Abstract

The "persistently full buffer" problem has been discussed in the IETF community since the early 80’s [RFC896]. The IRTF’s End-to-End Working Group called for the deployment of active queue management (AQM) to solve the problem in 1998 [RFC2309]. Despite the awareness, the problem has only gotten worse as Moore’s Law growth in memory density fueled an exponential increase in buffer pool size. Efforts to deploy AQM have been frustrated by difficult configuration and negative impact on network utilization. This problem, recently christened "bufferbloat", [TSVBB2011] [BB2011] has become increasingly important throughout the Internet but particularly at the consumer edge.

This document describes a general framework called CoDel (Controlled Delay) [CODEL2012] that controls bufferbloat-generated excess delay in modern networking environments. CoDel consists of an estimator, a setpoint, and a control loop. It requires no configuration in normal Internet deployments. CoDel comprises some major technical innovations and has been made available as open source so that the framework can be applied by the community to a range of problems. It has been implemented in Linux (and available in the Linux distribution) and deployed in some networks at the consumer edge. In addition, the framework has been successfully applied in other ways.

Note: Code Components extracted from this document must include the license as included with the code in Section 5.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute...
Internet-Draft CoDel October 2014

working documents as Internet-Drafts. The list of current Internet-
Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months
and may be updated, replaced, or obsoleted by other documents at any
time. It is inappropriate to use Internet-Drafts as reference
material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 3, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................... 3
2. Conventions used in this document .......................... 4
3. Building Blocks of Queue Management ........................ 5
   3.1. Estimator ........................................... 6
   3.2. Setpoint .......................................... 8
   3.3. Control Loop ....................................... 9
4. Putting it together: queue management for the network edge . 12
   4.1. Overview of CoDel AQM ............................... 12
   4.2. Non-starvation ..................................... 13
   4.3. Using the interval .................................. 13
   4.4. The target Setpoint ............................... 14
   4.5. Use with multiple queues ............................ 15
   4.6. Use of stochastic bins or sub-queues to improve
        performance ....................................... 15
   4.7. Setting up CoDel AQM ............................... 16
5. Annotated Pseudo-code for CoDel AQM ........................ 17
   5.1. Data Types ......................................... 18
   5.2. Per-queue state (codel_queue_t instance variables) .... 19
   5.3. Constants ......................................... 19
   5.4. Enque routine ..................................... 19
   5.5. Deque routine ..................................... 19
1. Introduction

The need for queue management has been evident for decades. Recently the need has become more critical due to the increased consumer use of the Internet mixing large video transactions with time-critical VoIP and gaming. Gettys [TSV2011, BB2011] has been instrumental in publicizing the problem and the measurement work [CHARB2007, NATAL2010] and coining the term bufferbloat. Large content distributors such as Google have observed that bufferbloat is ubiquitous and adversely affects performance for many users. The solution is an effective AQM that remediates bufferbloat at a bottleneck while "doing no harm" at hops where buffers are not bloated.

The development and deployment of effective active queue management has been hampered by persistent misconceptions about the cause and meaning of queues. Network buffers exist to absorb the packet bursts that occur naturally in statistically multiplexed networks. Short-term mismatches in traffic arrival and departure rates that arise from upstream resource contention, transport conversation startup transients and/or changes in the number of conversations sharing a link create queues. Unfortunately, other network behavior can cause queues to fill and their effects aren’t nearly as benign. Discussion of these issues and why the solution isn’t just smaller buffers can be found in [RFC2309], [VANQ2006], [REDL1998] and [CODEL2012]. It is critical to understand the difference between the necessary, useful "good" queue and the counterproductive "bad" queue.

Many approaches to active queue management (AQM) have been developed over the past two decades but none has been widely deployed due to performance problems. When designed with the wrong conceptual model for queues, AQMs have limited operational range, require a lot of configuration tweaking, and frequently impair rather than improve performance. Today, the demands on an effective AQM are even
greater: many network devices must work across a range of bandwidths, either due to link variations or due to the mobility of the device. The CoDel approach is designed to meet the following goals:

- is parameterless for normal operation - has no knobs for operators, users, or implementers to adjust
- treats "good queue" and "bad queue" differently, that is, keeps delay low while permitting necessary bursts of traffic
- controls delay while insensitive (or nearly so) to round trip delays, link rates and traffic loads; this goal is to "do no harm" to network traffic while controlling delay
- adapts to dynamically changing link rates with no negative impact on utilization
- is simple and efficient (can easily span the spectrum from low-end, linux-based access points and home routers up to high-end commercial router silicon)

Since April, 2012, when CoDel was published, a number of talented and enthusiastic implementers have been using and adapting it with promising results. Much of this work is collected at: http://www.bufferbloat.net/projects/codel. CoDel has five major innovations that distinguish it from prior AQMs: use of local queue minimum to track congestion ("bad queue"), use of an efficient single state variable representation of that tracked statistic, use of packet sojourn time as the observed datum, rather than packets, bytes, or rates, use of mathematically determined setpoint derived from maximizing the network power metric, and a modern state space controller.

CoDel configures itself based on a round-trip time metric which can be set to 100ms for the normal, terrestrial Internet. With no changes to parameters, we have found CoDel to work across a wide range of conditions, with varying links and the full range of terrestrial round trip times. CoDel has been implemented in Linux very efficiently and should lend itself to silicon implementation. CoDel is well-adapted for use in multiple queued devices and has been used by Eric Dumazet with multiple queues in sophisticated queue management approach, fq_codel (covered in another draft).

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].
In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. Building Blocks of Queue Management

Two decades of work on queue management failed to yield an approach that could be widely deployed in the Internet. With careful tuning for particular usages, queue management techniques have been able to "kind of" work, that is decrease queuing delays, but utilization and fairness suffer unduly. At the heart of queue management is the notion of "good queue" and "bad queue" and the search for ways to get rid of the bad queue (which only adds delay) while preserving the good queue (which provides for good utilization). This section explains queuing, both good and bad, and covers the innovative CoDel building blocks that can be used to manage packet buffers to keep their queues in the "good" range.

Packet queues form in buffers facing bottleneck links, i.e., where the line rate goes from high to low or many links converge. The well-known bandwidth-delay product (sometimes called "pipe size") is the bottleneck’s bandwidth multiplied by the sender-receiver-sender round-trip delay and is the amount of data that has to be in transit between two hosts in order to run at 100% utilization. To explore how queues can form, consider a long-lived TCP connection with a 25 packet window sending through a connection with a bandwidth-delay product of 20 packets. After an initial burst of packets the connection will settle into a five packet (+/-1) standing queue, the size determined by the window mismatch to the pipe size and unrelated to the connection’s sending rate. The connection has 25 packets in flight at all times, but only 20 packets arrive at the destination over a round trip time. If the TCP connection has a 30 packet window, the queue will be ten packets with no change in sending rate. Similarly, if the window is 20 packets, there will be no queue but the sending rate is the same. Nothing can be inferred about the sender rate from the queue and the existence of any queue at all other than transient bursts can only create delay in the network. The sender needs to reduce the number of packets in flight rather than sending rate.

In the above example, the five packet standing queue can be seen to contribute nothing but delay to the connection thus is clearly "bad queue". If, in our example, there is a single bottleneck link and it
is much slower than the link that feeds it (say, a high-speed ethernet link into a limited DSL uplink) a 20 packet buffer at the bottleneck might be necessary to temporarily hold the 20 packets in flight to keep the utilization high. The burst of packets should drain completely (to 0 or 1 packets) within a round trip time and this transient queue is "good queue" because it allows the connection to keep the 20 packets in flight and for the bottleneck link to be fully utilized. In terms of the delay experienced we can observe that "good queue" goes away in about a round trip time, while "bad queue" hangs around causing delays.

Effective queue management detects "bad queue" while ignoring "good queue" and takes action to get rid of the bad queue when it is detected. The goal is a queue controller that accomplishes this objective. To control queue, we need three basic components

- Estimator - figure out what we’ve got
- Setpoint - know what what we want
- Control loop - if what we’ve got isn’t what we want, we need a way to move it there

3.1. Estimator

The Estimator both observes the queue and detects when good queue turns to bad queue and vice versa. CoDel has two innovations in its Estimator: what is observed as an indicator of queue and how the observations are used to detect good/bad queue.

In the past, queue length has been widely used as an observed indicator of congestion and is frequently conflated with sending rate. Use of queue length as a metric is sensitive to how and when the length is observed. A high speed arrival link to a buffer serviced at a much lower rate can rapidly build up a queue that might disperse completely or down to a single packet before a round trip time has elapsed. If the queue length is monitored at packet arrival (as in original RED) or departure time, every packet will see a queue with one possible exception. If the queue length itself is time sampled (as recommended in [REDL1998], a truer picture of the queue’s occupancy can be gained but a separate process is required.

The use of queue length is further complicated in networks that are subject to both short and long term changes in available link rate (as in wifi). Link rate drops can result in a spike in queue length that should be ignored unless it persists. The length metric is problematic when what we really want to control is the amount of excess delay packets experience due to a persistent or standing
queue. The sojourn time that a packet spends in the buffer is exactly what we want to track. Tracking the packet sojourn times in the buffer observes the actual delay experienced by each packet. Sojourn time is independent of link rate, gives superior performance to use of buffer size, and is directly related to the user-visible performance. It works regardless of line rate changes or whether the link is shared by multiple queues (which the individual queues may experience as changing rates).

Consider a link shared by two queues, one priority queue and one of lower priority. Packets that arrive to the high priority queue are sent as soon as the link is available while packets of the other queue have to wait till the the priority queue is empty (i.e., a strict priority scheduler). The number of packets in the priority queue might be large but the queue is emptied quickly and the amount of time each packet spends enqueued (the sojourn time) is not large. The other queue might have a smaller number of packets, but packet sojourn times will include the wait for the high priority packets to be sent. This makes the sojourn times a good sample of the congestion that each separate queue is experiencing and shows how this metric is independent of the number of queues used or the service discipline and instead reflective of the congestion seen by the individual queue.

With sojourn time as the observation, how can it be used to separate good queue from bad queue? In the past, averages, in particular of queue length, have been used to determine bad queue. Consider the burst that disperses every round trip time. The average queue will be one-half the burst size, though this might vary depending on when the average is computed and the timing of arrivals. The average then would indicate a persistent queue where there is none. If instead we track the minimum observation, if there is one packet that has a zero sojourn time then there is no persistent queue. The value of the minimum in detecting persistent queue is apparent when looking at graphs of queue delay.

The standing queue can be detected by tracking the (local) minimum queue delay packets experience. To ensure that this minimum value does not become stale, it has to have been experienced recently, i.e. during an appropriate past time interval. This “interval” is the maximum amount of time a minimum is considered to be in effect. It is clear that this interval should be at least a round trip time to avoid falsely detecting a persistent queue and not a lot more than a round trip time to avoid delay in detecting the persistent queue. This suggests that the appropriate interval value is the maximum round-trip time of all the connections sharing the buffer. To avoid outlier values, the 95-99th percentile value is preferred rather than a strict maximum.
A key realization makes the local minimum an efficiently computed statistic. Note that it is sufficient to keep a single state variable of how long the minimum has been above or below a target value rather than retaining all the local values to compute the minimum, leading to both storage and computational savings.

These two innovations, use of sojourn time as observed values and the local minimum as the statistic to monitor queue congestion are key to CoDel’s Estimator building block. The local minimum sojourn time provides an accurate and robust measure of standing queue and has an efficient implementation. In addition, use of the minimum sojourn time has important advantages in implementation. The minimum packet sojourn can only be decreased when a packet is dequeued which means that all the work of CoDel can take place when packets are dequeued for transmission and that no locks are needed in the implementation. The minimum is the only statistic with this property.

A more detailed explanation with many pictures can be found at: http://pollere.net/Pdfdocs/GrantJul06.pdf and http://www.ietf.org/proceedings/84/slides/slides-84-tsvarea-4.pdf.

3.2. Setpoint

Now that we have a robust way of detecting standing queue, we need to have a Setpoint that tells us when to act. If the controller is set to take action as soon as the estimator has a non-zero value, the average drop rate will be maximized which minimizes TCP goodput [MACTCP1997]. Also, since this policy results in no backlog over time (no persistent queue), it also maximizes the bottleneck link bandwidth lost because of normal stochastic variation in packet interarrival time and obliterates much of the value of having a buffer. We want a setpoint that maximizes utilization while minimizing delay. Early in the history of packet networking, Kleinrock developed the analytic machinery to do this using a quantity he called _‘power’_ (the ratio of a normalized throughput to a normalized delay) [KLEIN81].

