Friendliness between AIMD Algorithms

Bob Briscoe* Olga Albisser†

16 Mar 2023

Abstract

This paper aims to provide a robust grounding for the additive increase factor used in the 'TCP-Friendly' mode of the CUBIC congestion control algorithm.

1 Introduction

The first IETF RFC to define the CUBIC [RXH+18] congestion control algorithm (CCA) was based on the original paper introducing CUBIC [HRX08]. For 'TCP-friendly' mode, both draw on an equation in an ACIRI technical report [FHP00] when they specify the additive increase factor. The derivation of the equation in that technical report has been called into question. So this report aims to more rigorously derive and evaluate the required additive increase factor. The ACIRI report assumes a deterministic dropping (or ECN-marking) algorithm at the bottleneck, which limits its applicability. Also it attempts to validate the theoretical formula empirically by simulation using a RED gateway at the bottleneck, but it results in flow rates that are inexplicably different (by a factor of more than 2×) when they should be the same.

Below, an equation for the additive increase factor is derived without the assumption of deterministic dropping. Instead it is assumed that drops are synchronized between flows, which is typically the case for tail-drop queues. The resulting equation turns out to be the same as that in the technical report [FHP00]. However, the derivation here is different. It has a straightforward geometric interpretation and it relies on fewer assumptions and no approximations; it considers variation of the RTT explicitly and it does not use loss probability at all.

The present paper is not intended to be ambitious or insightful, just pedestrian and rigorous. [BB01] is recommended for insight into the wider set of 'TCP-friendly' algorithms, but it does not go into depth on the simple linear cases analysed here.

*research@bobbriscoe.net,
†olga@albisser.org

2 Terminology

Nowadays, CUBIC's TCP-friendly mode is more accurately known as Reno-friendly mode, given its flow rate is intended to match that of the Reno CCA, and given that it is irrelevant which wire protocol is used, whether TCP, QUIC, SCTP, etc. The term C-Reno will be used for CUBIC in Reno-friendly mode.

This paper uses the variables defined below:

a : Additive increase factor;
b : Multiplicative decrease factor;
j : Round index;
J : Rounds per sawtooth cycle;
$R(j)$: Round trip time (RTT);
$W(j)$: Congestion window;
$\widehat{W}$: Maximum W;
$r(j)$: Packet rate;
X_r : Reno variant of any variable X;
X_c : C-Reno variant of any variable X.

3 AIMD-Friendliness

Consider two types of Additive Increase Multiplicative Decrease (AIMD) flow with parameters (a_r, b_r) and (a_c, b_c) competing at a bottleneck, under the following assumptions:

- The buffer is large enough not to drain completely, even if all flows reduce simultaneously (this assumption is relaxed later).
- All other factors of all the flows, particularly packet size and base RTT, are equal. When flows sharing the same bottleneck queue all have the same base RTT, they all have the same RTT, $R(j)$, at every stage, j, of their sawtooth cycles.
- The bandwidth-delay product (BDP) is low enough for all flows to remain in their AIMD mode throughout the cycle.
- All the flows have run long enough to converge to a steady state.
- All flows only respond to the presence of loss, not its extent.

