Technical Report

Rapid Signalling of Queue Dynamics

Bob Briscoe*

15 April 2019

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

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.

Propagation and queuing delay are the largest contributors to the overall delay experienced by network data. Propagation delay can be reduced by structural techniques, such as server placement, but queuing delay is a result of subtle interactions due to the system design.

This memo focuses on reducing the delay that congestion signals experience within the queuing algorithm, which 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.

The focus here is on congestion signals 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.

These congestion signals experience delay consisting of the following elements:

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

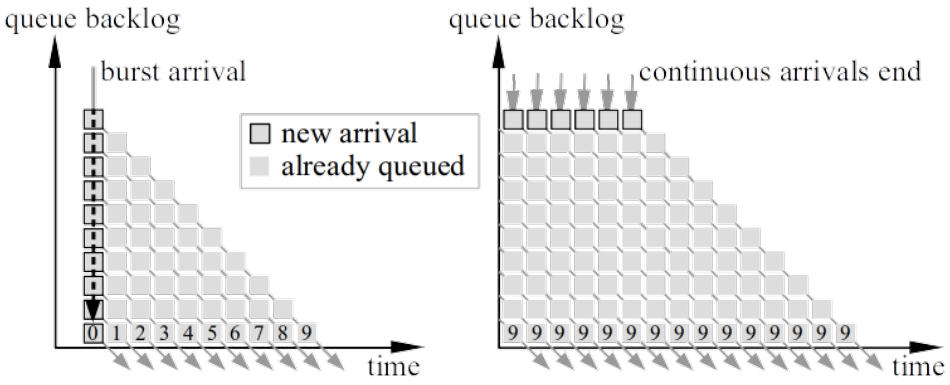

Figure 1: Schematic Illustrating Two Problems with the Sojourn Time Metric. a) It does not measure the full size of a burst until the end (left); b) It does not measure a draining queue (right). Draining is visualized at one equisized packet per timeslot. Sojourn time is represented just before each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

'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 focuses on reducing two of these: queuing and measurement. The other four are briefly surveyed in § 7.

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. So if a burst (or a reduction in drain rate) extends the queue, the sojourn time of the packet at the front of the burst (or the start of the reduction) will show no evidence of the queue that has built behind it. For instance, on the left of Figure 1, 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.

Conversely, consider a queue that has been stable then the flow ends, so that no further packets arrive (the right-hand schematic in Figure 1). 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.

This memo proposes simple techniques to cut that measurement delay by using all the information available in the queue at the point a packet is dequeued. At the moment a packet is dequeued there is very little time for additional processing, so considerable attention is also paid to minimizing execution time.

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 algoithm 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. It 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 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. Whereas sojourn time can only be measured once a packet is dequeued.

It minimizes delay if the signal is 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 (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 right hand column shows the effective delay added by the simple estimation technique proposed in the following section.

Added the other techniques, or remove them all

where signal is applied	Technique to measure queue delay		
	$\frac{\text{backlog}}{\text{drain\_rate}}$	sojourn time	scaled sojourn time
at enq	$t_r/2 + t_s$	$2t_s$	$3t_s/2$
at deq	$t_r/2$	t_s	$t_s/2$

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

It can be clearly seen that applying a signal at enqueue adds t_s to the signal delay.

The delay to measure the drain rate is given as $t_r/2$ because, although it takes t_r to gather the measurements, the resulting average rate metric can be considered to represent the rate half-way through the measurement period.

For example, the IETF specification of PIE [PNB+17] recommends that the drain rate needs 16 packets to get a representative estimate, so $t_r/2$ will be the serialization time of 8 packets, and it will become stale whenever there are less than 16 packets in the queue. In contrast, the sojourn time approaches apply down to a lone packet, but it takes longer to measure for longer queues.

3 Fairer Marking

This section introduces fairness problems when existing AQM approaches 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 is actually fair is deferred to a later discussion section (§ 5). Here

the discussion is confined to removing obvious fairness problems, which is why the title is not 'Fair Marking'.

When the Random Early Detection (RED) AQM algorithm was first proposed fairness was one evaluation factor [FJ93, § 8], where 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"[1].

Such fairness would be sufficient if all flows were long lived and smooth, but they are not. Wischik [Wis99] 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.

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.

Figure 2 shows a progression of four scenarios, reduced to their essentials by using equisized packets; one timeslot per delivered packet; a constant rate link; no congestion control; and a step ECN marking threshold. In the first three scenarios (a–c), two flows (grey and pink) fully utilize the link, consuming 80% and 20% each and the grey flow arrives in bursts that occupy half the queue delay threshold in the buffer. Below the progression of traffic scenarios is described, then the marking outcomes in a second pass:

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 one were to coincide with a grey burst).

