Data Center TCP (DCTCP)

Networking Services Team, Red Hat

Florian Westphal
1024D/F260502D fw@strlen.de
1C81 1AD5 EA8F 3047 7555
E8EE 5E2F DA6C F260 502D

February 6th 2015

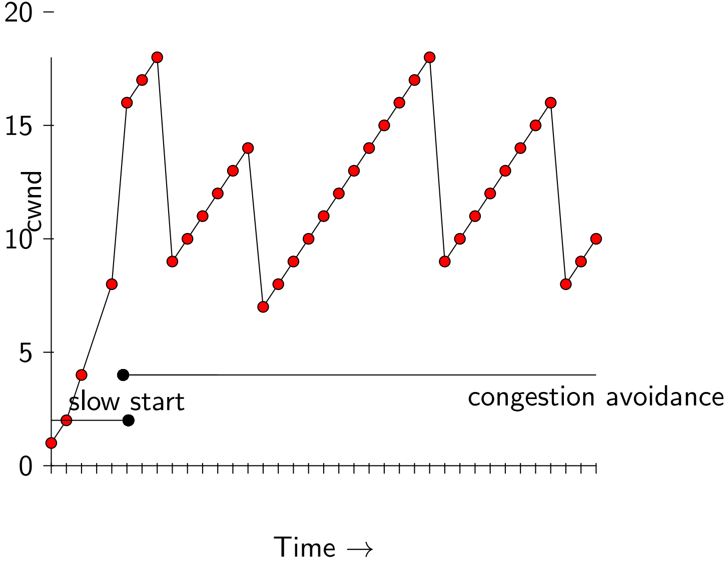
TCP Congestion Control – History

Did not exist in the early days.

- RFC 793: "[..] segments may be lost due to [..] network congestion, TCP uses retransmission (after a timeout) to ensure delivery.."
- Sender transmits as much data it has to send & current `rwnd` allows
- Massive problems, ca. 1986: → congestion collapse: most packets in networks were retransmits
- RFC 2581 – TCP congestion control
  - Slow Start, Congestion avoidance: `cwnd` – sender imposed flow control
  - Fast Retransmit & Fast Recovery

redhat.

Congestion Control (simplified, Reno)



Congestion Control Issues

- Not trivial to decide when to grow/shrink cwnd value
  - Link might have large delay
  - Packet reordering does happen
  - Even if packets have been lost: not neccessarily due to congestion
  - Available network capacity is not constant
- Active research topic, dozens of different algorithms

Linux TCP + Congestion Control: Architecture

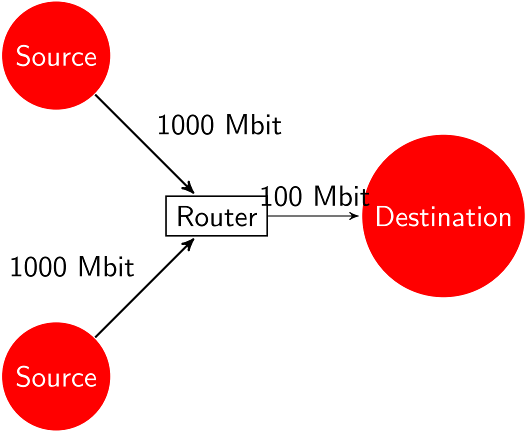
- Many different congestion control algorithms
- Default: CUBIC (since 2006)
- Plugin-based congestion control framework
- Different algorithms give different weight to the available information (e.g. rtt, duplicate acks, rwin, ...):
  - Hybla: don't penalize connections with high rtt
  - Veno: less aggressive cwnd decrease on loss
  - ...