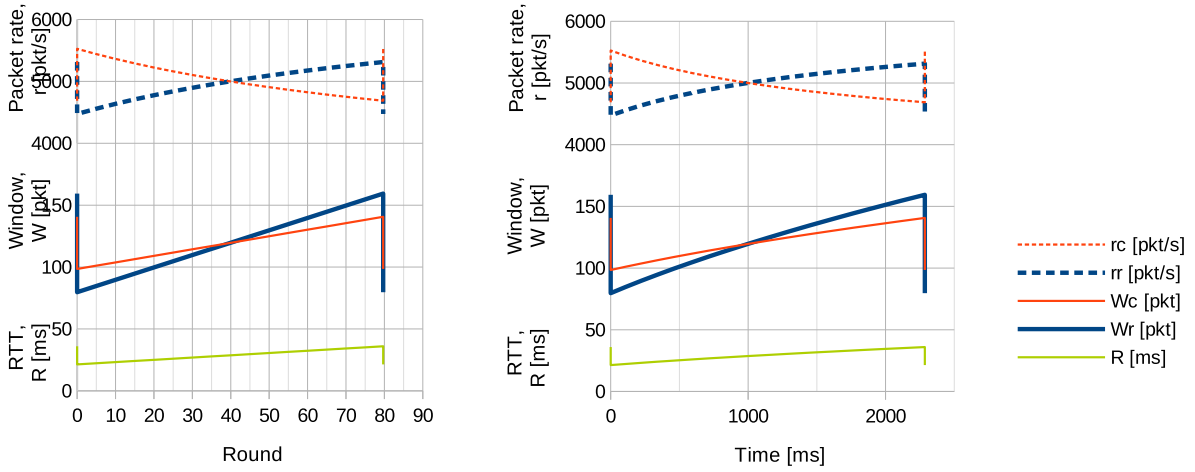

Figure 1: One synchronized sawtooth cycle of C-Reno and Reno plotted wrt. round trips (left) and wrt. time (right)

3.1 Synchronized (Tail-Drop) Case

For this case, it is additionally assumed that:

- All the flows are synchronized so that, whenever one flow experiences loss the others do too;
- All flows experience at least one loss at each congestion event (relaxation of this assumption is discussed later);

Steady state: For each flow, the additive increase of a cycle balances with its multiplicative decrease from the max, $\check{W}/b$, to the min, $\check{W}$.

$$\begin{aligned} a_r J &= \widehat{W}_r - b_r \widehat{W}_r \\ &= \widehat{W}_r (1 - b_r) \qquad (1) \\ a_c J &= \widehat{W}_c (1 - b_c) \qquad (2) \end{aligned}$$

Flow rate equality: Given the parameters a_r, b_r, b_c the aim is to derive a_c such that each flow's average rate is the same. This is equivalent to each flow transferring the same number of packets over a cycle.

As a cycle progresses, the RTT grows. So to derive the number of packets transferred over a cycle, the packet rate has to be weighted by the RTT in each cycle before being summed:

$$\begin{aligned} \sum_{j=0}^{J-1} r_c(j) R(j) &= \sum_{j=0}^{J-1} r_r(j) R(j) \\ \sum_{j=0}^{J-1} W_c(j) &= \sum_{j=0}^{J-1} W_r(j) \qquad (3) \\ \sum_{j=0}^{J-1} b_c \widehat{W}_c + a_c j &= \sum_{j=0}^{J-1} b_r \widehat{W}_r + a_r j \\ J b_c \widehat{W}_c + \frac{J^2 a_c}{2} &= J b_r \widehat{W}_r + \frac{J^2 a_r}{2} \end{aligned}$$

Dividing through by J and substituting from Equation 1 & Equation 2:

$$\begin{aligned} \widehat{W}_c \left(b_c + \frac{(1 - b_c)}{2} \right) &= \widehat{W}_r \left(b_r + \frac{(1 - b_r)}{2} \right) \\ \frac{\widehat{W}_c}{\widehat{W}_r} &= \frac{(1 + b_r)}{(1 + b_c)} \qquad (4) \end{aligned}$$

Returning to the steady state equations, we can divide Equation 2 by Equation 1, then substitute from Equation 4:

$$\begin{aligned} \frac{a_c}{a_r} &= \frac{\widehat{W}_c}{\widehat{W}_r} \frac{(1 - b_c)}{(1 - b_r)} \\ &= \frac{(1 - b_c)}{(1 + b_c)} \frac{(1 + b_r)}{(1 - b_r)} \qquad (5) \end{aligned}$$

