AQM
Internet-Draft
Intended status: Experimental
Expires: April 16, 2018

K. Nichols
Pollere, Inc.
V. Jacobson
A. McGregor, ed.
J. Iyengar, ed.
Google

October 13, 2017

Controlled Delay Active Queue Management
draft-ietf-aqm-codel-10

Abstract

This document describes a general framework called CoDel (Controlled Delay) 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.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The "persistently full buffer" problem has been discussed in the IETF community since the early 80's [RFC896]. The IRTF's End-to-End Research Group called for the deployment of active queue management (AQM) to solve the problem in 1998 [RFC2309]. Despite this 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 "bufferbloat" problem [TSV2011] [BB2011] has become increasingly important throughout the Internet but particularly at the consumer edge. Queue management has become more critical due to increased consumer use of the Internet, mixing large video transactions with time-critical VoIP and gaming.

An effective AQM remediates bufferbloat at a bottleneck while "doing no harm" at hops where buffers are not bloated. The development and deployment of AQM however is frequently subject to misconceptions about the cause of packet queues in network buffers. Network buffers exist to absorb the packet bursts that occur naturally in statistically multiplexed networks. Buffers helpfully absorb the queues created by such reasonable packet network behavior as 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. Unfortunately, other less useful network behaviors can cause queues to fill and their effects are not nearly as benign. Discussion of these issues and the reason why the solution is not simply smaller buffers can be found in [RFC2309], [VANQ2006], [REDL1998], and [CODEL2012]. To understand queue management, it is critical to understand the difference between the necessary, useful "good" queue, and the counterproductive "bad" queue.

Several approaches to 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. Learning from this past history, the CoDel approach is designed to meet the following goals:

- Making it parameterless for normal operation, with no knobs for operators, users, or implementers to adjust.

- Being able to distinguish "good queue" from bad queue and treat them differently, that is, keep delay low while permitting necessary bursts of traffic.

- Controlling 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.

- Adapting to dynamically changing link rates with no negative impact on utilization.

- Allowing simple and efficient implementation (can easily span the spectrum from low-end access points and home routers up to high-end router silicon).

CoDel has five major differences 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 network power [KLEIN81], 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, CoDel is expected to work across a wide range of conditions, with varying links and the full range of terrestrial round trip times.

CoDel is easily adapted to multiple queue systems as shown by [FQ-CODEL-ID]. Implementers and users SHOULD use the fq_codel multiple-queue approach as it deals with many problems beyond the reach of an AQM on a single queue.

CoDel was first published in [CODEL2012] and has been implemented in the Linux kernel.

Note that while this document refers to dropping packets when indicated by CoDel, it is reasonable to ECN-mark packets instead as well.

2. Conventions and terms 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.

The following terms are defined as used in this document:

sojourn time: the amount of time a packet has spent in a particular buffer, i.e. the time a packet departs the buffer minus the time the packet arrived at the buffer. A packet can depart a buffer via transmission or drop.

standing queue: a queue (in packets, bytes, or time delay) in a buffer that persists for a "long" time where "long" is on the order of the longer round trip times through the buffer as discussed in section 4.2. A standing queue occurs when the minimum queue over the "long" time is nonzero and is usually tolerable and even desirable as long as it does not exceed some target delay.

bottleneck bandwidth: the limiting bandwidth along a network path.

3. Understanding the Building Blocks of Queue Management

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 queueing, both good and bad, and covers the 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 where 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 the bottleneck link 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; this standing queue size is determined by the mismatch between the window size and the pipe size, and is 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 sending rate from the queue size, and any queue other than transient bursts only creates delays 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, and 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 bottleneck link's 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, the "good queue" goes away in about a round trip time, while "bad queue" hangs around for longer, 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 a queue, we need three basic components

- o Estimator - figure out what we’ve got.
- o Target setpoint - know what we want.
- o 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 parts to its estimator: what is observed as an indicator of queue and how the observations are used to detect good/bad queue.

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 at the expense of considerable implementation complexity.

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. It is not the queue length that should be controlled but the amount of excess delay packets experience due to a persistent or standing queue, which means that the packet sojourn time 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 allows queue management to be independent of link rate, gives superior performance to use of buffer size, and is directly related to user-visible performance. It works regardless of line rate changes or link sharing by multiple queues (which the individual queues may experience as changing rates).

Consider a link shared by two queues with different priorities. Packets that arrive at the high priority queue are sent as soon as the link is available while packets in the other queue have to wait until the high priority queue is empty (i.e., a strict priority scheduler). The number of packets in the high 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 waiting time for the high priority packets to be sent. This makes the sojourn time a good sample of the congestion that each separate queue is experiencing. This example also shows how the metric of sojourn time is independent of the number of queues or the service discipline used, and is instead indicative of congestion seen by the individual queues.

How can observed sojourn time be used to separate good queue from bad queue? Although averages, especially of queue length, have previously been widely used as an indicator of bad queue, their efficacy is questionable. 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 queue sojourn time would be one-half the time it takes to clear the burst. The average then would indicate a persistent queue where there is none. Instead of averages we recommend tracking the minimum sojourn time, then, if there is one packet that has a zero sojourn time then there is no persistent queue.

A persistent 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 value is considered to be in effect, and is related to the amount of time it takes for the largest expected burst to drain. Conservatively, 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.

(The following key insight makes computation of the local minimum efficient: It is sufficient to keep a single state variable of how long the minimum has been above or below the target value rather than retaining all the local values to compute the minimum, leading to both storage and computational savings. We use this insight in the pseudo-code for CoDel later in the document.)

These two parts, 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 in http://www.ietf.org/proceedings/84/slides/slides-84-tsvarea-4.pdf [1].

3.2. Target Setpoint

Now that we have a robust way of detecting standing queue, we need a target 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, this policy results in no backlog over time (no persistent queue), which negates much of the value of having a buffer, since it maximizes the bottleneck link bandwidth lost due to normal stochastic variation in packet interarrival time. We want a target 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', which is the ratio of a normalized throughput to a normalized delay [KLEIN81].

It is straightforward to derive an analytic expression for the average goodput of a TCP conversation at a given round-trip time r and target f (where f is expressed as a fraction of r). Reno TCP, for example, yields:

goodput = r (3 + 6f - f^2) / (4 (1+f))

Since the peak queue delay is simply the product of f and r, power is solely a function of f since the r's in the numerator and denominator cancel:

power is proportional to (1 + 2f - 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 Additive Increase Multiplicative Decrease (AIMD) TCP is monotone decreasing. But the curve is very flat for f < 0.1 followed by a increasing curvature with a knee around f = 0.2, then a steep, almost linear fall off [TSV84]. 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 0.1r 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 0.05r 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 of the bottleneck link. 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 target: the ideal range for the permitted standing queue, or the target setpoint, is between 5% and 10% of the TCP connection's RTT.

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 a value for the target 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 above.

A congested (but not overloaded) CoDel link with traffic composed solely or primarily of long-lived TCP flows will have a 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 the 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, as shown in [CODEL2012].

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 target setpoint for this measure that's a provably good balance between enhancing throughput and minimizing delay, and that this target 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 queueing system from any 'persistent queue above the target' state to a state where the persistent queue is below the 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 queueing 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 that 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 source-destination-source 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 does not know the number, duration, or kind of conversations creating its queue, so it has to learn the appropriate response. 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 the target is unlikely to overdrop and 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 drop 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. When the minimum sojourn time first crosses the target and CoDel drops a packet, the earliest the controller could see the effect of the drop is the round trip time (interval) + the local queue wait time (the target). If the next drop happens any earlier than this time (interval + target), CoDel will overdrop. In practice, the local queue waiting time tends to vary, so making the initial drop spacing (i.e., the time to the second drop) be exactly the minimum possible also leads to overdropping. Analysis of simulation and real-world measured data shows that the 75th percentile magnitude of this variation is less than the target, and so the initial drop spacing SHOULD be set to the estimator's interval (i.e., initial drop spacing = interval) to ensure that the controller has accounted for acceptable congestion delays.

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 target. Dropping at dequeue has both implementation and control advantages.

4. Overview of the Codel AQM

CoDel was initially designed as a bufferbloat solution for the consumer 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 CoDel algorithm described in the rest of this document uses two key variables: TARGET, which is the controller's target setpoint described in Section 3.2 and INTERVAL, which is the estimator's interval described in Section 3.3.

The only setting CoDel requires is the INTERVAL value, and as 100ms satisfies that definition for normal Internet usage, CoDel can be parameter-free for consumer use. To ensure that link utilization is not adversely affected, CoDel's estimator sets TARGET to one that optimizes power. 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 shows that an ideal TARGET is 5-10% of the connection round trip time

(RTT). In the open terrestrial-based Internet, especially at the consumer edge, we expect most unbloated RTTs to have a ceiling of 100ms [CHARB2007]. Using this RTT gives a minimum TARGET of 5ms and INTERVAL of 100ms. In practice, uncongested links will see sojourn times below TARGET more often than once per RTT, so the estimator is not overly sensitive to the value of 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 equal to INTERVAL. If the estimator’s output falls below 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 value, since an unnecessary drop would occur and lower link 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 drop state too soon after exiting it and resumes the drop state at a recent control level, if one exists. This logic is described more precisely in the pseudo-code below. Additional work is required to determine the frequency and importance of re-entering the drop state.

Note that CoDel AQM only enters its drop state when the local minimum sojourn delay has exceeded TARGET for a time period long enough for normal bursts to dissipate, ensuring that a burst of packets that fits in the pipe will not be dropped.

4.1. 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 bandwidths, including variable-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, therefore, 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 interface driver.

4.2. Setting INTERVAL

The INTERVAL value is chosen to give endpoints time to react to a drop without being so long that response times suffer. CoDel’s estimator, TARGET, and control loop all use 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 a priori. The best policy seems to be to use an INTERVAL value 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 (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). An INTERVAL 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 a 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.

4.3. Setting TARGET

TARGET is the maximum acceptable persistent queue delay above which CoDel is dropping or preparing to drop and below which CoDel will not drop. TARGET SHOULD be set to 5ms for normal Internet traffic.

The calculations of section 3.2 show that the best TARGET value is 5-10% of the RTT, with the low end of 5% preferred. Extensive simulations exploring the impact of different TARGET values when used with mixed traffic flows with different RTTs and different bandwidths show that below a TARGET of 5ms, utilization suffers for some conditions and traffic loads, and above 5ms showed very little or no improvement in utilization.

Sojourn times must remain above the TARGET for INTERVAL amount of time 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 TARGET. Since Internet traffic has very dynamic characteristics, the actual sojourn delay experienced by packets varies greatly and is often less than 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 total amount of queueing delay experienced by a packet should be less than 10ms even under extremely congested conditions. This net delay is substantially lower than common current queueing delays on the Internet that grow to orders of seconds [NETAL2010, CHARB2007].

A note on the roles of TARGET and the minimum-tracking INTERVAL. TARGET SHOULD NOT be increased in response to lower layers that 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 amount of time, and increasing TARGET only leads to longer queue delays. On the other hand, where a significant additional delay is added to the intrinsic RTT of most or all packets due to the waiting time for a transmission, it is necessary to increase INTERVAL by that extra delay. TARGET SHOULD NOT be adjusted for such short-term bursts, but INTERVAL MAY need to be adjusted if the path's intrinsic RTT changes.

