Congestion control, generalized

Ankit Singla

ETH Zürich Spring 2017

[Some slides are adapted from Mo Dong’s PCC presentation at NSDI ’15]
Learning goals

• TCP is not good enough (again)
• How do we fix it?!
• Does packet loss really mean congestion?
Congestion

Problem: what rate to send at?
The right rate is important

Rate > available capacity …
Packet loss and delay
Unfair capacity distribution

Rate < available capacity …
Inefficient use of the network
Probing the capacity
Probing the capacity
Feedback control loop

1. Send packets
2. Wait
3. ACKs return in an RTT (or not)
4. Adjust sending rate

The network as a black box
Loss-reactive congestion control

All packets ACK-ed …

Increase rate!

Some packets presumed lost …

Decrease rate!
TCP (extremely briefly)

Additive increase

Slow-start

Multiplicative decrease

Sending rate

Transmission number
Problems with TCP

- Congestion window
- Transmission number
- Additive increase
- Slow-start
- Multiplicative decrease

Sending rate vs. Transmission number
Problems with TCP

- Congestion window
- Additive increase
- Slow-start
- Reacting to losses
- Multiplicative decrease

Sending rate vs. Transmission number graph.
Long queues $\Rightarrow$ queueing delays
## Generic (whatever)-TCP strategy

<table>
<thead>
<tr>
<th>Low-level event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reno 1 pkt loss</td>
<td>cwnd/2</td>
</tr>
<tr>
<td>Scalable ACK</td>
<td>cwnd+1</td>
</tr>
<tr>
<td>CUBIC Time pass 1ms</td>
<td>cwnd+f(t,cwn,rtt)</td>
</tr>
<tr>
<td>FAST RTT increase x%</td>
<td>Reduce cwnd to f(x)%</td>
</tr>
<tr>
<td>HTCP 100 ACK</td>
<td>cwnd+f(cwnd)/cwnd</td>
</tr>
</tbody>
</table>

**Hard-wired mappings: low-level events to control actions**
Generic (whatever)-TCP strategy

<table>
<thead>
<tr>
<th>Low-level event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 pkt loss</td>
<td>cwnd/2</td>
</tr>
<tr>
<td></td>
<td>ACK</td>
</tr>
<tr>
<td></td>
<td>cwnd+1</td>
</tr>
</tbody>
</table>

Real Network -> [Diagram]

Real Network
Generic (whatever)-TCP strategy

The event-action mappings encode a model of the network.
Problem: this model is often wrong!

<table>
<thead>
<tr>
<th>Event</th>
<th>Model</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet loss</td>
<td>I’m causing congestion</td>
<td></td>
</tr>
<tr>
<td>Shallow buffer overflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Congestion from other flows</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loss is random</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
TCP

Est. 1988

High BDP

Wireless

Satellite

Inter-DC

Intra-DC

BIC
H-TCP
Compound
CUBIC
FAST TCP

Westwood
Vegas
Veno

Hybla
STAR

Illinois
SABUL

ICTCP
DCTCP
Generalizing congestion control?

TCP ex Machina: Computer-Generated Congestion Control

Keith Winston and Hari Balakrishnan

PCC: Re-architecting Congestion Control for Consistent High Performance

Mo Dong*, Qingxi Li*, Doron Zarchy**, P. Brighten Godfrey*, and Michael Schapira**

BBR: Congestion-Based Congestion Control

BY NEAL CARDWELL, YUCHUNG CHENG, C. STEPHEN GUNN, SOHEIL HASSAS YEGANEH, AND VAN JACOBSON

* Developed by Computer Science and Artificial Intelligence Laboratory
** Developed by the University of California, Berkeley

This paper describes a new approach to end-to-end congestion control on a multi-user network. Rather than manually formulate each congestion control method, we developed a program called Remy that generates congestion control algorithms given knowledge or assumptions about the network and an objective that must be achieved. We characterize the sensitivity of the resulting performance to changes in these assumptions, including both engineered and computer-generated algorithms. Remy can generate algorithms both for networks where some prior knowledge is less precise, such as cellular networks, and for networks where some assumptions are very difficult to achieve such as very high-speed networks. We argue this is indeed a very difficult task within TCP's rate control architecture. TCP reacts to events that can be as simple as "one packet loss" (TCP New Reno) or involve multiple signals like "one packet loss and RTT increased by 100 ms".

