# Improving DCTCP/Prague Responsiveness

Bob Briscoe*

07 Nov 2020

## 1 Problem

Common implementations of DCTCP [AGM+10] (e.g. in Linux or FreeBSD) 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 that the feedback loop adds.

Both extra rounds are due to the two-stage process for maintaining the EWMA then using it to respond to congestion. The first stage introduces a round of delay while it accumulates the marking fraction before passing the EWMA to the second stage that uses it to respond to congestion. The second extra round of delay is because the first stage clocks independently, while the second stage is triggered by the onset of congestion. 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.

## 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 `cwnd / gain` (where `cwnd` is the congestion window in packets, and `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 `cwnd` 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 (`v = p*cwnd`), but scaled up by the constant `1/gain`. So, on a congestion event, 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 is designed for smoothing in the network, with immediate response in the sender.

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

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, during the round, `v` will be growing if congestion is rising, and the response will naturally grow accordingly.

#### Definitions of variables

`g` = `1/gain`. By default `g = 16`.

`g_shift` = `lg(g)`

`av_up` : EWMA of marking probability upscaled by `cwnd * g`

`I` = 1 on feedback of a marked packet; 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 could be approximated by updating the EWMA on the feedback of every packet (for now assuming no delayed ACKs) as follows:

*research@bobbriscoe.net,

```
alpha += (I - alpha)/(cwnd*g),
```

where `cwnd` is in units of packets (the possibility of a circular dependency here is discussed later). In turn, this per-packet update of the upscaled EWMA is equivalent to the following integer arithmetic:

```
av_up += I - av_up/(cwnd*g).
```

## 2.2 Implementation

### 2.2.1 Maintaining the EWMA

The `I` term can be implemented by adding 1 to `av_up` on feedback of each CE-marked packet.

`cwnd` repetitive subtractions of `av_up/(cwnd*g)` in a round approximate to `av_up/g` decrements of 1, which in turn approximate to a decrement of 1 every `g*cwnd/av_up` packets.

Therefore, maintenance of the upscaled EWMA, `av_up`, can be implemented in integer arithmetic without any factoring down and therefore no truncation or rounding. This is achieved by the following repetitive additions, comparisons and decrements of a variable called `a_carry` that carries forward the remainder between ACKs:

```
On each ACK'd packet {
    if (echoed_ce) {
        av_up++;
    }
    a_carry += av_up;
    _cwnd_up = cwnd << g_shift;
    if (a_carry > _cwnd_up) {
        a_carry -= _cwnd_up;
        av_up--;
    }
}
```

### 2.2.2 Responding to Congestion

For research purposes, we ought not to introduce two changes at once, without evaluating each separately. Therefore, we can use the continually updated EWMA, `av_up` to reduce cwnd in the classical way—suppressing further response for a round then responding on the first subsequent feedback of a CE mark.

Then, the current EWMA `av_up` can be used to reduce the congestion window `cwnd` on the first signal of congestion after the sender had left a previous round of congestion window reduced (CWR) state. The reduction in `ssthresh` and therefore `cwnd` would be `(av_up / cwnd / g) * cwnd / 2` = `av_up / (2*g)`.

To spread the reduction out, note that this per-RTT reduction of `cwnd` by `av_up / (2*g)` is half the per-RTT reduction of `av_up` within the EWMA algorithm. So it could be implemented by decrementing `cwnd` and `ssthresh` by 1 every other time that `av_up` is decremented, for the duration of a round trip (while in CWR state).

Bob: ToDo: Give specific pseudocode for the reduction spread over time.

# 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.

 

# 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.

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

 

# Document history

| Version | Date | Author | Details of change |
|---|---|---|---|
| 00A | 07 Nov 2020 | Bob Briscoe | First draft. |

 