4.4. Use with multiple queues

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 what CoDel excels at: using a packet's sojourn time in the buffer completely circumvents this problem. In a multiple-queue setting, a separate CoDel algorithm runs on each queue, but each CoDel instance 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 does a multiple-queue CoDel system. As an optimization to avoid queueing more than necessary, when testing for queue occupancy before dropping, the total occupancy of all queues sharing the same output link SHOULD be used. This property of CoDel has been exploited in fq_codel [FQ-CODEL-ID], which hashes on the packet header fields to determine a specific bin, or sub-queue, for the packet, 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").

4.5. Setting up CoDel

CoDel is set for use in devices in the open Internet. An INTERVAL setting of 100ms is used, TARGET is set to 5% of INTERVAL, and the initial drop spacing is also set to the 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 value for 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 INTERVAL. But these settings will cause problems such as overdropping and low throughput if used on the open Internet, so devices that allow CoDel to be configured SHOULD 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 have attempted to highlight just these additions by presenting CoDel as a sub-class of a basic fifo-queue base class. 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 the Linux kernel version of CoDel - a well-written, well tested, real-world, C-based implementation. As of this writing, it is available at https://github.com/torvalds/linux/blob/master/net/sched/sch_codel.c.

5.1. Data Types

time_t is an integer time value in units convenient for the system. 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 enqueue() and dequeue() 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 dequeue() method but shouldn’t require a lock with the enqueue() method.

flag_t is a Boolean.

5.2. Per-queue state (codel_queue_t instance variables)

```
time_t first_above_time_ = 0; // Time to declare sojourn time above
                              // TARGET
time_t drop_next_ = 0;        // Time to drop next packet
uint32_t count_ = 0;          // Packets dropped in drop state
uint32_t lastcount_ = 0;      // Count from previous iteration
flag_t dropping_ = false;     // Set to true 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. Enqueue routine

All the work of CoDel is done in the dequeue routine. The only CoDel addition to enqueue is putting the current time in the packet’s tstamp field so that the dequeue routine can compute the packet’s sojourn time. Note that packets arriving at a full buffer will be dropped, but these drops are not counted towards CoDel’s computations.

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

5.5. Dequeue routine

This is the heart of CoDel. There are two branches based on whether the controller is in drop state: (i) if the controller is in drop state (that is, the minimum packet sojourn time is greater than TARGET) then the controller checks if it is time to leave drop state or schedules the next drop(s); or (ii) if the controller is not in drop state, it determines if it should enter drop state and do the initial drop.

```
packet_t* CoDelQueue::dequeue()
{
    time_t now = clock();
    dodequeue_result r = dodequeue(now);
    uint32_t delta;

    if (dropping_) {
```

```
            if (! r.ok_to_drop) {
                // sojourn time below TARGET - leave drop state
                dropping_ = false;
            }
            // 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);
                ++count_;
                r = dodequeue(now);
                if (! r.ok_to_drop) {
                    // leave drop state
                    dropping_ = false;
                } else {
                    // schedule the next drop.
                    drop_next_ = control_law(drop_next_, count_);
                }
            }
        // If we get here we're not in drop state. The 'ok_to_drop'
        // return from dodequeue means that the sojourn time has been
        // above 'TARGET' for 'INTERVAL' so enter drop state.
        } else if (r.ok_to_drop) {
            drop(r.p);
            r = dodequeue(now);
            dropping_ = true;

            // 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.) Implementations vary slightly here; this is
            // the Linux version, which is more widely deployed and
            // tested.
            delta = count_ - lastcount_;
            count_ = 1;
            if ((delta > 1) && (now - drop_next_ < 16*INTERVAL))
                count_ = delta;

            drop_next_ = control_law(now, count_);
            lastcount_ = count_;
        }
        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 can be computed efficiently using only integer multiplication.

```
time_t codel_queue_t::control_law(time_t t, uint32_t count)
{
    return t + INTERVAL / sqrt(count);
}
```

Next is a helper routine the 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 amount of time. 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;
} dodequeue_result;

