CS 514: Computer Networks
Lecture 7: Other Congestion Control Algorithms

Xiaowei Yang
Overview

• Other congestion algorithms
• TCP weaknesses
  – Slow convergence for large bandwidth-delay product networks
  – Unfairness among flows with different RTTs
  – Loss-based congestion detection may cause buffer bloat
• Solutions
  – XCP: an ideal solution
  – Other more practical solutions
One more bit is enough

• Variable Structure Congestion Control Protocol

• Key idea
  – Four bits to signal regions of action
  – 01: low load MI
  – 10: high load AI
  – 11: overload MD
- TCP uses binary congestion signals, such as loss or one-bit Explicit Congestion Notification (ECN).

- AI with a fixed step-size can be very slow for large bandwidth.
Key observation

Fairness is not critical in low-utilization region

- Use Multiplicative Increase (MI) for fast convergence onto efficiency in this region
- Handle fairness in high-utilization region
Variable structure control protocol

- Routers signal the level of congestion
- End-hosts adapt the control algorithm accordingly

Control

- Multiplicative Decrease (MD)
- Additive Increase (AI)
- Multiplicative Increase (MI)

Load factor

- Overload: (11)
- High-load: (10)
- Low-load: (01)

Traffic rate

2-bit ECN

Sender

Router

ACK

Receiver

Scale-free

Range of interest
- Use network link load factor as the congestion signal
- Decouple efficiency and fairness controls in different load regions
- Achieve high efficiency, low loss, and small queue
- Fairness model is similar to TCP:
  - Long flows get lower bandwidth than in XCP (proportional vs. max-min fairness)
  - Fairness convergence much slower than XCP (solvable with even more, e.g., 8 bits)
CUBIC: a new TCP-friendly high-speed TCP variant

by S. HaNorth, I. Rhee, and L. Xu
TCP congestion control

- Additive Increase Multiplicative Decrease
- for every ACK received
  - cwnd += 1/cwnd
  - cwnd measured in number of MSSes)
- For every packet lost
  - cwnd /= 2
TCP Cubic

- Implemented in Linux kernel and Windows 10
- Key features
  - Increase window sizes by real time rather than ACK driven
  - Faster window growth after a packet loss
(b) CUBIC window growth function.

\[ W(t) = C(t - K)^3 + W_{max} \]

\[ K = \sqrt[3]{\frac{W_{max} \beta}{C}} \]
Does it converge?

- Efficiency
- Fairness
  - Larger windows reduce more
  - And increase slowly
    - -K
Congestion-based Congestion Control

By Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson
Highlights

• Measuring Bottleneck Bandwidth And Round-Trip Propagation Time (BBR)

• Not widely deployed

• Private communication revealed it didn’t perform as well as CUBIC
Goal

• Achieving efficiency and fairness with a small queue at the bottleneck
  – Is it even possible?
  – At least one research paper (citation 14) claimed it was impossible
Congestion and bottlenecks

- A flow has exactly one slowest link
- It determines the maximum sending rate of the flow
- Persistent queues form
A flow’s physical constraints

- Round trip propagation delay
  - How fast data travel inside the links

- Available bandwidth at the bottleneck

- Delay $\times$ bandwidth = delay-bandwidth product (BDP)
An analogy

• “If the network path were a physical pipe, RTprop would be its length and BtlBw its minimum diameter.”

• The amount of data a flow can send is the amount of water the pipe can hold
FIGURE 1: DELIVERY RATE AND ROUND-TRIP TIME VS. INFLIGHT

- **BDP**
  - App limited
  - Bandwidth limited

- **BDP+ BtlneckBufSize**
  - Buffer limited

- **Round-trip time**
  - RTprop
  - Slope = 1 / BtlBw

- **Delivery rate**
  - Slope = 1 / RTprop

- **Amount inflight**
  - Optimum operating point is here
  - Loss-based congestion control operates here
How to achieve the goal

- Obtain link’s physical constraints
  - RTProp
  - BtlBw

- And send the BDP amount of data per RTProp time

- So a connection can send at its highest throughput with lowest latency
How to obtain RTprop

