# Applications - Tuning TCP

Markus Peuhkuri

2005-11-30

## Lecture topics

- Why QoS for TCP?
- TCP and quality needs
- Effect of QoS mechanisms on TCP
- Explicit Congestion Notification

Chapters from book: none (extra material)

## TCP is elastic . . .

- Dominant network transport protocol
- TCP is provides reliable byte stream
- Has two rate limiting mechanisms
  - window-based flow control: not to overrun receiver
  - AIMD (Adaptive Increase, Multiplicative Decrease) controlled congestion window: not overload network
    - probes available bandwidth
    - controls number of packets in network
- TCP congestion control reacts on packet losses
  ⇒ network must signal coming congestion by dropping packet
  ⇒ delay of RTT (round trip time) in feedback
- Round-trip time estimation essential
- Throughput depends on
  - round trip time
  - packet loss rate

  Throughput (in segments) is something like

  $$B < \min\left(\frac{W}{RTT}, \frac{1}{RTT\sqrt{p}}\right) \tag{1}$$

  $B$ number of segments in time unit, $RTT$ round trip time, $W$ window size in segments. Equation 1 is only approximate for steady-state and does not hold for large packet losses or during slow-start.

  Number of congestion signals (dropped packets or multiple acks) depends on size of a flow and for each signal rate is halved. This results two multiplicative effects and $1/\sqrt{p}$ term [5, footnote 6 on pages 4–5].

## . . . users are not

- TCP can live with packet rate of 0.01 Hz (<15 bit/s)
- Applications need some minimum bandwidth to maintain their fidelity
- Interactive applications need minimum response time
  ⇒ more on next lecture. . .

## How loss and delay affect TCP

- Both decrease throughput
- Loss rate

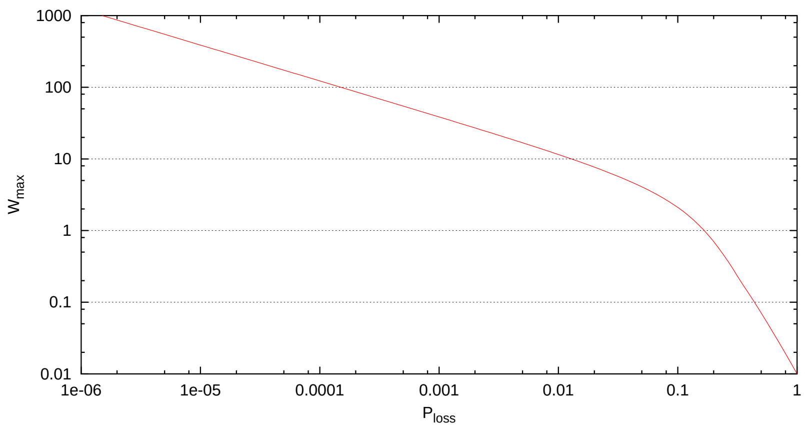
  $< \mathbf{1\,\%}$ very little effect

  $> \mathbf{20\,\%}$ throughput very low
- Delay
  - at longer delays window size is limiting factor
- Another approximation of bandwidth

$$B \approx \min\left(\frac{W}{RTT}, \frac{1}{RTT\sqrt{\frac{2p}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3p}{4}}\right) p(1+32p^2)}\right) \tag{2}$$

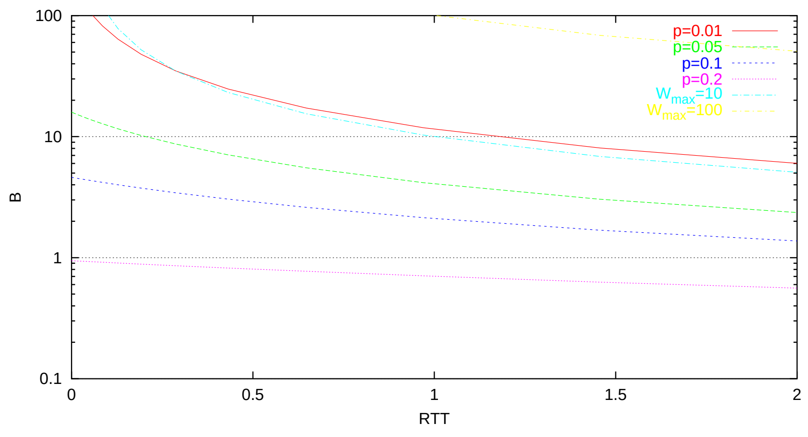
[7]