dodequeue_result codel_queue_t::dodequeue(time_t now)
{
    dodequeue_result r = { queue_t::dequeue(), false };
    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 = true;
        }
    }
    return r;
}
```

5.7. Implementation considerations

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.

Since CoDel requires relatively little per-queue state and no direct communication or state sharing between the enqueue and dequeue routines, it is relatively simple to add CoDel to almost any packet processing pipeline, including ASIC- or NPU-based forwarding engines. One issue to consider is dodequeue()'s use of a 'bytes()' function to determine the current queue size in bytes. 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 simply the difference between these two counters (32-bit counters should be adequate.) Enqueue has to update its counter once per packet queued but it does not 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. Further Experimentation

We encourage experimentation with the recommended values of TARGET and INTERVAL for Internet settings. CoDel provides general, efficient, parameterless building blocks for queue management that can be applied to single or multiple queues in a variety of data networking scenarios. CoDel's settings may be modified for other special-purpose networking applications.

7. Security Considerations

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

8. IANA Considerations

This document does not require actions by IANA.

9. Acknowledgments

The authors thank Jim Gettys for the constructive nagging that made us get the work "out there" before we thought it was ready. We 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 thank

Nandita Dukkipati for contributions to section 6 and for comments which helped to substantially improve this draft. We thank the AQM working group and the Transport Area shepherd, Wes Eddy, for patiently prodding this draft all the way to a standard.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key Words for use in RFCs to Indicate Requirement Levels", March 1997.

10.2. Informative References

[BB2011] Gettys, J. and K. Nichols, "Bufferbloat: Dark Buffers in the Internet", Communications of the ACM 9(11) pp. 57-65.

[BMPFQ] Suter, B., "Buffer Management Schemes for Supporting TCP in Gigabit Routers with Per-flow Queueing", IEEE Journal on Selected Areas in Communications Vol. 17 Issue 6, June, 1999, pp. 1159-1169.

[CHARB2007] Dischinger, M., "Characterizing Residential Broadband Networks", Proceedings of the Internet Measurement Conference San Diego, CA, 2007.

[CODEL2012] Nichols, K. and V. Jacobson, "Controlling Queue Delay", Communications of the ACM Vol. 55 No. 11, July, 2012, pp. 42-50.

[FQ-CODEL-ID] Hoeiland-Joergensen, T., McKenney, P., dave.taht@gmail.com, d., Gettys, J., and E. Dumazet, "FlowQueue-Codel", draft-ietf-aqm-fq-codel-06 (work in progress), March 2017.

[KLEIN81] Kleinrock, L. and R. Gail, "An Invariant Property of Computer Network Power", International Conference on Communications June, 1981, <http://www.lk.cs.ucla.edu/data/files/Gail/power.pdf>.

[MACTCP1997] Mathis, M., Semke, J., and J. Mahdavi, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", ACM SIGCOMM Computer Communications Review Vol. 27 no. 1, Jan. 2007.

[NETAL2010]
Kreibich, C., "Netalyzr: Illuminating the Edge Network", Proceedings of the Internet Measurement Conference Melbourne, Australia, 2010.

[REDL1998]
Nichols, K., Jacobson, V., and K. Poduri, "RED in a Different Light", Tech report, September, 1999, <http://www.cnaf.infn.it/~ferrari/papers/ispn/red_light_9_30.pdf>.

[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[RFC896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.

[SFQ1990] McKenney, P., "Stochastic Fairness Queuing", Proceedings of IEEE INFOCOMM 90 San Francisco, 1990.

[TSV2011] Gettys, J., "Bufferbloat: Dark Buffers in the Internet", IETF 80 presentation to Transport Area Open Meeting, March, 2011, <http://www.ietf.org/proceedings/80/tsvarea.html>.

[TSV84] Jacobson, V., "CoDel talk at TSV meeting IETF 84", <http://www.ietf.org/proceedings/84/slides/slides-84-tsvarea-4.pdf>.

[VANQ2006]
Jacobson, V., "A Rant on Queues", talk at MIT Lincoln Labs, Lexington, MA July, 2006, <http://www.pollere.net/Pdfdocs/QrantJul06.pdf>.

10.3. URIs

[1] http://www.ietf.org/proceedings/84/slides/slides-84-tsvarea-4.pdf

Appendix A. Applying CoDel in the datacenter

Nandita Dukkipati and her group at Google realized 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 is 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 setpoint 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 as there would be on the public Internet.

In several data center workload benchmarks, which are typically bursty, CoDel reduced the queueing latencies at the qdisc, and thereby improved the mean and 99th-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 qdisc buffers that are inherently bursty because of TCP Segmentation Offload (TSO).

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

Janardhan Iyengar
Google

Email: jri@google.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 27, 2016

G. Fairhurst
University of Aberdeen
M. Welzl
University of Oslo

November 24, 2015

The Benefits of using Explicit Congestion Notification (ECN)
draft-ietf-aqm-ecn-benefits-08

Abstract

The goal of this document is to describe the potential benefits when applications use a transport that enables Explicit Congestion Notification (ECN). The document outlines the principal gains in terms of increased throughput, reduced delay and other benefits when ECN is used over a network path that includes equipment that supports Congestion Experienced (CE) marking. It also discusses challenges for successful deployment of ECN. It does not propose new algorithms to use ECN, nor does it describe the details of implementation of ECN in endpoint devices (Internet hosts), routers or other network devices.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Internet Transports (such as TCP and SCTP) are implemented in endpoints (Internet hosts) and are designed to detect and react to network congestion. Congestion may be detected by loss of an IP packet or, if Explicit Congestion Notification (ECN) [RFC3168] is enabled, by the reception of a packet with a Congestion Experienced (CE) marking in the IP header. Both of these are treated by transports as indications of congestion. ECN may also be enabled by other transports: UDP applications that provide congestion control may enable ECN when they are able to correctly process the ECN signals [ID.RFC5405.bis] (e.g., ECN with RTP [RFC6679]).

Active Queue Management (AQM) [RFC7567] is a class of techniques that can be used by network devices (a router, middlebox, or other device that forwards packets through the network) to manage the size of queues in network buffers.

A network device that does not support AQM typically uses a drop-tail policy to drop excess IP packets when its queue becomes full. The discard of packets is treated by transport protocols as a signal that indicates congestion on the end-to-end network path. End-to-end transports, such as TCP, can cause a low level of loss while seeking to share capacity with other flows. Although losses are not always due to congestion (loss may be due to link corruption, receiver-overrun, etc) end points have to conservatively presume that all loss is potentially due to congestion and reduce their rate. Observed loss therefore results in a congestion control reaction by the transport to reduce the maximum rate permitted by the sending endpoint.

ECN makes it possible for the network to signal the presence of incipient congestion without incurring packet loss, it 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 receiving CE-marking instead of dropping packets. Several benefits reduce latency (e.g., reduced Head-of-Line Blocking).

The use of ECN is indicated in the ECN field [RFC3168], carried in the packet header of all IPv4 and IPv6 packets. This field may be set to one of four values shown in Table 1. The not-ECT codepoint '00' indicates a packet that is not using ECN. The ECT(0) codepoint '01' and the ECT(1) codepoint '10' both indicate that the transport protocol using the IP layer supports the use of ECN. The CE codepoint '11' is set by an ECN-capable network device to indicate congestion to the transport endpoint.

ECN FIELD		Name
0	0	Not-ECT
0	1	ECT(1)
1	0	ECT(0)
1	1	CE

Table 1: The ECN Field in the IP Packet Header (based on [RFC3168]).

When an application uses a transport that enables use of ECN [RFC3168], 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 the ECN-capable IP packets. An ECN-capable network device can then signal incipient congestion (network queueing) at a point before a transport experiences congestion loss or high queuing delay. The marking is generally performed as the result of various AQM algorithms [RFC7567], where the exact combination of AQM/ECN algorithms does not need to be known by the transport endpoints.

The focus of the document is on usage of ECN by transport and application layer flows, not its implementation in endpoint hosts, or in routers and other network devices.

1.1. Terminology

The following terms are used:

AQM: Active Queue Management.

CE: Congestion Experienced, a codepoint value ’11’ marked in the ECN field of the IP packet header.

ECN-capable IP Packet : A packet where the ECN field is set to a non-zero ECN value (i.e., with a ECT(0), ECT(1), or the CE codepoint).

ECN-capable network device : An ECN-capable network device may forward, drop, or queue an ECN-capable packet and may choose to CE-mark this packet when there is incipient congestion.

ECN-capable transport/application : A transport that sends ECN-capable IP Packets, and monitors reception of the ECN field and generates appropriate feedback to control the rate of the sending endpoint.

to provide end-to-end congestion control.

ECN field: A 2-bit field specified for use explicit congestion signalling in the IPv4 and IPv6 packet headers.

Endpoint: An Internet host that terminates a transport protocol connection across an Internet path.

Incipient Congestion: The detection of congestion when it is starting, perhaps by a network device noting that the arrival rate exceeds the forwarding rate.

Network device: A router, middlebox, or other device that forwards IP packets through the network.

non-ECN-capable: A network device or endpoint that does not interpret the ECN field. Such a device is not permitted to change the ECN codepoint.

not-ECN-capable IP Packet: An IP packet with the ECN field set to a value of zero ('00'). A not-ECN-capable packet may be forwarded, dropped or queued by a network device.

2. Benefit of using ECN to avoid Congestion Loss

An ECN-capable network device is expected to CE-mark an ECN-capable IP packet when an AQM method detects incipient congestion, rather than to drop the packet [RFC7567]. An application can benefit from this marking in several ways:

2.1. Improved Throughput

ECN seeks to avoid the inefficiency of dropping data that has already made it across at least part of the network path.

ECN can improve the throughput of an application, although this increase in throughput 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 would be 0.02 (i.e., 1 in 50 packets). This translates into an approximate 2% throughput gain if ECN is enabled. (Note that in heavy congestion, packet loss may be unavoidable with, or without, ECN.)

2.2. Reduced Head-of-Line Blocking

Many Internet transports provide in-order delivery of received data segments to the applications they support. For these applications, use of ECN can reduce the delay that can result when these applications experience packet loss.

Packet loss may occur for various reasons. One cause arises when an AQM scheme drops a packet as a signal of incipient congestion. Whatever the cause of loss, a missing packet needs to trigger a congestion control response. A reliable transport also triggers retransmission to recover the lost data. For a transport providing in-order delivery, this requires that the transport receiver stalls (or waits) for all data that was sent ahead of a lost segment to be correctly received before it can forward any later data to the application. A loss therefore creates a delay of at least one RTT after a loss event before data can be delivered to an application. We call this Head-of-Line (HOL) blocking. This is the usual requirement for TCP and SCTP. (PR-SCTP [RFC3758], UDP [RFC0768][ID.RFC5405.bis], and DCCP [RFC4340] provide a transport that does not provide re-ordering).

By enabling ECN, a transport continues to receive in-order data when there is incipient congestion, and can pass this data to the receiving application. Use of ECN avoids the additional reordering delay in a reliable transport. The sender still needs to make an appropriate congestion-response to reduce the maximum transmission rate for future traffic, which usually will require a reduction in the sending rate [ID.RFC5405.bis].)

2.3. Reduced Probability of RTO Expiry

Some patterns of packet loss can result in a Retransmission Time Out (RTO), which causes a sudden and significant change in the allowed rate at which a transport/application can forward packets. Because ECN provides an alternative to drop for network devices to signal incipient congestion, this can reduce the probability of loss and hence reduce the likelihood of RTO expiry.

