand half of the lowest row ($\beta_a + \beta_b = 2.25$), but also on the far-left of the other rows, where flow

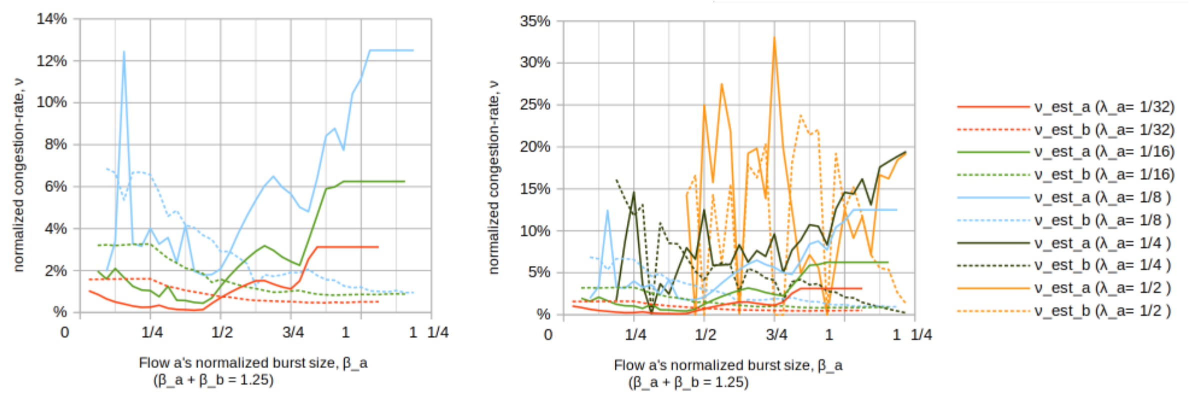

Figure 9: EST-based congestion-rate of two flows wrt capacity share, λ, and relative burstiness, β. $\lambda_a + \lambda_b = 100\%$; top :$\beta_a + \beta_b = 1.0625$; middle :$\beta_a + \beta_b = 1.25$ bottom :$\beta_a + \beta_b = 2.25$ (same as Figure 8). The left-hand charts are the same as the right, except they exclude two scenarios that otherwise obscure the other plots

a's marking exceeds b's even though its burstiness and capacity share are both lower. Similarly, on the right of the plots, no matter how bursty a becomes, its marking probability never exceeds b's. Thus, sojourn-based marking sometimes perversely rewards burstiness and penalizes smoothness.

The second most obvious failing is the much lower overall marking probability in any of the scenarios, given the high degree of burstiness in all scenarios. Even when the bursts of flow a alone all exceed twice the marking threshold ($\beta_a > 2$ in the bottom row), marking is no higher than 60% (in contrast, as a's bursts become smaller its marking perversely rises to 100%). And in the middle case ($\beta_a + \beta_b = 2.25$), even when all a's bursts exceed the threshold ($\beta_a > 1$ the marking probability only lies in the 10%–20% range.

5.2 What Marking would be Fair?

As remarked in § 2.3 it is obvious when marking is extremely unfair. For instance, if a bursty flow is using fraction λ of the capacity on average, but attracting less than fraction λ of the marking. However, it is not obvious how much more marking it would be fair to apply to a flow for bursts of a particular size relative to those of another flow.

The doctoral research of Wischik [Wis99a, Wis99b] and sample path shadow pricing [KMT98] on which it is based appear to be the only work to tackle the question of how to define fair marking for anything but smooth flows. Wischik considers a number of possible definitions of fair marking, which is reduced down to the following three candidates, all of which are intended for offline analysis only, and infeasible for a live marking algorithm. Also all are defined in relation to packet loss, and all assume high statistical multiplexing of flows at a resource:

Effective bandwidth: EB was developed in the context of flow admission control. The EB of a variable rate flow is somewhere between its mean and peak throughputs. A bursty flow can be replaced by another flow with the same effective bandwidth without altering the resulting loss probability, including replacement by a flow with constant throughput equal to the effective bandwidth. So it would be fair to mark flows in proportion to their effective bandwidths.

ΔL: The function $L(Y)$ is defined as the volume of loss at a resource when presented with load Y. Then $\Delta L(Y)$ is the change in the loss volume when the flow under consideration is removed, which is the standard definition of a shadow price, so should provide the basis for fair marking.