It’s straightforward to derive an analytic expression the average goodput of a TCP conversation for a given round-trip time _r_ and setpoint _f_ (where _f_ is expressed as a fraction of _r_) [VJTAR81]. Reno TCP, for example, yields:

\[
goodput = \frac{r (3 + 6f - f^2)}{4 (1+f)}
\]

Since the peak delay is just _f r_, it’s clear that _power_ is solely a function of _f_ since the _r_’s in the numerator and denominator cancel:
power = (1 + 2_f - 1/3 _f^2) / (1 + _f_)^2

As Kleinrock observed, the best operating point, in terms of bandwidth / delay tradeoff, is the peak power point since points off the peak represent a higher cost (in delay) per unit of bandwidth. The power vs. _f_ curve for any AIMD TCP is monotone decreasing. But the curve is very flat for _f_ < 0.1 followed by a increasing curvature with a knee around .2 then a steep, almost linear fall off [TSV84] [VJTARG14]. Since the previous equation showed that goodput is monotone increasing with _f_, the best operating point is near the right edge of the flat top since that represents the highest goodput achievable for a negligible increase in delay. However, since the _r_ in the model is a conservative upper bound, a target of .1_r_ runs the risk of pushing shorter RTT connections over the knee and giving them higher delay for no significant goodput increase. Generally, a more conservative target of .05_r _offers a good utilization vs. delay tradeoff while giving enough headroom to work well with a large variation in real RTT.

As the above analysis shows, a very small standing queue gives close to 100% utilization. While this result was for Reno TCP, the derivation uses only properties that must hold for any 'TCP friendly' transport. We have verified by both analysis and simulation that this result holds for Reno, Cubic, and Westwood[TSV84]. This results in a particularly simple form for the setpoint: the ideal range for the permitted standing queue is between 5 and 10% of the TCP connection RTT. Thus _target_ is simply 5% of the _interval_ of section 3.1.

3.3. Control Loop

Section 3.1 describes a simple, reliable way to measure bad (persistent) queue. Section 3.2 shows that TCP congestion control dynamics gives rise to a setpoint for this measure that’s a provably good balance between enhancing throughput and minimizing delay, and that this setpoint is a constant fraction of the same 'largest average RTT' interval used to distinguish persistent from transient queue. The only remaining building block needed for a basic AQM is a 'control loop' algorithm to effectively drive the queuing system from any 'persistent queue above target' state to a state where the persistent queue is below target.

Control theory provides a wealth of approaches to the design of control loops. Most of classical control theory deals with the control of linear, time-invariant, single-input-single-output (SISO) systems. Control loops for these systems generally come from a (well understood) class known as Proportional-Integral-Derivative (PID) controllers. Unfortunately, a queue is not a linear system and an
AQM operates at the point of maximum non-linearity (where the output link bandwidth saturates so increased demand creates delay rather than higher utilization). Output queues are also not time-invariant since traffic is generally a mix of connections which start and stop at arbitrary times and which can have radically different behaviors ranging from "open loop" UDP audio/video to "closed-loop" congestion-avoiding TCP. Finally, the constantly changing mix of connections (which can’t be converted to a single ‘lumped parameter’ model because of their transfer function differences) makes the system multi-input-multi-output (MIMO), not SISO.

Since queuing systems match none of the prerequisites for a classical controller, a modern state-space controller is a better approach with states ‘no persistent queue’ and ‘has persistent queue’. Since Internet traffic mixtures change rapidly and unpredictably, a noise and error tolerant adaptation algorithm like Stochastic Gradient is a good choice. Since there’s essentially no information in the amount of persistent queue [TSV84], the adaptation should be driven by how long it has persisted.

Consider the two extremes of traffic behavior, a single open-loop UDP video stream and a single, long-lived TCP bulk data transfer. If the average bandwidth of the UDP video stream is greater than the bottleneck link rate, the link’s queue will grow and the controller will eventually enter ‘has persistent queue’ state and start dropping packets. Since the video stream is open loop, its arrival rate is unaffected by drops so the queue will persist until the average drop rate is greater than the output bandwidth deficit (= average arrival rate - average departure rate) so the job of the adaptation algorithm is to discover this rate. For this example, the adaptation could consist of simply estimating the arrival and departure rates then dropping at a rate slightly greater than their difference. But this class of algorithm won’t work at all for the bulk data TCP stream. TCP runs in closed-loop flow balance [TSV84] so its arrival rate is almost always exactly equal to the departure rate - the queue isn’t the result of a rate imbalance but rather a mismatch between the TCP sender’s window and the src-dst-src round-trip path capacity (i.e., the connection’s bandwidth*delay product). The sender’s TCP congestion avoidance algorithm will slowly increase the send window (one packet per round-trip-time) [RFC2581] which will eventually cause the bottleneck to enter ‘has persistent queue’ state. But, since the average input rate is the same as the average output rate, the rate deficit estimation that gave the correct drop rate for the video stream would compute a drop rate of zero for the TCP stream. However, if the output link drops one packet as it enters ‘has persistent queue’ state, when the sender discovers this (via TCP’s normal packet loss repair mechanisms) it will reduce its window by a
factor of two [RFC2581] so, one round-trip-time after the drop, the
persistent queue will go away.

If there were N TCP conversations sharing the bottleneck, the
controller would have to drop O(N) packets, one from each
conversation, to make all the conversations reduce their window to
get rid of the persistent queue. If the traffic mix consists of
short (<= bandwidth*delay product) conversations, the aggregate
behavior becomes more like the open-loop video example since each
conversation is likely to have already sent all its packets by the
time it learns about a drop so each drop has negligible effect on
subsequent traffic.

The controller doesn’t know what type, how many or how long are the
conversations creating its queue so it has to learn that. Since
single drops can have a large effect if the degree of multiplexing
(the number of active conversations) is small, dropping at too high a
rate is likely to have a catastrophic effect on throughput. Dropping
at a low rate (< 1 packet per round-trip-time) then increasing the
drop rate slowly until the persistent queue goes below target is
unlikely to overdrop yet is guaranteed to eventually dissipate the
persistent queue. This stochastic gradient learning procedure is the
core of CoDel’s control loop (the gradient exists because a drop
always reduces the (instantaneous) queue so an increasing drop rate
always moves the system "down" toward no persistent queue, regardless
of traffic mix).

The next drop time is decreased in inverse proportion to the square
root of the number of drops since the dropping state was entered,
using the well-known nonlinear relationship of drop rate to
throughput to get a linear change in throughput. [REDL1998,
MACTCP1997]

Since the best rate to start dropping is at slightly more than one
packet per RTT, the controller’s initial drop rate can be directly
derived from the Estimator’s interval, defined in section 3.1. Where
the interval is likely to be very close to the usual round trip time,
the initial drop spacing SHOULD be set to the Estimator’s interval
plus twice the target (i.e., initial drop spacing = 1.1 * interval)
to ensure that acceptable congestion delays are covered.

Use of the minimum statistic lets the Controller be placed in the
dequeue routine with the Estimator. This means that the control
signal (the drop) can be sent at the first sign of bad queue (as
indicated by the sojourn time) and that the Controller can stop
acting as soon as the sojourn time falls below the Setpoint.
Dropping at dequeue has both implementation and control advantages.
4. Putting it together: queue management for the network edge

The CoDel building blocks are able to adapt to different or time-varying link rates, to be easily used with multiple queues, to have excellent utilization with low delay and to have a simple and efficient implementation. The only setting CoDel requires is its interval value, and as 100ms satisfies that definition for normal internet usage, CoDel can be parameter-free for consumer use. CoDel was released to the open source community where it has been widely promulgated and adapted to many problems. We can see how well these building blocks work in a simple CoDel queue management implementation. This AQM was designed as a bufferbloat solution and is focused on the consumer network edge.

4.1. Overview of CoDel AQM

To ensure that link utilization is not adversely affected, CoDel’s Estimator sets its target to the Setpoint that optimizes power and CoDel’s Controller does not drop packets when the drop would leave the queue empty or with fewer than a maximum transmission unit (MTU) worth of bytes in the buffer. Section 3.2 showed that the ideal Setpoint is 5-10% of the connection RTT. In the open Internet, in particular the consumer edge, we can use the "usual maximum" terrestrial RTT of 100 ms to calculate a minimum target of 5ms. Under the same assumptions, we compute the interval for tracking the minimum to be the nominal RTT of 100ms. In practice, uncongested links will see sojourn times under the target more often than once per RTT, so the Estimator is not overly sensitive to the value of the interval.

When the Estimator finds a persistent delay above target, the Controller enters the drop state where a packet is dropped and the next drop time is set. As discussed in section 3.3, the initial next drop spacing is intended to be long enough to give the endpoints time to react to the single drop so SHOULD be set to a value of 1.0 to 1.1 times the interval. If the Estimator’s output falls below the target, the Controller cancels the next drop and exits the drop state. (The Controller is more sensitive than the Estimator to an overly short interval, since an unnecessary drop could occur and lower utilization.) If next drop time is reached while the Controller is still in drop state, the packet being dequeued is dropped and the next drop time is recalculated. Additional logic prevents re-entering the dropping state too soon after exiting it and resumes the dropping state at a recent control level, if one exists.

Note that CoDel AQM only enters its dropping state when the local minimum sojourn delay has exceeded an acceptable standing queue target for a time interval long enough for normal bursts to dissipate.
ensuring that a burst of packets that fits in the pipe will not be dropped.

CoDel’s efficient implementation and lack of configuration are unique features and make it suitable to manage modern packet buffers. For more background and results on CoDel, see [CODEL2012] and http://pollere.net/CoDel.html.

4.2. Non-starvation

CoDel’s goals are to control delay with little or no impact on link utilization and to be deployed on a wide range of link bandwidth, including varying rate links, without reconfiguration. To keep from making drops when it would starve the output link, CoDel makes another check before dropping to see if at least an MTU worth of bytes remains in the buffer. If not, the packet SHOULD NOT be dropped and, currently, CoDel exits the drop state. The MTU size can be set adaptively to the largest packet seen so far or can be read from the driver.

4.3. Using the interval

The interval is chosen to give endpoints time to react to a drop without being so long that response times suffer. CoDel’s Estimator, Setpoint, and Control Loop all use the interval. Understanding their derivation shows that CoDel is the most sensitive to the value of interval for single long-lived TCPs with a decreased sensitivity for traffic mixes. This is fortunate as RTTs vary across connections and are not known apriori and it’s difficult to obtain a definitive histogram of RTTs seen on the normal consumer edge link. The best policy is to use an interval slightly larger than the RTT seen by most of the connections using a link, a value that can be determined as the largest RTT seen if the value is not an outlier (as in section 3.1, use of a 95-99th percentile value should work). In practice, this value is not known or measured (though see Section 6.2 for an application where interval is measured. Work-in-progress at Pollere may lead to a method of doing this in an Internet buffer). A setting of 100ms works well across a range of RTTs from 10ms to 1 second (excellent performance is achieved in the range from 10 ms to 300ms). For devices intended for the normal terrestrial Internet interval SHOULD have the value of 100ms. This will only cause overdropping where a long-lived TCP has an RTT longer than 100ms and there is little or no mixing with other connections through the link.

Some confusion concerns the roles of the target Setpoint and the minimum-tracking interval. In particular, some experimenters believe the value of target needs to be increased when the lower layers have a bursty nature where packets are transmitted for short periods.
interspersed with idle periods where the link is waiting for permission to send. CoDel’s Estimator will "see" the effective transmission rate over an interval and increasing target will just lead to longer queue delays. On the other hand, where a significant additional delay is added to the intrinsic round trip time of most or all packets due to the waiting time for a transmission, it is necessary to increase interval by that extra delay. That is, target SHOULD NOT be adjusted but interval MAY need to be adjusted. For more on this (and pictures) see http://pollere.net/Pdfdocs/noteburstymacs.pdf

4.4. The target Setpoint

The target is the maximum acceptable standing queue delay above which CoDel is dropping or preparing to drop and below which CoDel will not drop. The calculations of section 3.2 showed that the best setpoint is 5-10% of the RTT, with the low end of 5% preferred. We used simulation to explore the impact when TCPs are mixed with other traffic and with connections of different RTTs. Accordingly, we experimented extensively with values in the 5-10% of RTT range and, overall, used target values between 1 and 20 milliseconds for RTTs from 30 to 500ms and link bandwidths of 64Kbps to 100Mbps to experimentally explore the Setpoint that gives consistently high utilization while controlling delay across a range of bandwidths, RTTs, and traffic loads. Our results were notably consistent with the mathematics of section 3.2. Below a target of 5ms, utilization suffers for some conditions and traffic loads, above 5ms we saw very little or no improvement in utilization. Thus target SHOULD be set to 5ms for normal Internet traffic.

If a CoDel link has only or primarily long-lived TCP flows sharing a link to congestion but not overload, the median delay through the link will tend to the target. For bursty traffic loads and for overloaded conditions (where it is difficult or impossible for all the arriving flows to be accommodated) the median queues will be longer than target.