Internet transports/applications generally use a RTO timer as a last resort to detect and recover loss [ID.RFC5405.bis] [RFC5681]). Specifically, a RTO timer detects loss of a packet that is not followed by other packets, such as at the end of a burst of data segments or when an application 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). This loss of the last segment (or last few segments) of a traffic burst is also known as a "tail loss".

Standard transport recovery methods, such as Fast Recovery ([RFC5681], are often unable to recover from a tail loss. This is because the endpoint receiver is unaware that the lost segments were actually sent, and therefore generates no feedback [Fla13]. Retransmission of these segments therefore relies on expiry of a transport retransmission timer. This timer is also used to detect a lack of forwarding along a path. Expiry of the RTO therefore results in the consequent loss of state about the network path being used. This typically includes resetting path estimates such as the RTT, re-initialising the congestion window, and possibly updates to other transport state. This can reduce the performance of the transport until it again adapts to the path.

An ECN-capable network device cannot eliminate the possibility of tail loss, because a drop may occur due to a traffic burst exceeding the instantaneous available capacity of a network buffer or as a result of the AQM algorithm (overload protection mechanisms, etc [RFC7567]). However, an ECN-capable network device that observes incipient congestion may be expected to buffer the IP packets of an ECN-capable flow and set a CE-mark in one or more packet(s), rather than triggering packet drop. Setting a CE-mark signals incipient congestion without forcing the transport/application to enter retransmission timeout. This reduces application-level latency and can improve the throughput for applications that send intermittent bursts of data.

The benefit of avoiding retransmission loss is expected to be significant when ECN is used on TCP SYN/ACK packets [RFC5562] where the RTO interval may be large because TCP cannot base the timeout period on prior RTT measurements from the same connection.

2.4. Applications that do not Retransmit Lost Packets

A transport that enables ECN can receive timely congestion signals without the need to retransmit packets each time it receives a congestion signal.

Some latency-critical applications do not retransmit lost packets, yet may be able to adjust their sending rate following detection of incipient 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 the transport/application may therefore employ mechanisms (e.g., packet forward error correction, data duplication, or media codec error concealment) to mitigate the immediate effect of congestion loss on the application. Some mechanisms consume additional network capacity, some require additional processing and some contribute additional path latency when congestion is experienced. By decoupling congestion control from loss, ECN can allow transports that support these applications to reduce their rate before the application experiences loss from congestion. This can reduce the negative impact of triggering loss-hiding mechanisms with a direct positive impact on the quality experienced by the users of these applications.

2.5. Making Incipient 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 allowing information to be collected about the presence of incipient congestion.

Recording the presence of CE-marked packets can provide information about the current congestion level experienced on a network path. A network flow that only experiences CE-marking and no loss implies that the sending endpoint is experiencing only congestion. A network flow may also experience loss (e.g., due to queue overflow, AQM methods that protect other flows, link corruption or loss in middleboxes). When a mixture of CE-marking and packet loss is experienced, transports and measurements need to assume there is congestion [RFC7567]. An absence of CE-marks therefore does not indicate a path has not experienced congestion.

The reception of CE-marked packets can be used to monitor the level of congestion by a transport/application or a network operator. For example, ECN measurements are used by Congestion Exposure (ConEx) [RFC6789]. In contrast, metering packet loss is harder.

2.6. Opportunities for new Transport Mechanisms

ECN can enable design and deployment of new algorithms in network devices and Internet transports. Internet transports need to regard both loss and CE-marking as an indication of congestion. However, while the amount of feedback provided by drop ought naturally to be minimized, this is not the case for ECN. In contrast, an ECN-Capable network device could provide richer (more frequent and fine-grained) indication of its congestion state to the transport.

For any ECN-capable transport, the receiving endpoint needs to provide feedback to the transport sender to indicate that CE-marks have been received.[RFC3168] provides one method that signals once each round trip time that CE-marked packets have been received.

A receiving endpoint may provide more detailed feedback to the congestion controller at the sender (e.g., describing the set of received ECN codepoints, or indicating each received CE-marked packet). Precise feedback about the number of CE-marks encountered is supported by the Real Time Protocol (RTP) when used over UDP [RFC6679] and has been proposed for SCTP [ST14] and TCP [ID.Acc.ECN].

More detailed feedback is expected to enable evolution of transport protocols allowing the congestion control mechanism to make a more appropriate decision on how to react to congestion. Designers of transport protocols need to consider not only how network devices CE-mark packets, but also how the control loop in the application/transport reacts to reception of these CE-marked packets.

Benefit has been noted when packets are CE-marked early using an instantaneous queue, and if the receiving endpoint provides feedback about the number of packet marks encountered, an improved sender behavior has been shown to be possible, e.g, Datacenter TCP (DCTCP) [AL10]. DCTCP is targeted at controlled environments such as a datacenter. This is work-in-progress and it is currently unknown whether or how such behaviour could be safely introduced into the Internet. Any update to an Internet transport protocol requires careful consideration of the robustness of the behaviour when working with endpoints or network devices that were not designed for the new congestion reaction.

3. Network Support for ECN

For an application to use ECN requires that the endpoints first enable ECN within the transport being used, but also for all network devices along the path to at least forward IP packets that set a non-zero ECN codepoint.

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 [RFC7567].

- ECN may be deployed within a controlled environment, for example within a data centre or within a well-managed private network. This use of ECN may be tuned to the specific use-case. An example is DCTCP [AL10] [ID.DCTCP].

Early experience of using ECN across the general Internet encountered a number of operational difficulties when the network path either failed to transfer ECN-capable packets or inappropriately changed the ECN codepoints [BA11]. A recent survey reported a growing support for network paths to pass ECN codepoints [TR15].

The remainder of this section identifies what is needed for network devices to effectively support ECN.

3.1. The ECN Field

The current IPv4 and IPv6 specifications assign usage of 2 bits in the IP header to carry the ECN codepoint. This 2-bit field was reserved in [RFC2474] and assigned in [RFC3168].

[RFC4774] discusses some of the issues in defining alternate semantics for the ECN field, and specifies requirements for a safe coexistence in an Internet that could include routers that do not understand the defined alternate semantics.

Some network devices were configured to use a routing hash that included the set of 8 bits forming the now deprecated Type of Service (ToS) field [RFC1349]. The present use of this field assigns 2 of these bits to carry the ECN field. This is incompatible with use in a routing hash, because it could lead to IP packets that carry a CE-mark being routed over a different path to those packets that carried an ECT mark. The resultant reordering would impact the performance of transport protocols (such as TCP or SCTP) and UDP-based applications that are senstive to reordering. A network device that conforms to this older specification needs to be updated to the current specifications [RFC2474] to support ECN. Configuraton of network devices must note that the ECN field may be updated by any ECN-capable network device along a path.

3.2. Forwarding ECN-Capable IP Packets

Not all network devices along a path need to be ECN-capable (i.e., perform CE-marking). However, all network devices need to be configured not to drop packets solely because the ECT(0) or ECT(1) codepoints are used.

Any network device that does not perform CE-marking of an ECN-capable packet can be expected to drop these packets under congestion. Applications that experience congestion at these network devices do not see any benefit from enabling ECN. However, they may see benefit if the congestion were to occur within a network device that did support ECN.

3.3. Enabling ECN in Network Devices

Network devices should use an AQM algorithm that CE-marks ECN-capable traffic when making decisions about the response to congestion [RFC7567]. An ECN method should set a CE-mark on ECN-capable packets in the presence of incipient congestion. A CE-marked packet will be interpreted as an indication of incipient congestion by the transport endpoints.

There is opportunity to design an AQM method for an ECN-capable network device that differs from an AQM method designed to drop packets. [RFC7567] states that the network device should allow this behaviour to be configurable.

[RFC3168] describes a method in which a network device sets the CE-mark at the time that the network device 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, [RFC7567] recommends that network devices allow independent configuration of the settings for AQM dropping and ECN marking. Such separate configuration of the drop and mark policies is supported in some network devices.

3.4. Co-existance of ECN and non-ECN flows

Network devices need to be able to forward all IP flows and provide appropriate treatment for both ECN and non-ECN traffic.

The design considerations for an AQM scheme supporting ECN needs to consider the impact of queueing during incipient congestion. For example, a simple AQM scheme could choose to queue ECN-capable and non-ECN capable flows in the same queue with an ECN scheme that CE-mark packets during incipient congestion. The CE-marked packets that remain in the queue during congestion can continue to contribute to queueing delay. In contrast, non-ECN-capable packets would normally be dropped by an AQM scheme under incipient congestion. This difference in queueing is one motivation for consideration of more advanced AQM schemes, and may provide an incentive for enabling flow isolation using scheduling [RFC7567]. The IETF is defining methods to evaluate the suitability of AQM schemes for deployment in the general Internet [ID.AQM.eval].

3.5. Bleaching and Middlebox Requirements to deploy ECN

Network devices should not be configured to change the ECN codepoint in the packets that they forward, except to set the CE-codepoint to signal incipient congestion.

Cases have been noted where an endpoint sends a packet with a non-zero ECN mark, but the packet is received by the remote endpoint with a zero ECN codepoint [TR15]. This could be a result of a policy that erases or "bleaches" the ECN codepoint values at a network edge (resetting the codepoint to zero). Bleaching may occur for various reasons (including normalising packets to hide which equipment supports ECN). This policy prevents use of ECN by applications.

When ECN-capable IP packets, marked as ECT(0) or ECT(1), are remarked to non-ECN-capable (i.e., the ECN field is set to zero codepoint), this could result in the packets being dropped by ECN-capable network devices further along the path. This eliminates the advantage of using of ECN.

A network device must not change a packet with a CE mark to a zero codepoint, if the network device decides not to forward the packet with the CE-mark, it has to instead drop the packet and not bleach the marking. This is because a CE-marked packet has already received ECN treatment in the network, and remarking it would then hide the congestion signal from the receiving endpoint. This eliminates the benefits of ECN. It can also slow down the response to congestion compared to using AQM, because the transport will only react if it later discovers congestion by some other mechanism.

Prior to RFC2474, a previous usage assigned the bits now forming the ECN field as a part of the now deprecated Type of Service (ToS) field [RFC1349]. A network device that conforms to this older specification was allowed to remark or erase the ECN codepoints, and such equipment needs to be updated to the current specifications to support ECN.

3.6. Tunneling ECN and the use of ECN by Lower Layer Networks