Sample Path Shadow Pricing: SPSP (Figure 11) marks every packet that, if removed, would have resulted in one less packet being dropped.

Wischik explains that each definition has aspects in common with the others, but they differ in their goals, assumptions and user models.

ΔL suffers from not being 'incrementally fair'. As flows are added to the load, even if they are equally bursty and of equal average throughput, the rise in the volume of loss is greater for each additional flow added. So, with a certain load, imagine that the ΔL due to removing any one flow is 0.3%, but removing two equal flows reduces loss by only 0.5%. Then if two flows band together, and internally

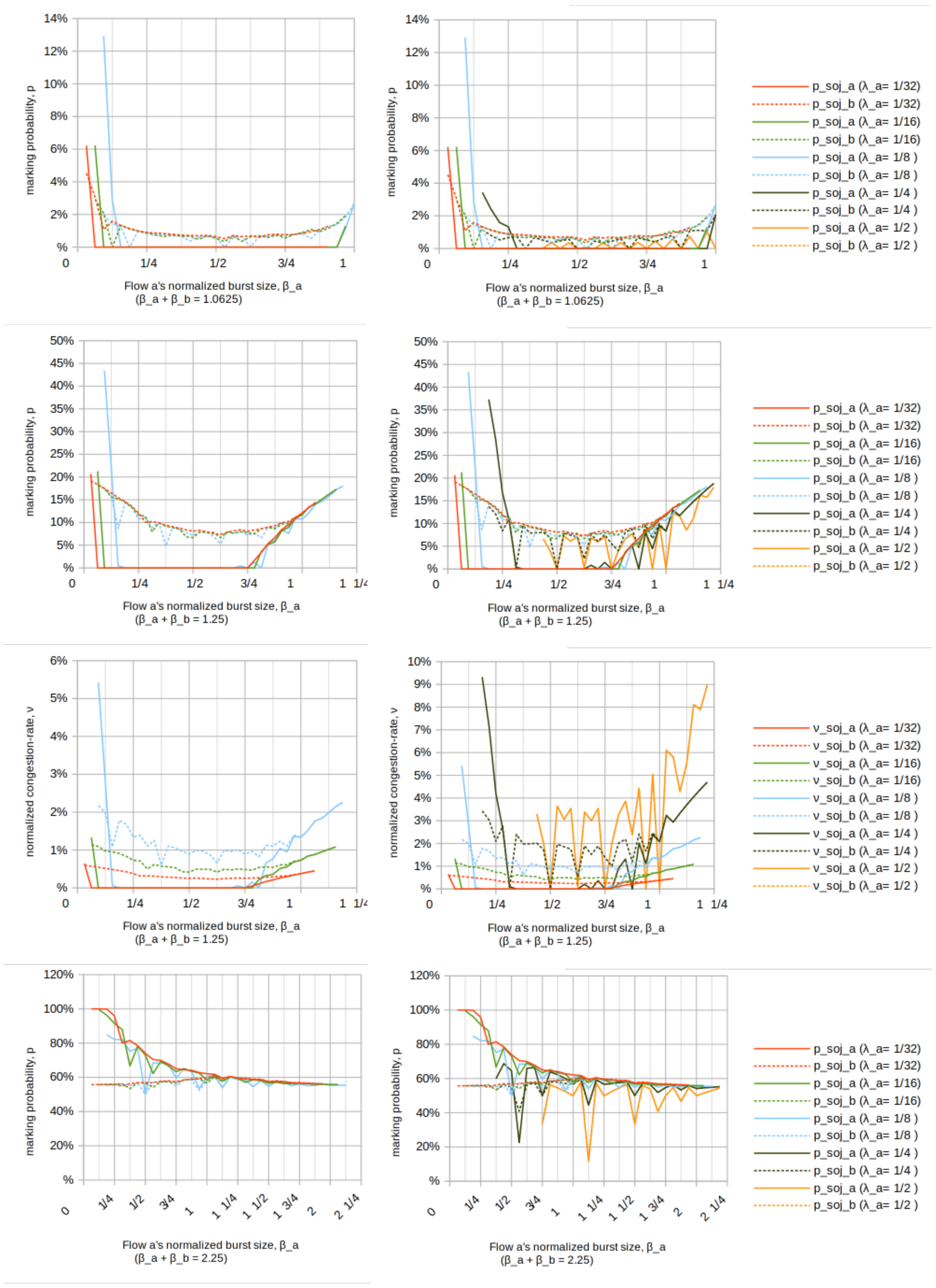

Figure 10: Sojourn-based marking fairness of two flows wrt capacity share, λ, and relative burstiness, β. $\lambda_a + \lambda_b = 100\%$; top :$\beta_a + \beta_b = 1.0625$; upper (marking prob) and lower (congestion-rate) middle: $\beta_a + \beta_b = 1.25$ bottom :$\beta_a + \beta_b = 2.25$ (same as Figure 8).

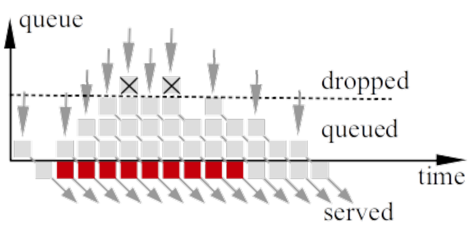

Figure 11: Schematic of Sample Path Shadow Pricing (SPSP) with one timeslot per equisized packet for illustration. A packet is marked (in red) if removing it would have resulted in one less packet being dropped

share the marks they get under the ΔL scheme between themselves, they would get only 0.25% marking each, thereby proving that ΔL is not incrementally fair.

In contrast, SPSP is incrementally fair because it acts at the granularity of packets, and is agnostic to whether packets band together under flows or users. This also means that SPSP is concerned with precisely which packets to mark, whereas ΔL only knows what proportion of marks to allocate to each flow.

Like ΔL, EB can only say what proportion of marks should be applied to each flow but EB only knows the relative proportions; it does not know how many marks to apply overall, although this can be remedied if an exchange rate between drops and marks is defined.

Thus, in the words of Wischik, "SPSP is best". However, like all the schemes, SPSP can only be applied offline—in retrospect. In the case of SPSP, this is because it is meant to mark the packets that were in the queue as it built up to the point of overflow, but many of those packets will already have left the queue by the time the queue does overflow (for instance the first four marked packets in Figure 11).