- At any time $t$, the measured RTT is

$$RTT_t = RT_{prop t} + \eta_t$$

- Using the minimum measurement over a time window to estimate

$$\hat{RT}_{prop} = RT_{prop} + \min(\eta_t) = \min(RTT_t) \quad \forall t \in [T - W_R, T]$$
How to estimate BtlBw

• Actually delivery rate cannot exceed the available bottleneck bandwidth

• Estimating the average delivery rate as the ratio of data sent over data acked: \( \text{deliveryRate} = \frac{\Delta \text{delivered}}{\Delta t} \)

• How to estimate \( \Delta t \)?
Comments

- Paper argues the measured time interval must be greater than the true arrival interval
- May not hold if ACK is compressed
The BtlBw estimate

• Since \( \text{deliveryRate} = \frac{\Delta \text{delivered}}{\Delta t} \leq \text{the bottleneck rate} \)

• And \( \Delta t \geq \text{the true arrival interval} \)

• It follows that
  – the bottleneck rate \( \geq \text{deliveryRate} \)

\[
\hat{\text{BtlBw}} = \max(\text{deliveryRate}_t) \quad \forall t \in [T - W_B, T]
\]
function onAck(packet)
    rtt = now - packet.sendtime
    update_min_filter(RTpropFilter, rtt)
    delivered += packet.size
    delivered_time = now
    deliveryRate = (delivered - packet.delivered) /
    (delivered_time - packet.delivered_time)
    if (deliveryRate > BtlBwFilter.currentMax || !
        packet.app_limited)
        update_max_filter(BtlBwFilter, deliveryRate)
    if (app_limited_until > 0)
        app_limited_until = app_limited_until - packet.size

• Each ack provides new RTT and average delivery rate measurements that update the RTprop and BtlBw estimates
How to send data

• Send data if the inflight data is less than BDP * a small gain

• Pace data to match the bottleneck bandwidth limit
How to send data

function send(packet)
    bdp = BtlBwFilter.currentMax × RTpropFilter.currentMin
    if (inflight >= cwnd_gain × bdp)
        // wait for ack or retransmission timeout
        return
    if (now >= nextSendTime)
        packet = nextPacketToSend()
        if (! packet)
            app_limited_until = inflight
            return
        packet.app_limited = (app_limited_until > 0)
        packet.sendtime = now
        packet.delivered = delivered
        packet.delivered_time = delivered_time
        ship(packet)
        nextSendTime = now + packet.size / (pacing_gain × BtlBwFilter.currentMax)
        timerCallbackAt(send, nextSendTime)
Steady state behavior
Comparison with a CUBIC Sender

**FIGURE 4: FIRST SECOND OF A 10-MBPS, 40-MS BBR FLOW**

- **startup**
- **drain**
- **probe BW**

- **data sent oracked (MB)**
- **time (sec.)**

- **RTT (ms)**
- **time (sec.)**

- **cwnd_gain clamps**
- **BBR inflight at 3 BDP**
- **CUBIC switches from exponential to linear inflight growth**
- **RTprop**
- **BBR operating at full BW with no queue**
Deployment

• Deployed at B4, Google’s wide area network
• Since 2016, all B4 traffic uses BBR
• BBR's throughput is consistently 2 to 25 times greater than CUBIC’s.
• Raising the receive buffer on one US-Europe path
  – BBR → 2 Gbps
  – CUBIC → 15 Mbps — the 133x relative improvement
Summary of BBR

• Goal is to send as fast as possible without building up a persistent queue

• Methods
  – Measuring RTprop & BtlBw
  – Limiting inflight data to a small multiple of BDP
Conclusion

• Discussed a few TCP variants
• We can modify the control laws to improve TCP’s performance
  – VCP
  – CUBIC
  – BBR
• Prior research may be proven wrong later
• Hopefully we can discuss QUIC later
The QUIC Transport Protocol: Design and Internet-Scale Deployment

By Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, Zhongyi Shi *
What’s QUIC

• Google’s latest HTTPs transport
• Can plug in various congestion control algorithms