Some networks may use ECN internally or tunnel ECN (e.g., for traffic engineering or security). These methods need to ensure that the ECN-field of the tunnel packets is handled correctly at the ingress and egress of the tunnel. Guidance on the correct use 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].

4. Using ECN across the Internet

A receiving endpoint needs to report the loss it experiences when it uses loss-based congestion control. So also, when ECN is enabled, a receiving endpoint must correctly report the presence of CE-marks by providing a mechanism to feed this congestion information back to the sending endpoint , [RFC3168], [ID.RFC5405.bis], enabling the sender to react to experienced congestion. This mechanism needs to be designed to operate robustly across a wide range of Internet path characteristics. This section describes partial deployment, how ECN-enabled endpoints can continue to work effectively over a path that experiences misbehaving network devices or when an endpoint does not correctly provide feedback of ECN congestion information.

4.1. Partial Deployment

Use of ECN is negotiated between the endpoints prior to using the mechanism.

ECN has been designed to allow incremental partial deployment [RFC3168]. Any network device can choose to use either ECN or some other loss-based policy to manage its traffic. Similarly, transport/ application negotiation allows senders and receiving endpoints to choose whether ECN will be used to manage congestion for a particular network flow.

4.2. Detecting whether a Path Really Supports ECN

Internet transport and applications need to be robust to the variety and sometimes varying path characteristics that are encountered in the general Internet. They need to monitor correct forwarding of ECN over the entire path and duration of a session.

To be robust, applications and transports need to be designed with the expectation of heterogeneous forwarding (e.g., where some IP packets are CE-marked by one network device, and some by another, possibly using a different AQM algorithm, or when a combination of CE-marking and loss-based congestion indications are used. ([ID.AQM.eval] describes methodologies for evaluating AQM schemes.)

A transport/application also needs to be robust to path changes. A change in the set of network devices along a path could 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 (see Section 3.5).

A sending endpoint can check that any CE-marks applied to packets received over the path are indeed delivered to the remote receiving endpoint and that appropriate feedback is provided. (This could be done by a sender setting known a CE codepoint for specific packets in a network flow and then checking whether the remote endpoint correctly reports these marks [ID.Fallback], [TR15].) If a sender detects persistent misuse of ECN, it needs to fall back to using loss-based recovery and congestion control. Guidance on a suitable tranport reaction is provided in [ID.Fallback].

4.3. Detecting ECN Receiver Feedback Cheating

Appropriate feedback requires that the endpoint receiver does not try to conceal reception of CE-marked packets in the ECN feedback information provided to the sending endpoint [RFC7567]. Designers of applications/transports are therefore encouraged to include mechanisms that can detect this misbehavior. If a sending endpoint detects that a receiver is not correctly providing this feedback, it needs to fall back to using loss-based recovery instead of ECN.

5. Summary: Enabling ECN in Network Devices and Hosts

This section summarises the benefits of deploying and using ECN within the Internet. It also provides a list of prerequisites to achieve ECN deployment.

Application developers should where possible use transports that enable ECN. Applications that directly use UDP need to provide support to implement the functions required for ECN [ID.RFC5405.bis]. 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 2 summarises the key benefits.

Section	Benefit
2.1	Improved throughput
2.2	Reduced Head-of-Line blocking
2.3	Reduced probability of RTO Expiry
2.4	Applications that do not retransmit lost packets
2.5	Making incipient congestion visible
2.6	Opportunities for new transport mechanisms

Table 2: Summary of Key Benefits

Network operators and people configuring network devices should enable ECN [RFC7567].

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

- o A network device that updates the ECN field in IP packets must use IETF-specified methods (see Section 3.1).

- A network device may support alternate ECN semantics (see Section 3.1).

- A network device must not choose a different network path solely because a packet carries has a CE-codepoint set in the ECN Field, CE-marked packets need to follow the same path as packets with an ECT(0) or ECT(1) codepoint (see Section 3.1).Network devices need to be configured not to drop packets solely because the ECT(0) or ECT(1) codepoints are used (see Section 3.2).

- A network device must not change a packet with a CE mark to a not-ECN-capable codepoint ('00'), if the network device decides not to forward the packet with the CE-mark, it has to instead drop the packet and not bleach the marking (see Section 3.5).

- An ECN-capable network device should correctly update the ECN codepoint of ECN-capable packets in the presence of incipient congestion (see Section 3.3).

- Network devices need to be able to forward both ECN-capable and not-ECN-capable flows (see Section 3.4).

Prerequisites for network endpoints to enable use of ECN include:

- An application should use an Internet transport that can set and receive ECN marks (see Section 4).

- An ECN-capable transport/application must return feedback indicating congestion to the sending endpoint and perform an appropriate congestion response (see Section 4).

- An ECN-capable transport/application should detect paths where there is there is persistent misuse of ECN and fall back to not sending ECT(0) or ECT(1) (see Section 4.2).

- Designers of applications/transports are encouraged to include mechanisms that can detect and react appropriately to misbehaving receivers that fail to report CE-marked packets (see Section 4.3).

6. 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, Colin Perkins, Richard Scheffenegger, Dave Taht, Wes Eddy, Fred Baker, Mikael Abrahamsson, Mirja Kuehlewind, John Leslie, and other members of the TSVWG and AQM working groups.

7. IANA Considerations

XX RFC Ed - PLEASE REMOVE THIS SECTION XXX

This memo includes no request to IANA.

8. Security Considerations

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

9. 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 restructured - 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.

Revision 03 includes updates from the mailing list and WG discussions at the Dallas IETF meeting.

The section "Avoiding Capacity Overshoot" was removed, since this refers primarily to an AQM benefit, and the additional benefits of ECN are already stated. Separated normative and informative references

Revision 04 (WG Review during WGLC)

Updated the abstract.

Added a table of contents.

Addressed various (some conflicting) comments during WGLC with new text.

The section on Network Support for ECN was moved, and some suggestions for rewording sections were implemented.

Decided not to remove section headers for 2.1 and 2.2 - to ensure the document clearly calls-out the benefits.

Updated references. Updated text to improve consistency of terms and added definitions for key terms.

Note: The group suggested this document should not define recommendations for end hosts or routers, but simply state the things needs to enable deployment to be successful.

Revision 05 (after WGLC comments)

Updated abstract to avoid suggesting that this describes new methods for deployment.

Added ECN-field definition, and sorted terms in order.

Added an opening para to each "benefit" to say what this is. Sought to remove redundancy between sections.

Added new section on Codepoints to avoid saying the same thing twice.

Reworked sections 3 and 4 to clarify discussion and to remove unnecessary text.

Reformatted Summary to refer to sections describing things, rather than appear as a list of new recommendations. Reordered to match the new document order.

Note: This version expects an update to RFC5405.bis that will indicate UDP ECN requirements (normative).

Revision 06

Corrections from Miria.

Revision 07

Update to include IESG feedback from: Spencer, Dan, Benoit, Joel. Corrected Non-ECN to Not-ECN where appropriate, added table of codepoints, clarified sentences describing "conservative" behaviour, added requirement to not do ToS-based routing (Junos enhanced hash), etc. Ammended Acknowledgments section.

Revision 08

Typo and definition correction from Bob Briscoe.

10. References

10.1. Normative References

[ID.RFC5405.bis]
Eggert, Lars., Fairhurst, Gorry., and Greg. Shepherd, "Unicast UDP Usage Guidelines", 2015.

[RFC2474] "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers".

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <http://www.rfc-editor.org/info/rfc3168>.

[RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <http://www.rfc-editor.org/info/rfc6040>.

[RFC7567] Baker, F. and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management", Internet-draft draft-ietf-aqm-recommendation-06, October 2014.

10.2. 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.

[Fla13] Flach, Tobias., Dukkipati, Nandita., Terzis, Andreas., Raghavan, Barath., Cardwell, Neal., Cheng, Yuchung., Jain, Ankur., Hao, Shuai., Katz-Bassett, Ethan., and Ramesh. Govindan, "Reducing web latency: the virtue of gentle aggression.", SIGCOMM 2013, October 2013.

[ID.Acc.ECN] Briscoe, Bob., Scheffeneger, Richard., and Mirja. Kuehlewind, "More Accurate ECN Feedback in TCP, Work-in-Progress".

[ID.AQM.eval] Kuhn, Nicolas., Natarajan, Preethi., Ros, David., and Naeem. Khademi, "AQM Characterization Guidelines (Work-in-progress, draft-ietf-aqm-eval-guidelines)", 2015.

[ID.DCTCP] Bensley, S., Eggert, Lars., and D. Thaler, "Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters (Work-in-progress, draft-bensley-tcpm-dctcp)", 2015.

[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.

[ID.Fallback] Kuehlewind, Mirja. and Brian. Trammell, "A Mechanism for ECN Path Probing and Fallback, draft-kuehlewind-tcpm-ecn-fallback, Work-in-Progress".

[RFC0768] Postel, J., "User Datagram Protocol", 1980.

[RFC1349] "Type of Service in the Internet Protocol Suite".

[RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <http://www.rfc-editor.org/info/rfc3649>.

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, DOI 10.17487/RFC3758, May 2004, <http://www.rfc-editor.org/info/rfc3758>.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <http://www.rfc-editor.org/info/rfc4340>.

[RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, DOI 10.17487/RFC4774, November 2006, <http://www.rfc-editor.org/info/rfc4774>.

[RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <http://www.rfc-editor.org/info/rfc5562>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <http://www.rfc-editor.org/info/rfc5681>.

[RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <http://www.rfc-editor.org/info/rfc6679>.

[RFC6789] Briscoe, B., Ed., Woundy, R., Ed., and A. Cooper, Ed., "Congestion Exposure (ConEx) Concepts and Use Cases", RFC 6789, DOI 10.17487/RFC6789, December 2012, <http://www.rfc-editor.org/info/rfc6789>.

[ST14] Stewart, R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", Internet-draft draft-stewart-tsvwg-sctpecn-05.txt, January 2014.

[TR15] Tranmmel, Brian., Kuehlewind, Mirja., Boppart, Damiano, Learmonth, Iain., and Gorry. Fairhurst, "Enabling internet-wide deployment of Explicit Congestion Notification Tramwell, B., Kuehlewind, M., Boppart, D., Learmonth, I., Fairhurst, G. & Scheffnegger, Passive and Active Measurement Conference (PAM)", March 2015.

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

Internet Engineering Task Force N. Kuhn, Ed.
Internet-Draft CNES, Telecom Bretagne
Intended status: Informational P. Natarajan, Ed.
Expires: December 16, 2016 Cisco Systems
N. Khademi, Ed.
University of Oslo
D. Ros
Simula Research Laboratory AS
June 14, 2016

AQM Characterization Guidelines
draft-ietf-aqm-eval-guidelines-13

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. This document describes various criteria for performing characterizations of AQM schemes, that can be used in lab testing during development, prior to deployment.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Active Queue Management (AQM) addresses the concerns arising from using unnecessarily large and unmanaged buffers to improve network and application performance, such as presented in the section 1.2 of the AQM recommendations document [RFC7567]. Several AQM algorithms have been proposed in the past years, most notably Random Early Detection (RED) [FLOY1993], BLUE [FENG2002], and Proportional Integral controller (PI) [HOLLO2001], and more recently CoDel [I-D.ietf-aqm-codel] and PIE [I-D.ietf-aqm-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 buffer occupancy, and therefore the end-to-end delay. 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.

A buffer is a physical volume of memory in which a queue or set of queues are stored. When speaking of a specific queue in this document, "buffer occupancy" refers to the amount of data (measured in bytes or packets) that are in the queue, and the "maximum buffer size" refers to the maximum buffer occupancy. In switches and routers, a global memory space is often shared between the available interfaces, and thus, the maximum buffer size for any given interface may vary over the time.

