TCP@fb

EXPERIENCES EVALUATING DCTCP

Lawrence Brakmo, Boris Burkov, Greg Leclercq and Murat Mugan

Facebook

INTRODUCTION

- Standard TCP congestion control, which only reacts to packet losses has many problems
  - Can result in standing queues (queues that do not dissipate)
  - Increases tail latencies due to loss recovery times
  - Penalizes smaller RPC flows
- Congestion avoidance, which reacts to increasing queues, have been proposed as a solution. Of these, DCTCP is one of the most commonly used in Data Centers.

ECN

- ECN (Early congestion notification) marks packets that arrive when a queue size threshold has been exceeded
- Original response to an ECN marking was to react as if the packet had been dropped: reduce cwnd by 50% (Reno)
- In many network topologies this response is too aggressive and can result in link under-utilization
- One problem is that it was not differentiating between transient and standing congestion

DCTCP

- DCTCP also uses ECN markings to detect queue build-up
- However, instead of always reacting in the same way (50% reduction of cwnd) DCTCP reacts to the level of congestion.
- Uses the percent of packets marked per RTT to determine response
  - If all packets marked, reduce cwnd by 50%
- To deal with transient congestion, the response is based on a moving average

OVERVIEW

- We describe our experiences evaluating DCTCP
- Initial tests were within a rack using Netesto* to create traffic and capture metrics
  - This uncovered various issues
- 6 rack test using DC services
  - Using an artificial load to fully saturate network
  - Corroborated higher CPU utilization seen in rack tests

* Network Testing Toolkit

TCP@fb

ISSUES

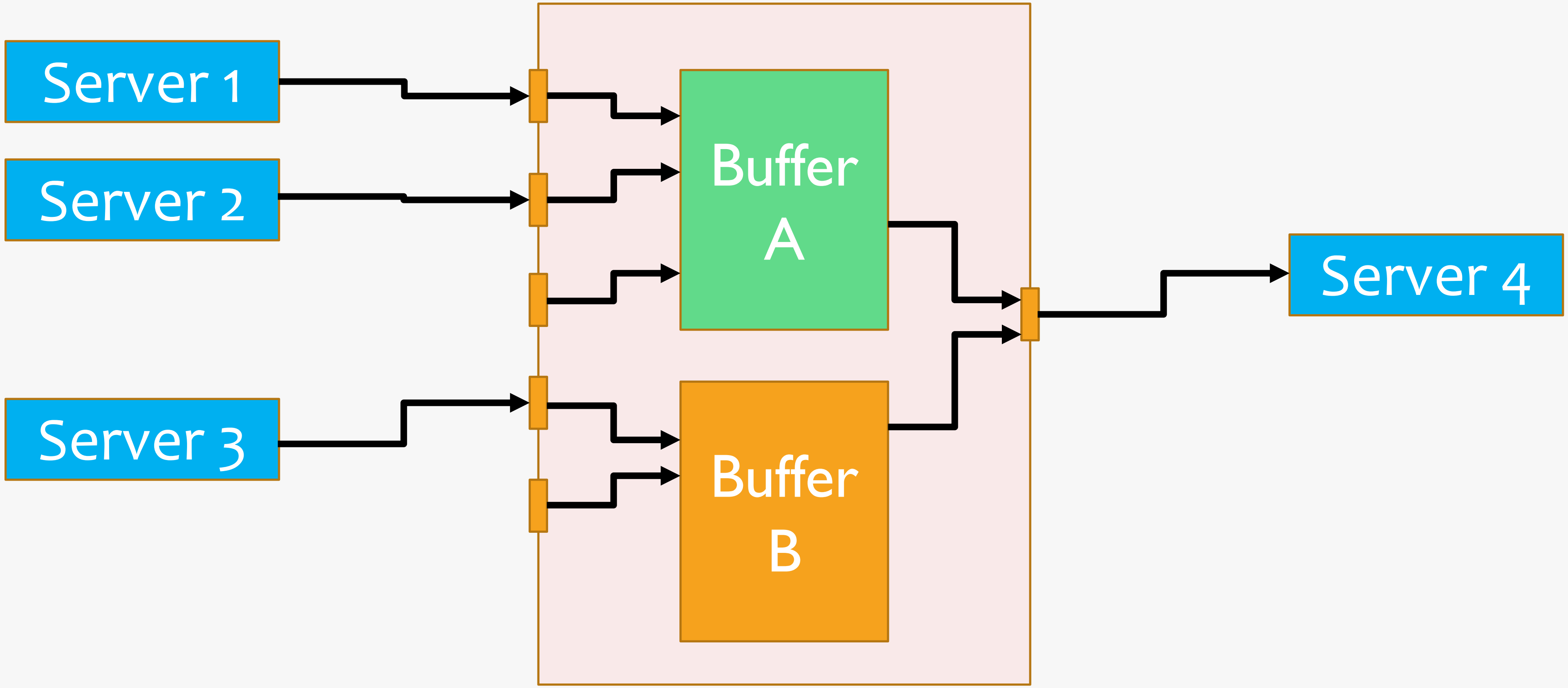
- Issues seen in rack testing
  - Unfairness between senders regardless of TCP congestion control
  - Unfairness between flows (even with only one sender) when using ECN
  - High tail latencies when using DCTCP
- Issues seen in 6-rack tests
  - Higher CPU utilization with DCTCP

