Notes on CoDel with Bursty MACs: "It's the RTT, stupid"[1]

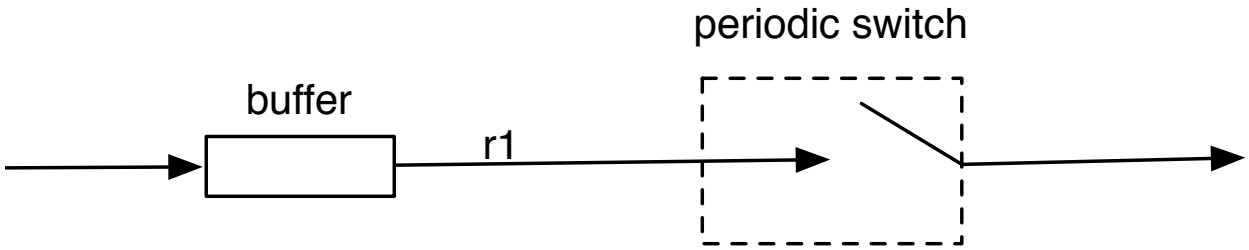
Kathleen Nichols
Pollere, Inc.
nichols@pollere.net

Overview

There has been confusion about the role of CoDel's *target* and *interval*, particularly for "bursty" MAC layers. Bursty MACs are those in which packet transmission does not happen smoothly but is governed by a stop-and-wait or request-and-grant structure that keeps the link idle and packets waiting in the buffer for periods of time that are long compared to the time to transmit a packet. Many experimenters believe the value of *target* needs to be increased independently of *interval* in this case. This note is to help explain why this is *not* so. Target should continue to be computed as 5-10% of interval (see draft-ietf-aqm-codel). CoDel "sees" the effective transmission rate over an *interval* and increasing *target* will just lead to longer queue delays. On the other hand, the time spent waiting for transmission by most or all of the packets adds additional delay to the intrinsic path (or physical link) delay, increasing conntection round trip times. It MAY be necessary to increase *interval* by that extra delay if it would otherwise be less than the round trip time. That is, *target* SHOULD NOT be adjusted independently but *interval* MAY need to be adjusted and target MAY be adjusted so that it remains at 5% of *interval*.

Bursty MAC Model

The figure below is an abstraction of a bursty MAC. Assume that the switch transfers *b* bits every s ms and that its output link rate *r1* >> *b*/*s*. When a packet arrives to an empty buffer, it has to wait *s* ms before being sent.



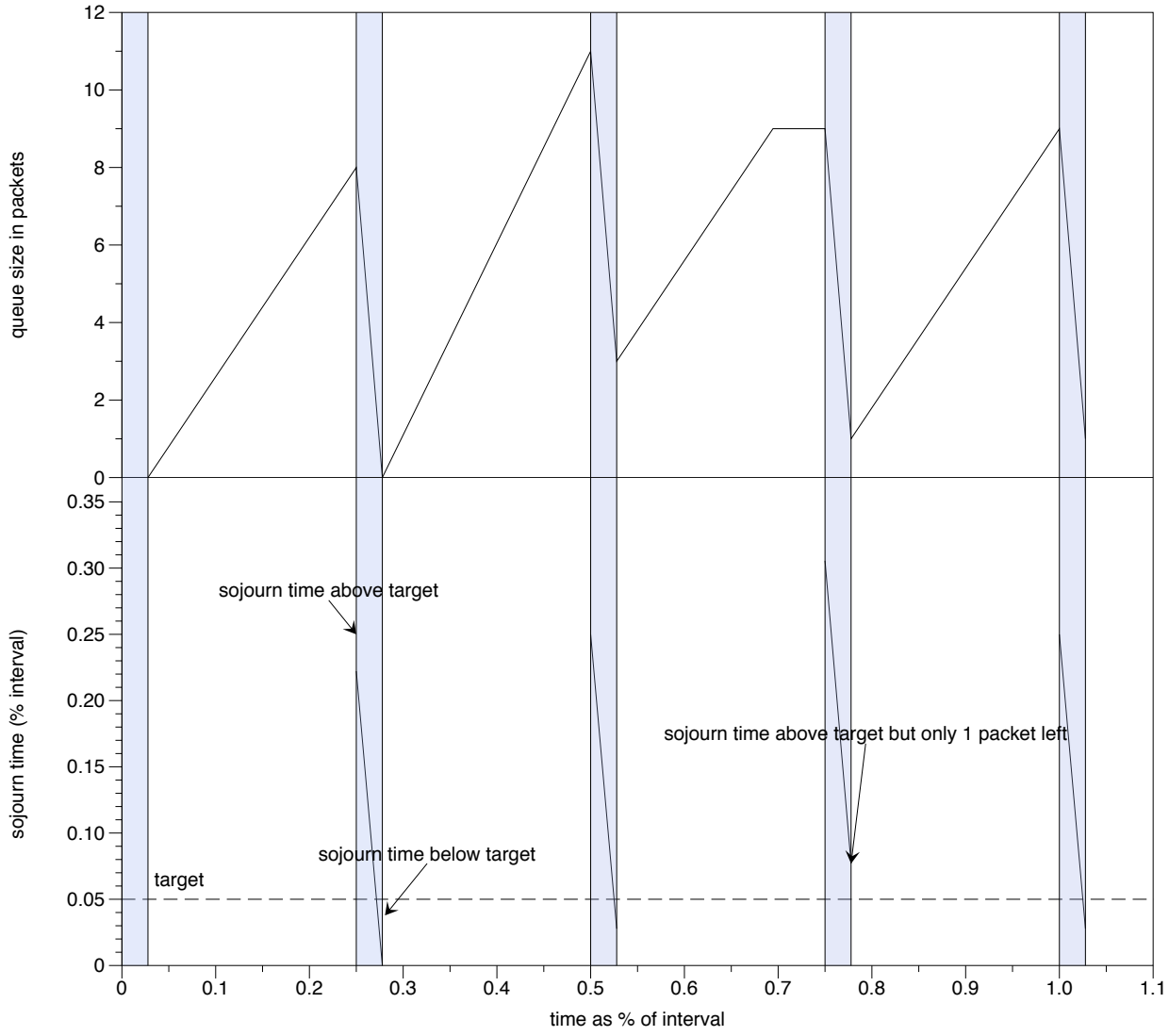
To operate correctly, a CoDel AQM operating on the buffer MUST have an *interval* that exceeds *s* (plus the round trip path delays).