Plugging in Reno's AIMD factors, $a_r = 1, b_r = 1/2$:

$$\boxed{a_c = \frac{3(1 - b_c)}{(1 + b_c)}} \qquad (6)$$

And plugging in the multiplicative decrease factor of C-Reno recommended in [RXH+18], $b_c = 0.7$:

$$\begin{aligned} a_c &= 9/17 \\ &\approx 0.53. \end{aligned}$$

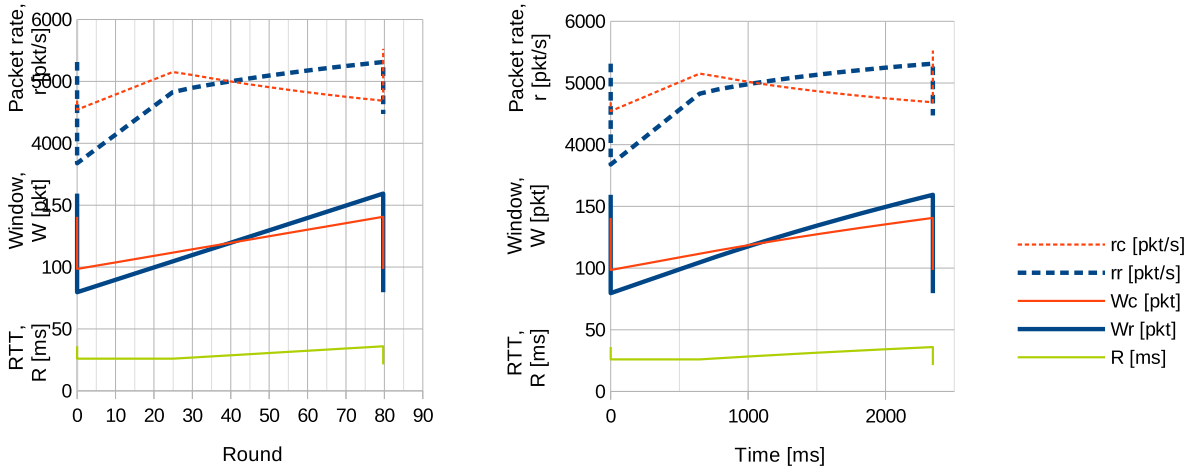

Figure 2: Similar synchronized sawtooth cycle of C-Reno and Reno to Figure 1, but with a 11 ms buffer that is too shallow to accommodate both sawteeth at once.

3.1.1 Geometric interpretation

Figure 1 shows one flow each of C-Reno and Reno competing over one synchronized sawtooth cycle. Superficially, the whole derivation of a_c above can be derived from simple triangle geometry, by drawing congestion window sawteeth that increase linearly wrt. round trips (mid-left) then setting the mid-points of the two ramps to the same height. Then Equation 4 gives the ratio between the heights of the bases of the triangles, and Equation 5 gives the ratio of the heights of the triangles themselves.

However, it is not enough to merely assert that the average heights of these sawteeth are equal, as [FHP00] does.[1] Strictly, it is necessary to start from the goal of equal flow rates averaged over time, as the above analysis does. Given RTT grows throughout the cycle, the plots of flow-rate against time (top right in Figure 1) stretch out more to the right, forming concave curves. It is not at all obvious how to equate the averages of these two curves until they are weighted by round trip duration, which transforms them into the linear plots of window wrt. rounds (mid-left), as in Equation 3.

It is also interesting to note from Figure 1 that C-Reno's packet rate *decreases* as its window increases over the sawtooth. This is because the competing Reno flow causes the RTT to grow faster than would be the case with only C-Reno flows.

If the buffer is not deep enough to hold all the synchronized sawteeth, it will be empty during the early part of the sawteeth. Then both flows will under-perform during the early part of the cycle when C-Reno would have achieved its highest packet rate and Reno would have achieved its lowest, as shown in Figure 2. Nonetheless, the rate of both flows reduces proportionately, so the ratio between their flow rates remains unaltered.