ISSUE: UNFAIRNESS BETWEEN SERVERS

- Noticed unfairness in experiment where 3 servers send to a 4th one
- 2 servers would get 25% of bandwidth each
- 1 server would get 50% of bandwidth
- Turned out to be due to the switch design
  - Switch uses 2 buffers for each output port
  - Input ports are assigned to one of the output buffers
  - 2 servers came on input ports assigned to buffer A
  - 1 server came to input port assigned to buffer B
  - Switch round-robins between buffers

TCP@fb

SWITCH ARCHITECTURE



- 2 Servers use Buffer A
- 1 Server uses Buffer B
- Output port round-robins between output buffers

TCP@fb

ISSUE: UNFAIRNESS BETWEEN FLOWS WITH ECN

- With only 2 flows, one flow would get much higher link utilization (23Gbps vs. 0.5 Gbps)
- Wrote a tool to analyze pcaps. For each flow it could show
  - Per RTT metrics
  - Per packet details
- Discovered that one flow's RTTs were bimodal: either 60us or 1.3ms (cwnd was small < 20)

BIMODAL RTT DISTRIBUITION

```
RTT  2.1    0.020255( 1.3ms)  out: 14.3KB rate: 85.83 Mbps
RTT  2.2    0.021586( 1.3ms)  out: 28.6KB rate:173.88 Mbps
RTT  2.3    0.022900(  40us)  out: 30.0KB rate:  5.85 Gbps
RTT  2.4    0.022941( 1.3ms)  out:  2.9KB rate: 17.35 Mbps
RTT  2.5    0.024258(  57us)  out: 31.4KB rate:  4.41 Gbps
RTT  2.6    0.024315( 1.3ms)  out:  1.4KB rate:  8.72 Mbps
RTT  2.7    0.025625(  76us)  out: 32.8KB rate:  3.46 Gbps
RTT  2.8    0.025701( 1.2ms)  out: 32.8KB rate:210.71 Mbps
RTT  2.9    0.026948(  85us)  out: 35.7KB rate:  3.36 Gbps
RTT  2.10   0.027033( 1.3ms)  out: 30.0KB rate:187.72 Mbps
RTT  2.11   0.028311( 1.4ms)  out: 38.6KB rate:222.87 Mbps
RTT  2.12   0.029695(  67us)  out: 41.4KB rate:  4.94 Gbps
```

Cause: NIC firmware using large coalescing values and 1ms timer

ISSUE: HIGH TAIL LATENCIES WITH DCTCP

- 1MB and 10KB RPCs had high (as compared to Cubic) tail latencies

	Cubic Latencies		DCTCP Latencies		DCTCP (fixed) Latencies	
	99%	99.9%	99%	99.9%	99%	99.9%
1-MB RPCs	2.6ms	5.5ms	43ms	208ms	5.8ms	6.9ms
10-KB RPCs	1.1ms	1.3ms	53ms	212ms	146us	203us