The non-starvation drop inhibit feature dominates where the link rate becomes very small. By inhibiting drops when there is less than an (outbound link) MTU worth of bytes in the buffer, CoDel adapts to very low bandwidth links. This is shown in [CODEL2012] and interested parties should see the discussion of results there. Unpublished studies were carried out down to 64Kbps. The drop inhibit condition can be expanded to include a test to retain sufficient bytes or packets to fill an allocation in a request-and-grant MAC.
Sojourn times must remain above target for an entire interval in order to enter the drop state. Any packet with a sojourn time less than target will reset the time that the queue was last below the target. Since Internet traffic has very dynamic characteristics, the actual sojourn delays experienced by packets varies greatly and is often less than the target unless the overload is excessive. When a link is not overloaded, it is not a bottleneck and packet sojourn times will be small or nonexistent. In the usual case, there are only one or two places along a path where packets will encounter a bottleneck (usually at the edge), so the amount of queuing delay experienced by a packet should be less than 10 ms even under extremely congested conditions. Contrast this to the queuing delays that grow to orders of seconds that have led to the "bufferbloat" term [NETAL2010, CHARRB2007].

4.5. Use with multiple queues

Unlike other AQMs, CoDel is easily adapted to multiple queue systems. With other approaches there is always a question of how to account for the fact that each queue receives less than the full link rate over time and usually sees a varying rate over time. This is exactly what CoDel excels at: using a packet’s sojourn time in the buffer completely bypasses this problem. A separate CoDel algorithm runs on each queue, but each CoDel uses the packet sojourn time the same way a single queue CoDel does. Just as a single queue CoDel adapts to changing link bandwidths [CODEL2012], so do the multiple queue CoDels. When testing for queue occupancy before dropping, the total occupancy of all bins should be used. This property of CoDel has been exploited in fq_codel, briefly discussed in the next section and the subject of another Internet Draft.

4.6. Use of stochastic bins or sub-queues to improve performance

 Shortly after the release of the CoDel pseudocode, Eric Dumazet created fq_codel, applying CoDel to each bin, or queue, used with stochastic fair queuing. (To understand further, see [SFQ1990] or the linux sfq at http://linux.die.net/man/8/tc-sfq.) Fq_codel hashes on the packet header fields to determine a specific bin, or sub-queue, for each five-tuple flow, and runs CoDel on each bin or sub-queue thus creating a well-mixed output flow and obviating issues of reverse path flows (including "ack compression"). Dumazet’s code is part of the CeroWrt project code at the bufferbloat.net’s web site and an Internet Draft has been submitted describing fq_codel, draft-hoeiland-joergensen-aqm-fq-codel. We’ve experimented with a similar approach by creating an ns-2 simulator code module, sfqcodel. This has provided excellent results thus far: median queues remain small across a range of traffic
patterns that includes bidirectional file transfers (that is, the same traffic sent in both directions on a link), constant bit-rate VoIP-like flows, and emulated web traffic and utilizations are consistently better than single queue CoDel, generally very close to 100%. Our version differs from Dumazet’s by preferring a packet-based round robin of the bins rather than byte-based DRR and there may be other minor differences in implementation. Our code, intended for simulation experiments, is available at http://pollere.net/CoDel.html and being integrated into the ns-2 distribution. Andrew McGregor has an ns-3 version of fq_codel.

Stochastic flow queuing provides better traffic mixing on the link and tends to isolate a larger flow or flows. For real priority treatment, use of DiffServ isolation is encouraged. We’ve experimented in simulation with creating a queue to isolate all the UDP traffic (which is all simulated VoIP thus low bandwidth) but this approach has to be applied with caution in the real world. Some experimenters are trying rounding with a small quantum (on the order of a voice packet size) but this also needs thorough study.

A number of open issues should be studied. In particular, if the number of different queues or bins is too large, the scheduling will be the dominant factor, not the AQM; it is NOT the case that more bins are always better. In our simulations, we have found good behavior across mixed traffic types with smaller numbers of queues, 8-16 for a 5Mbps link. This configuration appears to give the best behavior for voice, web browsing and file transfers where increased numbers of bins seems to favor file transfers at the expense of the other traffic. Our work has been very preliminary and we encourage others to take this up and to explore analytic modeling. It would be instructive to see the effects of different numbers of bins on a range of traffic models, something like an updated version of [BMPFQ].

Implementers SHOULD use the fq_codel multiple queue approach if possible as it deals with many problems beyond the reach of an AQM on a single queue.

4.7. Setting up CoDel AQM

CoDel’s is set for use in devices in the open Internet. An interval of 100ms is used, target is set to 5% of interval, and the initial drop spacing is also set to interval. These settings have been chosen so that a device, such as a small WiFi router, can be sold without the need for any values to be made adjustable, yielding a parameterless implementation. In addition, CoDel is useful in environments with significantly different characteristics from the normal Internet, for example, in switches used as a cluster.
interconnect within a data center. Since cluster traffic is entirely internal to the data center, round trip latencies are low (typically <100us) but bandwidths are high (1-40Gbps) so it’s relatively easy for the aggregation phase of a distributed computation (e.g., the Reduce part of a Map/Reduce) to persistently fill then overflow the modest per-port buffering available in most high speed switches. A CoDel configured for this environment (target and interval in the microsecond rather than millisecond range) can minimize drops (or ECN marks) while keeping throughput high and latency low.

Devices destined for these environments MAY use a different interval, where suitable. If appropriate analysis indicates, the target MAY be set to some other value in the 5-10% of interval and the initial drop spacing MAY be set to a value of 1.0 to 1.2 times the interval. But these settings will cause problems such as over dropping and low throughput if used on the open Internet so devices that allow CoDel to be configured MUST default to Internet appropriate values given in this document.

5. Annotated Pseudo-code for CoDel AQM

What follows is the CoDel algorithm in C++-like pseudo-code. Since CoDel adds relatively little new code to a basic tail-drop fifo-queue, we’ve tried to highlight just these additions by presenting CoDel as a sub-class of a basic fifo-queue base class. There have been a number of minor variants in the code and our reference pseudo-code has not yet been completely updated. The reference code is included to aid implementers who wish to apply CoDel to queue management as described here or to adapt its principles to other applications.

Implementors are strongly encouraged to also look at Eric Dumazet’s Linux kernel version of CoDel - a well-written, well tested, real-world, C-based implementation. As of this writing, it is at:

http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git;a=blob_plain;f=net/sched/sch_codel.c;hb=HEAD

This code is open-source with a dual BSD/GPL license:

Codel - The Controlled-Delay Active Queue Management algorithm

Copyright (C) 2011-2014 Kathleen Nichols <nichols@pollere.com>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
5.1. Data Types

"time_t" is an integer time value in units convenient for the system. Resolution to at least a millisecond is required and better resolution is useful up to the minimum possible packet time on the output link; 64- or 32-bit widths are acceptable but with 32 bits the resolution should be no finer than $2^{-16}$ to leave enough dynamic range to represent a wide range of queue waiting times. Narrower widths also have implementation issues due to overflow (wrapping) and underflow (limit cycles because of truncation to zero) that are not addressed in this pseudocode. The code presented here uses 0 as a flag value to indicate "no time set."

"packet_t*" is a pointer to a packet descriptor. We assume it has a tstamp field capable of holding a time_t and that field is available for use by CoDel (it will be set by the enqueue routine and used by the dequeue routine).
"queue_t" is a base class for queue objects (the parent class for
codel_queue_t objects). We assume it has enque() and deque() methods
that can be implemented in child classes. We assume it has a bytes()
method that returns the current queue size in bytes. This can be an
approximate value. The method is invoked in the deque() method but
shouldn’t require a lock with the enque() method.

"flag_t " is a Boolean.

5.2. Per-queue state (codel_queue_t instance variables)

time_t first_above_time; // Time to declare sojourn time above target
time_t drop_next; // Time to drop next packet
uint32_t count; // Packets dropped since entering drop state
flag_t dropping; // Equal to 1 if in drop state

5.3. Constants

time_t target = MS2TIME(5); // 5ms target queue delay
time_t interval = MS2TIME(100); // 100ms sliding-minimum window
u_int maxpacket = 512; // Maximum packet size in bytes
    // (should use interface MTU)

5.4. Enque routine

All the work of CoDel is done in the deque routine. The only CoDel
addition to enque is putting the current time in the packet’s tstamp
field so that the deque routine can compute the packet’s sojourn
time.

void codel_queue_t::enque(packet_t* pkt)
{
    pkt->timestamp() = clock();
    queue_t::enque(pkt);
}

5.5. Deque routine

This is the heart of CoDel. There are two branches: In packet-
dropping state (meaning that the queue-sojourn time has gone above
target and hasn’t come down yet), then we need to check if it’s time
to leave or if it’s time for the next drop(s); if we’re not in
dropping state, then we need to decide if it’s time to enter and do
the initial drop.
Packet* CoDelQueue::deque()
{
    double now = clock();
    dequeueResult r = dodeque(now);

    if (dropping_)
    {
        if (! r.ok_to_drop)
        {
            // sojourn time below target - leave dropping state
            dropping_ = 0;
        }
        // Time for the next drop. Drop current packet and dequeue
        // next. If the dequeue doesn't take us out of dropping
        // state, schedule the next drop. A large backlog might
        // result in drop rates so high that the next drop should
        // happen now, hence the 'while' loop.
        while (now >= drop_next_ && dropping_)
        {
            drop(r.p);
            r = dodeque(now);
            if (! r.ok_to_drop)
            {
                // leave dropping state
                dropping_ = 0;
            } else {
                ++count_;
                // schedule the next drop.
                drop_next_ = control_law(drop_next_);
            }
        }
        // If we get here we’re not in dropping state. The 'ok_to_drop'
        // return from dodeque means that the sojourn time has been
        // above 'target' for 'interval' so enter dropping state.
    } else if (r.ok_to_drop)
    {
        drop(r.p);
        r = dodeque(now);
        dropping_ = 1;

        // If min went above target close to when it last went
        // below, assume that the drop rate that controlled the
        // queue on the last cycle is a good starting point to
        // control it now. ('drop_next' will be at most 'interval'
        // later than the time of the last drop so 'now - drop_next'
        // is a good approximation of the time from the last drop
        // until now.)
        count_ = (count_ > 2 && now - drop_next_ < 8*interval_)? count_ - 2 : 1;
        drop_next_ = control_law(now);
    }
    return (r.p);
}
5.6. Helper routines

Since the degree of multiplexing and nature of the traffic sources is unknown, CoDel acts as a closed-loop servo system that gradually increases the frequency of dropping until the queue is controlled (sojourn time goes below target). This is the control law that governs the servo. It has this form because of the $\sqrt{p}$ dependence of TCP throughput on drop probability. Note that for embedded systems or kernel implementation, the inverse $\sqrt{p}$ can be computed efficiently using only integer multiplication. See Eric Dumazet’s excellent Linux CoDel implementation for example code (in file net/sched/sch_codel.c of the kernel source for 3.5 or newer kernels).

```c
time_t codel_queue_t::control_law(time_t t)
{
    return t + interval / sqrt(count);
}
```

Next is a helper routine that does the actual packet dequeue and tracks whether the sojourn time is above or below target and, if above, if it has remained above continuously for at least interval. It returns two values, a Boolean indicating if it is OK to drop (sojourn time above target for at least interval) and the packet dequeued.
typedef struct {
    packet_t* p;
    flag_t ok_to_drop;
} dodeque_result;

dodeque_result codel_queue_t::dodeque(time_t now)
{
    dodequeResult r = { NULL, queue_t::deque() };
    if (r.p == NULL) {
        // queue is empty - we can’t be above target
        first_above_time_ = 0;
        return r;
    }

    // To span a large range of bandwidths, CoDel runs two
    // different AQMs in parallel. One is sojourn-time-based
    // and takes effect when the time to send an MTU-sized
    // packet is less than target. The 1st term of the "if"
    // below does this. The other is backlog-based and takes
    // effect when the time to send an MTU-sized packet is >=
    // target. The goal here is to keep the output link
    // utilization high by never allowing the queue to get
    // smaller than the amount that arrives in a typical
    // interarrival time (MTU-sized packets arriving spaced
    // by the amount of time it takes to send such a packet on
    // the bottleneck). The 2nd term of the "if" does this.
    time_t sojourn_time = now - r.p->tstamp;
    if (sojourn_time_ < target_ || bytes() <= maxpacket_) {
        // went below - stay below for at least interval
        first_above_time_ = 0;
    } else {
        if (first_above_time_ == 0) {
            // just went above from below. if still above at
            // first_above_time, will say it’s ok to drop.
            first_above_time_ = now + interval_;
        } else if (now >= first_above_time_) {
            r.ok_to_drop = 1;
        }
    }

    return r;
}

5.7. Implementation considerations

Since CoDel requires relatively little per-queue state and no direct
communication or state sharing between the enqueue and dequeue
routines, it’s relatively simple to add it to almost any packet
processing pipeline, including ASIC- or NPU-based forwarding engines.
One issue to think about is dedeque’s use of a ‘bytes()’ function to find out about how many bytes are currently in the queue. This value does not need to be exact. If the enqueue part of the pipeline keeps a running count of the total number of bytes it has put into the queue and the dequeue routine keeps a running count of the total bytes it has removed from the queue, ‘bytes()’ is just the difference between these two counters. 32 bit counters are more than adequate. Enqueue has to update its counter once per packet queued but it doesn’t matter when (before, during or after the packet has been added to the queue). The worst that can happen is a slight, transient, underestimate of the queue size which might cause a drop to be briefly deferred.