[1] Here a 'connection' meant a unidirectional flow, and the paper went on to explain that "RED gateways do not attempt to ensure that each connection receives the same fraction of the total throughput"

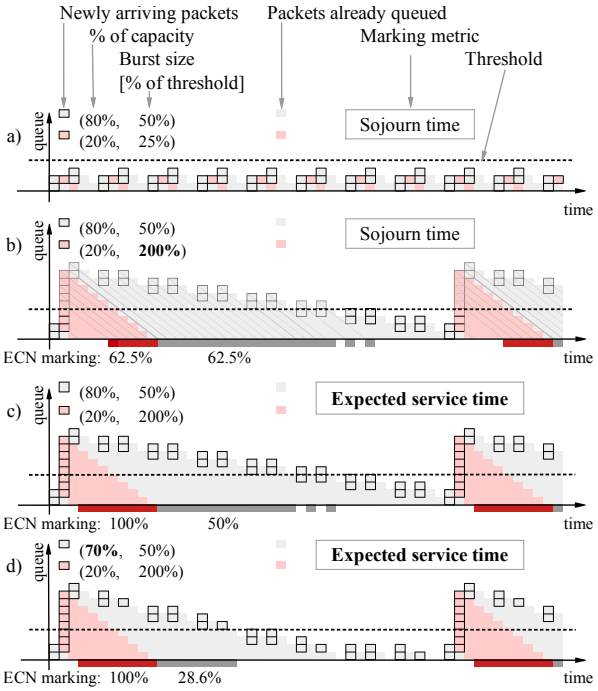

Figure 2: The Marking Fairness Problem with the Sojourn metric (b) and the ECN marking outcomes of a Solution (c–d) (see text for commentary)

b) In this case, each burst from the pink flow occupies 200% of the threshold in the buffer and arrives 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, but 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.

c) The arrivals in this scenario are identical to (b), but the marking algorithm is different (see later).

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 outcome of each is 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).

4 Solution: Expected Service Time

Three approaches will be proposed to minimize signalling delay, all starting with the backlog measured instantaneously at dequeue, then translating it into the expected time needed to drain this backlog—thus all three measure the expected service time:

- Time-based backlog;
- Time-adjusted thresholds;
- Scaled sojourn time.

4.1 Time-Based Backlog

4.2 Time-Adjusted Thresholds

4.3 Scaled Sojourn Time

It is proposed to solve this problem by scaling the sojourn time by the ratio of the backlogs at dequeue and enqueue. That is, the expected service time at any instant will be:

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

where backlog_at_enq can be written into the packet's metadata at enqueue.

4.3.1 Rationale for Scaling Sojourn Time

As Table 1 shows, it takes time to measure a representative rate and it takes time to measure a time. Ideally, but perhaps naïvely, just before forwarding a packet one could estimate the instantaneous drain rate as the serialization time of the previous head packet. Then one could calculate the instantaneous queuing delay as the instantaneous backlog divided by this instantaneous drain rate.

Then, for instance, if the arrival rate and drain rate have been constant while a packet works through the queue, but then the drain rate halves just before the packet is forwarded, the instantaneous queuing delay of the remaining queue will be double that measured by the sojourn time technique, whether or not it is scaled as proposed.

However, there is no reason to believe that the latest instantaneous rate measurement is the best estimate of the rate at which the remaining queue 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.

Therefore, an estimate based on how the rate varied while the current head packet worked through the queue is not necessarily less correct than an estimate based on the drain rate at the instant the previous head packet departed.

Also, rather than having to arbitrarily choose a number of packets to measure over, sojourn time techniques automatically tune the most commonly used measurement duration to the most common queuing delay.

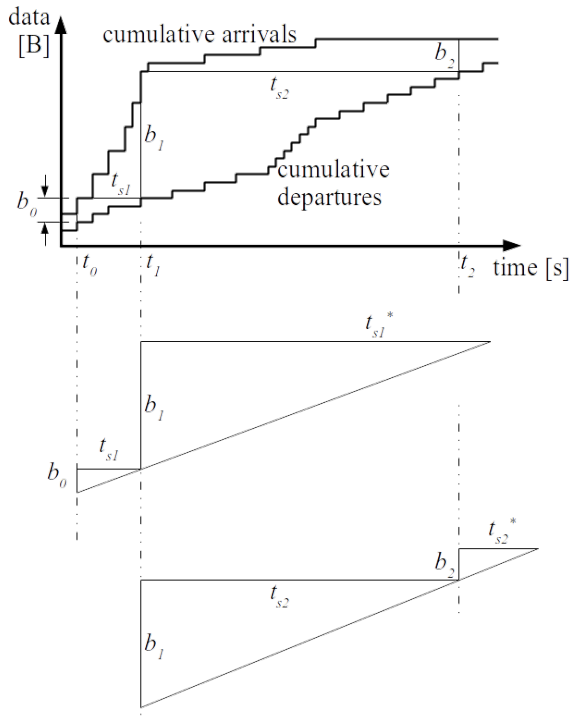

Figure 3: Rationale for Scaling Sojourn Time

Figure 3 visualizes 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 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

$$r_{d1} = \frac{b_0}{t_{s1}}. \tag{1}$$

The predicted sojourn time to drain the backlog at t_1 is

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

Substituting from Equation 1:

$$= t_{s1} \frac{b_1}{b_0}. \tag{2}$$

Another way to think of the scaling is as the 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:

$$b_1 = t_{s1} r_{a1}. \tag{3}$$

Substituting this into Equation 2, the scaled sojourn time at t_1,

$$t^*_{s1} = t_{s1} \frac{r_{a1}}{r_{d1}} \tag{4}$$

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.