We have studied congestion control, a fundamental problem in multi-user communication systems, for over thirty years. Our experiences have taught us that congestion originates at the senders of messages. The protocols developed so far allow the end systems to implement congestion control (DCCP, SCTP, the congestion control in the Internet). DCCP, SCTP, and the current best of breed, CUBIC, make use of a number of predefined control responses. TCP reacts to events that can be as simple as "one packet loss" (TCP New Reno). The result is a single numerical performance utility function that can be used to fully define the behavior of the algorithm.

TCP congestion control keeps them full, causing bufferbloat. When bottleneck buffers are small, loss-based congestion control misinterprets loss as a signal of traffic congestion, penalizes high-RTT flows, underutilizes high-speed links, and can collapse under data center incast. However, the performance-based congestion control methods? And just how well can we make computers perform this task?
TCP and its variants have suffered from surprisingly poor performance for decades. We argue the TCP family has to a fundamental architectural deficiency: hardwiring packet-level events to control responses. We propose Performance-oriented Congestion Control (PCC), a new congestion control architecture in which each sender continuously observes the connection between its sending and receiving parties. PCC's goal is to understand what rate control actions improve performance based on empirical experience, and then adaptively choose the rate that achieves consistently high performance over complex real-world network conditions.

In the roughly 25 years since its deployment, TCP's congestion control architecture has been notorious for degraded performance. TCP performs poorly on lossy links, penalizes high-RTT flows, underutilizes high bandwidth-delay product (BDP) connections, cannot handle rapidly changing networks, can collapse under data center incast [24] and incurs very high latency with bufferbloat [28] in the network. Protocol "patches" have addressed problems in specific network conditions such as high BDP wireless and lossy links [38, 39], and more. However, the fact that there are so many TCP variants suggests that each is only a point solution: they yield better performance in one scenario but are suboptimal in another. Indeed, using TCP variants' performance in many cases these TCP variants' performance is still far away from optimal even in the network conditions for which they are specially engineered.

As severe performance problems have accumulated over time, protocol "patches" have addressed problems in specific network conditions such as high BDP wireless and lossy links [38, 39], and more. However, the fact that there are so many TCP variants suggests that each is only a point solution: they yield better performance in one scenario but are suboptimal in another. Indeed, using TCP variants' performance in many cases these TCP variants' performance is still far away from optimal even in the network conditions for which they are specially engineered.

What is the right rate to send?
What is the right rate to send?

rate $r$ \rightarrow utility \, u

$U = f(\text{tpt, loss rate, latency, etc.})$

e.g. $U = \text{tpt} \times (1 - \text{loss rate})$
What is the right rate to send?

rate \( r_1 \) \( \rightarrow \) utility \( u_1 \)

\[ U = f(tpt, \text{loss rate, latency, etc.}) \]
\[ \text{e.g. } U = tpt \times (1 - \text{loss rate}) \]

No matter how complex the network, rate \( r \) \( \rightarrow \) utility \( u \)
PCC: control based on evidence

$r_1$ → $u_1$ → $r_1$

$r_2$ → $u_2$ → $r_2$

$u_1 > u_2$?

move to $r_1$

move to $r_2$
Performance-oriented congestion control

Observe real performance

Control based on empirical evidence

move to $r_1$

move to $r_2$

$u_1 > u_2$?
PCC: control based on evidence

This flow causing congestion

move to 98 Mbps

$U_1 > U_2$?

$U_1$

$U_2$
PCC: control based on evidence

random loss

\[ u_1 > u_2 \]?

move to 102 Mbps
Higher performance

- **4X** InterDC
- **17X** Satellite Networks
- **15X** Shallow Network Buffer
- **10X** Lossy Networks
- **Solves** RTT Unfairness
- Close to Optimal
- Rapidly Changing Networks
- Similar to ICTCP
- **5X in median** Global Commercial Internet
Software components

Sender

Performance Oriented Control Module

(Sending Rate, Utility)

Utility Function

Performance Metrics (tpt., loss rate, RTT)

Sending Rate Control

Sending Module

Sent Packet Log

Monitor Module

Data

SACK

Network

Receiver

Sent Packet Log

Performance Oriented Control Module

Sending Rate Control

Utility Function

Performance Metrics (tpt., loss rate, RTT)

Network

Sender

(Sending Rate, Utility)

Receiver

Data

SACK

Sent Packet Log

Performance Oriented Control Module

Sending Rate Control

Utility Function

Performance Metrics (tpt., loss rate, RTT)
Performance oriented control

\[ r = (1 - \epsilon) r (1 + \epsilon) \]

\[ \epsilon_{\text{min}} = 0.01 \]
Where is the congestion control?