Bufferbloat [BB2011] is the consequence of deploying large unmanaged buffers on the Internet -- the buffering has often been measured to be ten times or hundred times larger than needed. Large buffer sizes in combination with TCP and/or unresponsive flows increases end-to-end delay. This results 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 chartered 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. This document provides characterization guidelines that can be used to assess the applicability, performance and deployability of an AQM, whether it is candidate for standardization at IETF or not.

AQM algorithm implemented in a router can be separated from the scheduling of packets sent out by the router as discussed in the AQM recommendations document [RFC7567]. The rest of this memo refers to the AQM as a dropping/marking policy as a separate feature to any interface scheduling scheme. This document may be complemented with another one on guidelines for assessing combination of packet scheduling and AQM. We note that such a document will inherit all the guidelines from this document plus any additional scenarios relevant for packet scheduling such as flow starvation evaluation or impact of the number of hash buckets.

1.1. 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. To ensure the safety deployment of an AQM, its behaviour should be assessed in a variety of scenarios. Whenever possible, solutions ought to aim at both maximizing goodput and minimizing latency.

1.2. Goals of this document

This document recommends a generic list of scenarios against which an AQM proposal should be evaluated, considering both potential performance gain and safety of deployment. The guidelines help to quantify performance of AQM schemes in terms of latency reduction, goodput maximization and the trade-off between these two. The document presents central aspects of an AQM algorithm that should be considered whatever the context, such as burst absorption capacity, RTT fairness or resilience to fluctuating network conditions. The guidelines also discuss methods to understand the various aspects associated with safely deploying and operating the AQM scheme. Thus, one of the key objectives behind formulating the guidelines is to help ascertain whether a specific AQM is not only better than drop-tail (i.e. without AQM and with a BDP-sized buffer) but also safe to deploy: the guidelines can be used to compare several AQM proposals with each other, but should be used to compare a proposal with drop-tail.

This memo details generic characterization scenarios against which any AQM proposal should be evaluated, irrespective of whether or not an AQM is standardized by the IETF. This documents recommends the relevant scenarios and metrics to be considered. The document presents central aspects of an AQM algorithm that should be considered whatever the context, such as burst absorption capacity, RTT fairness or resilience to fluctuating network conditions.

These guidelines do not define and are not bound to a particular deployment scenario or evaluation toolset. Instead the guidelines can be used to assert the potential gain of introducing an AQM for the particular environment, which is of interest to the testers. These guidelines do not cover every possible aspect of a particular algorithm. These guidelines do not present context-dependent scenarios (such as 802.11 WLANs, data-centers or rural broadband networks). To keep the guidelines generic, a number of potential router components and algorithms (such as DiffServ) are omitted.

The goals of this document can thus be summarized as follows:

- o The present characterization guidelines provide a non-exhaustive list of scenarios to help ascertain whether an AQM is not only better than drop-tail (with a BDP-sized buffer), but also safe to deploy; the guidelines can also be used to compare several AQM proposals with each other.

- o The present characterization guidelines (1) are not bound to a particular evaluation toolset and (2) can be used for various deployment contexts; testers are free to select a toolset that is best suited for the environment in which their proposal will be deployed.

- o The present characterization guidelines are intended to provide guidance for better selecting an AQM for a specific environment; it is not required that an AQM proposal is evaluated following these guidelines for its standardization.

1.3. 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].

1.4. Glossary

- o application-limited traffic: a type of traffic that does not have an unlimited amount of data to transmit.

- o AQM: the Active Queue Managment (AQM) algorithm implemented in a router can be separated from the scheduling of packets sent by the router. The rest of this memo refers to the AQM as a dropping/ marking policy as a separate feature to any interface scheduling scheme [RFC7567].

- o BDP: Bandwidth Delay Product.

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

- o buffer occupancy: amount of data that are stored in a buffer, measured in bytes or packets.

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

- o IW10: TCP initial congestion window set to 10 packets.

- o latency: one-way delay of packets across Internet paths. This definition suits transport layer definition of the latency, that shall not be confused with an application layer view of the latency.

- o goodput: goodput is defined as the number of bits per unit of time forwarded to the correct destination minus any bits lost or retransmitted [RFC2647]. The goodput should be determined for each flow and not for aggregates of flows.

- o SQRT: the square root function.

- o ROUND: the round function.

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 schemes 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 could be used to better quantify (1) the reduction of latency, (2) maximization of goodput and (3) the trade-off between these two. This section provides normative requirements for metrics that can be used to assess the performance of an AQM scheme.

Some metrics listed in this section are not suited to every type of traffic detailed in the rest of this document. It is therefore not necessary to measure all of the following metrics: the chosen metric may not be relevant to the context of the evaluation scenario (e.g., latency vs. goodput trade-off in application-limited traffic scenarios). Guidance is provided for each metric.

2.1. Flow completion time

The flow completion time is an important performance metric for the end-user when the flow size is finite. The definition of the flow size may be source of contradictions, thus, this metric can consider a flow as a single file. 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 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:

FCT [s] = Fs [Byte] / (G [Bit/s] / 8 [Bit/Byte])

Where flow size is the size of the transport-layer payload in bits and goodput is the transport-layer payload transfer time (described in Section 2.5).

If this metric is used to evaluate the performance of web transfers, it is suggested to rather consider the time needed to download all the objects that compose the web page, as this makes more sense in terms of user experience than assessing the time needed to download each object.

2.2. Flow start up time

The flow start up time is the time between the request has been sent from the client and the server starts to transmit data. The amount of packets dropped by an AQM may seriously affect the waiting period during which the data transfer has not started. This metric would specifically focus on the operations such as DNS lookups, TCP opens and SSL handshakes.

2.3. Packet loss

Packet loss can occur en-route, this can impact the end-to-end performance measured at receiver.

The tester should evaluate loss experienced at the receiver using one of the two metrics:

- o the packet loss ratio: this metric is to be frequently measured during the experiment. The long-term loss ratio is of interest for steady-state scenarios only;

- o the interval between consecutive losses: the time between two losses is to be measured.

The packet loss ratio 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:

- o to check that for every packet, a corresponding packet was received within a reasonable time, as presented in the document that proposes a metric for one-way packet loss across Internet paths [RFC2680].

- o to keep a count of all packets sent, and a count of the non-duplicate packets received, as discussed in RFC that presents a benchmarking methodology [RFC2544].

The interval between consecutive losses, which is also called a gap, is a metric of interest for VoIP traffic [RFC3611].

2.4. Packet loss synchronization

One goal of an AQM algorithm is to help to avoid global synchronization of flows sharing a bottleneck buffer on which the AQM operates ([RFC2309],[RFC7567]). The "degree" of packet-loss synchronization between flows should be assessed, with and without the AQM under consideration.

Loss synchronization among flows may be quantified by several slightly different metrics that capture different aspects of the same issue [HASS2008]. However, in real-world measurements the choice of metric could be imposed by practical considerations -- e.g., whether fine-grained information on packet losses at the bottleneck is 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. This metric can be used in real-world experiments to characterize synchronization along arbitrary Internet paths [JAY2006].

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.

2.5. Goodput

The goodput has been defined as the number of bits per unit of time forwarded to the correct destination interface, minus any bits lost or retransmitted, such as proposed in the secton 3.17 of the RFC describing the benchmarking terminology for firewall performances [RFC2647]. This definition requires 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 provides an appreciation of how well an AQM scheme improves transport and application performance. The measured end-to-end goodput is linked to the dropping/marking policy of the AQM scheme -- e.g., the fewer the number of packet drops, the fewer packets need retransmission, minimizing the impact of AQM 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 of an AQM scheme in minimizing packet drops that impact application performance and to estimate how well the AQM scheme works with ECN.

The measurement of the goodput allows the tester to evaluate to which extent an AQM is able to maintain a high bottleneck utilization. This metric should also be obtained frequently during an 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. 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.

2.6. Latency and jitter

The latency, or the one-way delay metric, is discussed in [RFC2679]. 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) serves well all use cases [RFC5481].

The end-to-end latency includes components other than just the queuing delay, such as the signal processing delay, transmission delay and the processing delay. Moreover, the jitter is caused by variations in queuing and processing delay (e.g., scheduling effects). The introduction of an AQM scheme would impact end-to-end latency and jitter, and therefore these metrics should be considered in the end-to-end evaluation of performance.

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

The metrics presented in this section may be considered in order to discuss and quantify the trade-off between latency and goodput.

With regards to the goodput, and in addition to the long-term stationary goodput value, it is recommended to take measurements every multiple of the minimum RTT (minRTT) between A and B. It is suggested to take measurements at least every K x minRTT (to smooth out the fluctuations), with K=10. Higher values for K can be considered whenever it is more appropriate for the presentation of the results, since the value for K may depend 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 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.

From each of these sets of measurements, the cumulative density function (CDF) of the considered metrics should be computed. If the considered scenario introduces dynamically varying parameters, temporal evolution of the metrics could also be generated. For each scenario, the following graph may be generated: the x-axis shows queuing delay (that is the average per-packet delay in excess of minimum RTT), the y-axis the goodput. Ellipses are computed such as detailed in [WINS2014]: "We take each individual [...] run [...] as one point, and then compute the 1-epsilon elliptic contour of the maximum-likelihood 2D Gaussian distribution that explains the points. [...] we plot the median per-sender throughput and queueing delay as a circle. [...] The orientation of an ellipse represents the covariance between the throughput and delay measured for the protocol." This graph provides part of a better understanding of (1) the delay/goodput trade-off for a given congestion control mechanism (Section 5), and (2) how the goodput and average queue delay vary as a function of the traffic load (Section 8.2).

3. Generic setup 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

