Application Congestion Control

July 31, 2018

Ian Swett

Background

QUIC is a new transport
QUIC is used for user-facing flows
Nothing here is specific to QUIC

How many tabs do you have open?

4?

How many tabs do you have open?

4?

10?

How many tabs do you have open?

4?

10?

Me: 42

How many connections for a site?

1

How many connections for a site?

1

Maybe 3?

How many connections for a site?

1

Maybe 3?

YouTube: 13!

How many of those are fully-utilized?

1

How many of those are fully-utilized?

1

Or maybe 5?

How many of those are fully-utilized?

1

Or maybe 5?

All the possibilities

Thesis: App-limited is the norm…
Full utilization is the exception…

Types of traffic: Typical Applications

Tiny flows	Bursty flows	Long Fat flows	Realtime
Most Web Traffic	Video playbacks Image-heavy sites(ie: Maps)	Large Uploads and Downloads Datacenter transfers	WebRTC

Types of traffic: Typical Application Behavior

Tiny flows	Bursty flows	Long Fat flows	Realtime
Most Web Traffic	Video playbacks Image-heavy sites(ie: Maps)	Large Uploads and Downloads Datacenter transfers	WebRTC
Always App-Limited	**Intermittently App-Limited**	**Never App-Limited**	**Constantly on the edge**

Types of traffic: Operating Mode

Tiny flows	Bursty flows	Long Fat flows	Realtime
Loss Recovery, not Congestion Control	Approximates a long fat flow for a short period	Congestion Control Wheelhouse!	Constantly using the pipe Very poor information Bidirectional nature can cause RTT bloat

Goal: One congestion control that ‘just works’

What if we ignore it?

Window-based 1 loss can collapse CWND

Bandwidth-based 1 small request/response collapses bandwidth

- Even if there's no congestion!

What if we ignore loss/bw when app-limited?

Window-based No reaction to loss!

Bandwidth-based Discard a massive amount of bandwidth information

- Can go minutes without a ‘valid’ bandwidth sample
- When you DO finally get a sample, it may not be ‘good’

What if we ignore loss/bw when app-limited?

Window-based No reaction to loss!

Bandwidth-based Discard a massive amount of bandwidth information

- Can go minutes without a ‘valid’ bandwidth sample
- When you DO finally get a sample, it may not be ‘good’

What if we treat it as imperfect information?

Window-based Some reaction to loss

Bandwidth-based Increases error margins on the bandwidth estimate

- No longer sensitive to “one bad sample”
- Open area for research

Thought:
Imperfect information and approximate models aren’t unique to congestion control

Behavioral Economics

Theory(classic)

Perfectly Rational

Convex Utility Function

Perfect Information

Behavioral Economics

Theory(classic)	Practice
Perfectly Rational	Predictably Irrational
Convex Utility Function	Unknown Utility Function
Perfect Information	Limited information
	Even more limited attention span

Congestion control

Theory(classic)

Never App-Limited

Loss indicates congestion you're causing

Packets are acked two at a time

Buffers are ~=1 BDP

Congestion control

Theory(classic)	Practice
Never App-Limited	Constantly App-Limited
Loss indicates congestion you’re causing	Loss and RTT are not perfect signs of congestion
Packets are acked two at a time	Aggregation is constant
Buffers are ~=1 BDP	Buffers vary wildly

Hope:
Don’t treat application limited behavior as an afterthought

Backup Slides

Fun Pathological Issues: Cubic ‘bug’

Cubic has a time-fairness component

Slow start after idle is commonly disabled

=> Both QUIC and Linux TCP forgot to turn off time-fairness when SSAI was disabled

=> CWND rapidly increases after idle!

=> Huge losses for YouTube traffic

Fun Pathological Issues: WebRTC Startup

STARTUP + App-Limited + Bidi PROBE_RTT =

Runaway Buffer Bloat!

Why? It’s complicated

Why not Reno or Cubic for WebRTC?

- Buffer bloat
- Rapid reductions(ie: 30%, 50%) in sending rate are hard to adapt to.

Why not use BBR1 WebRTC?

- App-limited prevents exit from STARTUP
- App-limited behavior can create bandwidth crashes, keep under-filling the pipe
- Discard all app-limited samples, almost all info
- PROBE_RTT is too severe