Data Rate Less than b/s

The first case is where data arrives at the buffer at less than *b*/*s*. In this case, CoDel should not drop any packets. This is, in fact, the case, without changing *target*. The graphs below show the queue size of the buffer over time. To illustrate the different possibilities, we chose different rates for each period *s*: in the

[1] with apologies to Stuart Cheshire

first period, *b* packets arrive at a steady rate. In the second period, *b*+2 packets arrive, in the third period *b*-2 packets arrive, in the fourth period, *b* packets arrive. The time scale is in percentage of an *interval*, e.g. 100ms, with target 0.05 of an *interval*. All packets are the same size and *4s=interval*. When the first transmission starts, the first packet has waited *0.25interval* which is above *target* so CoDel record s this as the first_above_time_. The packets behind it arrived smoothly so waited less with the final packet only waiting for its transmission time. This gives a sojourn time below *target,* so first_above_time_ is reset. A similar situation happens on the second transmission time slot. During the third transmission time slot, the sojourn time remains above target, but there is only one packet (<= an MTU) left in the buffer, so first_above_time_ is reset.



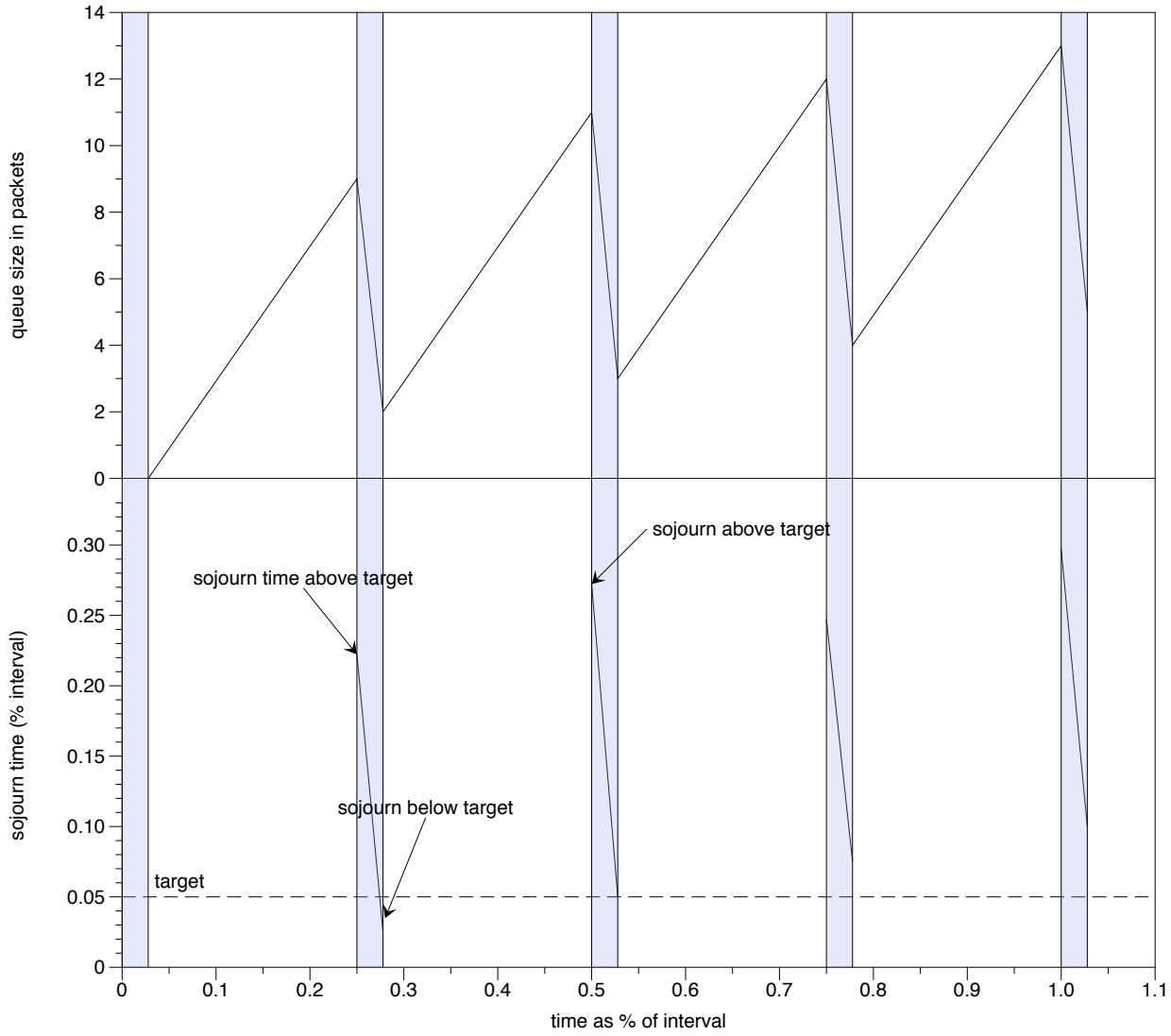
In the next example, the arrival rate is greater than *b/s* with 10 packets arriving every *s* ms and only 9 able to be sent. During the first transmission slot, the sojourn time of the first packet is above target so first_above_time_ is set. When the ninth packet is sent, there are now two packets in the buffer, but the sojourn time is below *target*, so first_above_time is reset. At the beginning of the second transmission slot, the sojourn time is above target, so the first_above_time_ is set to that time. The last packet sent during that transmission slot has a sojourn time equal to the target and there are three packets left in the

buffer, so first_above_time_ is not reset and (assuming this pattern continues) there will be a drop at 1.5 *interval*s.

