Create Design Code Build for everyone

Rethink e2e congestion control

Yuchung Cheng
What’s this lecture is about

- Quick BBR overview plus one interesting detail
- Why conventional congestion control like BBR is not enough
- Many problems I’d like you to help
My path to Google

https://ai.google/research/people/author27276/

IETF, Linux netdev, TCP, C.C., QUIC, ...
BBR: the story goes back to 2013...

Many Google services complained about TCP performance

- Internal backbone TCP throughput often < 10Mbps
- Youtube.com: terrible video quality sometimes, with RTT > 10 secs
- Google.com: poor latency in developing regions
- Google TCP congestion control was CUBIC (Linux default)
  - Packet loss is the sole signal
  - Services started to “work around” TCP
    - Use parallel connections, tweak TCP knobs, add more buffer to the network,
The problem: loss-based congestion control

  - Keeps sending faster until it sees a loss

- But packet loss alone is not a good proxy for congestion

- If loss comes before congestion, loss-based CC gets low throughput
  - 10Gbps over 100ms RTT needs <0.000003% packet loss (infeasible)
  - 1% loss (feasible) over 100ms RTT gets only 3Mbps

- If loss comes after congestion, loss-based CC bloats buffers, suffers high delays
Network congestion and bottlenecks
Loss-based congestion control in deep buffers

Loss-based CC (CUBIC / Reno)
Loss-based congestion control in shallow buffers

Multiplicative Decrease upon random burst losses

=> Poor utilization

Loss-based CC (CUBIC / Reno)
Optimal operating point

Optimal: max BW and min RTT (Kleinrock)

Delivery rate

RTT

BDP

amount in flight

BDP+BufSize
Estimating optimal point (max BW, min RTT)

BDP = (max BW) * (min RTT)

Est min RTT = windowed min of RTT samples

Est max BW = windowed max of BW samples
To see max BW, min RTT: probe both sides of BDP

- Only min RTT is visible
- Only max BW is visible
The devil is in the details...
TCP Data and ACK Aggregation (batch & burst)

- Commonly used for amortizing overheads to increase efficiency
  - In interrupt processing and hardware/software offload mechanisms (TSO, GRO)
  - In shared media like wifi, cellular, cable modems

- ACK aggregation severely limited throughput in initial BBR release
Full wifi trace: receiver

- Data (mostly) arrives smoothly
- TCP enqueues ACKs smoothly to wifi device
- Flat spots are from sender being cwnd-limited or rwnd-limited

later slides zoom in here
trace on TCP BBR **sender**

- Silences in ACK stream cause sender to be cwnd-limited or rwnd-limited
- Then ACKs arrive aggregated in bursts at sender
Zoomed-in wifi trace: sender

- Sender exhausts cwnd due to long gap in ACK stream

trace on TCP BBR sender

cwnd exhausted

burst of aggregated ACKs

long gap in ACK stream ...causes cwnd exhaustion
Zoomed in wifi trace: receiver

- Big aggregation visible to sender does not show up here

- Link underutilized due to sender cwnd exhaustion

- Wifi AP data transmission monopolizing link?
  - Data arrival is smooth
  - ACK generation is smooth
  - ACK transmission (previous slide) paused
Wifi min_rtt is far from the "typical" RTT
Wifi RTT samples are very noisy; e.g.:
RTT: 4ms to 80ms

Per-ACK RTT samples (ms)
1 TCP BBR sender active on wifi LAN
1 1G Ethernet hop; 1 802.11ac wifi hop
Solution

Provision cwnd based on the degree of ACK aggregation (extra_acked)

extra_acked = excess data ACKed beyond expected amount over this interval
            = actual_acked - estimated_BW * interval

cwnd = estimated_BW * min_RTT + max_filter(extra_acked, len=10RTT)
BBR aggregation estimator: visualizing the dynamics

Same zoomed-in sender-side wifi trace

extra_acked = actual_acked - expected_acked

= actual_acked - estimated_bw*interval
Discussion: pros & cons of congestion signals: loss, delay, BW, ECN
Lessons learned from developing BBR