6. Adapting and applying CoDel’s building blocks

CoDel is being implemented and tested in a range of environments. Dave Taht has been instrumental in the integration and distribution of bufferbloat solutions, including CoDel, and has set up a website and a mailing list for CeroWRT implementers. (www.bufferbloat.net/projects/codel) This is an active area of work and an excellent place to track developments.

6.1. Validations and available code

An experiment by Stanford graduate students successfully used the Linux CoDel to duplicate our published simulation work on CoDel’s ability to following drastic link rate changes which can be found at: http://reproducingnetworkresearch.wordpress.com/2012/06/06/solving-bufferbloat-the-codel-way/.

Our ns-2 simulations are available at http://pollere.net/CoDel.html. Cable Labs has funded some additions to the simulator sfqcode code which have been made public. The basic algorithm of CoDel remains unchanged, but we continue to experiment with drop interval setting when resuming the drop state, inhibiting or canceling drop state when bytes in the queue small, and other minor details. Our approach to changes is to only make them if we are convinced they do more good than harm, both operationally and in the implementation. With this in mind, some of these issues may continue to evolve as we get more deployment and as the building blocks are applied to a wider range of problems.

CoDel is being made available with the ns-2 distribution.

Andrew McGregor has an ns-3 implementation of both CoDel and FQ_CoDel (https://github.com/dtaht/ns-3-dev ).
CoDel is available in Linux. Eric Dumazet has put CoDel into the Linux distribution.

6.2. CoDel in the datacenter

Nandita Dukkipati’s team at Google was quick to realize that the CoDel building blocks could be applied to bufferbloat problems in datacenter servers, not just to Internet routers. The Linux CoDel queueing discipline (Qdisc) was adapted in three ways to tackle this bufferbloat problem.

1. The default CoDel action was modified to be a direct feedback from Qdisc to the TCP layer at dequeue. The direct feedback simply reduces TCP’s congestion window just as congestion control would do in the event of drop. The scheme falls back to ECN marking or packet drop if the TCP socket lock could not be acquired at dequeue.

2. Being located in the server makes it possible to monitor the actual RTT to use as CoDel’s interval rather than making a "best guess" of RTT. The CoDel interval is dynamically adjusted by using the maximum TCP round-trip time (RTT) of those connections sharing the same Qdisc/bucket. In particular, there is a history entry of the maximum RTT experienced over the last second. As a packet is dequeued, the RTT estimate is accessed from its TCP socket. If the estimate is larger than the current CoDel interval, the CoDel interval is immediately refreshed to the new value. If the CoDel interval is not refreshed for over a second, it is decreased it to the history entry and the process repeated. The use of the dynamic TCP RTT estimate lets interval adapt to the actual maximum value currently seen and thus lets the controller space its drop intervals appropriately.

3. Since the mathematics of computing the set point are invariant, a target of 5% of the RTT or CoDel interval was used here.

Non-data packets were not dropped as these are typically small and sometimes critical control packets. Being located on the server, there is no concern with misbehaving users scamming such a policy as there would be in an Internet router.

In several data center workload benchmarks, which are typically bursty, CoDel reduced the queueing latencies at the Qdisc, and thereby improved the mean and 99 percentile latencies from several tens of milliseconds to less than one millisecond. The minimum tracking part of the CoDel framework proved useful in disambiguating "good" queue versus "bad" queue, particularly helpful in controlling...
7. Security Considerations

This document describes an active queue management algorithm for implementation in networked devices. There are no specific security exposures associated with CoDel.

8. IANA Considerations

This document does not require actions by IANA.

9. Conclusions

CoDel provides very general, efficient, parameterless building blocks for queue management that can be applied to single or multiple queues in a variety of data networking scenarios. It is a critical tool in solving bufferbloat. CoDel's settings MAY be modified for other special-purpose networking applications.

On-going projects are creating a deployable CoDel in Linux routers and experimenting with applying CoDel to stochastic queuing with very promising results.

10. Acknowledgments

The authors wish to thank Jim Gettys for the constructive nagging that made us get the work "out there" before we thought it was ready. We also want to thank Dave Taht, Eric Dumazet, and the open source community for showing the value of getting it "out there" and for making it real. We also wish to thank Nandita Dukkipati for contribution to section 6 and for comments which helped to substantially improve this draft.

11. References

11.1. Normative References


11.2. Informative References


Authors’ Addresses

Kathleen Nichols
Pollere, Inc.
PO Box 370201
Montara, CA  94037
USA

Email: nichols@pollere.com

Van Jacobson
Google

Email: vanj@google.com

Andrew McGregor
Google

Email: andrewmcgr@google.com
The Benefits of using Explicit Congestion Notification (ECN)
draft-ietf-aqm-ecn-benefits-02

Abstract

This document describes the potential benefits when applications enable Explicit Congestion Notification (ECN). It outlines the principal gains in terms of increased throughput, reduced delay and other benefits when ECN is used over network paths that include equipment that supports ECN-marking. It also identifies some potential problems that might occur when ECN is used. The document does not propose new algorithms that may be able to use ECN or describe the details of implementation of ECN in endpoint devices, routers and other network devices.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 23, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents.
1. Introduction

Internet Transports (such as TCP and SCTP) have two ways to detect congestion: the loss of a packet and, if Explicit Congestion Notification (ECN) [RFC3168] is enabled, by reception of a packet with a Congestion Experienced (CE)-marking in the IP header. Both of these are treated by transports as indications of (potential) congestion. ECN may also be enabled by other transports: UDP applications may enable ECN when they are able to correctly process the ECN signals (e.g. ECN with RTP [RFC6679]).

A network device (router, middlebox, or other device that forwards packets through the network) that does not support AQM, typically uses a drop-tail policy to discard excess IP packets when its queue becomes full. The discard of packets serves as a signal to the end-to-end transport that there may be congestion on the network path being used. This triggers a congestion control reaction to reduce the maximum rate permitted by the sending endpoint.

When an application uses a transport that enables the use of ECN, the transport layer sets the ECT(0) or ECT(1) codepoint in the IP header of packets that it sends. This indicates to network devices that they may mark, rather than drop, packets in periods of congestion. This marking is generally performed by Active Queue Management (AQM) [RFC2309.bis] and may be the result of various AQM algorithms, where the exact combination of AQM/ECN algorithms does not need to be known by the transport endpoints. The focus of this document is on usage of ECN by transport and application layer flows, not its implementation in hosts, routers and other network devices.

ECN makes it possible for the network to signal the presence of congestion without incurring packet loss. This lets the network deliver some packets to an application that would otherwise have been dropped if the application or transport did not support ECN. This packet loss reduction is the most obvious benefit of ECN, but it is often relatively modest. However, enabling ECN can also result in a number of beneficial side-effects, some of which may be much more significant than the immediate packet loss reduction from ECN-marking instead of dropping packets. Several of these benefits have to do with reducing latency in some way (e.g., reduced Head-of-Line Blocking and potentially smaller queuing delay, depending on the marking rules in network devices).
The remainder of this document discusses the potential for ECN to positively benefit an application without making specific assumptions about configuration or implementation.

[RFC3168] describes a method in which a network device sets the CE codepoint of an ECN-Capable packet at the time that the router would otherwise have dropped the packet. While it has often been assumed that network devices should CE-mark packets at the same level of congestion at which they would otherwise have dropped them, separate configuration of the drop and mark thresholds is known to be supported in some network devices and this is recommended [RFC2309.bis]. Some benefits of ECN that are discussed rely upon network devices marking packets at a lower level of congestion, before they would otherwise drop packets from queue overflow [KH13].

The ability to use ECN relies upon using a transport that can support ECN. Some benefits are also only realized when the transport endpoint behaviour is also updated, this is discussed further in Section 5.

2. ECN Deployment

For an application to use ECN requires that the endpoint first enables ECN within the transport.

The ability to use ECN requires network devices along the path to at least pass IP packets that set ECN codepoints, and do not drop packets because these codepoints are used Section 2.2. This is the recommended behaviour for network devices [RFC2309.bis] [RFC3168]. Applications and transports (such as TCP or SCTP) can be designed to fall-back to not using ECN when they discover they are using a path that does not allow use of ECN (e.g., a firewall or other network device configured to drop the ECN codepoint) Section 6.1.

For an application to gain benefit from using a transport that enables ECN, network devices need to enable ECN marking. However, not all network devices along the path need to enable ECN, for the application to benefit. Any network device that does not mark an ECN-enabled packet with a CE-codepoint can be expected to drop packets under congestion. Applications that experience congestion in these network devices do not see any benefit from using ECN, but would see benefit if the congestion were to occur within a network device that did support ECN.

ECN can be deployed both in the general Internet and in controlled environments:
ECN can be incrementally deployed in the general Internet. The IETF has provided guidance on configuration and usage in [RFC2309.bis]. A recent survey reported growing support for ECN on common network paths [TR15].

ECN may also be deployed within a controlled environment, for example within a data centre or within a well-managed private network. In this case, the use of ECN may be tuned to the specific use-case. An example is Datacenter TCP (DCTCP) [AL10].

Some mechanisms that can assist in using ECN across paths that only partially supports ECN are noted in Section 6.

2.1. Enabling ECN in network devices

Network deployment needs also to consider the requirements for processing ECN at tunnel endpoints of network tunnels, and guidance on the treatment of ECN is provided in [RFC6040].

Further guidance on the encapsulation and use of ECN by non-IP network devices is provided in [ID.ECN-Encap].

2.2. Bleaching and middlebox requirements to deploy ECN

Cases have been noted where a sending endpoint marks a packet with a non-zero ECN mark, but the packet is received with a zero ECN value by the remote endpoint.

The current IPv4 and IPv6 specifications assign usage of 2 bits in the IP header to carry the ECN codepoint [RFC2474] [RFC3168]. A previous usage assigned these bits as a part of the now deprecated Type of Service (ToS) field [RFC1349]. Network devices that conform to this older specification may still remark or erase the ECN codepoints, and such equipment needs to be updated to the current specifications to support ECN. This remarking has also been called "ECN bleaching".

Some network devices have been observed to implement a policy that erases or "bleaches" the ECN marks at a network edge (resetting these to zero). This may be implemented for various reasons (including normalising packets to hide which equipment supports ECN). This policy prevents use of ECN by applications. A network device should therefore not remark an ECT(0) or ECT(1) mark to zero.

A network device must not change a packet with a CE mark to a zero codepoint (if the CE marking is not propagated, the packet must be discarded). Such a packet has already received ECN treatment in the
network, and remarking it would then hide the congestion signal from the endpoints.

Some networks may use ECN internally or tunnel ECN for traffic engineering or security. Guidance on the correct use of ECN in this case is provided in [RFC6040].

3. Benefit of using ECN to avoid congestion loss

When packet loss is a result of (mild) congestion, an ECN-enabled router may be expected to CE-mark, rather than drop an ECN-enabled packet [RFC2309.bis]. An application can benefit from this marking in several ways:

3.1. Improved Throughput

ECN can improve the throughput performance of applications, although this increase in throughput offered by ECN is often not the most significant gain.

When an application uses a light to moderately loaded network path, the number of packets that are dropped due to congestion is small. Using an example from Table 1 of [RFC3649], for a standard TCP sender with a Round Trip Time, RTT, of 0.1 seconds, a packet size of 1500 bytes and an average throughput of 1 Mbps, the average packet drop ratio is 0.02. This translates into an approximate 2% throughput gain if ECN is enabled. In heavy congestion, packet loss may be unavoidable with, or without, ECN.

3.2. Reduced Head-of-Line Blocking

Many transports provide in-order delivery of received data segments to the applications they support. This requires that the transport stalls (or waits) for all data that was sent ahead of a particular segment to be correctly received before it can forward any later data. This is the usual requirement for TCP and SCTP. PR-SCTP [RFC3758], UDP, and DCCP [RFC4340] provide a transport that does not have this requirement.

Delaying data to provide in-order transmission to an application results in additional latency when segments are dropped as indications of congestion. The congestive loss creates a delay of at least one RTT for a loss event before data can be delivered to an application. We call this Head-of-Line (HOL) blocking.

In contrast, using ECN can remove the resulting delay following a loss that is a result of congestion:
First, the application receives the data normally. This also avoids the inefficiency of dropping data that has already made it across at least part of the network path. It also avoids the additional delay of waiting for recovery of the lost segment.

Second, the transport receiver notes that it has received CE-marked packets, and then requests the sender to make an appropriate congestion-response to reduce the maximum transmission rate for future traffic.

3.3. Reduced Probability of RTO Expiry

In some situations, ECN can help reduce the chance of a retransmission timer expiring (e.g., expiry of the TCP or SCTP retransmission timeout, RTO [RFC5681]). When an application sends a burst of segments and then becomes idle (either because the application has no further data to send or the network prevents sending further data – e.g., flow or congestion control at the transport layer), the last segment of the burst may be lost. It is often not possible to recover this last segment (or last few segments) using standard methods such as Fast Recovery [RFC5681], since the receiver generates no feedback because it is unaware that the lost segments were actually sent.

