AC↯DC TCP

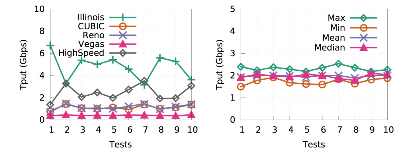
Virtual Congestion Control Enforcement for Datacenter Networks

Keqiang He, Eric Rozner, Kanak Agarwal, Yu (Jason) Gu, Wes Felter, John Carter, and Aditya Akella

Presented by: Allison McDonald and Andrew Quinn

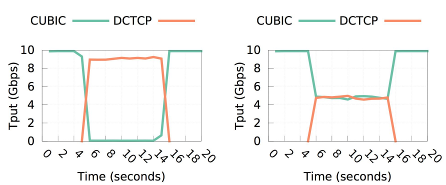
TCP Congestion Control in Public Datacenters

- Datacenter has no control over TCP/IP stack on VMs
- Dozens of different TCP Congestion Control algorithms exist and can interact with each other
- Ensuring that all VMs use up-to-date or uniform TCP/IP stacks is impossible



(a) 5 different CCs. (b) All CUBIC.
Figure 1: Different congestion controls lead to unfairness.

TCP Congestion Control in Public Datacenters



(a) Default. (b) AC↯DC.
Figure 15: (a) CUBIC gets little throughput when competing with DCTCP. (b) With AC↯DC, CUBIC and DCTCP flows get fair share.

Administrator Control over Datacenter (AC/DC) TCP

- Implement congestion control in vSwitch!
  - No changes to the VMs
  - Uniform congestion control across datacenter
  - Per-flow congestion control algorithm selection possible
  - Easy to move to vSwitch → congestion control is lightweight & portable

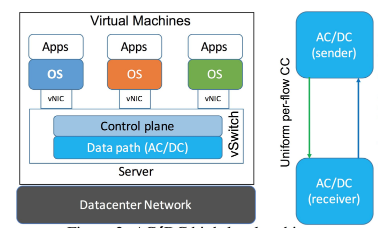


Figure 3: AC↯DC high-level architecture.

Bandwidth Allocation

- Transport layer schemes cannot enforce per-tenant bandwidth allocation
- But bandwidth allocation schemes cannot prevent congestion
  - Aggressive TCP/IP stacks can still flood switches “fairly”
- AC/DC aims to cooperate with or complement bandwidth allocation schemes

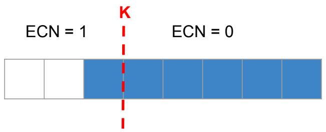
DCTCP

- Datacenter TCP (DCTCP) adjusts the sender’s rate based on the fraction of packets experiencing congestion
- Explicit Congestion Notification (ECN) bit set when switch queue length exceeds a congestion threshold

DCTCP Algorithm

Switch:

- Set ECN bit when Queue Length > K



Sender:

- Maintain fraction of marked packets (α)

For each RTT:

$$F = \frac{\#\, of\ marked\ ACKs}{Total\ \#\, of\ ACKs} \qquad \alpha \leftarrow (1-g)\alpha + gF$$

- Adaptive decrease

$$Cwnd \leftarrow (1 - \frac{\alpha}{2})Cwnd$$

AC/DC Design and Implementation

- Obtain congestion control state information for each flow
- Implement DCTCP at the vSwitch
- Enforce Congestion Control

- Implemented in Open vSwitch (OVS)
- Flows hashed on 5-tuple (dport, daddr, sport, saddr, VLAN)
- Each flow is tracked at receiver and sender

Congestion Control State

At vSwitch:

- cwnd is maintained; starts at 10
- Can see all traffic, so:
  - Loss: if ack_seq <= snd_una, then dupack is incremented
  - Timeouts: when snd_una < snd_nxt and inactivity timer fires

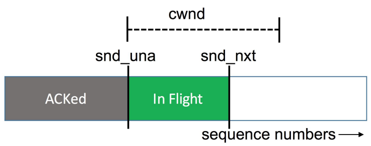


Figure 4: Variables for TCP sequence number space.