- Any algorithm can only work well with clear feedbacks
  - Delay often not reflecting queuing on the Internet
  - Loss is flaky if induced by transient burst
  - ECN is iffy if routers mark them differently
  - Bandwidth is highly dynamic and vulnerable to ACK compression

- The Internet is (and will always be) full of tricks
  - AQMs & policers assume TCP always backs off on drops
  - Middle-boxes delay (or even delete) ACKs for efficiency
  - ECN causes severe reorderings in old routers

- Any C.C. needs continuous refinement to perform well
  - Major performance bugs from unexpected places: delayed ACK, sender and receiver segment offload, socket options, even kernel memory management
Problem w sender-based congestion control

http://nytimes.com on iPhone 8  LTE

39 connections
106 requests
3MB total

39 senders performs C.C. independently at packet level

Receiver-driven congestion control at the browser to prioritize important objects?

https://www.webpagetest.org/
New landscape of congestion control
Google’s Network

- 20 Cloud Regions
- 125 POPs
- 90 CDN locations
- 72 Dedicated Interconnect locations
- 15 Google data centers
- 100,000s of miles of fiber optic cable
Scaling: data center network bandwidth growth

Traffic generated by servers in Google data centers

Aggregate traffic

Time
How would you design the e2e congestion control for this network?

- BW isolation & admission
- Network priorities
- WAN & LAN competition
- Performance
- Visibility & debuggability
- Incremental deployment
Problem: a lot of flows to control

Modern application communicate using many flows in highly correlated bursts

- Sometimes hundreds of millions flows of a single Google job
- Flows are mostly idle and burst in cascaded waves
- Feedback received per flow is too little and out-dated on next burst
Discussion

Do per-flow congestion control and fairness model make sense?
Problem: bursts are big and buffers are tiny

Server: many cores w/ multi Gbps NICs

Switch: MBs of buffers shared by tens of ports

RTT: <100 usecs

BDP of 10Tbps * 100 usec =≈ 833K packets (w/ 1.5KB MTU)

Often available BDP per (active) flow < 1 packet
De-burst by pacing: clock out packets precisely

Compute Earliest Departure Time based on shaping rate

Enqueue Packet in Timing Wheel

Dequeue packet and deliver completion

To NIC

Shaper

Timestamp
Design Principles of Carousel

1) Single, $O(1)$, Time-Indexed Queue, ordered by packet timestamps.
2) Apply Backpressure.

1) Single, $O(1)$, Time-Indexed Queue, ordered by packet timestamps.
1) Single, $O(1)$, Time-Indexed Queue, ordered by packet timestamps.

3) One Shaper per Core.

2) Apply Backpressure.
Discussion

pros & cons of pacing at packet granularity?
TCP congestion control is the last resort: data is at the door and expected to be sent ASAP.

TCP / Congestion Control

+ HoL blocking in the kernel.
+ No information about congestion control algorithm.
+ No visibility on short-lived jobs.

Often considered the “network” delay.
Move congestion control up

Transport is good at learning network and congestion details

Application schedules RPCs w/ informed network congestion and latencies

E.g. send higher priority RPCs and pause lower priority ones if congested
Problem: root-causing congestion control

Congestion is easy to detect and locate: drops, delay, ECN

Congestion control is hard to reason: why is my congestion window 2 pkt

Applications challenge evidence of slow-down due to congestion, but not bad congestion control

This gets harder when flows carry different network priorities
Problem: predictability is overlooked

Application care tail latency often because of predictability not performance

Highly predictable communication operations greatly simplify application designs

Halving mean latency is often moot if 99p doubles

How can congestion control provide highly predictable latency?

Potential direction: a tightly coupled BW admission control, congestion control, and job placement scheduler

1. Learn and build models of application behaviors
2. Schedule compute and storage with budgeted BW
Congestion control’s new mission in future networks

- Diagnosability to evaluate its performance
- Proactive efforts to prevent congestion
- Visibility of the congestion for app and ops
- Proper isolation for predictability
Thank you.

Questions?

Contact: ycheng@google.com