In addition to avoiding HOL blocking, this allows the transport to avoid the consequent loss of state about the network path it is using, which would have arisen had there been a retransmission timeout. Typical impacts of a transport timeout are to reset path estimates such as the RTT, the congestion window, and possibly other transport state that can reduce the performance of the transport until it again adapts to the path.

Avoiding timeouts can hence improve the throughput of the application. This benefits applications that send intermittent bursts of data, and rely upon timer-based recovery of packet loss. It can be especially significant when ECN is used on TCP SYN/ACK packets [RFC5562] where the RTO interval may be large because in this case TCP cannot base the timeout period on prior RTT measurements from the same connection.

3.4. Applications that do not retransmit lost packets

Some latency-critical applications do not retransmit lost packets, yet they may be able to adjust the sending rate in the presence of congestion. Examples of such applications include UDP-based services that carry Voice over IP (VoIP), interactive video or real-time data. The performance of many such applications degrades rapidly with increasing packet loss, and many therefore employ loss-hiding
mechanisms (e.g., packet forward error correction, or data duplication) to mitigate the effect of congestion loss on the application. However, such mechanisms add complexity and can themselves consume additional network capacity reducing the capacity for application data and contributing to the path latency when congestion is experienced.

By decoupling congestion control from loss, ECN can allow the transports supporting these applications to reduce their rate before the application experiences loss from congestion, especially when the congestion is mild and the application/transport can react promptly to reception of a CE-marked packet. Because this reduces the negative impact of using loss-hiding mechanisms, ECN can have a direct positive impact on the quality experienced by the users of these applications.

4. Benefit from Early Congestion Detection

An application can further benefit from using ECN, when the network devices are configured such that they mark packets at a lower level of congestion before they would otherwise have dropped packets from queue overflow:

4.1. Avoiding Capacity Overshoot

Internet transports do not know a priori how much capacity exists along a network path. Transports therefore try to measure the capacity available to an application by probing the network path with increasing traffic to the point where they detect the onset of congestion (such as TCP or SCTP Slow Start).

ECN can help capacity probing algorithms (such as Slow Start) from significantly exceeding the bottleneck capacity of a network path. Since a transport that enables ECN can receive congestion signals before there is significant congestion, an early-marking method in network devices can help a transport respond before it induces significant congestion with resultant loss to itself or other applications sharing a common bottleneck. For example, an application/transport can avoid incurring significant congestion during Slow Start, or a bulk application that tries to increase its rate as fast as possible, may quickly detect the presence of congestion, causing it to promptly reduce its rate.

Use of ECN is more effective than schemes such as Limited Slow-Start [RFC3742] because it provides direct information about the state of the network path. An ECN-enabled application/transport that probes for capacity can reduce its rate as soon as it discovers CE-marked packets are received, and before the applications increases its rate.
to the point where it builds a queue in a network device that induces congestion loss. This benefits an application seeking to increase its rate — but perhaps more significantly, it eliminates the often unwanted loss and queueing delay that otherwise may be inflicted on flows that share a common bottleneck.

4.2. Making Congestion Visible

A characteristic of using ECN is that it exposes the presence of congestion on a network path to the transport and network layers. This information can be used for monitoring performance of the path, and could be used to directly meter the amount of congestion that has been encountered upstream on a path; metering packet loss is harder. ECN measurements are used by Congestion Exposure (CoNex) [RFC6789].

A network flow that only experiences CE-marks and no loss implies that the sending endpoint is experiencing only congestion and not other sources of packet loss (e.g., link corruption or loss in middleboxes). The converse is not true — a flow may experience a mixture of ECN-marks and loss when there is only congestion or when there is a combination of packet loss and congestion [RFC2309.bis]. Recording the presence of CE-marked packets can therefore provide information about the performance of the network path.

5. Other forms of ECN-Marking/Reactions

ECN requires a definition of both how packets are CE-marked and how applications/transports need to react to reception of CE-marked packets. This section describes the benefits when updated methods are used.

ECN-capable receiving endpoints may provide more detailed feedback describing the ECN codepoints that they observe using [ID.Acc-ECN]. This can provide more information to a sending endpoint’s congestion control mechanism.

Benefit has been noted when packets are CE-marked earlier than they would otherwise be dropped, using an instantaneous queue, and if the receiver provides precise feedback about the number of packet marks encountered, a better sender behavior is possible. This has been shown by Datacenter TCP (DCTCP) [AL10].

Precise feedback about the number of packet marks encountered is supported by the Real Time Protocol (RTP) when used over UDP [RFC6779] and proposed for SCTP [ST14] and TCP [ID.Acc-ECN]. An underlying assumption of DCTCP is that it is deployed in confined environments such as a datacenter. It is currently unknown whether or how such behaviour could be safely introduced into the Internet.
6. ECN transport mechanisms for paths with partial ECN support

Early deployment of ECN encountered a number of operational difficulties when the network only partially supports the use of ECN, or to respond to the challenges due to misbehaving network devices and/or endpoints. These problems have been observed to diminish with time, but may still be encountered on some Internet paths [TR15]. This section describes transport mechanisms that allow ECN-enabled endpoints to continue to work effectively over a path with partial ECN support.

6.1. Verifying whether a path really supports ECN

ECN transport and applications need to implement mechanisms to verify ECN support on the path that they use and fallback to not using ECN when it would not work. This is expected to be a normal feature of IETF-defined transports supporting ECN.

Before a transport relies on the presence or absence of CE-marked packets, it may need to verify that any ECN marks applied to packets passed by the path are indeed delivered to the remote endpoint. This may be achieved by the sender setting known ECN codepoints into specific packets in a network flow and then verifying that these reach the remote endpoint [ID.Fallback], [TR15]. Endpoints also need to be robust to path changes. A change in the set of network devices along a path may impact the ability to effectively signal or use ECN across the path, e.g., when a path changes to use a middlebox that bleaches ECN codepoints. As a necessary, but short term fix, transports could implement mechanisms that detect this and fall-back to disabling use of ECN [BA11].

6.2. Detecting ECN receiver feedback cheating

It is important that receiving endpoints accurately report the loss they experience when using a transport that uses loss-based congestion control. So also, when using ECN, a receiver must correctly report the congestion marking that it receives and then provide a mechanism to feed the congestion information back to the sending endpoint.

The transport at endpoint receivers must not try to conceal reception of CE-marked packets in the ECN feedback information that they provide to the sending endpoint [RFC2309.bis]. Transport protocols are actively encouraged to include mechanisms that can detect and appropriately respond to such misbehavior (e.g., disabling use of ECN, and relying on loss-based congestion detection [TR15]).
7. Conclusion

Network devices should enable ECN and people configuring host stacks should also enable ECN. Specifically network devices must not change a packet with a CE mark to a zero codepoint (if the CE marking is not propagated, the packet must be discarded). These are prerequisites to allow applications to gain the benefits of ECN.

Prerequisites for network devices (including IP routers) to enable use of ECN include:

- should not reset the ECN codepoint to zero by default Section 2.2.
- should correctly update the ECN codepoint in the presence of congestion.
- should correctly support alternate ECN semantics ([RFC4774]).

Prerequisites for network endpoints to enable use of ECN include:

- should use transports that can set and receive ECN marks.
- should correctly return feedback of congestion to the sending endpoint.
- must use transports that react appropriately to received ECN feedback Section 6.2.
- should use transports that can detect misuse of ECN and detect paths that do not support ECN, providing fallback to loss-based congestion detection when ECN is not supported Section 6.1.

Application developers should where possible use transports that enable the benefits of ECN. Applications that directly use UDP need to provide support to implement the functions required for ECN. Once enabled, an application that uses a transport that supports ECN will experience the benefits of ECN as network deployment starts to enable ECN. The application does not need to be rewritten to gain these benefits. Table 1 summarises some of these benefits.
<table>
<thead>
<tr>
<th>Section</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Improved Throughput</td>
</tr>
<tr>
<td>3.2</td>
<td>Reduced Head-of-Line</td>
</tr>
<tr>
<td>3.3</td>
<td>Reduced Probability of RTO Expiry</td>
</tr>
<tr>
<td>3.4</td>
<td>Applications that do not retransmit lost packets</td>
</tr>
<tr>
<td>4.1</td>
<td>Avoiding Capacity Overshoot</td>
</tr>
<tr>
<td>4.2</td>
<td>Making Congestion Visible</td>
</tr>
</tbody>
</table>

Table 1: Summary of Key Benefits

8. Acknowledgements

The authors were part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the authors.

The authors would like to thank the following people for their comments on prior versions of this document: Bob Briscoe, David Collier-Brown, John Leslie, Colin Perkins, Richard Scheffenegger, Dave Taht, Wes Eddy.

9. IANA Considerations

XX RFC ED - PLEASE REMOVE THIS SECTION XXX

This memo includes no request to IANA.

10. Security Considerations

This document introduces no new security considerations. Each RFC listed in this document discusses the security considerations of the specification it contains.

11. Revision Information

XXX RFC-Ed please remove this section prior to publication.

Revision 00 was the first WG draft.

Revision 01 includes updates to complete all the sections and a rewrite to improve readability. Added section 2. Author list reversed, since Gorry has become the lead author. Corrections
following feedback from Wes Eddy upon review of an interim version of
this draft.

Note: Wes Eddy raised a question about whether discussion of the ECN
Pitfalls could be improved or restrucutured - this is expected to be
addressed in the next revision.

Revision 02 updates the title, and also the description of mechanisms
that help with partial ECN support.

We think this draft is ready for wider review. Comments are welcome
to the authors or via the IETF AQM or TSVWG mailing lists.

12. References

12.1. Normative References

[ RFC2309.bis ]
Baker, F. and G. Fairhurst, "IETF Recommendations

[ RFC3168 ]
Ramakrishnan, K., Floyd, S., and D. Black, "The Addition
of Explicit Congestion Notification (ECN) to IP", RFC

12.2. Informative References

[ AL10 ]
Alizadeh, M., Greenberg, A., Maltz, D., Padhye, J., Patel,
P., Prabhakar, B., Sengupta, S., and M. Sridharan, "Data
Center TCP (DCTCP)", SIGCOMM 2010, August 2010.

[ BA11 ]
Bauer, Steven., Beverly, Robert., and Arthur. Berger,
"Measuring the State of ECN Readiness in Servers, Clients,
and Routers, ACM IMC", 2011.

[ ID.Acc-ECN ]
Kuehlewind, Mirja., Scheffenegger, Richard., and Bob.
Briscoe, "Problem Statement and Requirements for a More
Accurate ECN Feedback", Internet-draft, IETF work-in-

[ ID.ECN-Encap ]
Briscoe, B., Kaippallimalil, J., and P. Thaler,
"Guidelines for Adding Congestion Notification to
Protocols that Encapsulate IP", Internet-draft, IETF work-
in-progress draft-ietf-tsvwg-ecn-encap-guidelines, .


"Type of Service in the Internet Protocol Suite", .

"Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", .


Authors' Addresses

Godred Fairhurst  
University of Aberdeen  
School of Engineering, Fraser Noble Building  
Aberdeen AB24 3UE  
UK  

Email: gorry@erg.abdn.ac.uk

Michael Welzl  
University of Oslo  
PO Box 1080 Blindern  
Oslo N-0316  
Norway  

Phone: +47 22 85 24 20  
Email: michawe@ifi.uio.no
Abstract

Unmanaged large buffers in today’s networks have given rise to a slew of performance issues. These performance issues can be addressed by some form of Active Queue Management (AQM) mechanism, optionally in combination with a packet scheduling scheme such as fair queuing. The IETF Active Queue Management and Packet Scheduling working group was formed to standardize AQM schemes that are robust, easily implementable, and successfully deployable in today’s networks. This document describes various criteria for performing precautionary characterizations of AQM proposals. This document also helps in ascertaining whether any given AQM proposal should be taken up for standardization by the AQM WG.
Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction ................................................. 3
   1.1. Guidelines for AQM designers ......................... 5
   1.2. Reducing the latency and maximizing the goodput ........ 5
   1.3. Glossary ............................................. 6
   1.4. Requirements Language ................................. 6
2. End-to-end metrics ........................................... 6
   2.1. Flow completion time ................................ 6
   2.2. Packet loss .......................................... 7
   2.3. Packet loss synchronization .......................... 7
   2.4. Goodput ............................................... 8
   2.5. Latency and jitter ................................... 9
   2.6. Discussion on the trade-off between latency and goodput 9
3. Generic set up for evaluations ............................. 10
   3.1. Topology and notations ............................... 10
   3.2. Buffer size .......................................... 11
   3.3. Congestion controls .................................. 12
4. Various TCP variants ....................................... 12
   4.1. TCP-friendly sender .................................. 13
   4.2. Aggressive transport sender ........................ 13
   4.3. Unresponsive transport sender ....................... 13
   4.4. TCP initial congestion window ...................... 14
5. RTT fairness ................................................ 14
   5.1. Motivation .......................................... 14
   5.2. Required tests ...................................... 15
   5.3. Metrics to evaluate the RTT fairness ............... 15
6. Burst absorption ............................................. 16
   6.1. Motivation .......................................... 16
   6.2. Required tests ...................................... 16
7. Stability ................................................... 17
   7.1. Motivation .......................................... 17
   7.2. Required tests ...................................... 18
1. Introduction

Active Queue Management (AQM) addresses the concerns arising from using unnecessarily large and unmanaged buffers, in order to improve network and application performance. Several AQM algorithms have been proposed in the past years, most notably Random Early Detection (RED), BLUE, and Proportional Integral controller (PI), and more recently CoDel [CODEL] and PIE [PIE]. In general, these algorithms actively interact with the Transmission Control Protocol (TCP) and any other transport protocol that deploys a congestion control scheme to manage the amount of data they keep in the network. The available
buffer space in the routers and switches should be large enough to accommodate the short-term buffering requirements. AQM schemes aim at reducing mean buffer occupancy, and therefore both end-to-end delay and jitter. Some of these algorithms, notably RED, have also been widely implemented in some network devices. However, the potential benefits of the RED scheme have not been realized since RED is reported to be usually turned off. The main reason of this reluctance to use RED in today’s deployments comes from its sensitivity to the operating conditions in the network and the difficulty of tuning its parameters.