Selfishly maximizing utility $\Rightarrow$ non-cooperative game

Does PCC converge to a fair Nash equilibrium?
Congestion control is in game theory

Some utility functions converge to a fair, efficient NE

\[ u_i(x) = T_i \text{observed loss rate} - x_i^* L_i \]

Cut off loss rate at 5%
Dynamic behavior and fairness

TCP uses AIMD for asymptotic fairness

Moving away from convergence
Dynamic behavior and fairness

PCC does not need AIMD because it looks at real performance

“Game Theory Force”

High utility
Dynamic behavior and fairness

500s Interval
2000s/each
100Mbps, 30ms
Dynamic behavior and fairness

TCP

PCC
Deployment

No hardwired support, packet header, protocol change needed

Where to deploy?

- CDN backbone, inter-data center, dedicated scientific network
- In the wild?
TCP friendliness

PCC's default utility function is not TCP friendly
**TCP friendliness**

Different utility functions could provide fairness

<table>
<thead>
<tr>
<th></th>
<th>30ms</th>
<th>60ms</th>
<th>90ms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>β = 10</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10Mbit/s</td>
<td>0.94</td>
<td>0.75</td>
<td>0.67</td>
</tr>
<tr>
<td>50Mbit/s</td>
<td>0.74</td>
<td>0.73</td>
<td>0.81</td>
</tr>
<tr>
<td>90Mbit/s</td>
<td>0.89</td>
<td>0.91</td>
<td>1.01</td>
</tr>
<tr>
<td><strong>β = 100</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10Mbit/s</td>
<td>0.71</td>
<td>0.58</td>
<td>0.63</td>
</tr>
<tr>
<td>50Mbit/s</td>
<td>0.56</td>
<td>0.58</td>
<td>0.54</td>
</tr>
<tr>
<td>90Mbit/s</td>
<td>0.63</td>
<td>0.62</td>
<td>0.88</td>
</tr>
</tbody>
</table>
### Performance: inter DC

Inter datacenter and dedicated high speed networks

<table>
<thead>
<tr>
<th>Transmission Pair</th>
<th>RTT</th>
<th>PCC</th>
<th>SABUL</th>
<th>CUBIC</th>
<th>Illinois</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPO → NYSERNet</td>
<td>12.1</td>
<td>818</td>
<td>563</td>
<td>129</td>
<td>326</td>
</tr>
<tr>
<td>GPO → Missouri</td>
<td>46.5</td>
<td>624</td>
<td>531</td>
<td>80.7</td>
<td>90.1</td>
</tr>
<tr>
<td>GPO → Illinois</td>
<td>35.4</td>
<td>766</td>
<td>664</td>
<td>84.5</td>
<td>102</td>
</tr>
<tr>
<td>NYSERNet → Missouri</td>
<td>47.4</td>
<td>816</td>
<td>662</td>
<td>108</td>
<td>109</td>
</tr>
<tr>
<td>Wisconsin → Illinois</td>
<td>9.01</td>
<td>801</td>
<td>700</td>
<td>547</td>
<td>562</td>
</tr>
<tr>
<td>GPO → Wisc.</td>
<td>38.0</td>
<td>783</td>
<td>487</td>
<td>79.3</td>
<td>120</td>
</tr>
<tr>
<td>NYSERNet → Wisc.</td>
<td>38.3</td>
<td>791</td>
<td>673</td>
<td>134</td>
<td>134</td>
</tr>
<tr>
<td>Missouri → Wisc.</td>
<td>20.9</td>
<td>807</td>
<td>698</td>
<td>259</td>
<td>262</td>
</tr>
<tr>
<td>NYSERNet → Illinois</td>
<td>36.1</td>
<td>808</td>
<td>674</td>
<td>141</td>
<td>141</td>
</tr>
</tbody>
</table>

123%
Performance: satellite network

WINDS System

42Mbps
0.74% packet loss
800ms RTT

Link capacity

TCP
TCP Hybla
TCP Illinois
TCP CUBIC
TCP New Reno

Throughput (Mbps)

Bottleneck buffer size (KB)
Performance: rapidly changing networks

BW: 10-100Mbps; RTT: 10-100ms; Loss Rate: 0-1%
Change every 5 seconds

Sending Rate (Mbps)