3.1.2 Synchronized losses?

We will now question the assumption that each flow always catches at least one loss at each congestion event. We still consider two flows with equal base RTTs: 1 Reno and 1 C-Reno. Between responses to losses, the queue grows inexorably by $(a_r + a_c \approx 1.53)$ pkt/RTT. Every time the buffer fills, one packet has to be dropped, but the queue continues to grow by 1.53 segments during the next round trip (until the resulting response reaches the queue). So it is likely that another packet will have to be discarded within the same RTT as the first.

The likelihood that a particular flow catches any one of the losses depends on its packet rate relative to the other.[2] That is,

$$\frac{p_r}{p_c} = \frac{\widehat{r_r}}{\widehat{r_c}}. \tag{7}$$

If the flows both have the same average window (the goal in Figure 1) then, by Equation 4 (or triangle geometry), the ratio between the packet rates of the two flows when they both reach their max is

$$\begin{aligned} \frac{\widehat{r_r}}{\widehat{r_c}} &= \frac{(1+b_c)}{(1+b_r)} \\ &= \frac{1.7}{1.5} \qquad \approx 1.13. \end{aligned} \tag{8}$$

[1] It is unlikely that the additional initial steps given here were implicit but unstated, because a high-level averaging approach was used.

[2] Irrespective of how rapidly the flow's own window grows.

When a loss occurs, from Equation 7 & Equation 8, we can say that:

$$p_r/p_c = 17/15 \quad (9)$$

$$\text{and} \quad p_r + p_c = 1, \quad (10)$$

because one or the other flow was hit with certainty.

Therefore

$$p_r = 17/32 \approx 53\%$$
$$p_c = 15/32 \approx 47\%$$

Then, for example, if there are two losses during a congestion event, the probabilities of each combination of two losses are:

$$p_{rr} = (17/32)^2 \approx 28\%$$
$$p_{rc} \vee p_{cr} = 17 * 15/32^2 + 15 * 17/32^2 \approx 50\%$$
$$p_{cc} = (15/32)^2 \approx 22\%,$$

where p_{ij} is the probability of a loss from flow i then j.

When there are two losses in a round and the same flow is hit twice, it doesn't reduce any more than if it's hit once, but the other flow doesn't reduce at all. So C-Reno is somewhat more likely than Reno to not get hit in some round. In such cases, only the Reno flow would reduce, then the queue would continue to grow by ($a_r + a_c \approx 1.53$) pkt/RTT, so the next cycle would be shorter and the C-Reno flow would be much more likely to be hit when it next filled the buffer — and more likely to be hit twice.

It would be possible to calculate the average rate of each type of flow by calculating the probabilities of each chain of events programmatically. However, such precision is unnecessary. For the case of tail-drop buffers, it will be sufficient to say:

- either that the AI factor of C-Reno should be slightly lower than that derived from Equation 6 to make C-Reno and Reno flow rates more precisely equal;
- or that the average rate of C-Reno flows will be slightly higher in comparison with Reno flows, if Equation 6 is used.

"Slightly" means within 10%, which conservatively accommodates the difference between p_{rr} and p_{cc}.

3.1.3 Empirical Results: Tail Drop

Testbed set-up: A single long-running C-Reno flow with MD factor $b_c = 0.7$ and AI factor $a_c = 1.53$ (the defaults in Linux derived using Equation 6) was tested against a single Reno flow over a tail-drop buffer. The buffer was at the bottleneck of a range of 25 different Ethernet paths with five different link rates and five different RTTs, as shown on the x-axis of Figure 3.[3]