A buffer is a physical volume of memory in which a queue or set of queues are stored. In real implementations of switches, a global memory is shared between the available devices: the size of the buffer for a given communication does not make sense, as its dedicated memory may vary over the time and real-world buffering architectures are complex. For the sake of simplicity, when speaking of a specific queue in this document, "buffer size" refers to the maximum amount of data the buffer may store, which can be measured in bytes or packets. The rest of this memo therefore refers to the maximum queue depth as the size of the buffer for a given communication.

In order to meet mostly throughput-based Service-Level Agreement (SLA) requirements and to avoid packet drops, many home gateway manufacturers resort to increasing the available memory beyond "reasonable values". This increase is also referred to as Bufferbloat [BB2011]. Deploying large unmanaged buffers on the Internet has lead to the increase in end-to-end delay, resulting in poor performance for latency-sensitive applications such as real-time multimedia (e.g., voice, video, gaming, etc). The degree to which this affects modern networking equipment, especially consumer-grade equipment’s, produces problems even with commonly used web services. Active queue management is thus essential to control queuing delay and decrease network latency.

The Active Queue Management and Packet Scheduling Working Group (AQM WG) was recently formed within the TSV area to address the problems with large unmanaged buffers in the Internet. Specifically, the AQM WG is tasked with standardizing AQM schemes that not only address concerns with such buffers, but also are robust under a wide variety of operating conditions. In order to ascertain whether the WG should undertake standardizing an AQM proposal, the WG requires guidelines for assessing AQM proposals. This document provides the necessary characterization guidelines.
1.1. Guidelines for AQM designers

One of the key objectives behind formulating the guidelines is to help ascertain whether a specific AQM is not only better than drop-tail but also safe to deploy. The guidelines help to quantify AQM schemes’ performance in terms of latency reduction, goodput maximization and the trade-off between these two. The guidelines also help to discuss AQM’s safe deployment, including self-adaptation, stability analysis, fairness, design and implementation complexity and robustness to different operating conditions.

This memo details generic characterization scenarios that any AQM proposal MUST consider for evaluation. Irrespective of whether or not an AQM is standardized by the WG, we RECOMMEND the relevant scenarios and metrics discussed in this document to be considered. This document presents central aspects of an AQM algorithm that MUST be considered whatever the context is such as, burst absorption capacity, RTT fairness or resilience to fluctuating network conditions. These guidelines do not cover every possible aspect of a particular algorithm. In addition, it is worth noting that the proposed criteria are not bound to a particular evaluation toolset. These guidelines do not present context dependent scenarios (such as 802.11 WLANs, data-centers or rural broadband networks).

This document details how an AQM designer can rate the feasibility of their proposal in different types of network devices (switches, routers, firewalls, hosts, drivers, etc) where an AQM may be implemented.

1.2. Reducing the latency and maximizing the goodput

The trade-off between reducing the latency and maximizing the goodput is intrinsically linked to each AQM scheme and is key to evaluating its performance. This trade-off MUST be considered in various scenarios to ensure the safety of an AQM deployment. Whenever possible, solutions should aim at both maximizing goodput and minimizing latency. This document proposes guidelines that enable the reader to quantify (1) reduction of latency, (2) maximization of goodput and (3) the trade-off between the two.

Testers SHOULD discuss in a reference document the performance of their proposal in terms of performance and deployment compared to those of drop-tail: basically, these guidelines provide the tools to understand the deployment costs versus the potential gain in performance due to the introduction of the proposed scheme.
1.3. Glossary

- **AQM**: there may be a debate on whether a scheduling scheme is additional to an AQM mechanism or is a part of an AQM scheme. The rest of this memo refers to AQM as a dropping/marking policy that does not feature a scheduling scheme.

- **buffer**: a physical volume of memory in which a queue or set of queues are stored.

- **buffer size**: the maximum amount of data that may be stored in a buffer, measured in bytes or packets.

1.4. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. End-to-end metrics

End-to-end delay is the result of propagation delay, serialization delay, service delay in a switch, medium-access delay and queuing delay, summed over the network elements along the path. AQM algorithms may reduce the queuing delay by providing signals to the sender on the emergence of congestion, but any impact on the goodput must be carefully considered. This section presents the metrics that MAY be used to better quantify (1) the reduction of latency, (2) maximization of goodput and (3) the trade-off between these two. These metrics MAY be considered to better assess the performance of an AQM scheme.

The metrics listed in this section are not necessarily suited to every type of traffic detailed in the rest of this document. It is therefore NOT REQUIRED to measure all of the following metrics in every scenario discussed in this document necessarily, if the chosen metric is not relevant to the context of the evaluation scenario (e.g. latency vs. goodput trade-off in application-limited traffic scenarios). The tester SHOULD however measure and report on all the metrics relevant to the context of the evaluation scenario.

2.1. Flow completion time

The flow completion time is an important performance metric for the end-user when the flow size is finite. Considering the fact that an AQM scheme may drop/mark packets, the flow completion time is directly linked to the dropping/marking policy of the AQM scheme. This metric helps to better assess the performance of an AQM scheme.
depending on the flow size. The Flow Completion Time (FCT) is related to the flow size (Fs) and the goodput for the flow (G) as follows:

\[ \text{FCT [s]} = \frac{\text{Fs [B]}}{\left( \frac{\text{G [Mbps]}}{8} \right)} \]

2.2. Packet loss

Packet losses, that may occur in a queue, impact on the end-to-end performance at the receiver's side.

The tester MUST evaluate, at the receiver:

- the packet loss probability: this metric should also be frequently measured during the experiment, since the long-term loss probability is only of interest for steady-state scenarios.
- the interval between consecutive losses: the time between two losses should be measured.

The packet loss probability can be assessed by simply evaluating the loss ratio as a function of the number of lost packets and the total number of packets sent. This might not be easily done in laboratory testing, for which these guidelines advice the tester:

- to check that for every packet, a corresponding packet was received within a reasonable time, as explained in [RFC2680].
- to keep a count of all packets sent, and a count of the non-duplicate packets received, as explained in the section 10 of [RFC2544].

The interval between consecutive losses, which is also called a gap, is a metric of interest for VoIP traffic and, as a result, has been further specified in [RFC3611].

2.3. Packet loss synchronization

One goal of an AQM algorithm should be to help with avoiding global synchronization of flows sharing the bottleneck buffer on which the AQM operates ([RFC2309]). It is therefore important to assess the "degree" of packet-loss synchronization between flows, with and without the AQM under consideration.

As discussed e.g. in [LOSS-SYNCH-MET-08], loss synchronization among flows may be quantified by several slightly different metrics that capture different aspects of the same issue. However, in real-world measurements the choice of metric may be imposed by practical
considerations -- e.g. whether fine-grained information on packet losses in the bottleneck available or not. For the purpose of AQM characterization, a good candidate metric is the global synchronization ratio, measuring the proportion of flows losing packets during a loss event. [YU06] used this metric in real-world experiments to characterize synchronization along arbitrary Internet paths; the full methodology is described in [YU06].

If an AQM scheme is evaluated using real-life network environments, it is worth pointing out that some network events, such as failed link restoration may cause synchronized losses between active flows and thus confuse the meaning of this metric.

2.4. Goodput

The goodput has been defined in the section 3.17 of [RFC2647] as the number of bits per unit of time forwarded to the correct destination interface of the Device Under Test (DUT) or the System Under Test (SUT), minus any bits lost or retransmitted. This definition induces that the test setup needs to be qualified to assure that it is not generating losses on its own.

Measuring the end-to-end goodput enables an appreciation of how well the AQM improves transport and application performance. The measured end-to-end goodput is linked to the AQM scheme’s dropping/marking policy -- e.g. the smaller the number of packet drops, the fewer packets need retransmission, minimizing AQM’s impact on transport and application performance. Additionally, an AQM scheme may resort to Explicit Congestion Notification (ECN) marking as an initial means to control delay. Again, marking packets instead of dropping them reduces the number of packet retransmissions and increases goodput. End-to-end goodput values help to evaluate the AQM scheme’s effectiveness in minimizing packet drops that impact application performance and to estimate how well the AQM scheme works with ECN.

The measurement of the goodput let the tester evaluate to which extent the AQM is able to maintain a high link utilization. This metric should be also obtained frequently during the experiment as the long-term goodput is relevant for steady-state scenarios only and may not necessarily reflect how the introduction of an AQM actually impacts the link utilization during at a certain period of time. It is worth pointing out that the fluctuations in the values obtained from these measurements may depend on other factors than the introduction of an AQM, such as link layer losses due to external noise or corruption, fluctuating bandwidths (802.11 WLANs), heavy congestion levels or transport layer’s rate reduction by congestion control mechanism.
2.5. Latency and jitter

The latency, or the one-way delay metric, is discussed in [RFC2680]. There is a consensus on an adequate metric for the jitter, that represents the one-way delay variations for packets from the same flow: the Packet Delay Variation (PDV), detailed in [RFC5481], serves well all use cases.

The end-to-end latency differs from the queuing delay: it is linked to the network topology and the path characteristics. Moreover, the jitter strongly depends on the traffic pattern and the topology as well. The introduction of an AQM scheme would impact on these metrics and therefore they SHOULD be considered in the end-to-end evaluation of performance.

The guidelines advice that the tester SHOULD measure the minimum, average and maximum as well as the coefficient of variation of the average values for these metrics.

2.6. Discussion on the trade-off between latency and goodput

The metrics presented in this section may be considered as explained in the rest of this document, in order to discuss and quantify the trade-off between latency and goodput.

This trade-off can also be illustrated with figures following the recommendations of the section 5 of [TCPEVAL2013]. Each of the end-to-end delay and the goodput SHOULD be measured frequently for every fixed time interval.

With regards to the goodput, and in addition to the long-term stationary goodput value, it is RECOMMENDED to take measurements every multiple of RTTs. We suggest a minimum value of 10 x RTT (to smooth out the fluctuations) but higher values are encouraged whenever appropriate for the presentation depending on the network’s path characteristics. The measurement period MUST be disclosed for each experiment and when results/values are compared across different AQM schemes, the comparisons SHOULD use exactly the same measurement periods.

With regards to latency, it is highly RECOMMENDED to take the samples on per-packet basis whenever possible depending on the features provided by hardware/software and the impact of sampling itself on the hardware performance. It is generally RECOMMENDED to provide at least 10 samples per RTT.

From each of these sets of measurements, the 10th and 90th percentiles and the median value SHOULD be computed. For each
scenario, a graph can be generated, with the x-axis showing the end-to-end delay and the y-axis the goodput. This graph provides part of a better understanding of (1) the delay/goodput trade-off for a given congestion control mechanism, and (2) how the goodput and average queue size vary as a function of the traffic load.

3. Generic set up for evaluations

This section presents the topology that can be used for each of the following scenarios, the corresponding notations and discusses various assumptions that have been made in the document.

3.1. Topology and notations

Figure 1: Topology and notations
Figure 1 is a generic topology where:

- various classes of traffic can be introduced;
- the timing of each flow (i.e., when does each flow start and stop) may be different;
- each class of traffic can comprise various number of flows;
- each link is characterized by a couple (RTT, capacity);
- flows are generated between A and B, sharing a bottleneck (Routers L and R);
- the tester SHOULD consider both scenarios of asymmetric and symmetric bottleneck links in terms of bandwidth. In case of asymmetric link, the capacity from senders to receivers is higher than the one from receivers to senders; the symmetric link scenario provides a basic understanding of the AQM mechanism’s operation whereas the asymmetric link scenario evaluates an AQM mechanism in a more realistic setup;
- in asymmetric link scenarios, the tester SHOULD study the bi-directional traffic between A and B (downlink and uplink) with the AQM mechanism deployed on one direction only. The tester MAY additionally consider a scenario with AQM mechanism being deployed on both directions. In each scenario, the tester SHOULD investigate the impact of AQM’s drop policy on TCP ACK packets and its impact on the performance.