4.3.2 Implementing Scaled Sojourn Time

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 following trick is likely to optimize execution of sojourn time scaling, although its efficiency will be machine-architecture-dependent:

```
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 equivalent, 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 further rationale for scaling the sojourn time is that an implementation that is already measuring the sojourn time does not need any additional measurement code, because it already has to maintain a count of the backlog to do basic queue handling. 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 3):

`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.

To implement scaling of the sojourn time, it is necessary to store `backlog_enq` in the packet's metadata when the packet is enqueued. Then at dequeue it can be combined with `backlog_deq` using the trick above.

4.3.3 Distributed Queues

Using sojourn time leverages the advantage that it can be measured across a complex set of queues, including the case where the inital enqueue and the final dequeue routines are distributed across different machines or processors, as already mentioned.

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 this case, the timestamp and backlog at enqueue would have to be included in the protocol data units being transmitted between machines (e.g. within the L2 protocol), not just in packet metadata held within one machine's memory space. Also the aggregation node would need high priority (pref. non-blocking) access to the `count_enq` variable on the input machine, in order to calculate `backlog_deq`. Certain access network technologies, e.g. those for cellular radio access networks, already include such a control channel. The delay to access a control variable at the 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 would still provide robust metrics to control queuing in the data channel.

Of course, the sojourn time based on just a timestamp at enqueue could be written into PDUs to control any of the above distributed access networks, without the extra need for a non-blocking control channel. However, this would not provide the extra timeliness that scaled sojourn time would.

The measured sojourn time would include the delay before a packet or frame was given access to the shared medium, which would be the main cause of the backlog at the client queue. As well as the aggregation node using (scaled) sojourn time to apply congestion signalling within the final dequeue routine (effectively on behalf of the input queue), the aggregation node could also use (scaled) sojourn time 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.

4.3.4 Applicability of Scaled Sojourn Time

Scaling the sojourn time improves its timeliness, so it is applicable wherever sojourn time itself is useful.

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, scaling the sojourn time 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 earlier). Scaling the sojourn time whenever it is sampled should eliminate the need for this metric because it takes account of the reducing backlog as the queue drains. Indeed, this was the original motivation for developing the scaled sojourn time metric.

Scaling the sojourn time is also highly 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. In CoDel's case, sojourn time is measured per packet, so the scaling would have to be applied per packet. Nonetheless, the trick above at least minimizes the cycles required.

Scaling the sojourn time should also be applicable to a simple low threshold algorithm like the time-based threshold recommended for DCTCP in [BCCW16] and proposed as the native AQM for more general, so-called 'L4S' traffic in DualPI2 [DSBEBT17], where L4S stands for Low Latency, Low Loss, Scalable throughput. It would be applicable whether the threshold is a simple step, or a probabilistic ramp like the RED function (but based on instantaneous sojourn time, not smoothed queue length), or a deterministic ramp or convex function of instantaneous queueing delay. However, given these schemes are intended to keep queue delay very low, there is less scope for widely varying queue dynamics, so the cost of the extra processing might not prove to be worth the benefit.

Scaling the sojourn time of a queue applies 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: Discussion

Give results of a scan of the range of mark-ing out-

[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.

Discuss lack of definition of marking fairness and discuss SPSP

Subsection to discuss potential for Queue Protection without per-flow state.

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 other work where they have been considered.

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.

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

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 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.[4]

Therefore, in this report we have assumed a standard unary encoding of congestion signals. 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].

6.1 Removing Randomness Delays

One of the main motivations for the design of Random Early Detection (RED) [FJ93] was to introduce randomness 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].

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.

The determinism would be lost wherever the AQM was controlling flows mulitplexed 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.

[4] However, encoding delay does increase with the degree of ACK coalescing.

Explain the distinction between the marking algo determining

PDPC+ [SLMP03] and CoDel [NJ12], which is very similar, use a deterministic rather the probabilistic algorithm to encode the congestion signal. However, they do not propose a way to introduce randomness in the end-systems instead. Therefore, they are likely to be prone to synchronization effects.

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 3).

8 Further Work

These ideas might not be novel, but no concerted effort has been made to search the literature. The ideas have not been evaluated either.

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.

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.

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

[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).

[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.

[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.

[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.

[Wis99] Damon Wischik. How to Mark Fairly. In *Workshop on Internet Service Quality Economics. MIT*, 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 April 2019	Bob Briscoe	Added abstract