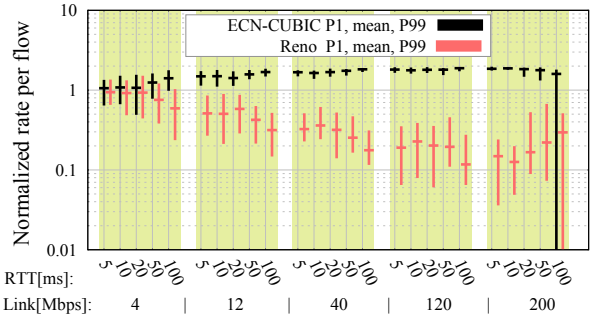

Figure 3: Empirical results comparing the flow rates of CUBIC and Reno competing 1:1 over a tail-drop queue in a 200 ms ('bloated') buffer with MD factor $b_c = 0.7$ and AI factor $a_c = 0.53$ for CUBIC's Reno-friendly algorithm

Two buffer sizing approaches were evaluated: 'bloated' and 'tuned'. In both the buffer was sized dependent on the link rate, but in the 'bloated' case it held 200 ms and in the 'tuned' case only 25 ms.[4] That is, buffer size B [byte] $= C\widehat{R}/8$, where C [b/s] is the link rate serving the buffer and $\widehat{R}$ is either 0.2 s or 0.025 s.

The 'bloated' case represents the outdated rule of thumb where an operator allows for 1 BDP of buffer at the maximum feasible RTT, where 200 ms approximates the RTT in glass fibre around half the earth's circumference and back. The 'tuned' case is tailored to CUBIC's MD factor of 0.7 and to its slow-start behaviour, on the assumption that the base RTT of most Internet traffic is under 50 ms.

In all cases the results are displayed as whisker plots that show 1st %-ile (P1), mean and 99th %-ile (P99) normalized flow rate. Normalized flow rate is defined as the flow rate relative to $1/N$ of the capacity, where N is the total number of flows.

The shaded background of part of the plots indicates those cases where CUBIC is not in it's Reno-

[3] In each run, flow rate measurement was delayed until the flows had converged, then taken every second for 250 s. Percentiles were derived from the resulting 250 measurements. The two types of flow were sent from two sending hosts to two receiving hosts via a bridge and a bottleneck node. All machines were running Linux kernel v5.10.31. All CCAs were configured with default parameters.

[4] The buffer was not sized to the RTT of each flow. Instead, to better match operational practice, it was sized to a worst-case RTT, given the RTT of each flow is hard to measure in the network. Then two different definitions of worst-case are compared.

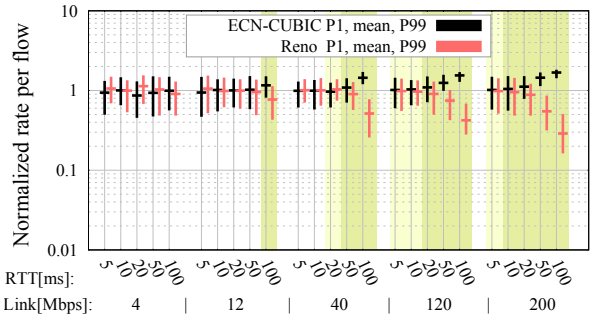

Figure 4: Empirical results with same set up as Figure 3 except buffer tuned to 25 ms

friendly mode. Darker shading represents true CUBIC mode, while lighter shading indicates where additive increase transitions from Reno-friendly to true CUBIC part-way up each saw-tooth. The switch-over RTT is taken from equation (6) in [Bri21] and it is assumed that all sawteeth peak at the tail of the buffer.

Interpretation of Results In the 'bloated' buffer case, Figure 3 shows that none of the CUBIC flows are ever in Reno-friendly mode. This tells us nothing relevant to the present paper, but it shows that CUBIC can significantly outcompete Reno (up to about 15:1 rate ratio) when not in Reno-friendly mode,[5] although at least Reno does not starve at low link rates.