This topology may not perfectly reflect actual topologies, however, this simple topology is commonly used in the world of simulations and small testbeds. This topology can be considered as adequate to evaluate AQM proposals, similarly to the topology proposed in [TCPEVAL2013]. The tester should carefully choose the topology that is going to be used to evaluate the AQM scheme.

3.2. Buffer size

The size of the buffers should be carefully chosen, and MAY be set to the bandwidth-delay product. However, if the context or the application requires a specific buffer size, the tester MUST justify and detail the way the maximum queue size is set. Indeed, the maximum size of the buffer may affect the AQM’s performance and its choice SHOULD be elaborated for a fair comparison between AQM proposals. While comparing AQM schemes the buffer size SHOULD remain the same across the tests.
3.3. Congestion controls

This memo features three kind of congestion controls:

- Standard TCP congestion control: the base-line congestion control is TCP NewReno with SACK, as explained in [RFC5681].

- Aggressive congestion controls: a base-line congestion control for this category is TCP Cubic.

- Less-than Best Effort (LBE) congestion controls: an LBE congestion control 'results in smaller bandwidth and/or delay impact on standard TCP than standard TCP itself, when sharing a bottleneck with it.' [RFC6297]

Recent transport layer protocols are not mentioned in the following sections, for the sake of simplicity.

4. Various TCP variants

Network and end-devices need to be configured with a reasonable amount of maximum available buffer space in order to absorb transient bursts. In some situations, network providers tend to configure devices with large buffers in order to avoid packet drops triggered by a full buffer and to maximize the link utilization for standard loss-based TCP traffic. Loss-based TCP congestion controls (including standard NewReno TCP) fill up these unmanaged buffers until the TCP sender receives a signal (packet drop) to decrease the sending rate. The larger the buffer is, the higher the buffer occupancy, and therefore the queuing delay. On the other hand, an efficient AQM scheme SHOULD convey early congestion signals to TCP senders so that the average queuing delay is brought under control.

Not all applications run over the same flavor of TCP or even necessarily use TCP. Variety of applications generate different classes of traffic which may not react to congestion signals (a.k.a unresponsive flows) or may not decrease their sending rate as expected (a.k.a aggressive flows); AQM schemes aim at maintaining the queuing delay under control, which is challenged if aggressive or unresponsive traffics are present.

This section provides guidelines to assess the performance of an AQM proposal for various traffic profiles -- different types of senders (with different TCP congestion control variants, unresponsive, aggressive).
4.1. TCP-friendly sender

This scenario helps to evaluate how an AQM scheme reacts to a TCP-friendly transport sender. A single long-lived, non application-limited, TCP NewReno flow transfers data between sender A and receiver B. Other TCP friendly congestion control schemes such as TCP-friendly rate control [RFC5348] etc MAY also be considered.

For each TCP-friendly transport considered, the graph described in Section 2.6 could be generated.

4.2. Aggressive transport sender

This scenario helps to evaluate how an AQM scheme reacts to a transport sender that is more aggressive than a single TCP-friendly sender. We define ‘aggressiveness’ as a higher increase factor than standard upon a successful transmission and/or a lower than standard decrease factor upon a unsuccessful transmission (e.g. in case of congestion controls with Additive-Increase Multiplicative-Decrease (AIMD) principle, a larger AI and/or MD factors). A single long-lived, non application-limited, TCP Cubic flow transfers data between sender A and receiver B. Other aggressive congestion control schemes MAY also be considered.

For each flavor of aggressive transports, the graph described in Section 2.6 could be generated.

4.3. Unresponsive transport sender

This scenario helps to evaluate how an AQM scheme reacts to a transport sender that is not responsive to congestion signals (ECN marks and/or packet drops) from the AQM scheme. Note that faulty transport implementations on end-hosts and/or faulty network elements on the path that modify congestion signals in packet headers (e.g. modifying the ECN-related bits) [I-D.ietf-aqm-recommendation] may also lead to a similar situation, such that the AQM scheme needs to adapt to the unresponsive traffic. To this end, these guidelines propose the two following scenarios.

The first scenario aims at creating a test environment that results in constant queue build up; we consider unresponsive flow(s) with an overall sending rate that is greater than the bottleneck’s link capacity between routers L and R. This scenario consists of a long-lived non application-limited UDP flow that transfers data between sender A and receiver B. Graphs described in Section 2.6 could be generated.
The second scenario aims to test to which extent the AQM scheme is able to keep the responsive fraction of overall traffic load under control, this scenario considers a mixture of TCP-friendly and unresponsive traffics. It consists of a long-lived non application-limited UDP flow and a single long-lived, non application-limited, TCP NewReno flow that transfer data between sender A and receiver B. As opposed to the first scenario, the rate of the UDP traffic should be less than or equal to half of the bottleneck capacity. For each type of traffic, the graph described in Section 2.6 could be generated.

4.4. TCP initial congestion window

This scenario helps to evaluate how an AQM scheme adapts to a traffic mix consisting of TCP flows with different values of the Initial congestion Window (IW).

For this scenario, we consider two types of flows that MUST be generated between sender A and receiver B:

- A single long-lived non application-limited TCP NewReno flow;
- A single long-lived application-limited TCP NewReno flow, with an IW set to 3 or 10 packets. The size of the data transferred MUST be strictly higher than 10 packets and should be lower than 100 packets.

The transmission of the non application-limited flow MUST start before the transmission of the application-limited flow and only after the steady state has been reached by non application-limited flow.

For each of these scenarios, the graph described in Section 2.6 could be generated for each class of traffic (application-limited and non application-limited). The completion time of the application-limited TCP flow could be measured.

5. RTT fairness

5.1. Motivation

The capability of AQM schemes to control the queueing delay highly depends on the way end-to-end transport protocols react to congestion signals. When network path’s intrinsic RTT varies, the behavior of congestion control is impacted and so the capability of AQM schemes to control the queueing level. It is therefore important to assess the AQM schemes against a set of intrinsic RTTs common in the Internet transfers (e.g. from 5 ms to 500 ms).
Also, asymmetry in terms of difference in intrinsic RTT between various paths sharing the same bottleneck SHOULD be considered and the fairness between the flows SHOULD be discussed since in this scenario, a flow traversing on shorter RTT path may react faster to congestion and recover faster from it compared to another flow on a longer RTT path. The introduction of AQM schemes may potentially improve this type of fairness.

Moreover, introducing an AQM scheme may cause the unfairness between the flows, even if the RTTs are identical. This potential unfairness SHOULD be investigated as well.

5.2. Required tests

The topology that SHOULD be used is presented in Figure 1:

- To evaluate the inter-RTT fairness, for each run, two flows divided into two categories. Category I which RTT between sender A and Router L SHOULD be 100ms. Category II which RTT between sender A and Router L SHOULD be in [5ms;560ms]. The maximum value for the RTT represents the RTT of a satellite link that, according to the section 2 of [RFC2488] should be at least 558ms.

- To evaluate the impact of the RTT value on the AQM performance and the intra-protocol fairness (the fairness for the flows using the same paths/congestion control), for each run, two flows (Flow1 and Flow2) SHOULD be introduced. For each experiment, the set of RTT SHOULD be the same for the two flows and in [5ms;560ms]. These flows MUST use the same congestion control algorithm.

5.3. Metrics to evaluate the RTT fairness

The output that MUST be measured is:

- for the inter-RTT fairness: (1) the cumulative average goodput of the flow from Category I, goodput_Cat_I (Section 2.4); (2) the cumulative average goodput of the flow from Category II, goodput_Cat_II (Section 2.4); (3) the ratio goodput_Cat_II/goodput_Cat_I; (4) the average packet drop rate for each category (Section 2.2).

- for the intra-protocol RTT fairness: (1) the cumulative average goodput of the two flows (Section 2.4); (2) the average packet drop rate for the two flows (Section 2.2).
6. Burst absorption

6.1. Motivation

Packet arrivals can be bursty due to various reasons. A packet burst can push the AQM schemes to drop/mark packets momentarily even though the average queue length may still be below the AQM’s target queuing thresholds. Dropping/marking one or more packets within a burst may result in performance penalties for the corresponding flows since the dropped/marked packets cause unnecessary rate reduction by congestion control as well as retransmission in case of drop only. Performance penalties may turn into unmet SLAs and become disincentives for the AQM adoption. Therefore, an AQM scheme SHOULD be designed to accommodate transient bursts. AQM schemes do not present the same tolerance to packet bursts arriving at the buffer, therefore this tolerance MUST be quantified.

Note that accommodating bursts translates to higher queue length and queuing delay. Naturally, it is important that the AQM scheme brings bursty traffic under control quickly. On the other hand, spiking packet drops in order to bring packet bursts quickly under control could result in multiple drops per flow and severely impact transport and application performance. Therefore, an AQM scheme SHOULD bring bursts under control by balancing both aspects -- (1) queuing delay spikes are minimized and (2) performance penalties for ongoing flows in terms of packet drops are minimized.

An AQM scheme maintains short average queues to allow the remaining space in the queue for temporary bursts of packets. The tolerance to packet bursts depends on the number of packets in the queue, which is directly linked to the AQM algorithm. Moreover, one AQM scheme may implement a feature controlling the maximum size of accepted bursts, that may depend on the buffer occupancy or the currently estimated queuing delay. Also, the impact of the buffer size on such feature (a.k.a burst allowance) MAY be evaluated.

6.2. Required tests

For this scenario, the following traffic MUST be generated from sender A to receiver B:

- Web traffic with IW10: Web transfer of 100 packets with initial congestion window set to 10;
- Bursty video frames;
- Constant bit rate UDP traffic.
A single bulk TCP flow as background traffic.

Figure 2 presents the various cases for the traffic that MUST be generated between sender A and receiver B.

<table>
<thead>
<tr>
<th>Case</th>
<th>Traffic Type</th>
<th>Video</th>
<th>Webs (IW 10)</th>
<th>CBR</th>
<th>Bulk TCP Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>II</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>III</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>IV</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2: Bursty traffic scenarios

For each of these scenarios, the graph described in Section 2.6 could be generated. In addition, other metrics such as end-to-end latency, jitter, flow completion time MUST be generated. For the cases of frame generation of bursty video traffic as well as the choice of web traffic pattern, we leave these details and their presentation to the testers.

7. Stability

7.1. Motivation

Network devices experience varying operating conditions depending on factors such as time of the day, deployment scenario, etc. For example:

- Traffic and congestion levels are higher during peak hours than off-peak hours.
- In the presence of scheduler, a queue’s draining rate may vary depending on other queues: a low load on a high priority queue implies higher draining rate for lower priority queues.
- The available capacity on the physical layer may vary over time such as in the context of lossy channels.

Whether the target context is a not stable environment, the capability of an AQM scheme to maintain its control over the queuing
delay and buffer occupancy is challenged. This document proposes guidelines to assess the behavior of AQM schemes under varying congestion levels and varying draining rates.

7.2. Required tests

Note that the traffic profiles explained below comprises non application-limited TCP flows. For each of the below scenarios, the results described in Section 2.6 SHOULD be generated. For Section 7.2.5 and Section 7.2.6 they SHOULD incorporate the results in per-phase basis as well.

Wherever the notion of time has explicitly mentioned in this subsection, time 0 starts from the moment all TCP flows have already reached their congestion avoidance phase.

7.2.1. Definition of the congestion Level

In these guidelines, the congestion levels are represented by the projected packet drop rate, had a drop-tail queue was chosen instead of an AQM scheme. When the bottleneck is shared among non-application-limited TCP flows. $l_r$, the loss rate projection can be expressed as a function of $N$, the number of bulk TCP flows, and $S$, the sum of the bandwidth-delay product and the maximum buffer size, both expressed in packets, based on Eq. 3 of [SCL-TCP]:

\[
l_r = 0.76 * N^2 / S^2
\]

\[
N = S * \sqrt{1/0.76} * \sqrt{l_r}
\]

7.2.2. Mild Congestion

This scenario helps to evaluate how an AQM scheme reacts to a light load of incoming traffic resulting in mild congestion -- packet drop rates around 0.1%. The number of bulk flows required to achieve this congestion level, $N_{mild}$, is then:

\[
N_{mild} = \text{round}(0.036*S)
\]

7.2.3. Medium Congestion

This scenario helps to evaluate how an AQM scheme reacts to incoming traffic resulting in medium congestion -- packet drop rates around 0.5%. The number of bulk flows required to achieve this congestion level, $N_{med}$, is then:

\[
N_{med} = \text{round}(0.081*S)
\]
7.2.4. Heavy Congestion

This scenario helps to evaluate how an AQM scheme reacts to incoming traffic resulting in heavy congestion -- packet drop rates around 1%. The number of bulk flows required to achieve this congestion level, \( N_{\text{heavy}} \), is then:

\[ N_{\text{heavy}} = \text{round} \left( 0.114 \times S \right) \]

7.2.5. Varying congestion levels

This scenario helps to evaluate how an AQM scheme reacts to incoming traffic resulting in various levels of congestions during the experiment. In this scenario, the congestion level varies within a large time-scale. The following phases may be considered: phase I - mild congestion during 0-20s; phase II - medium congestion during 20-40s; phase III - heavy congestion during 40-60s; phase I again, and so on.