Nonetheless, SPSP represents an ideal scheme that a practical marking algorithm ought to aspire to. However, the goals of the present work are wider than those in Wischik. We particularly want to maintain very low queuing delay, but we do also want to ensure marking is as fair as possible. Therefore, it will be worth taking note of the two questions that Wischik poses to assess the fairness of a marking algorithm:

1. Does it marks packets that caused overflow, or does it mark innocent packets that arrived later?
2. Does it mark in the busy period leading up to the overflow?

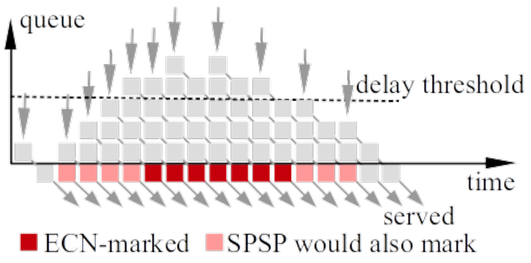

Figure 12: Marking based on Expected Service Time compared against Sample Path Shadow Pricing

For marking based on expected service time (EST), we redefine overflow as exceeding the queuing delay target (in time units not bytes). Then, using the example scenario in Figure 12, we can answer the questions and justify where EST diverges.

Q1. EST does not mark any innocent packets; however it does not mark the packets that contributed to excess delay but were dequeued after excess delay had ended (the pink packets to the right of the red ECN-marked packets).

Q2. EST marks as many packets in the busy period leading up to excess delay as possible, but it cannot mark those that had already been served when excess delay was first detected (the pink packets to the left).

From the earlier analysis of EST-based marking of traffic bursts (§ 5.1.1), it can be seen that EST will rarely miss any packets in the build-up to excess delay, because most of the packets of a burst are still queued when the tail arrives.[5] Also stopping EST marking as soon as the excess delay has gone away seems preferable for two reasons:

- It seems less likely to hit smoother flows, which tend to back up behind a burst, as explained in § 2.3;
- When there has been a period of marking, the load arriving from sources will reduce a round trip later. Thus if marking were to continue until the queue was empty, it would tend to cause under-utilization in the following round trip.