## Loss and maximum window

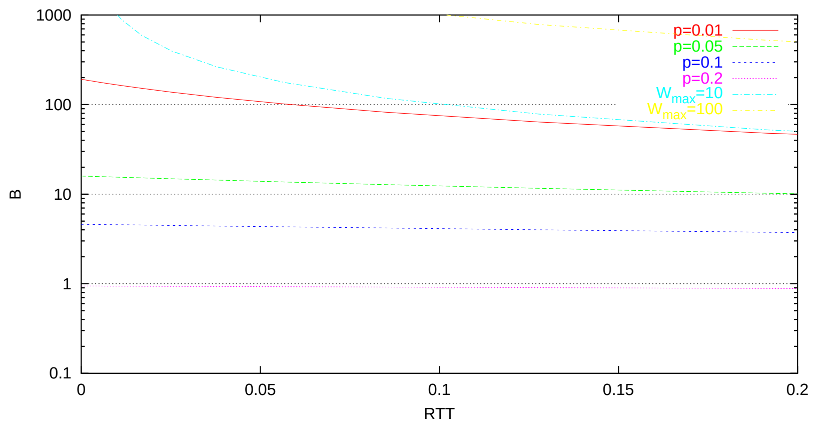


$RTT = 1$

## Throughput: loss and round trip time



## Throughput: loss and round trip time



## Estimating round trip delay

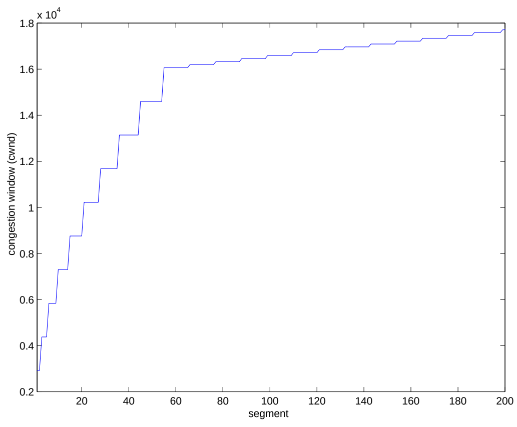
- *Too short* results superfluous retransmissions
- *Too long* reduces network utilisation and throughput
- Jacobson/Karels [5]

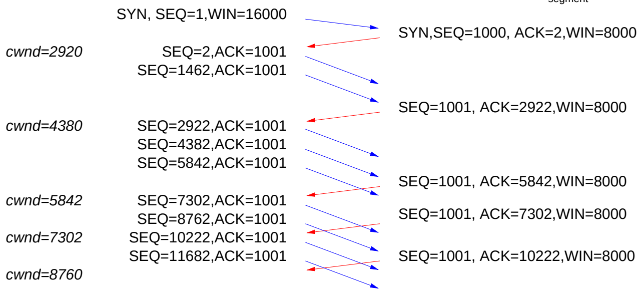
$$\begin{aligned} Diff &= RTT_{Sample} - RTT_{Est} \\ RTT_{Est} &= RTT_{Est} + \delta Diff \\ Dev &= Dev + \delta(Diff - Dev) \\ RTO &= RTT_{Est} + \phi Dev \\ & 0 \leq \delta \leq 1 \\ & \phi = 4 \end{aligned}$$

- Many different estimators proposed

## Estimating network capacity

- Initially no knowledge of network status
  $\Rightarrow cwnd \leq \min\{2SMSS, 2segment\}$
- Try to use as much network as possible
  $\Rightarrow cwnd = cwnd + SMSS$ by each acknowledgement
- After a point ($cwnd > ssthresh$) limit rate of increase
  $\Rightarrow cwnd = cwnd + SMSS^2/cwnd$ by each acknowledgement
- *ssthresh* is threshold value between “slow start” and “congestion avoidance”