In the 'tuned case', Figure 4 shows that CUBIC is either in or partly in its Reno-friendly mode over a larger part of the scenario space (respectively unshaded or lightly shaded background). In all such cases, the mean normalized flow rate of both flows is close to 1. This validates the model of tail drop behind Equation 6, at least for the case when $b_c = 0.7$.

Outside its Reno-friendly mode, CUBIC increasingly dominates Reno, but that is outside the AIMD-specific focus of the present paper.

3.2 Desynchronized (AQM) Case

For this case, instead of the assumption of synchronization, it is assumed that:

- As the queue grows, Active Queue Management (AQM) at the bottleneck selects single packets to drop or mark, so that the congestion responses of each flow tend not to coincide.[6]

The AQM case is harder to analyse than the synchronized case with tail-drop. Superficially, one could use the transformation from equal average flow rates (wrt. time) into equal window (wrt. rounds). However, there is no guarantee that the number of rounds per cycle, J is the same in each case.

If we assumed it was, we would end up with Equation 6 for C-Reno's additive increase factor a_c. Then, as shown in Figure 5, the phasing between the sawteeth would evolve so that the queue reached roughly the same depth before each reduction, i.e. the tips of the RTT sawteeth will all align at roughly the same level—the operating point of the AQM.

However, although Figure 5 shows the sawtooth reductions alternating Reno – C-Reno – Reno, this need not be the case. In the cycle after 400 ms in the top-right plot, the ratio between C-Reno's and Reno's packet rates is about 51:49. So it is nearly as likely that the AQM will hit a Reno packet as a C-Reno packet, causing Reno to reduce twice in a row. If the AQM did hit C-Reno around 400 ms, at around 600 ms the ratio would be about 42:58, making Reno more likely to be hit.

The probability of each outcome at each stage could be derived programmatically (a Markov chain), but it will suffice to test empirically whether the model in Equation 6 fits the AQM case. We can at least suppose that, if a_c is set as for the synchronized case (Equation 6), an AQM is likely to hit Reno somewhat more often than C-Reno.

3.2.1 Empirical Results: PIE AQM

Testbed set-up: To test the AQM case, similar experiments to those in § 3.1.3 were run, but this time with a PIE AQM not a tail-drop buffer.[7]

Also, various combinations of numbers of each flow type were tested, as shown along the x-axis of Figure 7. For instance, in the bottom plot of Figure 7, A2-B8 means 2 ECN-CUBIC flows vs. 8 (non-ECN) Reno flows. The top plot of the same figure shows the results of equivalent control experiments where the Reno flows were replaced by non-ECN CUBIC flows. This served to check that Reno and C-Reno flows were equally agressive against the same set of ECN C-Reno flows.

the ECN-capability of the C-Reno flows was not altering their aggressiveness.

[5] These experiments ensured that both flows remained in steady state, but the argument for CUBIC rests on Reno being more sensitive to disturbances that knock it out of its steady state [HLRX07].

[6] In desynchronized cases, the RTT varies less than in synchronized cases—on average the amplitude is respectively $1/\sqrt{n}$ vs. n times that of a single flow, where n is the number of flows [AKM04]. Therefore, the first assumption listed in § 3 (that the buffer is large enough not to drain completely) is much more likely to hold in desynchronized cases.

[7] On a v5.10.31 Linux kernel with default parameters.