Linux TCP + Congestion Control: Configuration

- sysctl `net.ipv4.tcp_congestion_control=vegas`
- `ip route add $dst dev $dev congctl vegas`[1]
- `setsockopt(.., TCP_CONGESTION, "vegas", ..);`
- Every tcp connection has exactly one assigned algorithm
- But individual connection can each use different one

[1] 3.20, `http://git.kernel.org/cgit/linux/kernel/git/davem/net-next.git/commit/?id=81164413ad096bafe8ad1068f3f095a7dd081d8b`

Congestion Control – Queueing



ingress: median arrival rate in a given interval

ingress > Egress: queue starts to form

Issues for senders

- Bufferbloat: too big buffers take sender longer to realize loss when it occurs
- Too small buffers can cause needless loss during short bursts
- Simple "drop when buffers are full" can affect many flows $\rightarrow$ global synchronization, incast

Wishlist

- Want to know about loss asap
- Active queue management in switches/routers
  - e.g. RED: drop with increasing probability once buffers start filling up
  - also: sfb, codel, fq_codel, choke, . . .

Ideally . . .

Allow detection of imminent congestion before loss occurs redhat

Explicit Congestion Notification (ECN)

- Extension to IP to allow routers/switches to signal congestion before packets are dropped
- Uses two bits in the IPv4 header TOS octet, 3 states:
  1. ECN-unaware (00)
  2. Two ECN-Nonces, 01 and 10 – ECN-aware
  3. Congestion experienced (11)
- receiver can detect when congestion occurs
- but only sender could do something about it

ECN & TCP

- Two new TCP header flags: ECN-echo & Congestion Window Reduced
- ECE: Used by receiver to inform sender that it received CE-marked packet
- CWR: Used by sender to tell Receiver that congestion window was reduced
- Use is "negotiated" during three-way-handhake

To enable on Linux:

`net.ipv4.tcp_ecn=1` or

`ip route change 192.168.2.0/24 dev eth0 features ecn`[2]

[2] 3.20, `http://git.kernel.org/cgit/linux/kernel/git/davem/net-next.git/commit/?id=f7b3bec6f5167efaf56b756abfafb924cb1d3050`

ECN issues

- bugs in middleboxes (e.g. firewalls, tcp ”accelerators“, etc):
  - ECN Blackholes: packets with SYN+CWR+ECE bits set are dropped
  - All packets might get CE marked (even more frequent with ipv6).
  - Even if signalling would work: proper marking (virtually) never happens
- Design:
  - Doesn’t quantify the extent of the congestion, only presence

lots of pain for little gain

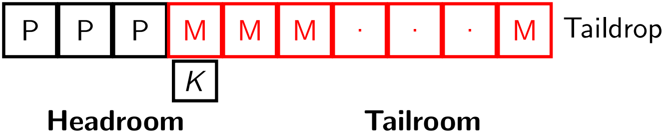
. . . And thus virtually no default-on

Summary

- Current tcp stacks are very good at detecting loss & loss recovery
- But loss still bad for latency
- ECN supported by all major OS and switch firmware
- Problematic due to myriad of implementation bugs / misconfigurations
- But if you have full control over all nodes involved (e.g. within datacenter . . .)

Datacenter TCP

- Designed as improvement to TCP Congestion Control for DC traffic
  1. High burst tolerance (incast due to to partition/aggregate)
  2. Low latency (short flows, queries)
  3. High throughput (large file transfers)
- ECN is used to estimate amount of bytes that experienced congestion (i.e., extent, not just presence)



enough tolerance to absorb bursts

Suggested mark threshold k for 10Gbit Ethernet: 65 packets ($\approx$ 100KB)

DCTCP: congestion estimate

`SND.UNA`, `SND.NXT` used as 'observation window'
Add counters for marked and total bytes
for each acceptable ack:

1. Count the bytes acked
2. If ack has ECE set, also count those bytes as "marked"
3. If `SND.UNA` not yet reached, stop; else update alpha:
   1. Compute F: $(\frac{marked}{total})$
   2. Compute $\alpha_{new} = (1 - g) * \alpha_{prev} + g * F$
   3. Start new observation window, valid until current `SND.NXT` acknowledged

