Insights from Curvy Random Early Detection (RED)

Bob Briscoe*

08 Jun 2015

Abstract

Active queue management (AQM) drops packets early in the growth of a queue, to prevent a capacity-seeking sender (e.g. TCP) from keeping the buffer full. An AQM can mark instead of dropping packets if they indicate support for explicit congestion notification (ECN [RFB01]). Two modern AQMs (PIE [PNB+15] and CoDel [NJ12]) are designed to keep queuing delay to a target by dropping packets as load varies.

This memo uses Curvy RED and an idealised but sufficient model of TCP traffic to explain why attempting to keep delay constant is a bad idea, because it requires excessively high drop at high loads. This high drop itself takes over from queuing delay as the dominant cause of delay, particularly for short flows. A link is better able to preserve reasonable performance at high load if the delay target is softened into a curve rather than a hard cap.

Another surprising corollary of this analysis concerns cases where a bottleneck is highly aggregated. Although aggregation reduces queue variation, if the target queuing delay of the AQM at that bottleneck is reduced to take advantage of this aggregation, TCP will still increase the loss level because of the reduction in round trip time. The only way to resolve this dilemma is to overprovision (a formula is provided).

Nonetheless, for traffic with ECN enabled, there is no harm in an AQM holding queuing delay constant or configuring an AQM to take advantage of any reduced delay due to aggregation without over-provisioning. A corollary is that the dropping and marking behaviours of an AQM should be different. Recently, the requirement of the ECN standard [RFB01] that ECN must be treated the same as drop has been questioned. The insight that the goals of an AQM for drop and for ECN should be different proves that this doubt is justified.

1 Curvy RED

Curvy RED is an active queue management (AQM) algorithm that is both a simplification and a a generalisation of Random Early Detection (RED [FJ93]). Two examples are shown in Figure 1 and Figure 2 shows a close-up of their normal operating regions.

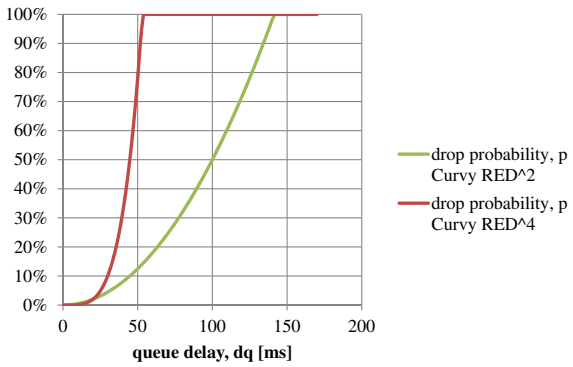

Figure 1: Two Example Curvy RED algorithms

The drop probability, p, of a Curvy RED AQM is:

$$p = \left(\frac{d_q}{D_q}\right)^u, \tag{1}$$

where d_q is averaged queue delay and:

u the exponent (cUrviness) of the AQM;

D_q the slope of the AQM, i.e. d_q where p hits 100%.

The queuing delay used by Curvy RED is averaged, but averaging is not relevant to this memo, which is only concerned with insights from steady-state conditions.

The slopes D_q of the curves in Figure 1 are arranged so that they both pass through (20 ms, 2%), which we call the design point. All the curves in Figure 2 and Figure 3 are arranged to pass through this same operating point, which makes them comparable over the operating region of about 0–40 ms either side of this point, shown in Figure 2.

In the following, the dependence of both queuing delay and drop on load will be derived, assuming the load consists solely of equal TCP Reno flows.