Implementing DCTCP

- Add and remove ECN bits when packets go through the vSwitch
- Receiver module monitors congestion and reports it to sender using ACK packets
  - Piggy-back ACK (PACK): add data to ACK's skb headroom
  - Fake ACK (FACK) when PACK creates larger MTU than allowed
  - IP header checksum, IP packet length, and TCP data offset are recalculated; TCP checksum calculated by NIC

Implementing DCTCP

- At sender, cwnd calculated
- If no congestion was encountered, tcp_cong_avoid() expands cwnd based on TCP's New Reno algorithm

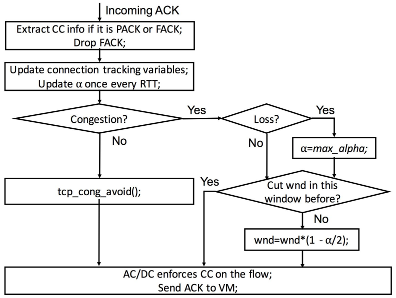


Figure 5: DCTCP congestion control in AC⚡DC.

Enforcing Congestion Control

- vSwitch commandeers sender's advertised rwnd to push its cwnd to receiver
  - Only overwritten when AC/DC cwnd < sender's rwnd
- Well-behaved TCP stacks will follow the standard and adhere to rwnd
- vSwitch can identify misbehaving TCP stacks (sending more than rwnd) and drop excess packets
- Because VM-level ECN feedback is removed, AC/DC's cwnd is the limiting factor, allowing more data to be sent (allegedly)

Potential Extensions: Per-Flow Congestion Control

- Per-flow bandwidth allocation easy by capping cwnd
- Congestion control algorithm can be chosen based on flow
  - For example, CUBIC for flows to the WAN, DCTCP for internal flows
- Priority possible for service classes

$\beta \in [0,1]$

$$\mathrm{rwnd} = \mathrm{rwnd}(1 - (\alpha - \frac{\alpha\beta}{2}))$$

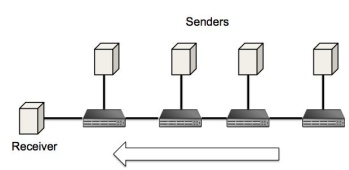
Performance Evaluation

- Is AC/DC underneath regular Linux TCP comparable to DCTCP performance?
  - TCP throughput
  - Loss rate
  - Jain's fairness index
  - Flow completion time
- Across microbenchmarks and macrobenchmarks?

Microbenchmarks

Each sender starts long lived flow:

- DCTCP and AC/DC have .03 (1%) lower throughput than standard TCP, and .05 (5%) better fairness
- RTT for 50th and 99th:
  - AC/DC: 124us & 279us
  - DCTCP: 136us & 301us
  - CUBIC: 3.3ms & 3.9ms

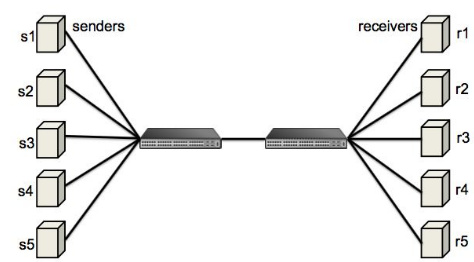


(b) Multi-hop, multi-bottleneck (parking lot) topology.
Figure 7: Experiment topologies.

Why is AC/DC better?
- More on this later

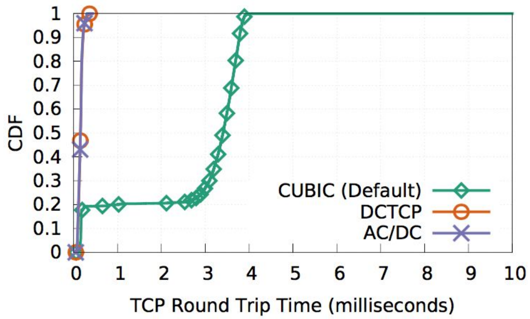
Microbenchmarks

flows from each s_i to r_i

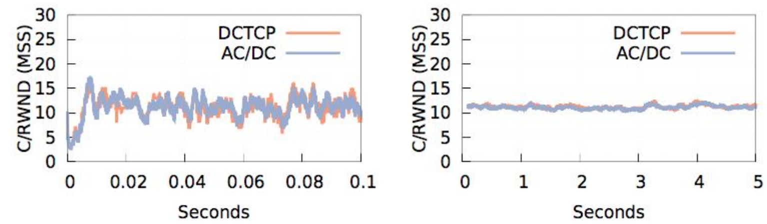


(a) Dumbbell topology.