5.3 Queue Protection and Marking Fairness

The goal of queue protection (QProt) is to try to eliminate bursty traffic from a shared low latency

[5] Where the difference between the burst rate and the service rate of the queue is less pronounced, this would not be true. However, in the earlier analysis, bursts were always assumed to arrive in one timeslot.

queue. The ability of EST-based marking to focus on bursts raises the possibility of a crude form of queue protection without having to identify layer-4 flows (in contrast to per-flow QProt specified for DOCSIS [BW19]).

Here an example design for aggregate QProt is proposed, for instance as an extension to the DualQ Coupled AQM. When a packet is about to be dequeued from the L queue, if the EST of the backlog behind it exceeds the maximum ECN-marking threshold[6] by some margin, the head packet would be redirected to the back of the C queue.[7] The margin might be set, for instance, at twice the depth of the maximum marking threshold, or it might be slightly randomized to discouraging sources from 'flying just under the radar'. Once redirection had started, it would continue until the EST at least reduced below the allowed margin, or preferably it could continue until the EST had been brought down to the maximum ECN marking threshold.

Then, any large bursts would be more likely to be ejected from the L queue, while traffic keeping within the maximum ECN threshold would be more likely to remain in the L queue, without having to sit behind large bursts. Of course there is no guarantee that smooth traffic would not sometimes be redirected, when it happened to arrive in front of a burst. But smooth, well-behaved traffic should never build up behind a burst that had been large enough to be ejected from the L queue.

This technique would cause reordering of any flow it redirected, However, any application receiving data in bursts should not be particularly sensitive to reordering within the bursts, given one assumes it would have to buffer the bursts as they arrive anyway. Nonetheless, any even slight harm due to reordering would help to disincentivize an application from sending in bursts.

If all the traffic was bursty, perhaps because it had passed through a bursty link upstream, aggregated QProt would redirect some traffic from all flows into the C queue, which would just add reordering to all traffic. Per-flow QProt is designed to redirect the most bursty flows until queue delay reduces, so it would not necessarily fare much better in a scenario where burstiness was added to all traffic in the aggregate. However, QProt attempts to focus redirection on only a few flows, even if they are only slightly worse than others, which might help in this case. In practice, most flows would be responsive to congestion signalling so, as long as the marking was fair, aggregate burstiness should result in general under-utilization, rather than endemic reordering.[8]

[6] The step threshold, or the top of a ramp function.

[7] Preferably by rearranging pointers, rather than moving the packet data.

The above aggregate QProt technique is not necessarily only applicable to a DualQ. It is simple enough that it could potentially also be included in a per-flow queue, to protect against the possibility of a bursty flow encapsulated within a VPN alongside smooth flows. For instance, the L4S architecture allows for the possibility that VPNs are separated into two sister per-flow queues for L4S and non-L4S packets. Then the L4S per-'flow' queue could redirect bursts into its non-L4S sister. However, a bursty flow within a VPN is perhaps a corner case that would not warrant such an addition.

6 Other Signalling Delays

The introduction enumerated six causes of delay to congestion signals and highlighted two that this memo would focus on. The other four sources of signalling delay are briefly surveyed below, with pointers to where they have been considered in other work.

6.1 Propagation Delay

Numerous proposals have been made to speed up signalling by sending the signal from the queue back against the flow of traffic, direct to the sender. This can be done in a pure L2 network, e.g. backwards congestion notification (BCN) in IEEE 802.1Qau [FE10] a.k.a. Quantized Congestion Notification (QCN), which is now rolled into 802.1Q-2011 and 802.1Q-2014. However, in general signalling backwards is problematic in IP networks, amongst other reasons because the sender has to accept out-of-band packets from any arbitrary source in the middle of the network, which makes it vulnerable to DoS attacks [Gon12].

Therefore, here we will assume that signals are piggy-backed on the forward traffic flow then fed back to the sender via the receiver. However, this does not preclude a solution to the problems of backwards congestion notification.