The load on a link is proportional to the number of simultaneous flows being transmitted, n. If the

*ietf@bobbriscoe.net, BT Research & Technology, B54/77, Adastral Park, Martlesham Heath, Ipswich, IP5 3RE, UK

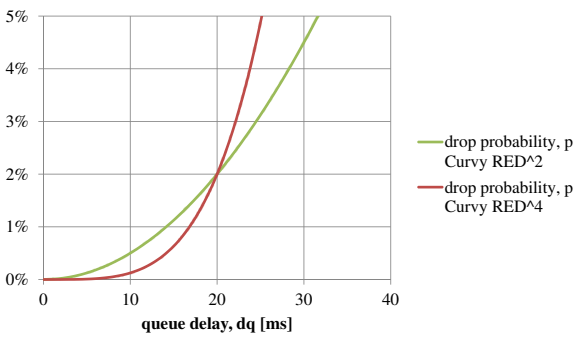

Figure 2: Usual Operating Region of The Same Two Example Curvy RED AQM algorithms

capacity of the link (which may vary) is X and the bit-rate of each flow is x, then:

$$n = \frac{X}{x}. \tag{2}$$

The rate of a TCP flow depends on round trip delay, d_R and drop probability, p. An accurate formula has been derived, but the simplest model [MSMO97] will suffice for insight purposes:

$$x = \frac{s}{d_R}\sqrt{\frac{3}{2p}}, \tag{3}$$

where s is the average packet size.

The round trip time, d_R consists of the base RTT D_R plus queuing delay d_q, such that:

$$d_R = D_R + d_q \tag{4}$$

Substituting Equation 3 Equation 4 in Equation 2:

$$\begin{aligned} n &= \frac{X(D_R + d_q)}{s}\sqrt{\frac{2p}{3}} \\ &= \frac{D_R + d_q}{KD_s}\sqrt{p}, \end{aligned} \tag{5}$$

where constant $K = \sqrt{3/2}$ and $D_s = s/X$, which is the serialisation delay for an average sized packet, which is constant while capacity is constant.

By substituting from Equation 1 into Equation 5, the number of flows can be given as a function of either queueing delay d_q or loss probability, p:

$$n = \frac{(D_R + d_q)}{KD_s}\left(\frac{d_q}{D_q}\right)^{u/2} \tag{6}$$

$$n = \frac{(D_R + D_q.p^{1/u})p^{1/2}}{KD_s} \tag{7}$$

Equation 6 & Equation 6 are plotted against normalised load in Figure 3 for an example set of

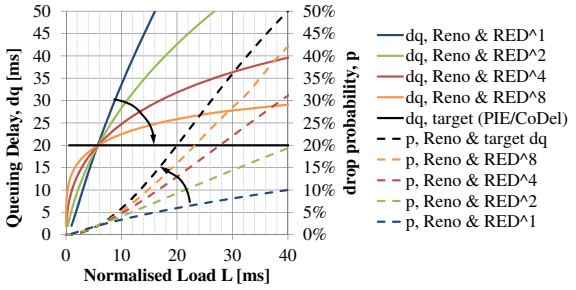

Figure 3: Dilemma between Two Impairments: Delay and Loss against Normalised Load; for a Set of Curvy RED Algorithms with Increasing Curviness

Curvy RED algorithms with curviness parameters $u = 1, 2, 4, 8, \infty$, with base RTT $D_R = 20\,\text{ms}$. $u = \infty$ represents an algorithm like PIE or CoDel that attempts to clamp queuing delay to a target value whatever the load. Normalised load L is used as the horizontal axis, because it is proportional to the number of flows but scales with link capacity and packet size, by the following relationship:

$$L = KD_s n, \tag{8}$$

where KD_s is a constant, at least as long as link capacity and average packet size is constant.

It can be seen that as the curviness parameter is increased, the AQM pushes harder against growth of queuing delay as load increases. However, given TCP is utilising the same capacity, it has to cause more loss (right axis) if it cannot cause more queuing delay (left axis).

In the extreme, infinite curviness represents the intent of AQMs such as CoDel and PIE that aim to clamp queueing delay to a target value (the black horizontal straight line). As a consequence, the TCP flows force loss to rise more quickly with load (the black dashed line).

Normalised load is plotted on the horizontal axis to make the plots independent of capacity or average packet size. Normalised load has units of time and is defined as $L = nKD_s$, with D_s the serialisation delay of an average packet.

As an example, if link capacity $X = 40\,\text{Mb/s}$ and average packet size $s = 820\,B$, then for one Reno flow, $L = 820 * 8 * \sqrt{3/2}/40M = 0.20\,\text{ms}$. Thus, for these example values, $L = 1$ on the horizontal axis represents $1/0.20 = 5$ flows. For a $10\times$ faster link of $400\,\text{Mb/s}$, $L = 1$ on the horizontal axis would represent $10\times$ more, i.e. 50 flows.