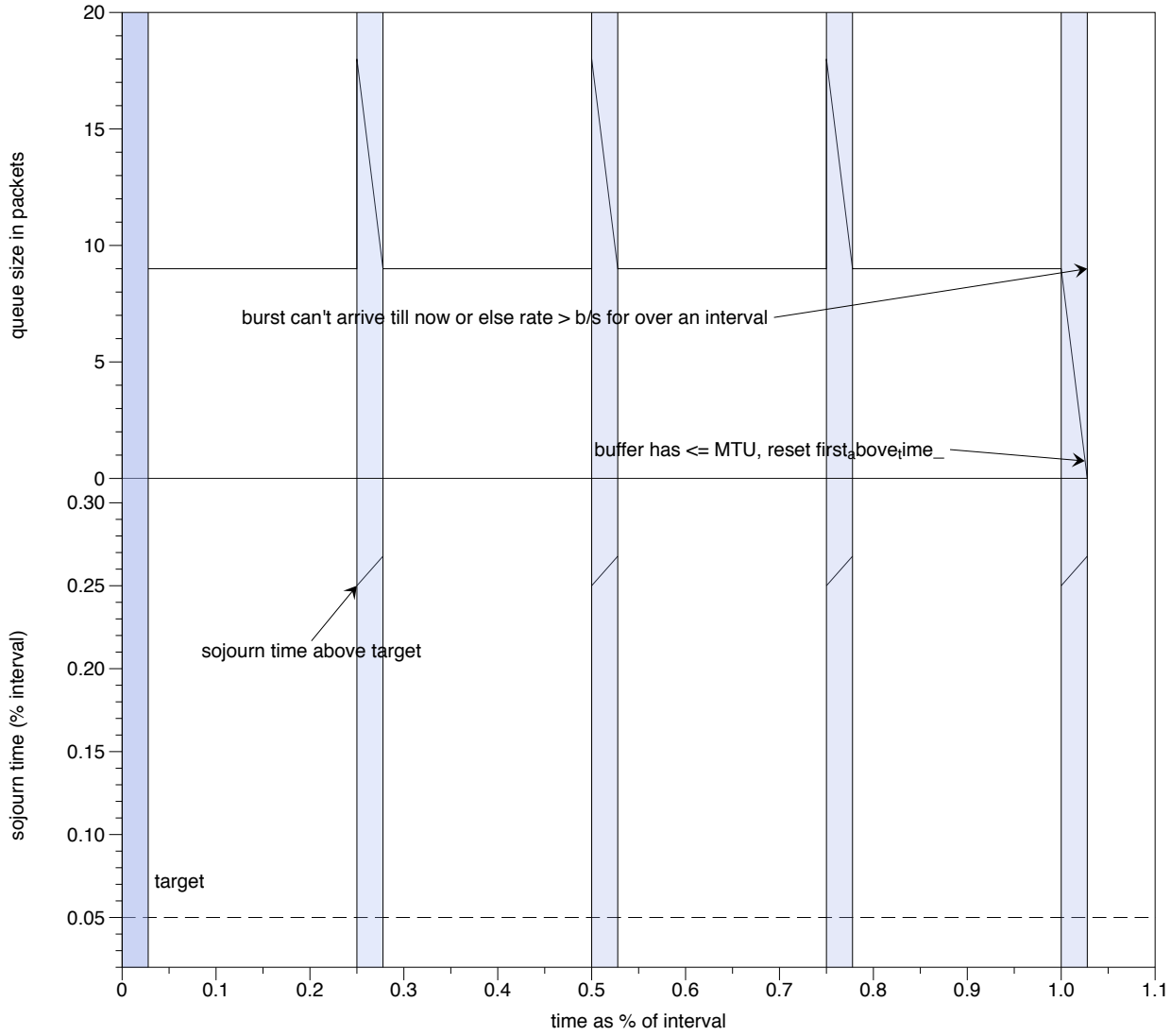
In these examples, the *target* is fine as it is, but if the original *interval* only just covered the intrinsic path round trip time, we must add *s* ms to the *interval* to allow sufficient time for drop responses to propagate back to the source. Although these examples are contrived, they can be used as a starting point to understand the basic principle that:

> If the transmission rate over an *interval* is greater than the arrival rate, then the drop state should not persist long enough for a drop to occur.

In addition to increasing the *interval* by the waiting delay *s*, another adjustment might be useful for certain kinds of bursty MACs. If the MAC is a request-and-grant type, as wifi in infrastructure mode, cable modems and some satellite modems, the allocation of bytes or packets that can be sent during each transmission