6.2 Smoothing Delay

AQMs designed for the Internet's classic congestion controls (TCP Reno, Cubic, Compound, etc.)

[8] Where the IETF draft on L4S [DSBET17] discusses burstiness from upstream links, it suggests that the pragmatic solution is to improve the configuration of the system as a whole, for instance reducing the burstiness of an upstream link, or increasing the L4S marking threshold at the bottleneck.

filter out fluctuations in the queue by smoothing it before using the smoothed measurement as a measure of load to drive the congestion signal. DCTCP [AGM+10] proposed to smooth the signal at the sender, so that the network could send out the signal immediately, without smoothing, and L4S followed this approach [BEDSB17]. This allows the sender to receive the signal without smoothing delay, which is particularly useful in cases where the sender might not need to smooth the signal itself, e.g. to detect overshoot when accelerating to start a new flow. Shifting the smoothing function from the network to the sender also makes sense because the network does not know the round trip time (RTT) of each flow, so it has to smooth over the maximum likely RTT. Whereas a sender knows its own RTT and can smooth over this timescale.

6.3 Signal encoding delay

Previous research has proposed to change the IP wire protocol to provide more bits to signal congestion. Nonetheless, it has been pointed out[9] that the delay of a unary encoding is inversely proportional to the value being encoded, and the congestion window of a scalable congestion control is also inversely proportional to the value of the congestion signal. So, as flow rates (and consequently congestion windows) increase over the years, at least in general the delay to encode the signal does not increase.[10]

Therefore, in this report we have assumed the unary encoding of congestion signals standardized as ECN by the IETF [RFB01]. This does not preclude other encodings, e.g. the multi-bit encoding of QCN or minor alterations to the decoding to avoid saturation, such as that in [BDS17].

6.4 Removing Randomness Delays

One of the main motivations for the design of Random Early Detection (RED) [FJ93] was to break up synchronization between the sawteeth of TCP flows driving the same queue. This still remains an important requirement for all AQM algorithms [BF15].

AQMs mitigate synchronization by introducing marking or dropping more gradually than a tail-drop buffer would, and to a certain extent by randomizing the marking.

[9] Matt Mathis is believed to have pointed this out first.
[10] However, encoding delay does increase with the degree of ACK coalescing.

With clean-slate approaches such as DCTCP in private networks, or incrementally deployable clean-slate approaches like L4S [BEDSB17] for the public Internet, requirements for the network and for end-systems are still in the process of definition. In these clean-slate or slightly dirty-slate cases, it would be possible to require the sender's congestion control to dither its response to congestion signals, so that it would not be necessary to introduce randomness in the network, which adds uncertainty and therefore delay to the congestion signalling channel.

Any AQM that probabilistically signals congestion with probability p could deterministically signal congestion by introducing an interval of $1/p$ packets between each drop or mark. PDPC+ [SLMP03] and CoDel [NJ12], which is very similar, use a deterministic rather the probabilistic algorithm to encode the congestion signal. Both AQMs in DualPI2 [DSBEBT17, Appx. A] also uses a deterministic algorithm.

The determinism would be lost wherever the AQM was controlling flows multiplexed within one queue without per-flow state, because assignment of each deterministic congestion signal to each flow would become randomized by even slightly random packet arrivals from the different flows [Bri15].

Nonetheless, whenever a flow is on its own in an AQM, which is a common case for the traffic patterns in many access network designs, deterministic congestion signalling would reduce signalling delay. This could particularly ease the design of new flow-start algorithms, where the flow introduces microbursts or chirps to sense at what level it starts to congest the link.

Determinism of an AQM is of less importance when the congestion control rather than the AQM determines the spacing between marks. For instance, the duration of the sawteeth of a classical congestion control scales with BDP. So at low BDP, the AQM determines the spacing between marks, but as BDP scales, the congestion control sawteeth move in and out of closed loop control, which determines the duration between 'congestion events' with the AQM inactive between times [Bri21, § 3.3].