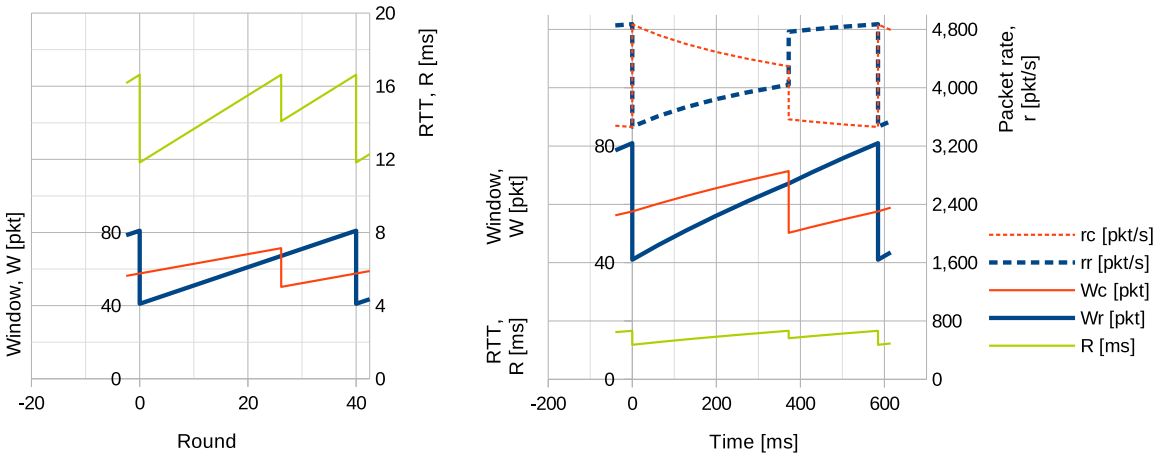

Figure 5: One desynchronized sawtooth cycle of C-Reno and Reno plotted wrt. round trips (left) and wrt. time (right)

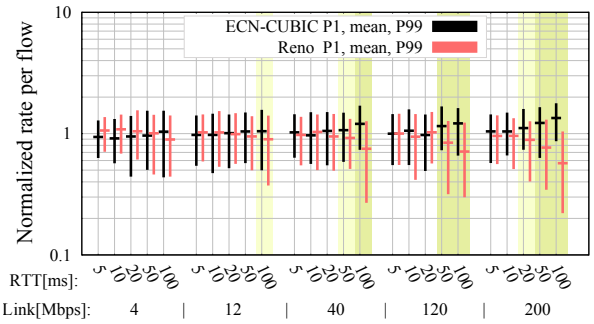

Figure 6: Empirical results with set up similar to Figure 3 except over a PIE AQM

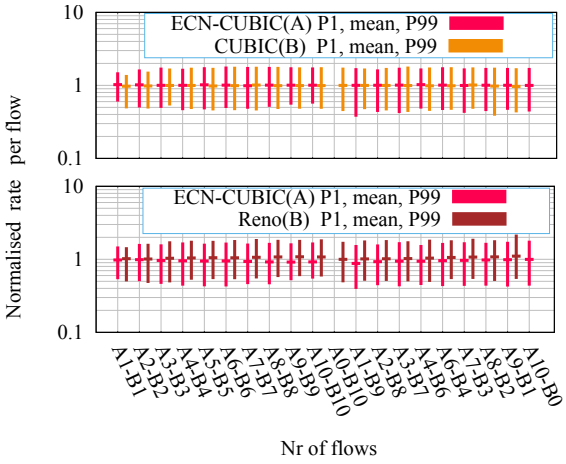

Figure 7: Average flow rates of different numbers of long-running CUBIC and Reno flows. AQM: PIE; link capacity: 40 Mb/s; Base RTT: 10 ms.

All the different combinations of flow numbers were run over a path with base RTT 10 ms over a 40 Mb/s bottleneck, which was chosen to keep CUBIC in its Reno-friendly mode.

See § 3.1.3 for the meanings of the whiskers on the plots, the 'normalized flow rate' metric and the shaded regions of the background.[8]

Interpretation of Results: In the 1 CUBIC vs. 1 Reno tests (Figure 6) over a PIE AQM, it can be seen that C-Reno and Reno share the bandwidth roughly equally. However, as BDP increases, CUBIC can be seen to start taking a greater share of capacity, as it increasingly operates beyond its Reno-friendly mode.[9]

Thus, in practice, the AI factor derived from the tail drop model keeps the flow rates roughly equal whether the bottleneck is tail drop or a PIE AQM.

