Improving DCTCP/Prague Responsiveness

Bob Briscoe*

27 Nov 2020

Abstract

This paper shows that Data Center TCP (DCTCP) introduces approximately 1.5–2 unnecessary rounds of lag due to the way it processes congestion feedback. This is in addition to the inherent round trip of lag in the feedback loop. A per-packet moving average is proposed that removes one of these rounds of lag, while still smoothing the average over a set number of rounds. The rest of the lag is addressed by spreading the congestion response over a round.

1 Problem

Common implementations of DCTCP [AGM+10] (e.g. in Windows, Linux, or FreeBSD [BTB+17]) take two rounds before a change in congestion on the path fully feeds into the moving average that regulates its response. This is on top of the inherent round trip of delay in the feedback loop.

Both extra rounds are due to the two-stage process for maintaining the EWMA then using it to respond to congestion (see Figure 1). The first stage introduces a round of delay (RTT_i) while it accumulates the marking fraction before it can calculate the EWMA (α_i), which it passes to the second stage that uses the EWMA to reduce the congestion window (W) by $\alpha_i W_i$. The second extra round of delay is because the first stage clocks independently, while the second stage is triggered by the arrival of congestion feedback (a red-coloured ACK). So it takes two rounds before a full round of the congestion that triggered the start of the second stage has fed through into the EWMA that the first stage passes to the second. This is further exacerbated by entering congestion window reduced (CWR) state as soon as the initial inadequate response has been applied, which suppresses any further response for a round, while all the congestion feedback arrives.

The problem seems to boil down to how to update an EWMA of marking probability on a per-packet basis. But, more specifically, the problem is how to ensure that the time constant over which the per-packet EWMA smooths itself keeps to a set number of rounds, even though the number of packets per round varies.

2 Per-ACK EWMA

Instead of the EWMA of the marking probability being upscaled by a constant factor, it is proposed to upscale it by `flight_ / gain`, where `flight_` is the packets in flight, and `gain` is a constant (`gain < 1`). Then, as shown below, the EWMA can be maintained by a single continuous set of repetitive increments or decrements determined on each ACK.

As we shall see, scaling up the EWMA by `flight_` enables the repetitive per-packet operations to *implicitly* smooth the EWMA with a characteristic timescale proportionate to the RTT (specifically, `RTT/gain`).

This contrasts with the two-stage process of accumulating the marking fraction for a round before feeding it into an EWMA that is *explicitly* clocked per round.

The scaled up EWMA is effectively a smoothed count of the number of congestion marks per RTT, but scaled up by the constant `1/gain`. The number of marks per RTT (`v`) is related to the marking probability (`p`) by `v = p*flight_`.

Common implementations of DCTCP incorrectly mimic the timing but not the size of a classical congestion control's response. Classical congestion controls suppress any further response for a round because their initial response is large and fixed. On first onset of congestion, it seems wrong for DCTCP to immediately respond with a tiny reduction (based on the previous absence of congestion), then suppress any further response for a round trip.

So, once we have an EWMA of congestion marks that is updated continually on every ACK, it becomes possible to spread the reduction over the round. Then, if congestion continues to rise during the round, the EWMA will grow, and the response can pick up this growth as it proceeds.