slot is generally known at the beginning of transmission and may vary for each transmission slot. In that case, it MAY be useful to use that value instead of the MTU value to reset first_above_time_.



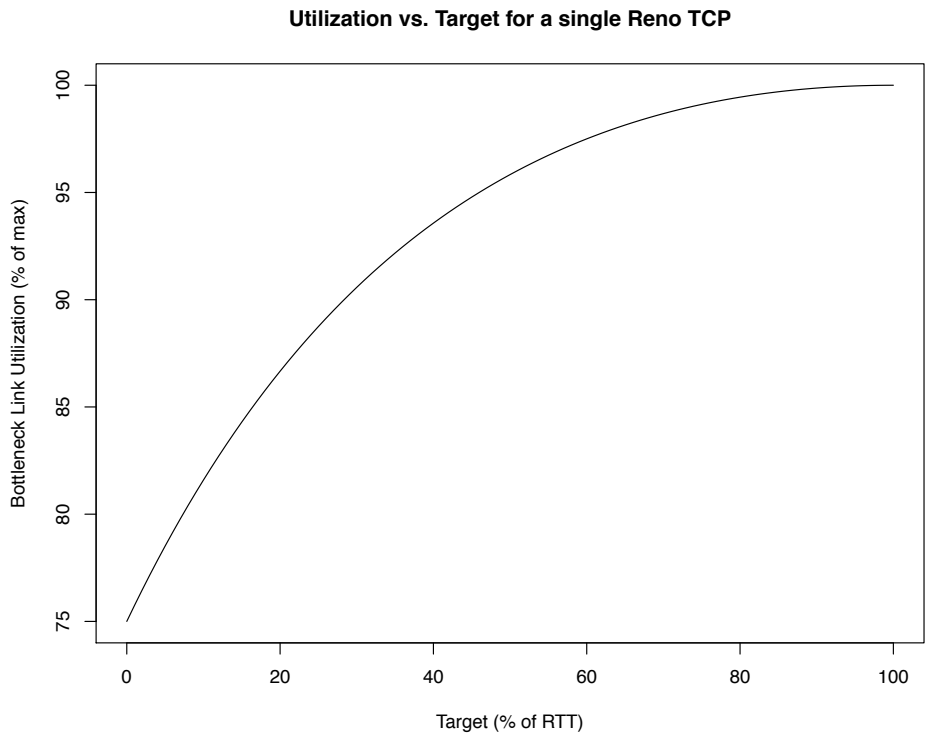
It is possible to contrive a more problematic example if the departure rate is exactly equal to the arrival rate. Only 9 packets arrive every s ms, which is equal to the transmission rate during the transmission slot, but they all arrive in a burst. The first one arrives just after the transmission slot and the second burst arrives during the second transmission slot so that the buffer occupancy doesn't empty or fall to a single packet. However, if the rate is not to exceed b/s over *interval*, there has to be a point where the burst doesn't come until the end of the transmission slot which will allow first_above_time_ to be reset.