7.2.6. Varying Available Bandwidth

This scenario helps to evaluate how an AQM scheme adapts to varying available bandwidth on the outgoing link.

To emulate varying draining rates, the bottleneck bandwidth between nodes ‘Router L’ and ‘Router R’ varies over the course of the experiment as follows:

- Experiment 1: the capacity varies between two values within a large time-scale. As an example, the following phases may be considered: phase I - 100Mbps during 0-20s; phase II - 10Mbps during 20-40s; phase I again, and so on.

- Experiment 2: the capacity varies between two values within a short time-scale. As an example, the following phases may be considered: phase I - 100Mbps during 0-100ms; phase II - 10Mbps during 100-200ms; phase I again, and so on.

The tester MAY choose a phase time-interval value different than what is stated above, if the network’s path conditions (such as bandwidth-delay product) necessitate. In this case the choice of such time-interval value SHOULD be stated and elaborated.

The tester MAY additionally evaluate the two mentioned scenarios (short-term and long-term capacity variations), during and/or including TCP slow-start phase.
More realistic fluctuating bandwidth patterns MAY be considered. The tester MAY choose to incorporate realistic scenarios with regards to common fluctuation of bandwidth in state-of-the-art technologies.

The scenario consists of TCP NewReno flows between sender A and receiver B. In order to better assess the impact of draining rates on the AQM behavior, the tester MUST compare its performance with those of drop-tail.

7.3. Parameter sensitivity and stability analysis

An AQM scheme’s control law is the primary means by which the queuing delay is controlled. Hence understanding the control law is critical to understanding the AQM scheme’s behavior. The control law may include several input parameters whose values affect the AQM scheme’s output behavior and its stability. Additionally, AQM schemes may auto-tune parameter values in order to maintain stability under different network conditions (such as different congestion levels, draining rates or network environments). The stability of these auto-tuning techniques is also important to understand.

AQM proposals SHOULD provide background material showing control theoretic analysis of the control law and the input parameter space within which the control law operates as expected; or could use other ways to discuss its stability. For parameters that are auto-tuned, the material SHOULD include stability analysis of the auto-tuning mechanism(s) as well. Such analysis helps to understand an AQM scheme’s control law better and the network conditions/deployments under which the AQM scheme is performing stably.

8. Various traffic profiles

This section provides guidelines to assess the performance of an AQM proposal for various traffic profiles such as traffic with different applications or bi-directional traffic.

8.1. Traffic Mix

This scenario helps to evaluate how an AQM scheme reacts to a traffic mix consisting of different applications such as:

- Bulk TCP transfer
- Web traffic
- VoIP
- Constant Bit Rate (CBR) UDP traffic
Various traffic mixes can be considered. These guidelines RECOMMEND to examine at least the following example: 1 bi-directional VoIP; 6 Webs; 1 CBR; 1 Adaptive Video; 5 bulk TCP. Any other combinations could be considered and should be carefully documented.

For each scenario, the graph described in Section 2.6 could be generated for each class of traffic. In addition, other metrics such as end-to-end latency, jitter and flow completion time MUST be reported.

8.2. Bi-directional traffic

Control packets such as DNS requests/responses, TCP SYNzs/ACKzs are small, but their loss can severely impact the application performance. The scenario proposed in this section will help in assessing whether the introduction of an AQM scheme increases the loss probability of these important packets.

For this scenario, traffic MUST be generated in both downlink and uplink, such as defined in Section 3.1. These guidelines RECOMMEND to consider a mild congestion level and the traffic presented in section Section 7.2.2 in both directions. In this case, the metrics reported MUST be the same as in section Section 7.2 for each direction.

The traffic mix presented in section Section 8.1 MAY also be generated in both directions.

9. Implementation cost

9.1. Motivation

An AQM scheme’s successful deployment is directly related to its cost of implementation. Network devices may need hardware or software implementations of the AQM mechanism. Depending on a device’s capabilities and limitations, the device may or may not be able to implement some or all parts of the AQM logic.

AQM proposals SHOULD provide pseudo-code for the complete AQM scheme, highlighting generic implementation-specific aspects of the scheme such as "drop-tail" vs. "drop-head", inputs (e.g. current queuing delay, queue length), computations involved, need for timers, etc. This helps to identify costs associated with implementing the AQM scheme on a particular hardware or software device. Also, it helps the WG to understand which kind of devices can easily support the AQM and which cannot.
9.2. Required discussion

AQM proposals SHOULD highlight parts of AQM logic that are device
dependent and discuss if and how AQM behavior could be impacted by
the device. For example, a queueing-delay based AQM scheme requires
current queuing delay as input from the device. If the device
already maintains this value, then it is trivial to implement the AQM
logic on the device. On the other hand, if the device provides
indirect means to estimate the queuing delay (for example:
timestamps, dequeuing rate etc), then the AQM behavior is sensitive
to how accurate enough the queuing delay estimations are on that
device. Highlighting the AQM scheme’s sensitivity to queuing delay
estimations helps implementers to identify optimal means of
implementing the mechanism on the device.

10. Operator control knobs and auto-tuning

One of the biggest hurdles of RED deployment was/is its parameter
sensitivity to operating conditions -- how difficult it is to tune
RED parameters for a deployment in order to get maximum benefit from
the RED implementation. Fluctuating congestion levels and network
conditions add to the complexity. Incorrect parameter values lead to
poor performance. This is one reason why RED is reported to be
usually turned off by the network operators.

Any AQM scheme is likely to have parameters whose values affect the
AQM’s control law and behavior. Exposing all these parameters as
control knobs to a network operator (or user) can easily result in a
unsafe AQM deployment. Unexpected AQM behavior ensues when parameter
values are set improperly. A minimal number of control knobs
minimizes the number of ways a possibly naive user can break a system
where an AQM scheme is deployed at. Fewer control knobs make the AQM
scheme more user-friendly and easier to deploy and debug.

We highly recommend that an AQM scheme SHOULD minimize the number of
control knobs exposed for the operator’s tuning. An AQM scheme
SHOULD expose only those knobs that control the macroscopic AQM
behavior such as queue delay threshold or queue length threshold and
so on.

Additionally, an AQM scheme’s safety is directly related to its
stability under varying operating conditions such as varying traffic
profiles and fluctuating network conditions, as described in
Section 7. Operating conditions vary often and hence it is necessary
that the AQM scheme MUST remain stable under these conditions without
the need for additional external tuning. If AQM parameters require
tuning under these conditions, then the AQM MUST self-adapt necessary
parameter values by employing auto-tuning techniques.
11. Interaction with ECN

11.1. Motivation

Apart from packet drops, Explicit Congestion Notification (ECN) is an alternative mean to signal the data senders about network congestion. The AQM recommendation document [I-D.ietf-aqm-recommendation] describes some of the benefits of using ECN coupled with an AQM mechanism.

11.2. Required discussion

An AQM scheme SHOULD support ECN and the testers MUST discuss and describe the support of ECN.

12. Interaction with scheduling

12.1. Motivation

Coupled with an AQM scheme, a router may schedule the transmission of packets in a specific manner by introducing a scheduling scheme. This algorithm may create sub-queues and integrate a dropping policy on each of these sub-queues. Another scheduling policy may modify the way packets are sequenced, modifying the timestamp of each packet.

12.2. Required discussion

The scheduling and the AQM conjointly impact on the end-to-end performance. During the characterization process of a dropping policy, the tester MAY discuss the feasibility to add scheduling to its algorithm. This discussion as an instance, MAY explain whether the dropping policy is applied when packets are being enqueued or dequeued.

13. Discussion on methodology, metrics, AQM comparisons and packet sizes

13.1. Methodology

A sufficiently detailed description of the test setup MUST be provided which facilitates other testers to replicate the tests if required. The test setup description MUST include software and hardware specifications and versions. TCP congestion controls implementations, TCP ACKing mechanisms or TCP default options may differ in evaluation toolsets: the chosen mechanisms and options MUST be carefully reported as they may have non negligible impacts on the performances of the AQM scheme.
The proposals SHOULD be experimented on real-life systems, or they MAY be evaluated with event-driven simulations (such as ns-2, ns-3, OMNET, etc). The proposed scenarios are not bound to a particular evaluation toolset.

The tester is encouraged to make the detailed test setup and the results publicly available.

13.2. Comments on metrics measurement

In this document, we presented the end-to-end metrics that SHOULD be used to evaluate the trade-off between latency and goodput in Section 2. In addition to the end-to-end metrics, the queue-level metrics (normally collected at the device operating the AQM) provide a better understanding of the AQM behavior under study and the impact of its internal parameters. Whenever it is possible (e.g. depending on the features provided by the hardware/software), these guidelines RECOMMEND to collect queue-level metrics, such as link utilization, queuing delay, queue size or packet drop/mark statistics in addition to the AQM-specific parameters. However, the evaluation MUST be primarily based on externally observed end-to-end metrics.

These guidelines do not aim to detail on the way these metrics can be measured, since they highly depend on the evaluation toolset and/or hardware.

13.3. Comparing AQM schemes

This memo recognizes that the guidelines mentioned above may be used for comparing AQM schemes. It recommends that AQM schemes MUST be compared against both performance and deployment categories. In addition, this section details how best to achieve a fair comparison of AQM schemes by avoiding certain pitfalls.

13.3.1. Performance comparison

AQM schemes MUST be compared against all the generic scenarios presented in this memo. AQM schemes MAY be compared for specific network environments such as data centers, home networks, etc. If an AQM scheme’s parameter(s) were externally tuned for optimization or other purposes, these values MUST be disclosed.

Note that AQM schemes belong to different varieties such as queue-length based schemes such as RED or queueing-delay based scheme such as CoDel and PIE. Also, AQM schemes expose different control knobs associated with different semantics. For example, while both PIE and CoDel are queueing-delay based schemes and each expose a knob to control the queueing delay -- PIE’s "queueing delay reference" vs.
CoDel’s "queueing delay target", the two schemes’ knobs have different semantics resulting in different control points. Such differences in AQM schemes SHOULD not be overlooked while making comparisons.

This document recommends the following procedures for a fair performance comparison between the AQM schemes:

1. Comparable control parameters and comparable input values: carefully identify the set of parameters that control similar behavior between the two AQM schemes and ensure these parameters have comparable input values. For example, while comparing how well a queue-length based AQM scheme controls queueing delay vs. a queueing-delay based AQM scheme, identify the two schemes’ parameters that control queueing delay and ensure that their input values are comparable. Similarly, to compare two AQM schemes on how well they accommodate packet bursts, identify burst-related control parameters and ensure they are configured with similar values.

2. Compare over a range of input configurations: there could be situations when the set of control parameters that affect a specific behavior have different semantics between the two AQM schemes. As mentioned above, PIE’s knob to control queueing delay has different semantics from CoDel’s. In such situations, these schemes MUST be compared over a range of input configurations. For example, compare PIE vs. CoDel over the range of target delay input configurations.

13.3.2. Deployment comparison

AQM schemes MUST be compared against deployment criteria such as the parameter sensitivity (Section 7.3), auto-tuning (Section 10) or implementation cost (Section 9).

13.4. Packet sizes and congestion notification

An AQM scheme may be considering packet sizes while generating congestion signals. [RFC7141] discusses the motivations behind this. For example, control packets such as DNS requests/responses, TCP SYN/ACKs are small, but their loss can severely impact the application performance. An AQM scheme may therefore be biased towards small packets by dropping them with smaller probability compared to larger packets. However, such an AQM scheme is unfair to data senders generating larger packets. Data senders, malicious or otherwise, are motivated to take advantage of such AQM scheme by transmitting smaller packets, and could result in unsafe deployments and unhealthy transport and/or application designs.
An AQM scheme SHOULD adhere to the recommendations outlined in [RFC7141], and SHOULD NOT provide disproportionate advantage to flows with smaller packets.

14. Acknowledgements

This work has been partially supported by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

15. Contributors


16. IANA Considerations

This memo includes no request to IANA.

17. Security Considerations

This document, by itself, presents no new privacy nor security issues.

18. References

18.1. Normative References


18.2. Informative References


Authors’ Addresses

Nicolas Kuhn (editor)
Telecom Bretagne
2 rue de la Chataigneraie
Cesson-Sevigne 35510
France
Phone: +33 2 99 12 70 46
Email: nicolas.kuhn@telecom-bretagne.eu

Naeem Khademi (editor)
University of Oslo
Department of Informatics, PO Box 1080 Blindern
N-0316 Oslo
Norway
Phone: +47 2285 24 93
Email: naeemk@ifi.uio.no

Preethi Natarajan (editor)
Cisco Systems
510 McCarthy Blvd
Milpitas, California
United States
Email: prenatar@cisco.com
David Ros  
Simula Research Laboratory AS  
P.O. Box 134  
Lysaker, 1325  
Norway  

Phone: +33 299 25 21 21  
Email: dros@simula.no