Datacenter TCP (DCTCP):
TCP Congestion Control for Datacenters
draft-bensley-tcpm-dctcp-00

Stephen Bensley, Lars Eggert and Dave Thaler
TCPM, IETF-89, London, UK
March 6, 2014

RAND-Z IPR disclosure! https://datatracker.ietf.org/ipr/2319/
DCTCP recap

• **TCP variant for datacenters**

• **Goals**
  – Low latencies for short flows
  – High throughputs for long flows

• **Approach**
  – Use ECN to quantify the extent of congestion
  – Scale `cwnd` proportionally
Known DCTCP implementations

• **Microsoft Windows Server 2012**
• **FreeBSD-CURRENT** (by Midori Kato, Keio)
  – Conforms to draft (plus some variants; ODCTCP)
• **Linux 2.6.38.3 and ns2** (by Stanford & NEC)
  – Unclear conformance to draft
Architectural components

- **Switch fabric**
  - Detects congestion and sets CE bit
- **Receiver**
  - Echoes ECE
- **Sender**
  - Adjusts cwnd
Congestion marking

• Switch marks per RFC 3168, e.g., with threshold
Echoing congestion information

• RFC3168 does not suffice
• Introduce DCTCP . CE state flag
• When sending ACK
  – Set ECE in ACK iff DCTCP . CE == true
• Upon segment reception
  – If CE is true and DCTCP . CE is false, send ACK and set DCTCP . CE to true
  – If CE is false and DCTCP . CE is true, send ACK and set DCTCP . CE to false
  – Otherwise, ignore CE
Sender-side processing

• Sender estimates fraction of sent bytes that saw congestion, stored in $\text{DCTCP \cdot Alpha}$
  
  \[
  \text{DCTCP \cdot Alpha} = \text{DCTCP \cdot Alpha} \times (1-g) + g \times M
  \]

• $g$ is estimation gain, real number between 0-1 (implementation-specific; Windows uses 1/16)

• $M$ is the fraction of sent bytes that saw congestion during the previous observation window ($\approx$ RTT)
Sender cwnd update

- Update `cwnd` when `DCTCP.Alpha` updates
  
  \[ cwnd = cwnd \times (1 - DCTCP.Alpha/2) \]

- When no sent byte experienced congestion, `DCTCP.Alpha` is 0, and `cwnd` is left unchanged

- When all sent bytes experienced congestion, `DCTCP.Alpha` is 1, and `cwnd` is reduced by half

- Levels of congestion between the extremes will result in a proportional reduction to `cwnd`
Updating DCTCP.Alpha

• Three additional TCP state variables

• **DCTCP.WindowEnd**
  – TCP sequence number threshold for beginning a new observation window; initialized to **SND.UNA**

• **DCTCP.BytesSent**
  – Number of bytes sent during the current window; initialized to zero

• **DCTCP.BytesMarked**
  – Number of bytes sent during the current window that encountered congestion; initialized to zero
Process acceptable ACKs

1. Compute the bytes acknowledged
   \[\text{BytesAcked} = \text{SEG.ACK} - \text{SND.UNA}\]

2. Update the bytes sent
   \[\text{DCTCP.BytesSent} += \text{BytesAcked}\]

3. If ECE flag is set, update the bytes marked
   \[\text{DCTCP.BytesMarked} += \text{BytesAcked}\]
   - If \(\text{SEG.SEQ} \leq \text{DCTCP.WindowEnd}\), stop
   - Otherwise, observation window has ended; update the congestion estimate (steps 4-7)
Update congestion estimate

4. Compute congestion for the current window
   \[ M = \frac{\text{DCTCP.BytesMarked}}{\text{DCTCP.BytesSent}} \]

5. Update the congestion estimate
   \[ \text{DCTCP.Alpha} = \text{DCTCP.Alpha} \times (1-g) + g \times M \]

6. Set the end of the next window
   \[ \text{DCTCP.WindowEnd} = \text{SND.NXT} \]

7. Reset the byte counters
   \[ \text{DCTCP.BytesSent} = 0 \]
   \[ \text{DCTCP.BytesMarked} = 0 \]
Next steps

• Improve draft for completeness & clarity
  – Initial goal: describe MS implementation

• Eventually, would like to know if there is interest in this as a WG item
  – And then maybe incorporate improvements