"I increased the target and it works"

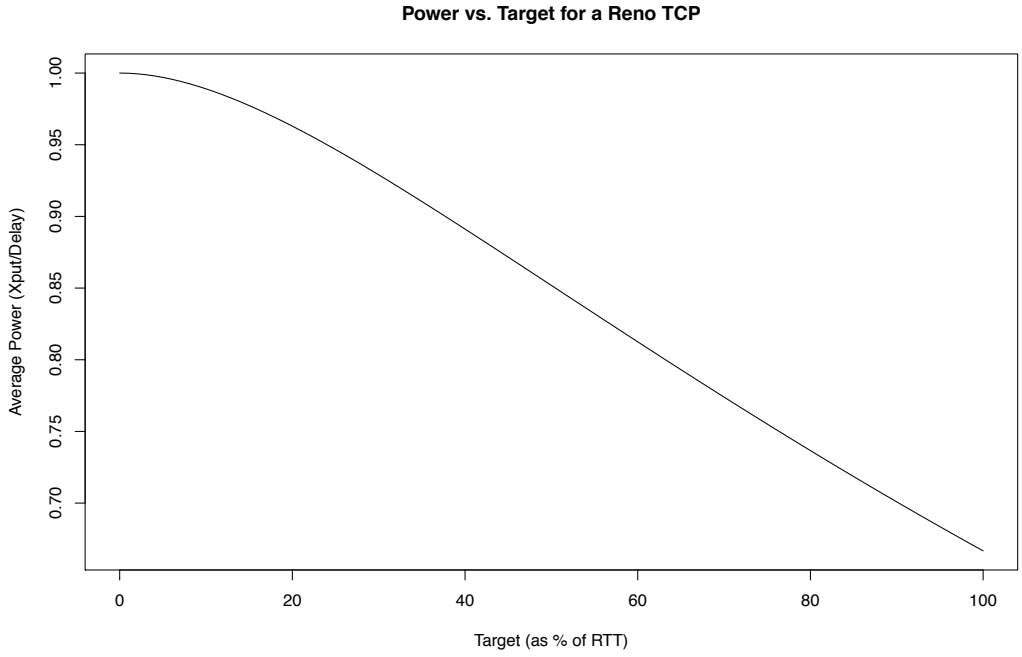
Okay, please define what "it works" means to you. What measurements were taken to decide this was an improvement? Was the packet trace examined to determine what was happening?

A larger target means it takes longer until the first drop occurs and more packets are in flight in the connection. When the drop occurs and the number of packets in flight is halved, there will still be more packets in flight than for a lower target and thus utilization will be higher. The cost is a longer delay (by the difference in the two targets) in the connection in steady state. This effect is evident in the figures



reproduced below from Van's IETF talk. Increasing the target moves the operating point to the right; the target increases and the bottleneck link utilization increases for a single TCP.

Increase the target and an increased link utilization can be observed for a single TCP. This thinking led to bufferbloat in the first place. As seen below, this ignores the effects of increased delay. As the target



increases, the delay increases and the *power* of the connection declines. Furthermore, the utilization improvement decreases with each additional TCP flow through the bottleneck, any increase in utilization from adding delay goes to zero.