*research@bobbriscoe.net,

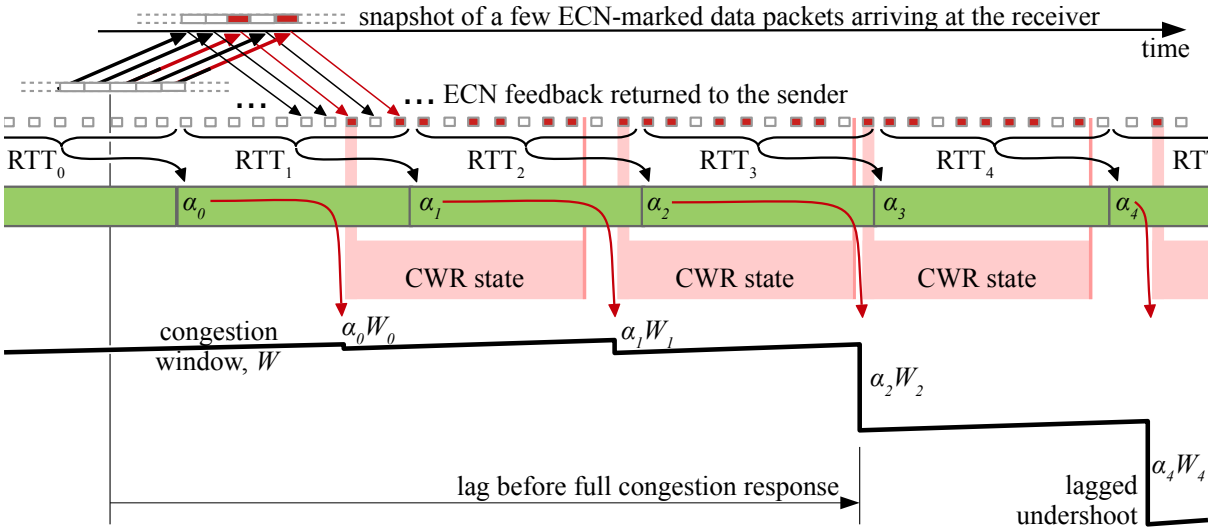

Figure 1: The problem: DCTCP's two stages for processing congestion feedback: 1) gathering feedback in a fixed sequence of rounds (RTT_i to calculate the EWMA (α_i); 2) applying this EWMA on the first feedback mark, when it has had no time to gather enough feedback, which leads to a typically inadequate congestion response before entering congestion window reduced (CWR) state, which suppresses any further response for a round. See text for full commentary.

Definitions of variables

`g` = `1/gain`. By default `g = 16`;
`g_shift` = `lg(g)`;
`av_up` : EWMA of the marks per round upscaled by `g`; Alternatively, it might help to think of this as the EWMA of the marking probability (alpha in DCTCP) upscaled by `g * flight_`.
`flight_` : the number of packets in flight when the marking probablity was fed into the EWMA;
`flight` : the number of packets in flight now;
`ce_fb` = 1 if ECN packet feedback; 0 otherwise.

2.1 Intuition

In DCTCP, the EWMA, `alpha`, is maintained per round trip as follows (in floating point arithmetic):

```
alpha += (F - alpha)/g,
```

where F is the fraction of marked bytes accumulated over the last round trip.

This can be approximated (see § A) by repeatedly updating the EWMA on the feedback of every packet, but scaling down each update by the number of packets in that round, `flight_`. That is, the following per-packet update:

```
alpha += (ce_fb - alpha) / (flight_*g).
```

The above per-packet update of the EWMA is roughly equivalent to the following per-packet update of the upscaled EWMA, `av_up`:

```
av_up += ce_fb - av_up/(flight_*g).
```

2.2 Implementation

2.2.1 Maintaining the EWMA

The `ce_fb` term can be implemented by adding 1 to `av_up` on feedback of each CE-marked packet. If the upscaling of `av_up` by `g` were removed to produce an EWMA of marks per round, this would be equivalent to adding `1/g` of a mark.

The number of packets acknowledged in the current round is `flight`. So repeatedly subtracting `av_up/(flight_*g)` on the arrival of every ACK would reduce `av_up` by `av_up*flight/(flight_*g)` in a round. This approximates to `av_up/g` per round. In integer arithmetic, that can be implemented by `av_up/g` decrements of 1.

```
On_each_ACK'd_packet {
    av_up += ce_fb - repetitive_div(
                av_up, flight*g, &av_carry);
}

// Repeated execution outputs true with
//  frequency proportional to num/denom
bool repetitive_div(u32 num, u32 denom,
                                u32 *carry) {
    *carry += num;
    if (*carry > denom) {
        *carry -= denom;
        return true;
    }
    return false;
}
```


This is achieved as shown in the C-like pseudocode above by the repetitive additions, comparisons and decrements of a variable called `av_carry`, which carries forward the remainder between ACKs.

We have defined the general-purpose function `repetitive_div()` for the repetitive division, which will be reused later for maintaining the congestion window as well.

Figure 2 compares toy simulations of the above EWMA and the DCTCP EWMA (without changing cwnd). It can be seen that, when whenever marks arrive, the algorithm always moves immediately, whereas DCTCP's EWMA does nothing until the next round trip cycle.