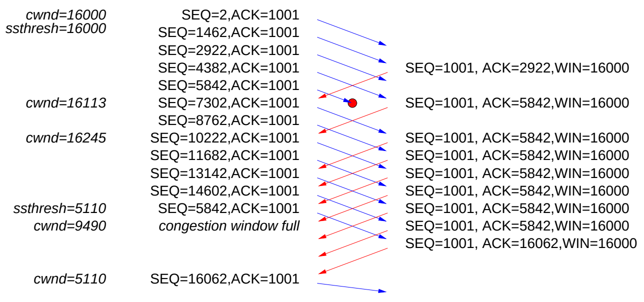


## Fast Retransmit / Fast Recovery

If a segment is lost
⇒ gap in byte sequence

- Receiver acks last byte of *continuous* sequence, if receives bytes after hole
  ⇒ segment loss is identified by 3 duplicates
- Retransmit Timeout if window size small, because there are no segments in flight to trigger duplicate acks

1. $sstresh = \max\{FlightSize/2, 2SMSS\}$
2. retransmit of lost, $cwnd = sstresh + 3SMSS$
3. for each duplicate received $cwnd = cwnd + SMSS$
4. continue sending if $cwnd$ and receiver window allows
5. set $cwnd = sstresh$ when new data is acknowledged



## TCP and QoS marking

- TCP bursty by design
- For each ack, available window is sent
  - ACK compression makes larger bursts

- Burst may exceed allowed *burst size*
  ⇒ Tail of bursts gets marked out-profile
  ⇒ Loss of multiple segments
  ⇒ Possibly Retransmit Timeout

## Partial ACK

- With a large *FlightSize* there may be several holes
  ⇒ a new fast recovery for each hole
  ⇒ *cwnd* reduced for each, maybe too much
- Reduce only once for each *FlightSize*
- Still problems identifying which segment(s) to resend

## Selective Acknowledgement

- Helps to identify lost segments [6]
- Use agreed on SYN-segments with *TCP Sack-Permitted Option*
- In case of loss, receiver sends ACK (as normal), and a partial list (TCP maximum option size of 40 bytes allows 4 blocks; 3 if Timestamp option is used) of some segments received
- First block includes SACK relevant for this ACK
- Receiver *may drop* some data that is SACKed but not ACKed
- D-SACK (Duplicate SACK [4]) reports duplicates received
  ⇒ info about spurious retransmits

## Elephants and mices

**Elephant** A flow which lasts for a long time and has many bytes in it

- terminal sessions
- usenet news server-server traffic
- database synchronisation
  ⇒ steady-state communication

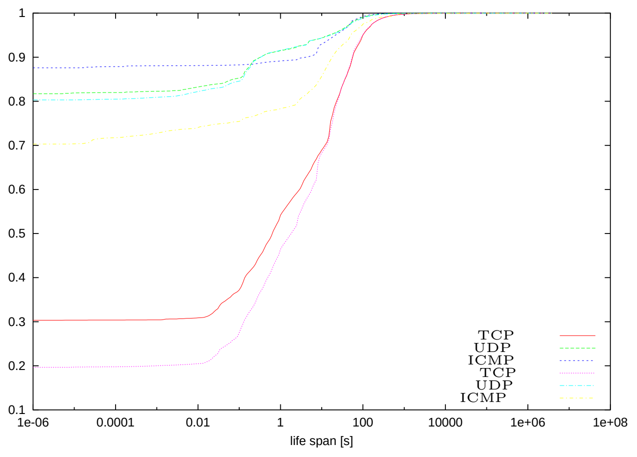
**Mice** Short-lived flow with only few segments

- HTTP requests
- DNS (on top of UDP)
  ⇒ only in slow-start phase
  ⇒ does not react to congestion control, unfairness
- majority of flows
- state sharing between TCP flows?