Time(s)
Performance: rapidly changing networks

BW: 10-100Mbps; RTT: 10-100ms; Loss Rate: 0-1%
Change every 5 seconds

Sending Rate (Mbps)

Time(s)
Performance: rapidly changing networks

BW: 10-100Mbps; RTT: 10-100ms; Loss Rate: 0-1%
Change every 5 seconds
PCC vs TCP vs Take a flight: 100 GB

- Utah, U.S. → Berlin, Germany
- Illinois, US → Waseda, Japan
- Georgia, US → Stockholm, Sweden
- Georgia, US → Ljubljana, Slovenia
- Missouri, US → Rennes, France
- Massachusetts US → Seoul, Korea

**Comparison Times (in minutes):**
- PCC: 0:0:00, 1:18:27:30, 3:12:55:00, 5:7:22:30, 7:1:50:00
- TCP CUBIC: 0:0:00, 1:18:27:30, 3:12:55:00, 5:7:22:30, 7:1:50:00
- Take a Flight: 0:0:00, 1:18:27:30, 3:12:55:00, 5:7:22:30, 7:1:50:00
Long list of things in paper ...

- More stories about the fact that TCP is broken
- Proof of Nash Equilibrium and Convergence
- Concrete Implementation of PCC
  - Performance monitoring
  - Details of learning control algorithm
  - Implementation designs and optimizations
- Performance Evaluation
  - Inter data center networks
  - Small buffer networks
  - Reactiveness and stability tradeoff
  - Jain index fairness
  - Benefit of Randomized Control Trials
  - Details of TCP friendliness evaluation
  - Emulated satellite networks
  - Emulated datacenter networks
  - Cure RTT unfairness
  - Does not fundamentally harm short flow FCT
  - Evaluation in the wild vs non-TCP protocols
- Flexibility by pluggable utility function
Different utility functions

Same rate control algorithm

+ 

Different utility function

= 

Flexibility
How do we operate at (nearly) zero congestion?
The target operational point

RTT vs. Data in flight

Throughput vs. Data in flight
The target operational point

RTT

Throughput

Data in flight
The target operational point

In-flight = RTT x Capacity
The target operational point

Kleinrock [1979]: This operating point is **optimal**!

Jaffe [1981]: … but **impossible for distributed algorithms**
How do we estimate RTT and Capacity?

BY NEAL CARDWELL, YUCHUNG CHENG, C. STEPHEN GUNN, SOHEIL HASSAS YEGANEH, AND VAN JACOBSON

BBR: Congestion-Based Congestion Control

ACM Queue, 2016
BBR: two target conditions

• Full pipe
  • \textit{in-flight} = \textit{min-RTT} \times \textit{bottleneck-bw}

• Rate balance
  • buffer fill rate = emptying rate
BBR: how it works

- BtlBw
  - Probe around [1, 1.25, 0.75] of current

- RTProp
  - (occasionally) drop rate to see an empty queue

- Needs pacing of packets!
BBR: how it works

- BtlBw doubled to 20Mbps
- BW estimate increases 1.95x (=1.25^3) in 3 cycles
- inflight (kB) vs. RTT (ms)

- BtlBw halved; inflight doesn’t fit in pipe, increasing RTT
- inflight reduction lowers RTT which lowers inflight...
- until optimum regained

BBR Congestion-Based Congestion Control, ACM Queue ‘16
Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson
BBR: what about the startup phase?

- BtlBw
  - like slow-start until the estimate plateaus (not just until loss)

- Drain queue created in startup
BBR: how it works

BBR Congestion-Based Congestion Control, ACM Queue '16
Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson
BBR: how it works

packet loss and recovery episodes

bottleneck’s 250 ms buffer limit
BBR: with many flows
BBR: under loss
BBR: in actual deployment (probes)
BBR: still not devoid of problems!

Current dynamics w/ with loss-based CC

CUBIC vs BBR goodput: bw = 10Mbps, RTT = 40ms, 4 min. bulk xfer, varying buffer sizes
BBR: still not devoid of problems!

**BBR multi-flow behavior: RTT fairness**

Compare the goodput of two competing BBR flows with short (A) and long (B) min_RTT

- **Flow B** (varying min_RTTs, start t = 2 sec)
- **Flow A** (min_RTT=10ms, start t = 0 sec)