7 Related Work

Scaling sojourn time seems superficially similar to combined enqueue and dequeue ECN marking (CEDM) [SR17], because CEDM marks a packet at enqueue if the queue is over a threshold, but then unmarks it at dequeue if the backlog has dropped below the threshold. However the two are significantly different. Firstly, CEDM has to be based on queue length in order to mark at enqueue. But also CEDM is intentionally asymmetric, in that it unmarks packets if the backlog at dequeue has dropped below the threshold, but it does not mark packets at dequeue if they have risen above the threshold. In contrast, scaling sojourn time is deliberately symmetric, meaning it compensates for growth or shrinkage of the backlog (Figure 7).

8 Further Work

The author is not aware of any additional related work, but so far only superficial efforts have been made to search the literature to check.

The responsiveness of an AQM based on scaled sojourn time has been compared to that with just sojourn time in unpublished work with Tom Henderson using the ns3 simulator. However, the potential for expected service time to improve marking fairness under bursty traffic loads has only been superficially evaluated so far.

9 Acknowledgements

The scaling of the service time of the queue was based on discussions with Henrik Steen, an MSc student of the author, in Nov 2016 & May 2017. Asad Ahmed originally proposed the idea of using EST for aggregate queue protection.kj

References

[Ada13] Richelle Adams. Active Queue Management: A Survey. *IEEE Communications Surveys & Tutorials*, 15(3):1425–1476, 2013.

[AGM+10] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitu Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). *Proc. ACM SIGCOMM'10, Computer Communication Review*, 40(4):63–74, October 2010.

[BBP+16] Bob Briscoe, Anna Brunstrom, Andreas Petlund, David Hayes, David Ros, Ing-Jyh Tsang, Stein Gjessing, Gorry Fairhurst, Carsten Griwodz, and Michael Welzl. Reducing Internet Latency: A Survey of Techniques and their Merits. *IEEE Communications Surveys & Tutorials*, 18(3):2149–2196, Q3 2016. (publication mistakenly delayed since Dec 2014).

[BCCW16] Wei Bai, Li Chen, Kai Chen, and Haitao WuHaitao. Enabling ECN in Multi-Service Multi-Queue Data Centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 537–549, Santa Clara, CA, March 2016. USENIX Association.

[BDS17] Bob Briscoe and Koen De Schepper. Resolving Tensions between Congestion Control Scaling Requirements. Technical Report TR-CS-2016-001, Simula, July 2017.

[BEDSB17] Bob Briscoe (Ed.), Koen De Schepper, and Marcelo Bagnulo. Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture. Internet Draft draft-ietf-tsvwg-l4s-arch-00, Internet Engineering Task Force, May 2017. (Work in Progress).

[BF15] Fred Baker and Gorry Fairhurst. IETF Recommendations Regarding Active Queue Management. Request for Comments RFC7567, RFC Editor, July 2015.

[Bri07] Bob Briscoe. Flow Rate Fairness: Dismantling a Religion. *ACM SIGCOMM Computer Communication Review*, 37(2):63–74, April 2007.

[Bri15] Bob Briscoe. Review: Proportional Integral controller Enhanced (PIE) Active Queue Management (AQM). Technical Report TR-TUB8-2015-001, BT, May 2015.

[Bri21] Bob Briscoe. PI^2 Parameters. Technical Report TR-BB-2021-001; arXiv:2107.01003 [cs.NI], bobbriscoe.net, October 2021.

[BW19] Bob Briscoe and Greg White. Queue Protection to Preserve Low Latency. Internet Draft draft-briscoe-q-protection-00, Internet Engineering Task Force, July 2019. (Work in progress).

[DOC19] Data-Over-Cable Service Interface Specifications DOCSIS® 3.1; MAC and Upper Layer Protocols Interface Specification. Specification CM-SP-MULPIv3.1-I17-190121, CableLabs, January 2019.

[DSBEBT17] Koen De Schepper, Bob Briscoe (Ed.), Olga Bondarenko, and Ing-Jyh Tsang. DualQ Coupled AQM for Low Latency, Low Loss and Scalable Throughput. Internet Draft draft-ietf-tsvwg-aqm-dualq-coupled-01, Internet Engineering Task Force, July 2017. (Work in Progress).

[DSBET17] Koen De Schepper, Bob Briscoe (Ed.), and Ing-Jyh Tsang. Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay. Internet Draft draft-ietf-tsvwg-ecn-l4s-id-00, Internet Engineering Task Force, May 2017. (Work in Progress).