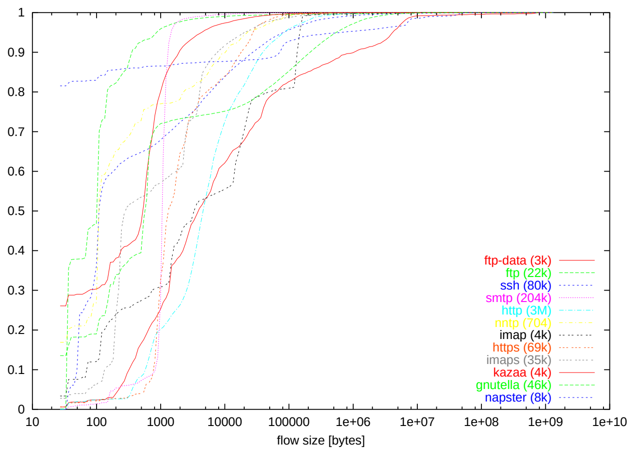
## Flow lifetime

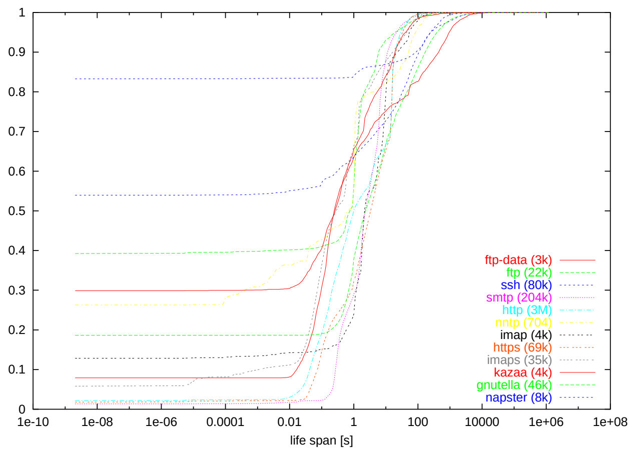
- Lifetimes vary
  - two packets exchanged in few milliseconds: one DNS query
  - millions of packets in a month: several TCP connections between two servers
- Flow timeout depends on application