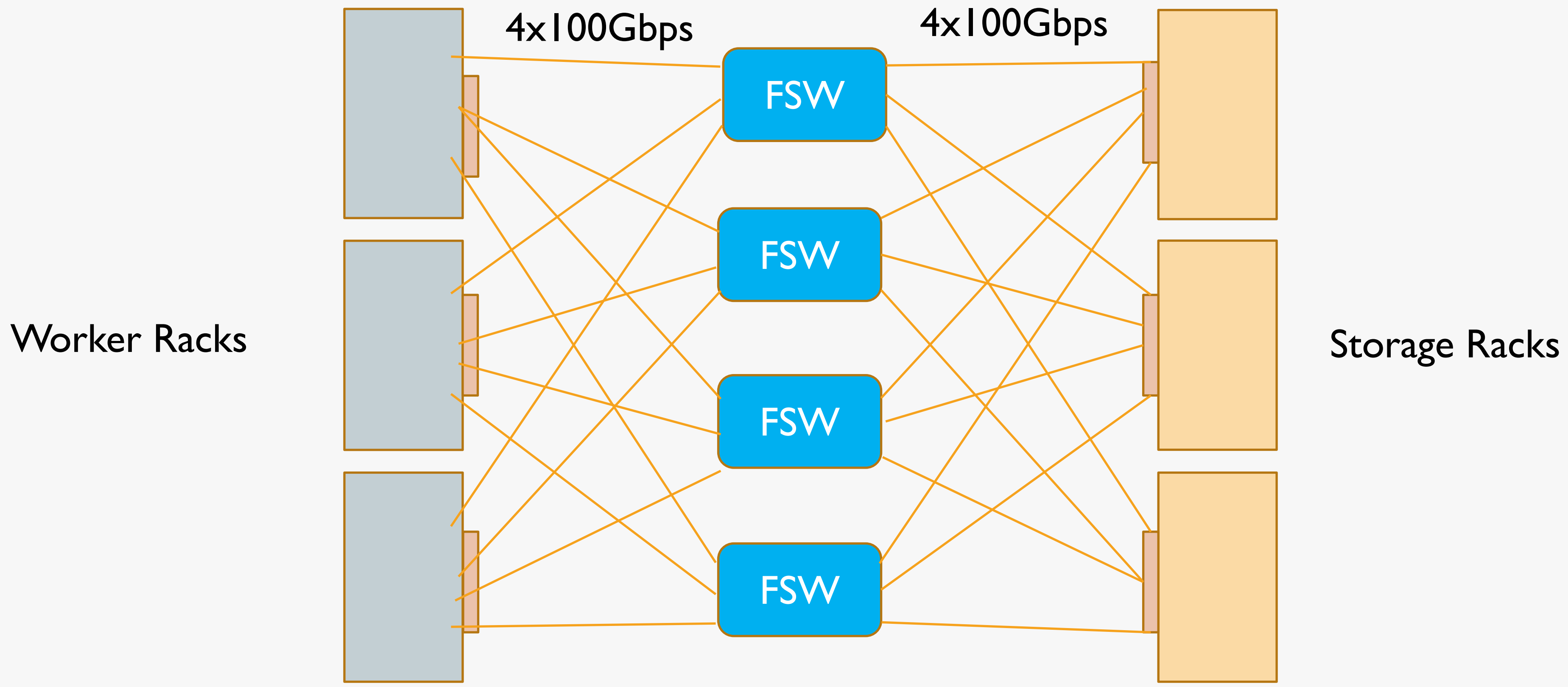
TCP@fb

ISSUE: HIGH TAIL LATENCIES WITH DCTCP (2)

- There were 2 issues increasing tail latencies
  - RTOs caused by the receiver sending a dup ACK and not ACKing the last (and only) packet sent
  - Delaying ACKs when the sender has a cwnd of 1, so everything pauses for the duration of the delayed ACK
- Triggered by kernel patches in 2015
- Fixes are now upstream (patches by Yuchung Cheng, Neal Cardwell and Lawrence Brakmo).

6-RACK TESTS

- 3 racks are store servers
- 1-3 racks (workers) read data from store servers
- Cross traffic between workers



TCP@fb

3 WORKER RACKS (LESS CONGESTION)

	Cubic	DCTCP
FSW to Worker Max Link Util %	69.9	69.8
FSW Discards (bits)	89M	236K (0.3%)
Worker rack discards (bits)	417M	0
Storage Retransmits	0.020	0.000
Worker Retransmits	0.173	0.078
Storage CPU (%)	X	X
Worker CPU (%)	Y	Y + 1%
Storage ECN CE Marked (%)		6.5
Worker ECN CE Marked (%)		12.8

2 WORKER RACKS (MORE CONGESTION)

	Cubic	DCTCP
FSW to Worker Max Link Util %	99.1	98.7
FSW Discards (bits)	160B	157M (0.1%)
Worker rack discards (bits)	2.2B	0
Storage Retransmits	0.590	0.001
Worker Retransmits	0.376	0.035
Storage CPU (%)	X	X + 14%
Worker CPU (%)	Y	Y + 4%
Storage ECN CE Marked (%)		5.5
Worker ECN CE Marked (%)		63.7

TCP@fb

1 WORKER RACKS (VERY CONGESTED)

	Cubic	DCTCP
FSW to Worker Max Link Util %	99.9	98.1
FSW Discards (bits)	235B	19B (8.1%)
Worker rack discards (bits)	1.1B	0
Storage Retransmits	1.020	0.125
Worker Retransmits	0.620	0.125
Storage CPU (%)	X	X + 10%
Worker CPU (%)	Y	Y + 3%
Storage ECN CE Marked (%)		18
Worker ECN CE Marked (%)		73

RESULT SUMMARY

- Fewer switch discards for DCTCP
  - 10x to 1000x fewer depending on load
- Higher CPU utilization for DCTCP at high link utilization
  - At 70% link utilization, CPU use is similar
  - At 99% link utilization, DCTCP uses up to 14% more CPU
    - Depends on the percent of ECN congestion markings
  - Not clear whether this is an issue on production traffic

TCP@fb

ISSUE: HIGH CPU UTILIZATION

- CPU Utilization increases as link utilization increased
  - But then decreases as load increased further
- Seems to be caused by smaller packet coalescence
  - LRO/GRO cannot coalesce packets with different ECN values
    - => more packets handled by receiver
    - => more ACK packets handled by sender
  - Worst case scenario is every other packet (50%) has ECN congestion marking
- Not sure how much of an issue on production workloads

TCP@fb

FUTURE WORK

- Explore techniques for reducing CPU overhead when using DCTCP
- Run cluster wide experiments with production workloads

SUMMARY

- Need better network testing for the kernel
  - I.e. DCTCP bug triggered by patch in 2015
- DCTCP reduces packet drops and retransmissions significantly (up to 1000x)
- DCTCP increases fairness between RPC sizes
- DCTCP increases CPU utilization due to reduced packet coalescing