RTT of different schemes

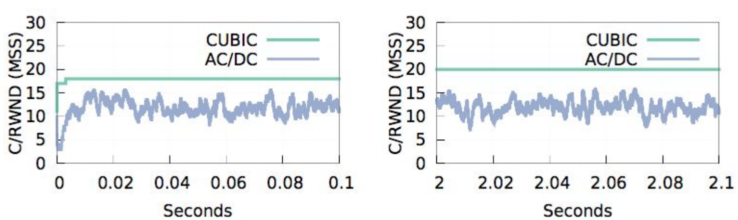


CWND



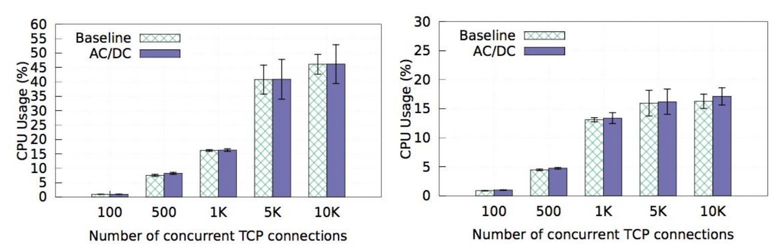
(a) First 100 ms of a flow. (b) Moving average.

Who limits TCP throughput?



(a) Starting from 0 sec. (b) Starting from 2 sec.

AC/DC CPU overhead



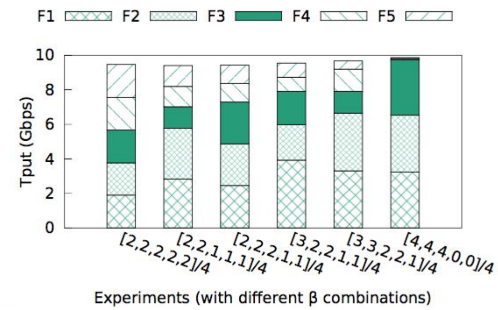
Sender Receiver

AC/DC flexibility

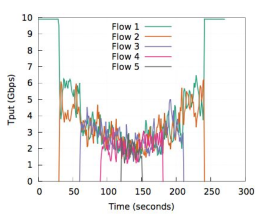
CC Variants	50th percentile RTT (µs)		99th percentile RTT (µs)		Avg Tput (Gbps)		Fairness Index	
	mtu=1.5KB	mtu=9KB	mtu=1.5KB	mtu=9KB	mtu=1.5KB	mtu=9KB	mtu=1.5KB	mtu=9KB
CUBIC*	3232	3448	3641	3865	1.89	1.98	0.85	0.98
DCTCP*	128	142	232	259	1.89	1.98	0.99	0.99
CUBIC	128	142	231	252	1.89	1.98	0.99	0.99
Reno	120	149	235	248	1.89	1.97	0.99	0.99
DCTCP	129	149	232	266	1.88	1.98	0.99	0.99
Illinois	134	152	215	262	1.89	1.97	0.99	0.99
HighSpeed	119	147	224	252	1.88	1.97	0.99	0.99
Vegas	126	143	216	251	1.89	1.97	0.99	0.99

Table 1: AC⚡DC works with many congestion control variants. Legend: CUBIC*: CUBIC + standard OVS, switch WRED/ECN marking off. DCTCP*: DCTCP + standard OVS, switch WRED/ECN marking on. Others: different CCs + AC⚡DC, switch WRED/ECN marking on.

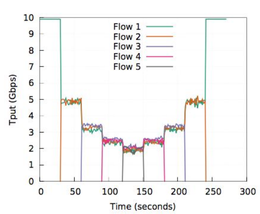
AC/DC flexibility



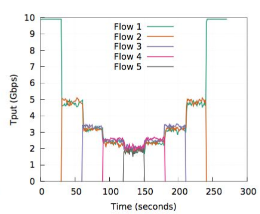
AC/DC Fairness



(a) CUBIC convergence test.

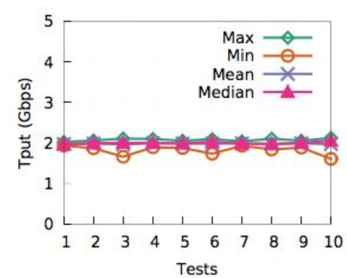


(b) DCTCP convergence test.

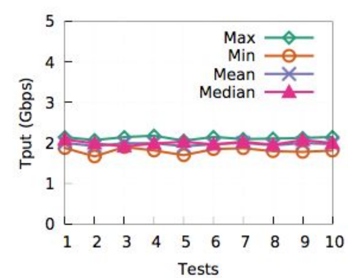


(c) AC⚡DC convergence test.