## Protocol-level flows with 60-second timeout

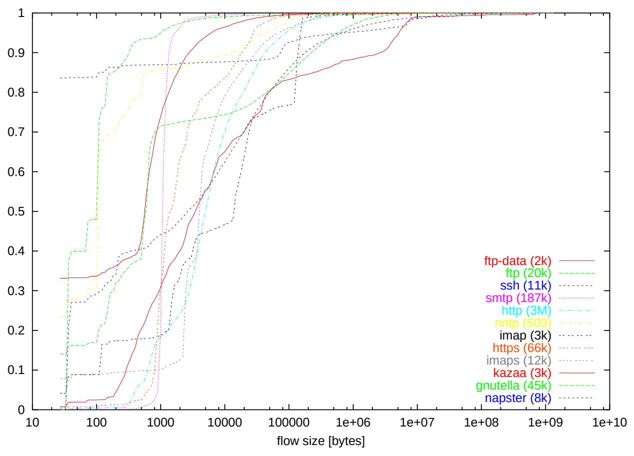


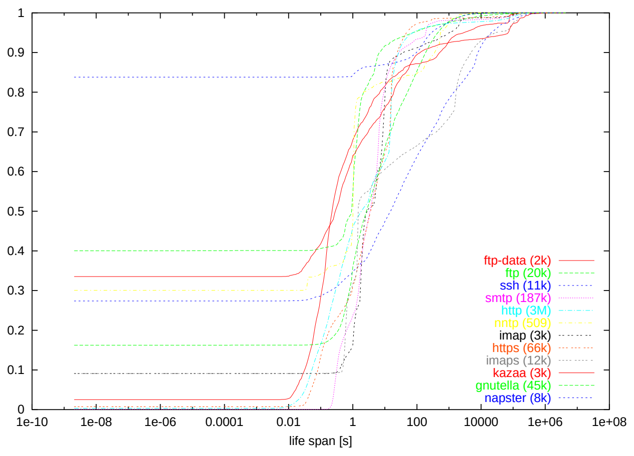
## TCP, 60-second timeout, 5-tuple



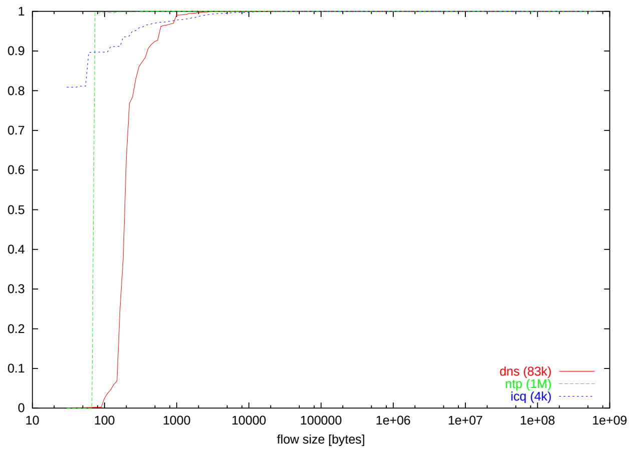


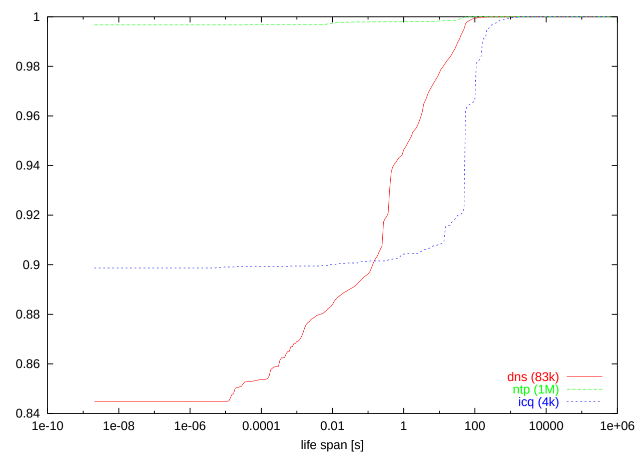
## TCP, 48-hour timeout, 4-tuple





## UDP, 60-second timeout, 5-tuple





## To be or not to be TCP-friendly

- TCP is the most important transport protocol
  $\Rightarrow$ network optimised for TCP: this includes buffer dimensioning and packet drop algorithms
- A protocol (application) can be TCP friendly
  – behaves similarly in event of congestion
  – uses *fair share* of resources
- Or not
  – gets more than fair share of bandwidth
  – causes fluctuations in network load
  – may result in *congestion collapse*

## DCCP – Datagram Congestion Control Protocol

- Congestion-controlled, unreliable datagrams
  – unreliable flows of datagrams, with acknowledgements
    ∗ packet loss and ECN information
    ∗ which data was received and dropped (optional)
  – handshakes for connection setup and teardown
  – negotiation of options, like a suitable congestion control mechanism
  – stateless mechanisms for unacknowledged connection attempts and already-finished connections on servers
  – ECN support
  – PMTU discovery
- Two congestion control mechanisms specified
  – TCP-like congestion control
  – TFRC (TCP-Friendly Rate Control) congestion control

## Should connections be limited?

- Limit number of TCP connections at link
  – guarantee minimum bandwidth
  – goodput rate is low with high losses
- Limit maximum flow speed

  - if a flow takes a great partition of link capacity, slow it
  - charge only high-bandwidth flows
  - go around by using multiple TCP flows
    ⇒ limit by host, network…

- Report congestion to edge routers to enforce policing for misbehaving flows
- Some utility, however problems with
  1. scalability
  2. fairness
  3. accuracy

## Active queue management

- RED (Random Early Detection) drops packet even if queue not full
  - statistical dropping
    ⇒ dropping proportional to bandwidth used (independent of flow count)
  - aim to avoid synchronisation of flows
  - see lecture 3: Mechanisms - Egress traffic processing
- Packet drop crude *signal* of congestion
  - delivered packet has better utility than dropped
    * has already used some network resources
    * better fidelity without retransmissions
    * TCP must wait for multiple packets before it can distinguish between reorder and drop
- Some better indication needed

## Signals of Congestion

**IP** ICMP Source Quench

- router sends if its resources are exhausted [2]
- sender *must* limit transmission rate; for TCP react as
  - retransmission timeout had occurred[1], or
  - cut congestion window into half as in Fast Retransmit
- adds traffic to congested network
  ⇒ needs rate limiting
- fast feedback: less than RTT

**Packet networks** DECbit[11]

- a bit is set by average queue size
- if more than half of packets have bit set
  ⇒ decrease congestion window multiplicatively, otherwise increase additively