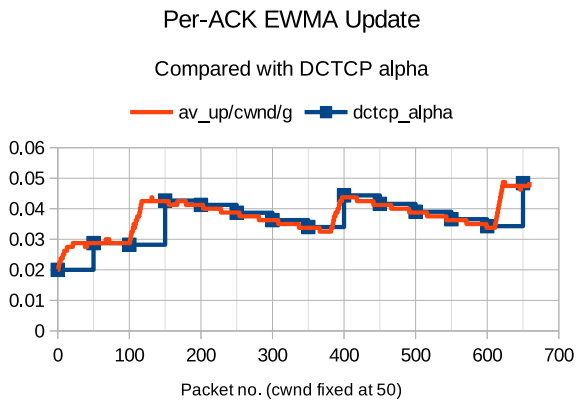

Figure 2: Initial verification of per-ACK EWMA algorithm, with constant `cwnd`

2.2.2 Responding to Congestion

The proposed approach uses the teaching of DCTCP selectively. DCTCP reduces `cwnd` by `alpha*cwnd/2` in any round trip in which CE feedback is present. The proposed approach reduces `cwnd` by half the average number of marked packets per round, or `av_up / (2*g)`.

This is broadly equivalent to DCTCP in that it maintains the scalable '$1/p$' response function, but with two main quantitative differences:

- The window reduction is taken as a proportion of what the window has been in recent rounds, not what it is now (as in DCTCP) (see § 3).
- The window reduction is taken as a proportion of the amount of the window that has been *used* in recent rounds, not the maximum that the flow was entitled to use, i.e. packets in flight not `cwnd`. Thus, if an application-limited flow has only used a quarter of the available window in recent rounds, the proposed reduction of `cwnd` will be only a quarter of that applied by DCTCP.

The main departure from DCTCP is in the speed of response. Rather than reduce `cwnd` on the first sign of CE feedback then suppress further response for a round trip, it is proposed to spread the reduction over the round following the first sign of CE feedback. In other words, use the whole round while in CWR state to reduce `cwnd` as the EWMA updates, such that, by the end of the round it will still have reduced as much as it would have done if the whole reduction had been applied at the end of CWR state.

This will exploit the fact that the EWMA (`av_up`) is continually updated on every ACK. So, at one extreme, if the first CE mark is immediately followed by many others, the EWMA will rapidly increase early in the round of CWR, and `cwnd` can be rapidly decreased accordingly. While, at the other extreme, if the first CE mark is the only CE mark in the round, `cwnd` will still have reduced by `alpha*cwnd/2` by the end of the round, but the EWMA will hardly have increased above the value it took when the CWR round started.

To spread the reduction over the round, the proposed algorithm below does not divide the round into an arbitrary number of points where `cwnd` is altered by varying amounts. Instead, it alters `cwnd` on the first ACK when the algorithm calculates that at least one packet of movement is in order.

```
On_each_ACK'd_packet {
    av_up += ce_fb - repetitive_div(
              av_up, flight*g, &av_carry);
    if (!cwr && ce_fb) {
        // Record start of CWR state
        next_seq = snd_next;
        cwr = true;
    }
    if (cwr) {
        cwnd -= repetitive_div(
          av_up, flight*g*2, &cwnd_carry);
        // Check for end of CWR round
        if (snd_una > next_seq) {
            cwr = false;
        }
    }
}
```

Bob: SHould CWR be checked before the reduction?

Bob: Consider inflating cwnd reduction by cwnd/flight.

3 (Non-)Concerns

There seems to be a circular dependency, because `av_up` is both upscaled by `cwnd` then used to update `cwnd`.

In fact, `av_up` is upscaled by what `cwnd` *was* at the time of each repetitive decrement or increment of `av_up`. So `av_up` depends on an implicit exponentially weighted moving average of `cwnd` with a characteristic smoothing timescale of `g` round trips. This stabilizes the circular dependency.

4 Evaluation

For research purposes, we ought not to introduce two changes at once, without evaluating each separately. Therefore, initially, we ought to use the continually updated EWMA, `av_up` to reduce `cwnd` in the classical way. That is, on the first feedback of a CE mark, reduce `cwnd` once by `av_up/(2*g)`. Then suppress further response for a round (CWR state). This should remove one round of lag (originally spent accumulating the marking fraction), but not the other (reducing cwnd in response to a single mark, then doing nothing for a round while the extent of marking is becoming apparent).