```
   +--------------+                                +--------------+
   |sender A_i    |                                |receive B_i   |
   |--------------|                                |--------------|
   | SEN.Flow1.1 +---------+            +-----------+ REC.Flow1.1 |
   |        +     |        |            |          |        +     |
   |        |     |        |            |          |        |     |
   |        +     |        |            |          |        +     |
   | SEN.Flow1.X +-----+   |            |  +--------+ REC.Flow1.X |
   +--------------+    |   |            |  |       +--------------+
        +            +-+---+---+     +--+--+---+            +
        |            |Router L |     |Router R |            |
        |            |---------|     |---------|            |
        |            | AQM     |     |         |            |
        |            | BuffSize|     | BuffSize|            |
        |            | (Bsize) +-----+ (Bsize) |            |
        |            +-----+--++     ++-+------+            |
        +                  |  |       | |                   +
   +--------------+        |  |       | |          +--------------+
   |sender A_n    |        |  |       | |          |receive B_n   |
   |--------------|        |  |       | |          |--------------|
   | SEN.FlowN.1 +---------+  |       | +-----------+ REC.FlowN.1 |
   |        +     |           |       |            |        +     |
   |        |     |           |       |            |        |     |
   |        +     |           |       |            |        +     |
   | SEN.FlowN.Y +------------+       +-------------+ REC.FlowN.Y |
   +--------------+                                +--------------+
```

Figure 1: Topology and notations

Figure 1 is a generic topology where:

- o traffic profile is a set of flows with similar characteristics - RTT, congestion control scheme, transport protocol, etc.;
- o senders with different traffic characteristics (i.e., traffic profiles) can be introduced;
- o the timing of each flow could be different (i.e., when does each flow start and stop);
- o each traffic profile can comprise various number of flows;
- o each link is characterized by a couple (one-way delay, capacity);
- o sender A_i is instantiated for each traffic profile. A corresponding receiver B_i is instantiated for receiving the flows in the profile;

- o flows sharing a bottleneck (the link between routers L and R);

- o 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 operation of the AQM mechanism whereas the asymmetric link scenario evaluates an AQM mechanism in a more realistic setup;

- o 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 drop policy of the AQM on TCP ACK packets and its impact on the performance (Section 9.2).

Although this topology may not perfectly reflect actual topologies, the simple topology is commonly used in the world of simulations and small testbeds. It can be considered as adequate to evaluate AQM proposals [I-D.irtf-iccrg-tcpeval]. Testers ought to pay attention to the topology that has been used to evaluate an AQM scheme when comparing this scheme with a newly proposed AQM scheme.

3.2. Buffer size

The size of the buffers should be carefully chosen, and may be set to the bandwidth-delay product; the bandwidth being the bottleneck capacity and the delay the largest RTT in the considered network. The size of the buffer can impact the AQM performance and is a dimensioning parameter that will be considered when comparing AQM proposals.

If a specific buffer size is required, 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 document considers running three different congestion control algorithms between A and B

- o Standard TCP congestion control: the base-line congestion control is TCP NewReno with SACK [RFC5681].

- o Aggressive congestion controls: a base-line congestion control for this category is TCP Cubic [I-D.ietf-tcpm-cubic].

- o 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.': a base-line congestion control for this category is LEDBAT [RFC6817].

Other transport congestion controls can OPTIONALLY be evaluated in addition. Recent transport layer protocols are not mentioned in the following sections, for the sake of simplicity.

4. Methodology, Metrics, AQM Comparisons, Packet Sizes, Scheduling and ECN

4.1. Methodology

A description of each test setup should be detailed to allow this test to be compared with other tests. This also allows others to replicate the tests if needed. This test setup should detail software and hardware versions. The tester could make its data available.

The proposals should be evaluated 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.

4.2. Comments on metrics measurement

The document presents the end-to-end metrics that ought to 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 advise to consider 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 the way these metrics are measured is expected to depend on the evaluation toolset.

4.3. Comparing AQM schemes

This document recognizes that these guidelines may be used for comparing AQM schemes.

AQM schemes need to 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.

4.3.1. Performance comparison

AQM schemes should be compared against the generic scenarios that are summarized in Section 13. AQM schemes may be compared for specific network environments such as data centers, home networks, etc. If an AQM scheme has parameter(s) that were externally tuned for optimization or other purposes, these values must be disclosed.

AQM schemes belong to different varieties such as queue-length based schemes (ex. RED) or queueing-delay based scheme (ex. CoDel, PIE). 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 tuning parameters of the two schemes have different semantics, resulting in different control points. Such differences in AQM schemes can be easily overlooked while making comparisons.

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

1. similar control parameters and implications: Testers should be aware of the control parameters of the different schemes that control similar behavior. Testers should also be aware of the input value ranges and corresponding implications. For example, consider two different schemes - (A) queue-length based AQM scheme, and (B) queueing-delay based scheme. A and B are likely to have different kinds of control inputs to control the target delay - target queue length in A vs. target queuing delay in B, for example. Setting parameter values such as 100MB for A vs. 10ms for B will have different implications depending on evaluation context. Such context-dependent implications must be considered before drawing conclusions on performance comparisons. Also, it would be preferable if an AQM proposal listed such

parameters and discussed how each relates to network characteristics such as capacity, average RTT etc.

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 has tuning parameters to control queue delay that has a different semantics from those used in CoDel. In such situations, these schemes need to be compared over a range of input configurations. For example, compare PIE vs. CoDel over the range of target delay input configurations.

4.3.2. Deployment comparison

AQM schemes must be compared against deployment criteria such as the parameter sensitivity (Section 8.3), auto-tuning (Section 12) or implementation cost (Section 11).

4.4. Packet sizes and congestion notification

An AQM scheme may be considering packet sizes while generating congestion signals [RFC7141]. For example, control packets such as DNS requests/responses, TCP SYNs/ACKs are small, but their loss can severely impact application performance. An AQM scheme may therefore be biased towards small packets by dropping them with lower 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 the best current practive for dropping and marking packets document [RFC7141], and should not provide undue advantage to flows with smaller packets, such as discussed in the section 4.4 of the AQM recommendation document [RFC7567]. In order to evaluate if an AQM scheme is biased towards flows with smaller size packets, traffic can be generated, such as defined in Section 8.2.2, where half of the flows have smaller packets (e.g. 500 bytes packets) than the other half of the flow (e.g. 1500 bytes packets). In this case, the metrics reported could be the same as in Section 6.3, where Category I is the set of flows with smaller packets and Category II the one with larger packets. The bidirectional scenario could also be considered (Section 9.2).

4.5. Interaction with ECN

ECN [RFC3168] is an alternative that allows AQM schemes to signal receivers about network congestion that does not use packet drop. There are benefits of providing ECN support for an AQM scheme [WELZ2015].

If the tested AQM scheme can support ECN, the testers must discuss and describe the support of ECN, such as discussed in the AQM recommendation [RFC7567]. Also, the AQM's ECN support can be studied and verified by replicating tests in Section 8.1 with ECN turned ON at the TCP senders. The results can be used to not only evaluate the performance of the tested AQM with and without ECN markings, but also quantify the interest of enabling ECN.

4.6. Interaction with Scheduling

A network device may use per-flow or per-class queuing with a scheduling algorithm to either prioritize certain applications or classes of traffic, limit the rate of transmission, or to provide isolation between different traffic flows within a common class, such as discussed in the section 2.1 of the AQM recommendation document [RFC7567].

The scheduling and the AQM conjointly impact on the end-to-end performance. Therefore, the AQM proposal must discuss the feasibility to add scheduling combined with the AQM algorithm. It can be explained whether the dropping policy is applied when packets are being enqueued or dequeued.

These guidelines do not propose guidelines to assess the performance of scheduling algorithms. Indeed, as opposed to characterizing AQM schemes that is related to their capacity to control the queuing delay in a queue, characterizing scheduling schemes is related to the scheduling itself and its interaction with the AQM scheme. As one example, the scheduler may create sub-queues and the AQM scheme may be applied on each of the sub-queues, and/or the AQM could be applied on the whole queue. Also, schedulers might, such as FQ-CoDel [HOEI2015] or FavorQueue [ANEL2014], introduce flow prioritization. In these cases, specific scenarios should be proposed to ascertain that these scheduler schemes not only helps in tackling the bufferbloat, but also are robust under a wide variety of operating conditions. This is out of the scope of this document that focus on dropping and/or marking AQM schemes.

5. Transport Protocols

Network and end-devices need to be configured with a reasonable amount of buffer space to absorb transient bursts. In some situations, network providers tend to configure devices with large buffers to avoid packet drops triggered by a full buffer and to maximize the link utilization for standard loss-based TCP traffic.

AQM algorithms are often evaluated by considering Transmission Control Protocol (TCP) [RFC0793] with a limited number of applications. TCP is a widely deployed transport. It fills up available buffers until a sender transfering a bulk flow with TCP receives a signal (packet drop) that reduces the sending rate. The larger the buffer, the higher the buffer occupancy, and therefore the queuing delay. An efficient AQM scheme sends out early congestion signals to TCP to bring the queuing delay under control.

Not all endpoints (or applications) using TCP use the same flavor of TCP. Variety of senders generate different classes of traffic which may not react to congestion signals (aka non-responsive flows in the section 3 of the AQM recommendation document [RFC7567]) or may not reduce their sending rate as expected (aka Transport Flows that are less responsive than TCP, such as proposed in the section 3 of the AQM recommendation document [RFC7567], also called "aggressive flows"). In these cases, AQM schemes seek to control the queuing delay.

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).

5.1. TCP-friendly sender

5.1.1. TCP-friendly sender with the same initial congestion window

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, with an Initial congestion Window (IW) set to 3 packets, 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 2.7 could be generated.

5.1.2. TCP-friendly sender with different initial congestion windows

This scenario can be used to evaluate how an AQM scheme adapts to a traffic mix consisting of TCP flows with different values of the IW.

For this scenario, two types of flows must be generated between sender A and receiver B:

o A single long-lived non application-limited TCP NewReno flow;

o A single 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 first and the transmission of the application-limited flow starts after the non application-limited flow has reached steady state. The steady state can be assumed when the goodput is stable.

For each of these scenarios, the graph described in Section 2.7 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.

5.2. Aggressive transport sender

This scenario helps testers 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 2.7 could be generated.

5.3. Unresponsive transport sender

This scenario helps testers to evaluate how an AQM scheme reacts to a transport sender that is less responsive than TCP. Note that faulty transport implementations on an end host and/or faulty network elements en-route that "hide" congestion signals in packet headers may also lead to a similar situation, such that the AQM scheme needs to adapt to unresponsive traffic (see the section 3 of the AQM recommendation document [RFC7567]). To this end, these guidelines propose the two following scenarios.

The first scenario can be used to evaluate queue build up. It considers unresponsive flow(s) whose sending rate is greater than the bottleneck link capacity between routers L and R. This scenario consists of a long-lived non application limited UDP flow transmits data between sender A and receiver B. Graphs described in Section 2.7 could be generated.

The second scenario can be used to evaluate if the AQM scheme is able to keep the responsive fraction under control. This scenario considers a mixture of TCP-friendly and unresponsive traffics. It consists of a long-lived UDP flow from unresponsive application and a single long-lived, non application-limited (unlimited data available to the transport sender from application layer), TCP New Reno flow that transmit data between sender A and receiver B. As opposed to the first scenario, the rate of the UDP traffic should not be greater than the bottleneck capacity, and should be higher than half of the bottleneck capacity. For each type of traffic, the graph described in Section 2.7 could be generated.

5.4. Less-than Best Effort transport sender

This scenario helps to evaluate how an AQM scheme reacts to LBE congestion controls that 'results in smaller bandwidth and/or delay impact on standard TCP than standard TCP itself, when sharing a bottleneck with it.' [RFC6297]. There are potential fateful interactions when AQM and LBE techniques are combined [GONG2014]; this scenario helps to evaluate whether the coexistence of the proposed AQM and LBE techniques may be possible.

A single long-lived non application-limited TCP NewReno flow transfers data between sender A and receiver B. Other TCP-friendly congestion control schemes may also be considered. Single long-lived non application-limited LEDBAT [RFC6817] flows transfer data between sender A and receiver B. We recommend to set the target delay and gain values of LEDBAT respectively to 5 ms and 10 [TRAN2014]. Other LBE congestion control schemes may also be considered and are listed in the IETF survey of LBE protocols [RFC6297].

For each of the TCP-friendly and LBE transports, the graph described in Section 2.7 could be generated.

6. Round Trip Time Fairness

6.1. Motivation

An AQM scheme's congestion signals (via drops or ECN marks) must reach the transport sender so that a responsive sender can initiate its congestion control mechanism and adjust the sending rate. This procedure is thus dependent on the end-to-end path RTT. When the RTT varies, the onset of congestion control is impacted, and in turn impacts the ability of an AQM scheme to control the queue. It is therefore important to assess the AQM schemes for a set of RTTs between A and B (e.g., from 5 ms to 200 ms).

The asymmetry in terms of difference in intrinsic RTT between various paths sharing the same bottleneck should be considered, so that the fairness between the flows can be discussed. 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 the RTT fairness.

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.

6.2. Recommended tests

The recommended topology is detailed in Figure 1.

To evaluate the RTT fairness, for each run, two flows are divided into two categories. Category I whose RTT between sender A and receiver B should be 100ms. Category II which RTT between sender A and receiver B should be in the range [5ms;560ms] inclusive. The maximum value for the RTT represents the RTT of a satellite link [RFC2488].

A set of evaluated flows must use the same congestion control algorithm: all the generated flows could be single long-lived non application-limited TCP NewReno flows.

6.3. Metrics to evaluate the RTT fairness

The outputs that must be measured are: (1) the cumulative average goodput of the flow from Category I, goodput_Cat_I (Section 2.5); (2) the cumulative average goodput of the flow from Category II, goodput_Cat_II (Section 2.5); (3) the ratio goodput_Cat_II/ goodput_Cat_I; (4) the average packet drop rate for each category (Section 2.3).

7. Burst Absorption

"AQM mechanisms need to control the overall queue sizes, to ensure that arriving bursts can be accommodated without dropping packets" [RFC7567].

7.1. Motivation

An AQM scheme can face bursts of packet arrivals due to various reasons. Dropping one or more packets from a burst can result in performance penalties for the corresponding flows, since dropped packets have to be retransmitted. Performance penalties can result in failing to meet SLAs and be a disincentive to AQM adoption.

The ability to accommodate bursts translates to larger queue length and hence more queuing delay. On the one hand, it is important that an AQM scheme quickly brings bursty traffic under control. On the other hand, a peak in the packet drop rates to bring a packet burst quickly under control could result in multiple drops per flow and severely impact transport and application performance. Therefore, an AQM scheme ought to 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 that maintains short queues allows some remaining space in the buffer for bursts of arriving packets. The tolerance to bursts of packets depends upon the number of packets in the queue, which is directly linked to the AQM algorithm. Moreover, an AQM scheme may implement a feature controlling the maximum size of accepted bursts, that can depend on the buffer occupancy or the currently estimated queuing delay. The impact of the buffer size on the burst allowance may be evaluated.

7.2. Recommended tests

For this scenario, tester must evaluate how the AQM performs with a traffic mixed that could be composed of (from sender A to receiver B):

- o Burst of packets at the beginning of a transmission, such as web traffic with IW10;
- o Applications that send large bursts of data, such as bursty video frames;
- o Background traffic, such as Constant Bit Rate (CBR) UDP traffic and/or A single non application-limited 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.

Case	Video	Web (IW 10)	CBR	Bulk TCP Traffic
I	0	1	1	0
II	0	1	1	1
III	1	1	1	0
IV	1	1	1	1

Figure 2: Bursty traffic scenarios

A new web page download could start after the previous web page download is finished. Each web page could be composed by at least 50 objects and the size of each object should be at least 1kB. 6 TCP parallel connections should be generated to download the objects, each parallel connections having an initial congestion window set to 10 packets.

For each of these scenarios, the graph described in Section 2.7 could be generated for each application. Metrics such as end-to-end latency, jitter, flow completion time may be generated. For the cases of frame generation of bursty video traffic as well as the choice of web traffic pattern, these details and their presentation are left to the testers.

8. Stability

8.1. Motivation

The safety of an AQM scheme is directly related to its stability under varying operating conditions such as varying traffic profiles and fluctuating network conditions. Since operating conditions can vary often the AQM needs to remain stable under these conditions without the need for additional external tuning.

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

- o Traffic and congestion levels are higher during peak hours than off-peak hours.

- o In the presence of a scheduler, the draining rate of a queue can vary depending on the occupancy of other queues: a low load on a high priority queue implies a higher draining rate for the lower priority queues.

- o The capacity available can vary over time (e.g., a lossy channel, a link supporting traffic in a higher diffserv class).

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

8.2. Recommended tests

Note that the traffic profiles explained below comprises non application-limited TCP flows. For each of the below scenarios, the graphs described in Section 2.7 should be generated, and the goodput of the various flows should be cumulated. For Section 8.2.5 and Section 8.2.6 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.

8.2.1. 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 [MORR2000]:

l_r = 0.76 * N^2 / S^2

N = S * SQRT(1/0.76) * SQRT (l_r)

These guidelines use the loss rate to define the different congestion levels, but they do not stipulate that in other circumstances, measuring the congestion level gives you an accurate estimation of the loss rate or vice-versa.

8.2.2. Mild congestion

This scenario can be used 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 = ROUND (0.036*S)

8.2.3. Medium congestion

This scenario can be used 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 = ROUND (0.081*S)

8.2.4. Heavy congestion

This scenario can be used 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_heavy, is then:

N_heavy = ROUND (0.114*S)

8.2.5. Varying the congestion level

This scenario can be used to evaluate how an AQM scheme reacts to incoming traffic resulting in various levels of congestion 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.

8.2.6. Varying available capacity

This scenario can be used to help characterize how the AQM behaves and adapts to bandwidth changes. The experiments are not meant to reflect the exact conditions of Wi-Fi environments since it is hard to design repetitive experiments or accurate simulations for such scenarios.

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

- o 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.

- o 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 capacity 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. To better assess the impact of draining rates on the AQM behavior, the tester must compare its performance with those of drop-tail and should provide a reference document for their proposal discussing performance and deployment compared to those of drop-tail. Burst traffic, such as presented in Section 7.2, could also be considered to assess the impact of varying available capacity on the burst absorption of the AQM.

8.3. Parameter sensitivity and stability analysis

The control law used by an AQM is the primary means by which the queuing delay is controlled. Hence understanding the control law is critical to understanding the behavior of the AQM scheme. The control law could 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.

Transports operating under the control of AQM experience the effect of multiple control loops that react over different timescales. It is therefore important that proposed AQM schemes are seen to be stable when they are deployed at multiple points of potential congestion along an Internet path. The pattern of congestion signals (loss or ECN-marking) arising from AQM methods also need to not adversely interact with the dynamics of the transport protocols that they control.

AQM proposals should provide background material showing control theoretic analysis of the AQM control law and the input parameter space within which the control law operates as expected; or could use another way to discuss the stability of the control law. 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 control law better and the network conditions/deployments under which the AQM is stable.

9. 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.

9.1. Traffic mix

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

- o Bulk TCP transfer
- o Web traffic
- o VoIP
- o Constant Bit Rate (CBR) UDP traffic
- o Adaptive video streaming (either unidirectional or bidirectional)

Various traffic mixes can be considered. These guidelines recommend to examine at least the following example: 1 bi-directional VoIP; 6 Web pages download (such as detailed in Section 7.2); 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 2.7 could be generated for each class of traffic. Metrics such as end-to-end latency, jitter and flow completion time may be reported.

9.2. Bi-directional traffic

Control packets such as DNS requests/responses, TCP SYNs/ACKs 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. The amount of asymmetry between the uplink and the downlink depends on the context. These guidelines recommend to consider a mild congestion level and the traffic presented in Section 8.2.2 in both directions. In this case, the metrics reported must be the same as in Section 8.2 for each direction.

The traffic mix presented in Section 9.1 may also be generated in both directions.

10. Example of multi-AQM scenario

10.1. Motivation

Transports operating under the control of AQM experience the effect of multiple control loops that react over different timescales. It is therefore important that proposed AQM schemes are seen to be stable when they are deployed at multiple points of potential congestion along an Internet path. The pattern of congestion signals (loss or ECN-marking) arising from AQM methods also need to not adversely interact with the dynamics of the transport protocols that they control.

10.2. Details on the evaluation scenario