In the multi-flow tests over the PIE AQM (Figure 7), it can be seen that C-Reno gives roughly the same rate as Reno over the full range of tests. Indeed, if anything, C-Reno seems to be slightly less aggressive than Reno. However, the discrepancy is hardly visible, and more than an order of magnitude smaller than the range of the ratio's natural variability between its P1 and P99.

[8] With an AQM, it would seem less straightforward to determine whether CUBIC will have started to transition from C-Reno to pure CUBIC mode at a particular base RTT, because it is hard to estimate where the tips of the sawteeth sit in relation to the 15 ms AQM target (see § 3.3 of [Bri21] for background). However, at the selection of base RTTs shown, it turned out that using any feasible estimate didn't change whether CUBIC was in transition. In other words, the mode CUBIC is in is fairly insensitive to this estimate.

[9] Further discussion of pure CUBIC mode is outside the scope of the present paper, which focuses on AIMD.

4 Conclusion

This report provides:

- a formula (Equation 6) for the additive increase parameter of an AIMD algorithm as a function of its chosen multiplicative decrease factor that should maintain an equal flow rate with another AIMD flow, specifically a Reno flow.
- a derivation of the formula that relies on fewer assumptions and is more rigorous than that in Floyd *et al* [FHP00]. It applies to tail drop buffers whereas that in Floyd *et al* relied on an AQM with deterministic marking. Nonetheless the formula turns out to be the same.
- a testbed validation of the formula (at least for the case where the MD factor is 0.7) over a range of 25 different path characteristics (5 rates and 5 base RTTs) with a tail-drop bottleneck buffer.
- a testbed evaluation of the formula over the same range of scenarios but with a PIE AQM at the bottleneck. This shows that the AI factor works correctly over a PIE AQM, even though it was derived assuming tail-drop.

These result show that, with the widely deployed decrease factor of $b_c = 0.7$, the Reno-Friendly mode in CUBIC is sufficiently well modelled by Equation 6 that the Additive Increase factor it produces ($a_c = 0.53$) ensures that TCP CUBIC competes roughly equally with Reno across its intended operating range, whether with a tail-drop queue or a single-queue AQM (PIE) at the bottleneck.

References

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

[BB01] Deepak Bansal and Hari Balakrishnan. Binomial Congestion Control Algorithms. In *Proc. IEEE Conference on Computer Communications (Infocom'01)*, pages 631–640. IEEE, April 2001.

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

[FHP00] Sally Floyd, Mark Handley, and Jitendra Padhye. A Comparison of Equation-Based and AIMD Congestion Control. Technical report, ACIRI, May 2000.

[HLRX07] Sangtae Ha, Long Le, Injong Rhee, and Lisong Xu. Impact of background traffic on performance of high-speed TCP variant protocols. *Computer Networks*, 51(7):1748–1762, 2007. Protocols for Fast, Long-Distance Networks.

[HRX08] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Operating Systems Review*, 42(5):64–74, July 2008.

[RXH+18] I. Rhee, L. Xu, S. Ha, A. Zimmerman, L. Eggert, and R. Scheffenegger. CUBIC for Fast Long-Distance Networks. Request for Comments RFC8312, RFC Editor, August 2018.

Document history

Version	Date	Author	Details of change
00A	29 Sep 2021	Bob Briscoe	First draft.
00B	01 Oct 2021	Bob Briscoe	Added geometric interpretation and deterministic case.
00C	03 Aug 2022	Bob Briscoe	Added shallow buffer case, empirical results over PIE and discussion of the applicability of the synchronized loss model. Added Acks section. Altered analysis to use max window not min.
00D	08 Aug 2022	Bob Briscoe	Added explicit step at Equation 7 that was previously implicit.
00E	16 Mar 2023	Bob Briscoe	Added text around results in § 3.1.3. Improvements throughout. Expanded conclusions.