2 Invariance with Capacity

2.1 Does Flow Aggregation Increase or Decrease the Queue?

There used to be a rule of thumb that a router buffer should be sized (in bytes) for the bandwidth delay product. Then, in 2004 Appenzeller et al. [AKM04, GM06] pointed out that for large aggregates BDP/$\sqrt{n}$ would be sufficient. This applies only if TCP's sawteeth are desynchronized so that the variance of all the sawteeth does not grow linearly with the number of flows, but instead with the square root (by the Central Limit Theorem).

A link for 100× more flows will typically be sized with 100× more capacity, so the queue will drain 100× faster. Therefore, if the buffer is sized for BDP/$\sqrt{n}$, queuing delay will be $\sqrt{n}$ × (= 10×) lower.

Note that this square-root law does not apply indefinitely; only for medium levels of aggregation. Once the queue size has reduced to about half a dozen packets, it does not get any smaller with increased aggregation [GM06].

In the previous section we showed that queuing delay *grows* with number of flows. How does this reconcile with the idea that queuing delay *shrinks* as more flows are aggregated?

There is no paradox; the two ideas are reconciled by noting that queuing delay only shrinks when the number of flows grows *and* more capacity is added to accommodate them, which occurs at the timescale of provisioning, not usually at the timescale of queue control.

Therefore, for a specific link with a known capacity, the previous section has explained that an AQM should temporarily accommodate more flows by allowing a little more queuing delay. But, if a certain number of flows is expected permanently, the link has to be sized accordingly.

This is why we have plotted queuing delay and drop against *normalised* load L in Figure 3. Normalised load stays the same if a link is sized proportionate to the expected number of flows. The explanation is as follows.

Our definition of normalised load is $L = nKD_s$. So if the serialisation delay D_s is held constant (implying link capacity is constant), then increasing n increases L. But normally a large permanent increase in n would be accommodated by more link capacity, implying lower serialisation delay D_s. For instance, for 100× more flows, link capacity would be 100× larger, D_s would therefore be 100× smaller, so normalised load would be unchanged.

Note that our definition of normalised load also factors out a change in average packet size. For a given capacity and number of flows, increasing the packet size increases the normalised load. It may be counter-intuitive that the same bit-rate in larger packets will change normalised load. This is an artefact of the window-based design of the TCP Reno algorithm (and most other TCP variants), which behaves as if a link is more congested if it has to reduce the packet rate, even if it only had to because it increased packet sizes and its bit-rate did not change.

2.2 AQM Configuration and Scale

AQM configuration should be invariant with link capacity, but it will need to change in environments where the expected range of RTTs is significantly different. Therefore, AQM configurations in a data centre will be different, not because of high link capacities, but because of shorter RTTs.

The AQM should be configured against a design point, such as $d_q = 20\,\text{ms}, p = 2\%$. The operator will set this design point where delay and loss start to become troublesome for the most sensitive applications (e.g. interactive voice). By setting the AQM to pass through this point, it will ensure a good compromise between delay and loss. Then the most sensitive application will survive at the highest possible load, rather than holding one impairment unnecessarily low so that the other is pushed so high that it breaks the sensitive app.

The curviness of the AQM depends on the dominant congestion control regime (this memo is written assuming TCP Reno is dominant). For a particular dominant congestion control, curviness depends only on the best compromise to resolve the dilemma in Figure 3, and can otherwise be assumed constant.

Further research is needed to understand how best to average the queue length, and how best to configure the averaging parameter.

Having determined an AQM configuration as above, it will be applicable for all link rates as long as expected round trip times are unchanged. If there are too many flows for the capacity, no amount of AQM configuration will help. Capacity needs to be appropriately upgraded. The following formula can be used to determine capacity X given the expected number of flows n and expected average base RTT D_R, where d_q^* and p^* are the values of queuing delay and drop at the design point, as determined above.

$$K\frac{ns}{X} = (D_R + d_q^*)\sqrt{p^*}.$$

Alternatively, it can be used to determine the number of flows n that a particular capacity X can reasonably be expected to support. For convenience, serialisation delay has been broken down into its component parts, $D_s = s/X$.