**Frame Relay** FECN/BECN

- based on virtual channels, set up by signalling or network management
- FECN set if packet experienced congestion in transit
- BECN set if congestion in reverse direction

## IP Explicit Congestion Notification

- Possibility to indicate congestion in network by marking packets[10]
  ⇒ no traffic added
- Internet routes asymmetric
  ⇒ recipient must echo; needs support in both end systems
  ⇒ delay of RTT
- If transport protocol is *ECN capable*, it may set ECT bit ECN Capable Transport
- If router *would drop* and *ECT is set*
  ⇒ router sets CE bit (Congestion Experienced)
- Transport protocol *must* react if packet had *been drop*

  **TCP** reduce congestion window

  **VoIP** reduce sending rate; possibly using higher compression rates

  **multicast** select stream with lower rate, if there is one available (as should be because ECN is activated)

- Redefined ToS field

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| DS field, DSCP | | | | | | ECN field | |

- ECN codepoints

| | | |
|---|---|---|
| 0 | 0 | Not-ECT |
| 0 | 1 | ECT(1) (ECN-Capable Transport) |
| 1 | 0 | ECT(0) |
| 1 | 1 | CE (Congestion Experienced) |

  Older specification [9] used ECT bit (bit 6) to indicate ECT and if congestion was experienced, CE bit (bit 7) was set. Value "01" (currently ECT(1)) was not defined.

  Use of two different ECT values can serve 1-bit nonce to protect end systems from misbehaving network elements.

## TCP and ECN

| 0 | | | | 1 | | | | | | 2 3 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 1 2 3 | 4 5 6 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 |
| Data Offset | *Reser-ved* | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window |

- Use negotiated at connection setup
  1. connection initiator sets CWR (Congestion Window Reduced) and ECE (ECN-echo) bits
  2. recipient replies with CWR clear and ECE set
  3. connection may use CE codepoint

  Note that use of ECN differs from other TCP extension using options. This may result some problems with non-compliant firewalls and end systems.

- Congestion signalled once for RTT
  1. receives TCP segment with CE bit set, sets ECE bit for all TCP segment it sends
  2. receives TCP segment with ECE bit set, reduces congestion window and sets CWR bit; ignores ECE until next RTT
  3. receives TCP segment with CWR bit set, stops setting ECE bit

## Possible problems in ECN

- Unresponsive hosts
  - host may report it honours ECN
    ⇒ packet not dropped but marked
  - ignores CE, does not reduce rate
  - host can behave badly without ECN by increasing sending rate with FEC (Forward Error Correction)
- Feedback delay
  - full RTT before indication
  - asymmetric routing; a router may not see other direction
- IP tunnels; IPSec
  - DS byte "volatile" (not covered by AH or ESP headers)
  - in tunnel mode IPSec outer header discarded at end of tunnel
  - should ECN or DiffServ codepoints be copied to inner header?
  - depends on situation
    ⇒ ECN Tunnel attribute for IPSec SA (Security Association)
- TCP specification

  > *If an incoming segment has a security level, or compartment, or precedence which does not exactly match the level, and compartment, and precedence requested for the connection, a reset is sent and connection goes to the CLOSED state. The reset takes its sequence number from the ACK field of the incoming segment. [8, p. 37]*

  - problems with both DiffServ and ECN
  - only few implementations check for those
    ⇒ TCP updated in RFC2873 to ignore precedence [12]

## Tuning TCP for high capacity connections

- All recent operating systems support SACK, window scaling and timestamps
  - Windows 98 ⇒
  - Linux 2.1.90 ⇒
- Default flow control window size quite small (64 KiB)
- There are also tunable parameters
  - Windows: registry `http://rdweb.cns.vt.edu/public/notes/win2k-tcpip.htm` `http://support.microsoft.com/kb/q224829/`
  - Linux: sysctl or `/proc/sys/net` Variables available in each kernel version can be found from Documentation/networking/ip-sysctl.txt file in kernel source distribution, typically under /usr/src/linux or from `http://www.kernel.org/git/?p=linux/kernel/git/torvalds/linux-2.6.git;a=history;f=Documentation/networking/ip-sy`

See `http://www.psc.edu/networking/projects/tcptune/` for some details on tuning TCP.