[DSBTB16] Koen De Schepper, Olga Bondarenko, Ing-Jyh Tsang, and Bob Briscoe. PI^2 : A Linearized AQM for both Classic and Scalable TCP. In *Proc. ACM CoNEXT 2016*, pages 105–119, New York, NY, USA, December 2016. ACM.

[FE10] Norm Finn (Ed.). IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks - Amendment: 10: Congestion Notification. Draft standard 802.1Qau, IEEE, April 2010.

[FJ93] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[Flo94] Sally Floyd. TCP and Explicit Congestion Notification. *ACM SIGCOMM Computer Communication Review*, 24(5):10–23, October 1994. (This issue of CCR incorrectly has '1995' on the cover).

[Gon12] Fernando Gont. Deprecation of ICMP Source Quench Messages. Request for Comments 6633, RFC Editor, May 2012.

[HM21] Pete Heist and Jonathan Morton. L4S Tests. Online https://github.com/heistp/l4s-tests/#readme, May 2021.

[KF02] Minseok Kwon and Sonia Fahmy. A Comparison of Load-based and Queue-based Active Queue Management Algorithms. In *Proc. Int'l Soc. for Optical Engineering (SPIE)*, volume 4866, pages 35–46, 2002.

[KMT98] Frank P. Kelly, Aman K. Maulloo, and David K. H. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3):237–252, 1998.

[MS10] Andrew McGregor and Derek Smithie. Rate Adaptation for 802.11 Wireless Networks: Minstrel. http://blog.cerowrt.org/papers/minstrel-sigcomm-final.pdf, 2010? (Rejected conference submission).

[NJ12] Kathleen Nichols and Van Jacobson. Controlling Queue Delay. *ACM Queue*, 10(5), May 2012.

[PNB+17] Rong Pan, Preethi Natarajan, Fred Baker, Greg White, Bill Ver Steeg, Mythili Prabhu, Chiara Piglione, and Vijay Subramanian. PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem. Request for Comments RFC 8033, RFC Editor, February 2017.

[PPP+13] Rong Pan, Preethi Natarajan Chiara Piglione, Mythili Prabhu, Vijay Subramanian, Fred Baker, and Bill Ver Steeg. PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem. In *High Performance Switching and Routing (HPSR'13)*. IEEE, 2013.

[RFB01] K. K. Ramakrishnan, Sally Floyd, and David Black. The Addition of Explicit Congestion Notification (ECN) to IP. Request for Comments 3168, RFC Editor, September 2001.

[SLMP03] M. Sågfors, R. Ludwig, M. Meyer, and J. Peisa. Buffer Management for Rate-Varying 3G Wireless Links Supporting TCP Traffic. In *Proc Vehicular Technology Conference*, April 2003.

[SR17] Danfeng Shan and Fengyuan Ren. Improving ECN Marking Scheme with Micro-burst Traffic in Data Center Networks. In *Proc. IEEE Conference on Computer Communications (Infocom'17)*, May 2017.

[TST10] Yifeng Tan, Riikka Susitaival, and Johan Torsner. Active Queue Management for Wireless Communication Network Uplink. Patent WO2010107355, 2010.

[Wis99a] Damon Wischik. How to Mark Fairly. In *Workshop on Internet Service Quality Economics. MIT*, 1999.

[Wis99b] Damon Wischik. *Large Deviations and Internet Congestion*. PhD dissertation, University of Cambridge, September 1999.

Document history

Version	Date	Author	Details of change
00A	04 Sep 2017	Bob Briscoe	First draft.
01	05 Sep 2017	Bob Briscoe	First complete version.
02	07 Sep 2017	Bob Briscoe	Added a couple of refs. Qualified claims about clz().
03	16 Jan 2018	Bob Briscoe	Completed the algebraic rationale for scaling sojourn time.
04	15 Apr 2019	Bob Briscoe	Added abstract
04A	26 Nov 2021	Bob Briscoe	Restructured. Generalized from Scaled Sojourn to Expected Service Time, and added Time-based Backlog as better approach. Added sections on Fairer Marking and Marking Fairness.
04B	28 Nov 2021	Bob Briscoe	Added marking fairness results based on sojourn for comparison, and discussion and plots of congestion-rate as well as probability.