For highly aggregated links, the queuing delay at the design point can be reduced proportionate to $1/\sqrt{n}$ without compromising utilisation. However, *the dilemma in Figure 3 still applies, so loss probability will increase.* The only way to take advantage of the $1/\sqrt{n}$ reduction in queuing delay due to aggregation without increasing loss is to over-provide capacity, so that n/X reduces as much as $(D_R + d_q^*)$ reduces. Specifically, if the design point for queuing delay at low aggregation, d_q^* is reduced to $d_q^*/\sqrt{n}$ at high aggregation, then the over-provisioning factor should be:

$$\frac{X'}{X} = \frac{(D_R + d_q^*)}{(D_R + d_q^*/\sqrt{n})}.$$

For example, if average base RTT, $D_R = 20ms$ a link for 100× more flows could have the design point for queuing delay reduced from 20 ms to 2 ms while keeping the loss design point constant at 2%. But only if aggregated capacity is over-provided by $(20+20)/(20+2) = 1.8$, relative to a link designed for 100× fewer flows.

3 Conclusions & Further Work

Although Curvy RED seems to be a useful AQM, we are not necessarily recommending it here. We are merely using the concept of curviness to draw insights.

The curviness parameter of Curvy RED can be considered to represent the operator's policy for the tradeoff between delay and loss whenever load exceeds the intended design point. Conversely, PIE and RED embody a hard-coded policy, which dictates that holding down delay is paramount, at the expense of more loss.

Given losses from short interactive flows (e.g. Web) cause considerable delay to session completion, sacrificing loss for delay is unlikely to be the optimal policy to minimise delay. Also, it may lead real-time applications such as conversational video and VoIP to degrade or fail sooner as load increases. Allowing some additional flex in queueing delay with consequently less increase in loss is likely to give more favourable performance for a mix of Internet applications.

In addition, when load is below the design point, both PIE and CoDel tend to allow the queue to grow towards the target delay. Whereas Curvy RED gives lower queuing delay when load is light.

We intend to conduct experiments to give advice on a compromise level of curviness that best protects a range of delay-sensitive and loss-sensitive applications during high load.

Another insight that can be drawn from this analysis is that the dilemma in Figure 3 disappears with ECN. For ECN, the AQM can mark packets without introducing any impairment. There is therefore no downside to clamping down queuing delay for ECN packets. This further supports the idea that ECN should not be treated equivalently to drop.

The analysis also shows that, once a design point defining an acceptable queuing delay and loss has been defined, the same configuration can be used for the AQM at any link rate, but only for a similar RTT environment. A shallower queuing delay configuration can be used at high aggregation, but only if the higher loss is acceptable, or if the capacity is suitably over-provisioned. Formulae for all these configuration trade-offs have been provided.

References

[AKM04] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing Router Buffers. *Proc. ACM SIGCOMM'04, Computer Communication Review*, 34(4), September 2004.

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

[GM06] Yashar Ganjali and Nick McKeown. Update on Buffer Sizing in Internet Routers. *ACM SIGCOMM Computer Communication Review*, 36, October 2006.

[MSMO97] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithms. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, 1997.

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

[PNB+15] Rong Pan, Preethi Natarajan, Fred Baker, Bill Ver Steeg, Mythili Prabhu, Chiara Piglione, Vijay Subramanian, and Greg White. PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem. Internet Draft draft-ietf-aqm-pie-01, Internet Engineering Task Force, March 2015. (Work in progress).

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

Document history

Version	Date	Author	Details of change
Draft 00A	19 May 2015	Bob Briscoe	First Draft
Draft 00B	23 May 2015	Bob Briscoe	Explained normalised load
Draft 00C	23 May 2015	Bob Briscoe	Changed notation & added Aggregation section
Draft 00D	08 Jun 2015	Bob Briscoe	Corrected explanation about packet size, plus minor corrections