Scalable TCP: Improving Performance in HighSpeed Wide Area Networks

PFLDnet 2003

CERN, Geneva

Tom Kelly

`ctk21@cam.ac.uk`

CERN

and

Laboratory for Communication Engineering

University of Cambridge

Motivation

- Poor performance of TCP in high bandwidth wide area networks due to TCP congestion control algorithm
  - for each ack in a RTT without loss:
    $cwnd_r \mapsto cwnd_r + \frac{1}{cwnd}$
  - for each window experiencing loss:
    $cwnd_r \mapsto cwnd_r - \frac{1}{2} cwnd_r$

Throughput	Window	Loss recovery time	Supporting loss rate
10Mbps	170pkts	17s	5.4×10^{-5}
100Mbps	1700pkts	2mins 50s	5.4×10^{-7}
1Gbps	17000pkts	28mins	5.4×10^{-9}
10Gbps	170000pkts	4hrs 43mins	5.4×10^{-11}

Characteristics of a 200ms, 1500 MTU TCP connection

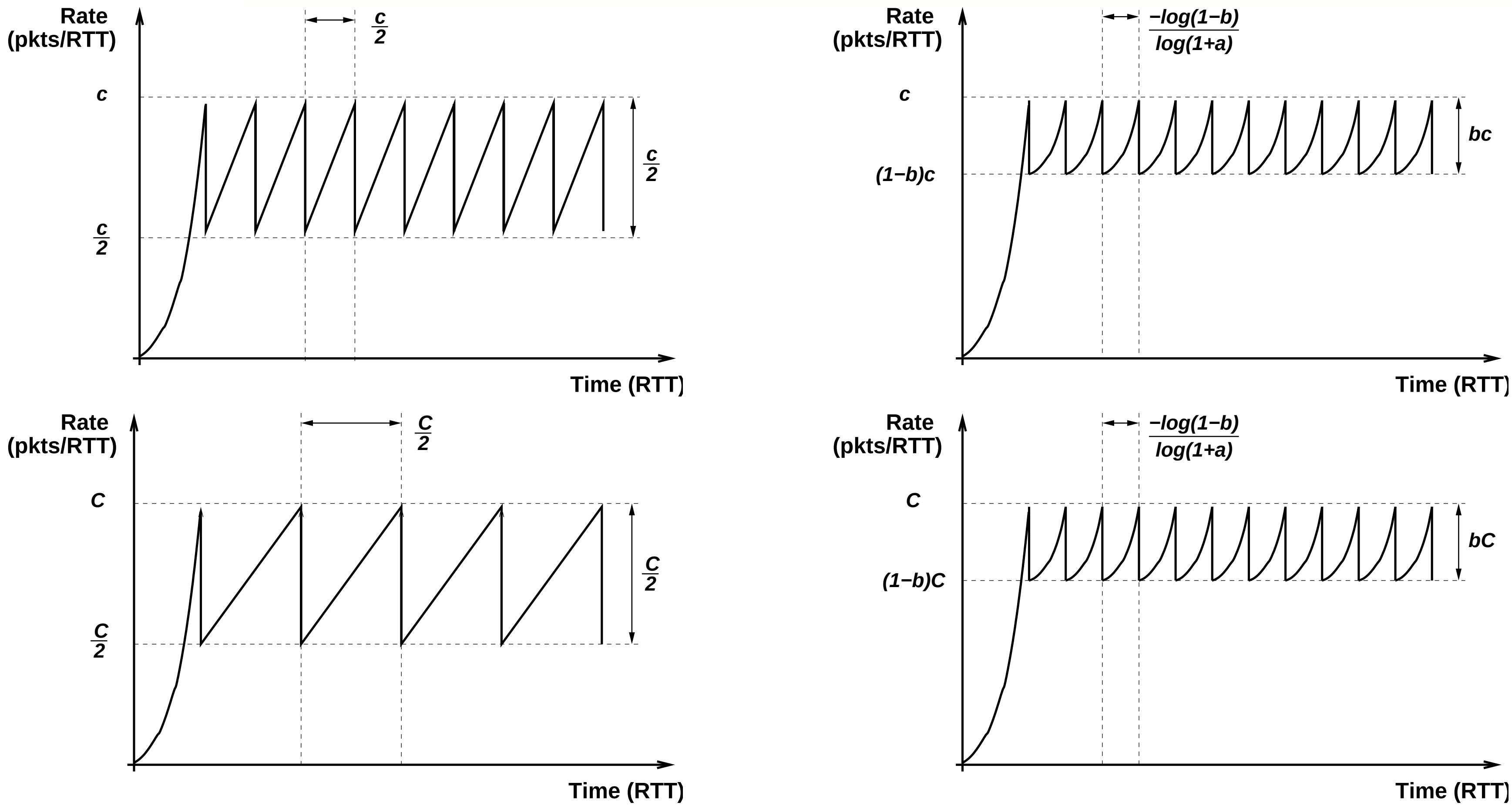
Changing congestion control - aims and assumptions