Basic experiments will need to compare how quickly a DCTCP flow in congestion avoidance can reduce in response to a newly arriving flow. It will also be necessary to check performance in the following more cases that might expose poor approximations in the algorithm (relative to DCTCP):

1. When the packets in flight has been growing for some time;
2. . . . or shrinking for some time;
3. When the flow is application limited, with packets in flight varying wildly, rather than tracking the smoother evolution of `cwnd`.

5 Related Work

Reducing `cwnd` by half of `v` per round trip is similar to Relentless TCP [Mat09], which reduces `cwnd` by half a segment on feedback of each CE-marked packet. The difference is that `v` is a moving average, so it implements smoothing in the sender, whereas Relentless immediately applies a full congestion response without smoothing, because it is designed for smoothing in the network.

Like DCTCP, the per-ACK congestion response proposed in section 5.2 of [AJP11] maintains an EWMA of congestion marking probability, `alpha`. But it reduces `cwnd` by half of `alpha` (in units of packets) on feedback of each ECN mark. This causes jumpiness, because marks tend to be bunched into one round then clear for a few rounds, particularly with step-marking. So, even though this algorithm uses the smoothed EWMA for each reduction, it applies many more reductions in those rounds with more marks. In contrast, the approach proposed in the present paper limits the reduction to `alpha` within a round trip (as DCTCP itself does). Nonetheless, it removes the round of lag that DCTCP's EWMA uses to accumulate the marking probability, and it spreads the reduction over a round to ensure that it picks up feedback as soon as it arrives.

6 Future Work

Bob: Possibility of increasing gain (reducing g).

Bob: Possibility of adding reduction proportional to growth in av_up (adding the P part of a PI controller).

Bob: Possibility of using av_up to detect that the flow probably gone open-loop, and should enter a different mode to find a faster operating point. E.g. measuring the number of unmarked packets since the last mark, and using the difference between that and the average distance implied by av_up.

References

[AGM+10] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitu Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). *Proc. ACM SIGCOMM'10, Computer Communication Review*, 40(4):63–74, October 2010.

[AJP11] Mohammad Alizadeh, Adel Javanmard, and Balaji Prabhakar. Analysis of DCTCP: Stability, Convergence, and Fairness. In *Proc. ACM SIGMETRICS'11*, 2011.

[BTB+17] Stephen Bensley, Dave Thaler, Praveen Balasubramanian, Lars Eggert, and Glenn Judd. Data Center TCP (DCTCP): TCP Congestion Control for Data Centers. Request for Comments RFC8257, RFC Editor, October 2017.

[Mat09] Matt Mathis. Relentless Congestion Control. In *Proc. Int'l Wkshp on Protocols for Future, Large-scale & Diverse Network Transports (PFLDNeT'09)*, May 2009.

A Approximations

The per-ACK EWMA is not intended to precisely mimic a per-RTT EWMA. Otherwise, the per-ACK EWMA would have to reach the same value by the end of the round, irrespective of whether markings arrived early or late in the round. It is more important for the EWMA to quickly accumulate any markings early in the round than it is to ensure that the EWMA reaches the same value by the end of the round.

It is not important that a per-ACK EWMA decays at the same rate as a per-round EWMA using the same gain. The gain is not precisely chosen, so if a per-ACK EWMA decays more slowly, a higher gain value can be used for a per-ACK algorithm.

It *is* important that a per-ACK EWMA decays at about the same rate however many ACKs there are per round, although the decay rate does not have to be precisely the same.

The per-ACK approach uses the approximation that one reduction with gain $1/g$ is roughly equivalent to n repeated reductions with $1/n$ of the gain. Specifically, that $(1 - 1/ng)^n \approx 1 - 1/g$, for $g \geq 2; n \geq 2$.

$$\begin{aligned}(1 - 1/ng)^n &= 1 + \frac{n}{-ng} + \frac{(n-1)}{(-ng)^2} + \dots \\ &= 1 - \frac{1}{g} + O\left(\frac{1}{n(-g)^2}\right) \\ &\approx 1 - \frac{1}{g}\end{aligned}$$

.

Document history

Version	Date	Author	Details of change
00A	07 Nov 2020	Bob Briscoe	First draft.
00B	27 Nov 2020	Bob Briscoe	Added `cwnd` reduction. Defined reusable function `repetitive_div()`. Corrected use of `cwnd` to `flight`, and distinguished current `flight`, from `flight_` when marks were averaged.