## TCP optimisation for High-speed networks

- Standard TCP does not scale for high-speed networks
  - for 10 Gbit/s thoughput with 100 ms RTT delay, 1500 byte packets
    $\Rightarrow$ average congestion window: 83,333 segments
    $\Rightarrow$ $P_{loss} < 2 * 10^{-10}$ (one loss / 100 minutes)
- Solutions
  - use parallel TCP connections
  - use shared state for multiple TCP connectins
  - change TCP function on high congestion window values
  - develop another protocol
- Some proposals
  - binary increase congestion control (BIC): uses combination of additive and binary search increase
  - high speed TCP
  - TCP-hybla for long-delay paths
  - scalable TCP uses MIMD-congestion control, has issues with fairness (depends on loss model)

## High Speed TCP

- Goals for HighSpeed TCP [3]
  - achieve high per-connection throughput in real networks
  - pass slow-start phase quickly and recover fast from small congestion windows
  - no changes to routers
  - no additional feedback from receivers
  - TCP-equavalent performance in typical Internet environment $P_{loss} > 1\,\%$
  - as good performance in moderate or high loss enviroment
  - behaves well in transistent situations
- Non-goals
  - not to outperform TCP in very low packet drop rates. Standard TCP cannot fully utilize loss rates of $< 10^{-5}$
  - faster increase in slow-start, as HighSpeed TCP should be utilized in present Internet and not only in high-bandwidth high-delay paths.
  - protecting parallel flow oscillations
- Change AIMD parameters if congestion window exceeds threshold

## Summary

- While elastic, TCP needs some QoS
- Bursty losses bad; especially with small window
- Losses in wireless network may trigger backoff
- ECM provides gentler signal of congestion

## References

[1] R. Braden and Ed. *Requirements for Internet Hosts - Communication Layers*, October 1989. RFC 1122. URL:`http://www.ietf.org/rfc/rfc1122.txt`.

[2] R.T. Braden and J. Postel. *Requirements for Internet gateways*, June 1987. RFC 1009. URL:`http://www.ietf.org/rfc/rfc1009.txt`.

[3] S. Floyd. *HighSpeed TCP for Large Congestion Windows*, December 2003. RFC 3649. URL:`http://www.ietf.org/rfc/rfc3649.txt`.

[4] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. *An Extension to the Selective Acknowledgement (SACK) Option for TCP*, July 2000. RFC 2883. URL:`http://www.ietf.org/rfc/rfc2883.txt`.

[5] V. Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM Conference*, pages 314–329, August 1988.

[6] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. *TCP Selective Acknowledgment Options*, October 1996. RFC 2018. URL:`http://www.ietf.org/rfc/rfc2018.txt`.

[7] Jitedra Padhye, Victor Firoiu, Don Towsley, and Jim Krusoe. Modeling TCP throughput: A simple model and its empirical validation. In *ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 303–314, Vancouver, CA, 1998. URL:`ftp://gaia.cs.umass.edu/pub/Padhye-Firoiu98-TCP-throughput-TR.ps`.

[8] J. Postel. *Transmission Control Protocol*, September 1981. RFC 793. URL:`http://www.ietf.org/rfc/rfc793.txt`.

[9] K. Ramakrishnan and S. Floyd. *A Proposal to add Explicit Congestion Notification (ECN) to IP*, January 1999. RFC 2481. URL:`http://www.ietf.org/rfc/rfc2481.txt`.

[10] K. Ramakrishnan, S. Floyd, and D. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*, September 2001. RFC 3168. URL:`http://www.ietf.org/rfc/rfc3168.txt`.

[11] K. Ramakrishnan and R. Jain. A binary feeback scheme for congestion avoidance in computer networks. *ACM Transactions on Computer Systems*, 8(2):158–181, May 1990. URL:`http://portal.acm.org/citation.cfm?id=78955`.

[12] X. Xiao, A. Hannan, V. Paxson, and E. Crabbe. *TCP Processing of the IPv4 Precedence Field*, June 2000. RFC 2873. URL:`http://www.ietf.org/rfc/rfc2873.txt`.