- Make effective use of high bandwidth links
- Changes need to be robust in a wide variety of networks and traffic conditions
  - L2 switches, bugs, packet corruption, reordering and jitter
- Do not adversely damage existing network traffic
- Do not require manual tuning to achieve reasonable performance
  - 80% of maximal performance for 95% of the people is fine

The generalised Scalable TCP algorithm

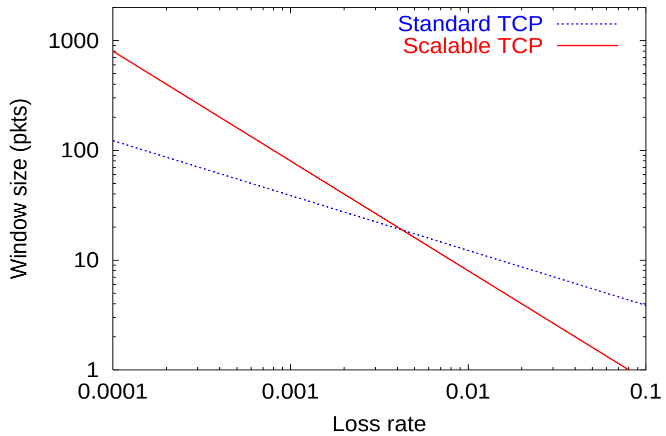
- Let a and b be constants and $cwnd$ be the congestion window
  - for each ack in a RTT without loss: $cwnd \mapsto cwnd + a$
  - for each window experiencing loss: $cwnd \mapsto cwnd - b \times cwnd$
- Loss recovery times for RTT 200ms and MTU 1500bytes
  - Scalable TCP: $\frac{log(1-b)}{log(1+a)}$ RTTs
    e.g. if $a = 0.01, b = 0.125$ then it is about 2.7s
  - Traditional: at 50Mbps about 1min 38s, at 500Mbps about 27min 47s!

The Scalable TCP algorithm



Fairness

- Choose a legacy window size, $lwnd$
- When $cwnd > lwnd$ use the Scalable TCP algorithm
- When $cwnd \leq lwnd$ use traditional TCP algorithm



- Same argument used in the HighSpeed TCP proposal
- Fixing $lwnd$, fixes the ratio $\frac{a}{b}$

Control Theoretic Stability

- Theorem (Vinnicombe): The generalised Scalable TCP algorithm is locally stable about equilibrium, if

$$a < \frac{p_j(\hat{y}_j)}{\hat{y}_j p'_j(\hat{y}_j)} \qquad \forall j \in J$$

  where $\hat{y}_j$ is the equilibrium rate at each link, $p_j(y)$ is the probability of loss at link j for an arrival rate y, and J is the set of all links

- With appropriate buffer sizes or AQM stability can be ensured

Variance and Convergence

b	a	Rate CoV	Loss recovery time	Rate halving time	Rate doubling time
$\frac{1}{2}$	$\frac{2}{50}$	0.50	$17.7T_r$ (3.54s)	T_r (0.20s)	$17.7T_r$ (3.54s)
$\frac{1}{4}$	$\frac{1}{50}$	0.35	$14.5T_r$ (2.91s)	$2.41T_r$ (0.48s)	$35T_r$ (7.00s)
$\frac{1}{8}$	$\frac{1}{100}$	0.25	$13.4T_r$ (2.68s)	$5.19T_r$ (1.04s)	$69.7T_r$ (13.9s)
$\frac{1}{16}$	$\frac{1}{200}$	0.18	$12.9T_r$ (2.59s)	$10.7T_r$ (2.15s)	$139T_r$ (27.8s)

Parameter choice and Implementation

- $lwnd = 16, a = 0.01, and b = 0.125$ represents a good trade off of concerns
- Patch against Linux 2.4.19 implements Scalable TCP algorithm
  - Linux already implements reordering detection, SACK, and rate halving
- Some driver details (bugs?) fixed for Gbps operations

Bulk throughput

- DataTAG 2.4Gbps link and minimal buffers (2048/40)
- Flows transfer 2 gigabytes and start again for 1200s

Number of flows	2.4.19 TCP	2.4.19 TCP & giga-bit device buffer	Scalable TCP
1	7	16	44
2	14	39	93
4	27	60	135
8	47	86	140
16	66	106	142

Web traffic results

- DataTAG 2.4Gbps link and minimal buffers (2048/40)
- 4 bulk concurrent flows across 2 machines for 1200s
- 4200 concurrent web users across 3 machines

Type of bulk transfer users	Web traffic transferred	2 Gigabyte transfers completed
No bulk transfers	65GB	n/a
TCP in 2.4.19	65GB	36
TCP in 2.4.19 & gigabit device buffers	65GB	58
Scalable TCP	65GB	96

Conclusion

- Strong theoretical framework behind the algorithm
- Offers an easy evolution from the traditional TCP AMID scheme
- Freely available working code
  `http://www-lce.eng.cam.ac.uk/~ctk21/scalabl`

Where from here

- Correcting RTT bias in throughput allocation; methods similar to the parameter scaling used in previous ECN work
- Better code efficiency to improve robustness and performance of implementation
- AQM and ECN evolutions that can give extra performance in some scenarios

More at

http://www-lce.eng.cam.ac.uk/˜ctk21/scalable