Fairness across ECN support

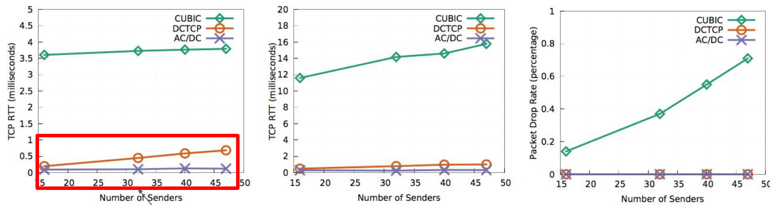


(a) All DCTCP.



(b) 5 different CCs (AC⚡DC).

Macrobenchmarks (Incast)



(a) 50[th] percentile RTT. (b) 99.9[th] percentile RTT. (c) Packet drop rate.
Figure 19: Many to one incast: RTT and packet drop rate. AC⚡DC can reduce DCTCP's RTT by limiting window sizes.

Why is AC/DC better?
- DCTCP has a lower bound on CWND, which is too high when under extreme congestion

Macrobenchmarks (Incast)

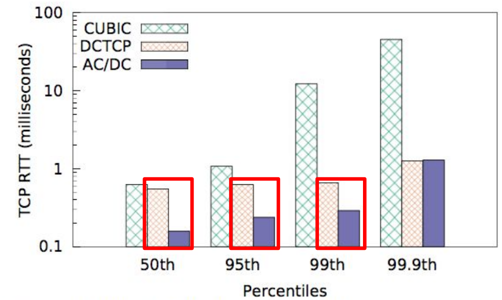
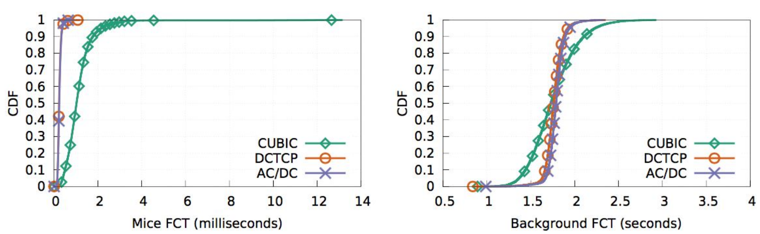


Figure 20: TCP RTT when almost all ports are congested.

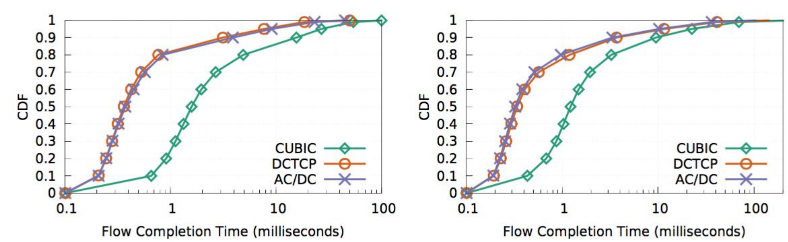
Macrobenchmarks (Stride)



(a) Mice flow completion times. (b) Background flow completion times.
Figure 21: CDF of mice and background FCTs in concurrent stride workload.

Note: Shuffle Not Shown (check out the paper)

Trace-Driven workload



(a) Web-search workload. (b) Data-mining workload.
Figure 23: CDF of mice (flows < 10KB) FCT in web-search and data-mining workloads.

Summary

- Operators need control of their networks to improve data center performance, despite diverse tenants running arbitrary networking stacks
- AC/DC allows operators to control the TCP congestion control algorithm of arbitrary tenants by implementing congestion control at the vSwitch
- AC/DC has the performance of specialized transport layer protocols like DCTCP without requiring tenant adoption, new networking hardware or software

Discussion

- What about distributed vSwitches?
- “cannot force an application to send more data than the VM’s CWND allows”
  - AC/DC increases traffic b/c TCP only reduces CWND on loss or ECN feedback
  - In other words, CWND of sender is always less than AC/DC’s RWND
  - Will this type of approach work on other protocols? (UDP) Does it need to?
- Operating on Datapath… didn’t we just learn that kernel is bottleneck on datapath!?
  - what is AC/DC overhead beneath an optimized networking stacks (Arrakis, IX)?