```
+---------+                             +-----------+
|senders A|---+                     +---|receivers A|
+---------+   |                     |   +-----------+
        +-----+---+  +---------+  +--+-----+
        |Router L |--|Router M |--|Router R|
        |AQM A    |  |AQM M    |  |No AQM  |
        +---------+  +--+------+  +--+-----+
+---------+             |            |   +-----------+
|senders B|-------------+            +---|receivers B|
+---------+                              +-----------+
```

Figure 3: Topology for the Multi-AQM scenario

Figure Figure 3 describes topology options for evaluating multi-AQM scenarios. The AQM schemes are applied in sequence and impact the induced latency reduction, the induced goodput maximization and the trade-off between these two. Note that AQM schemes A and B introduced in Routers L and M could be (I) same scheme with identical parameter values, (ii) same scheme with different parameter values, or (iii) two different schemed. To best understand the interactions and implications, the mild congestion scenario as described in Section 8.2.2 is recommended such that the number of flows is equally shared among senders A and B. Other relevant combination of congestion levels could also be considered. We recommend to measure the metrics presented in Section 8.2.

11. Implementation cost

11.1. Motivation

Successful deployment of AQM is directly related to its cost of implementation. Network devices can 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 their 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. This also facilitates discsusions around which kind of devices can easily support the AQM and which cannot.

11.2. Recommended discussion

AQM proposals should highlight parts of their 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 can be trivial to implement the their AQM logic on the device. If the device provides indirect means to estimate the queuing delay (for example: timestamps, dequeuing rate), then the AQM behavior is sensitive to the precision of the queuing delay estimations are for that device. Highlighting the sensitivity of an AQM scheme to queuing delay estimations helps implementers to identify appropriate means of implementing the mechanism on a device.

12. Operator Control and Auto-tuning

12.1. Motivation

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 to achieve acceptable benefit from using RED. Fluctuating congestion levels and network conditions add to the complexity. Incorrect parameter values lead to poor performance.

Any AQM scheme is likely to have parameters whose values affect the control law and behaviour of an AQM. Exposing all these parameters as control parameters 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 parameters minimizes the number of ways a user can break a system where an AQM scheme is deployed at. Fewer control parameters make the AQM scheme more user-friendly and easier to deploy and debug.

"AQM algorithms should not require tuning of initial or configuration parameters in common use cases." such as stated in the section 4.3 of the AQM recommendation document [RFC7567]. A scheme ought to expose only those parameters that control the macroscopic AQM behavior such as queue delay threshold, queue length threshold, etc.

Additionally, the safety of an AQM scheme is directly related to its stability under varying operating conditions such as varying traffic profiles and fluctuating network conditions, as described in Section 8. Operating conditions vary often and hence the AQM needs to 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.

12.2. Recommended discussion

In order to understand an AQM's deployment considerations and performance under a specific environment, AQM proposals should describe the parameters that control the macroscopic AQM behavior, and identify any parameters that require tuning to operational conditions. It could be interesting to also discuss that even if an AQM scheme may not adequately auto-tune its parameters, the resulting performance may not be optimal, but close to something reasonable.

If there are any fixed parameters within the AQM, their setting should be discussed and justified, to help understand whether a fixed parameter value is applicable for a particular environment.

If an AQM scheme is evaluated with parameter(s) that were externally tuned for optimization or other purposes, these values must be disclosed.

13. Summary

Figure 4 lists the scenarios for an extended characterization of an AQM scheme. This table comes along with a set of requirements to present more clearly the weight and importance of each scenario. The requirements listed here are informational and their relevance may depend on the deployment scenario.

Scenario	Sec.	Informational requirement
Interaction with ECN	4.5	must be discussed if supported
Interaction with Scheduling	4.6	should be discussed
Transport Protocols	5.	
TCP-friendly sender	5.1	scenario must be considered
Aggressive sender	5.2	scenario must be considered
Unresponsive sender	5.3	scenario must be considered
LBE sender	5.4	scenario may be considered
Round Trip Time Fairness	6.2	scenario must be considered
Burst Absorption	7.2	scenario must be considered
Stability	8.	
Varying congestion levels	8.2.5	scenario must be considered
Varying available capacity	8.2.6	scenario must be considered
Parameters and stability	8.3	this should be discussed
Various Traffic Profiles	9.	
Traffic mix	9.1	scenario is recommended
Bi-directional traffic	9.2	scenario may be considered
Multi-AQM	10.2	Scenario may be considered

Figure 4: Summary of the scenarios and their requirements

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).

Many thanks to S. Akhtar, A.B. Bagayoko, F. Baker, R. Bless, D. Collier-Brown, G. Fairhurst, J. Gettys, P. Goltsman, T. Hoiland-Jorgensen, K. Kilkki, C. Kulatunga, W. Lautenschlager, A.C. Morton, R. Pan, G. Skinner, D. Taht and M. Welzl for detailed and wise feedback on this document.

15. IANA Considerations

This memo includes no request to IANA.

16. Security Considerations

Some security considerations for AQM are identified in [RFC7567].This document, by itself, presents no new privacy nor security issues.

17. References

17.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, 1997.

[RFC2544] Bradner, S. and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices", RFC 2544, DOI 10.17487/RFC2544, March 1999, <http://www.rfc-editor.org/info/rfc2544>.

[RFC2647] Newman, D., "Benchmarking Terminology for Firewall Performance", RFC 2647, DOI 10.17487/RFC2647, August 1999, <http://www.rfc-editor.org/info/rfc2647>.

[RFC2679] Almes, G., Kalidindi, S., and M. Zekauskas, "A One-way Delay Metric for IPPM", RFC 2679, DOI 10.17487/RFC2679, September 1999, <http://www.rfc-editor.org/info/rfc2679>.

[RFC2680] Almes, G., Kalidindi, S., and M. Zekauskas, "A One-way Packet Loss Metric for IPPM", RFC 2680, DOI 10.17487/RFC2680, September 1999, <http://www.rfc-editor.org/info/rfc2680>.

[RFC5481] Morton, A. and B. Claise, "Packet Delay Variation Applicability Statement", RFC 5481, DOI 10.17487/RFC5481, March 2009, <http://www.rfc-editor.org/info/rfc5481>.

[RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <http://www.rfc-editor.org/info/rfc7567>.

17.2. Informative References

[ANEL2014] Anelli, P., Diana, R., and E. Lochin, "FavorQueue: a Parameterless Active Queue Management to Improve TCP Traffic Performance", Computer Networks vol. 60, 2014.

[BB2011] "BufferBloat: what’s wrong with the internet?", ACM Queue vol. 9, 2011.

[FENG2002] Feng, W., Shin, K., Kandlur, D., and D. Saha, "The BLUE active queue management algorithms", IEEE Trans. Netw. , 2002.

[FLOY1993] Floyd, S. and V. Jacobson, "Random Early Detection (RED) Gateways for Congestion Avoidance", IEEE Trans. Netw. , 1993.

[GONG2014] Gong, Y., Rossi, D., Testa, C., Valenti, S., and D. Taht, "Fighting the bufferbloat: on the coexistence of AQM and low priority congestion control", Computer Networks, Elsevier, 2014, 60, pp.115 - 128 , 2014.

[HASS2008] Hassayoun, S. and D. Ros, "Loss Synchronization and Router Buffer Sizing with High-Speed Versions of TCP", IEEE INFOCOM Workshops , 2008.

[HOEI2015] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "FlowQueue-Codel", IETF (Work-in-Progress) , January 2015.

[HOLLO2001]
Hollot, C., Misra, V., Towsley, V., and W. Gong, "On Designing Improved Controller for AQM Routers Supporting TCP Flows", IEEE Infocom , 2001.

[I-D.ietf-aqm-codel]
Nichols, K., Jacobson, V., McGregor, A., and J. Iyengar, "Controlled Delay Active Queue Management", draft-ietf-aqm-codel-04 (work in progress), June 2016.

[I-D.ietf-aqm-pie]
Pan, R., Natarajan, P., Baker, F., and G. White, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem", draft-ietf-aqm-pie-08 (work in progress), June 2016.

[I-D.ietf-tcpm-cubic]
Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", draft-ietf-tcpm-cubic-01 (work in progress), January 2016.

[I-D.irtf-iccrg-tcpeval]
Hayes, D., Ros, D., Andrew, L., and S. Floyd, "Common TCP Evaluation Suite", draft-irtf-iccrg-tcpeval-01 (work in progress), July 2014.

[JAY2006] Jay, P., Fu, Q., and G. Armitage, "A preliminary analysis of loss synchronisation between concurrent TCP flows", Australian Telecommunication Networks and Application Conference (ATNAC) , 2006.

[MORR2000]
Morris, R., "Scalable TCP congestion control", IEEE INFOCOM , 2000.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[RFC2488] Allman, M., Glover, D., and L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanisms", BCP 28, RFC 2488, DOI 10.17487/RFC2488, January 1999, <http://www.rfc-editor.org/info/rfc2488>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <http://www.rfc-editor.org/info/rfc3168>.

[RFC3611] Friedman, T., Ed., Caceres, R., Ed., and A. Clark, Ed., "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, DOI 10.17487/RFC3611, November 2003, <http://www.rfc-editor.org/info/rfc3611>.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <http://www.rfc-editor.org/info/rfc5348>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <http://www.rfc-editor.org/info/rfc5681>.

[RFC6297] Welzl, M. and D. Ros, "A Survey of Lower-than-Best-Effort Transport Protocols", RFC 6297, DOI 10.17487/RFC6297, June 2011, <http://www.rfc-editor.org/info/rfc6297>.

[RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <http://www.rfc-editor.org/info/rfc6817>.

[RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", RFC 7141, 2014.

[TRAN2014]
Trang, S., Kuhn, N., Lochin, E., Baudoin, C., Dubois, E., and P. Gelard, "On The Existence Of Optimal LEDBAT Parameters", IEEE ICC 2014 - Communication QoS, Reliability and Modeling Symposium , 2014.

[WELZ2015]
Welzl, M. and G. Fairhurst, "The Benefits to Applications of using Explicit Congestion Notification (ECN)", IETF (Work-in-Progress) , June 2015.

[WINS2014]
Winstein, K., "Transport Architectures for an Evolving Internet", PhD thesis, Massachusetts Institute of Technology , 2014.

Authors' Addresses

Nicolas Kuhn (editor)
CNES, Telecom Bretagne
18 avenue Edouard Belin
Toulouse 31400
France

Phone: +33 5 61 27 32 13
Email: nicolas.kuhn@cnes.fr

Preethi Natarajan (editor)
Cisco Systems
510 McCarthy Blvd
Milpitas, California
United States

Email: prenatar@cisco.com

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

David Ros
Simula Research Laboratory AS
P.O. Box 134
Lysaker, 1325
Norway

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