- F fraction of packets marked in last window
- g is weight given to new samples (default: $\frac{1}{16}$)

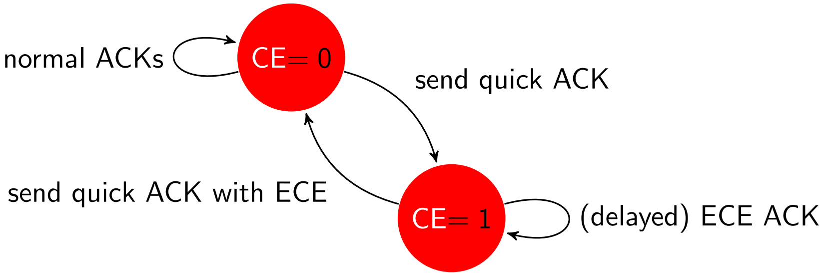
DCTCP: cwnd computation

α represents fraction of marked packets
Congestion window is computated as follows:

$$cwnd_{new} = cwnd_{prev} * 1 - \frac{\alpha}{2}$$

- $\alpha \approx 0$ little/no congestion, $\alpha \approx 1$: high/full congestion
- Everything depends on realistic estimate of the marked bytes
- How to infer when one of our data packets was marked?
- Simple and wrong solution: send ack for every single packet

DCTCP: ACK generation state machinery



quickacks are only sent when state changes.

DCTCP: Implementation

- DCTCP congestion control module
- Stack was extended to provide a couple of more events to modules
- CC modules can now indicate (force) ECN
- Fallback to Reno CC if peer doesn't support ECN
- Easiest way to enable:
  `ip route change dev eth0 10.0.0.7/24 congctl dctcp`

redhat.

DCTCP: Operation

- Read `Documentation/networking/dctcp.txt`
- Suggest to only enable it for local network(s), not globally:
  `ip route ... congctl dctcp`
- Don't need extra ecn-tuning on end-hosts, ecn will be used automatically

```
$ ss -nite
Send-Q  Local Address:Port Peer Address:Port
12408   192.168.7.10:22    192.168.7.1:35274 [..]
 dctcp [..] ce_state 0 alpha 312 ab_ecn 2896 ab_tot 0
```

DCTCP: Problems

```
$ ss -nite
Send-Q  Local Address:Port Peer Address:Port
12408   192.168.7.10:22    192.168.7.1:35274 [..]
 dctcp [..] ce_state 0 alpha 312 ab_ecn 2896 ab_tot 0
```

- `dctcp-reno`: fallback mode: other host using e.g. CUBIC with ecn off
- `alpha`: if large (max 1024): huge congestion or middlebox marking all packets

DCTCP: results from data center deployment[4]

Latency (in ms):

	CUBIC	DCTCP
Mean	4.0088	0.04219
Median	4.055	0.0395
Max	4.2	0.085
Min	3.32	0.028
Stddev	0.1666	0.01064

Throughput[3], in Mbps:

	CUBIC	DCTCP
Mean	521.684	521.895
Median	464	523
Max	776	527
Min	403	519
Stddev	105.891	2.601

[3] per flow, 19 senders in parallel

[4] `http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=e3118e8359bb7`

DCTCP: Issues

- Must configure all routers/switches to mark at k
- Must separate DCTCP and TCP traffic on switches (e.g. via DSCP marking) to maintain fairness
- Pure ACK loss breaks congestion estimate
- Both paper and ietf draft are not clear on a few details, e.g.
  - Should α be changed on loss?
  - ...only on timeout?

Questions?

Bibliography

- M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan:
  Data Center TCP (DCTCP), Data Center Networks session
  Proc. ACM SIGCOMM, New Delhi, 2010.
  `http://simula.stanford.edu/~alizade/Site/DCTCP_files/dctcp-final.pdf`
- S. Bensley, L. Eggert, D. Thaler
  Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters
  `http://tools.ietf.org/html/draft-bensley-tcpm-dctcp-02`