4. Behind Traditional TCP: protocols for high-throughput and high-latency networks

PA159: Net-Centric Computing I.

Eva Hladká

Slides by: Petr Holub, Tomáš Rebok

Faculty of Informatics Masaryk University

Autumn 2014

Lecture overview

Lecture overview

Protocols for reliable data transmission

Protocols for reliable data transmission have to:

- ensure the reliability of the transfer
  - retransmissions of lost packets
  - FEC might be usefully employed
- a protection from congestion
  - network, receiver

Behavior evaluation:

- *aggressiveness* – ability to utilise available bandwidth
- *responsiveness* – ability to recover from a packet loss
- *fairness* – getting a fair portion of network throughput when more streams/participants use the network

Problem statement

- network links with *high capacity* and *high latency*
  - iGrid 2005: San Diego $\leftrightarrow$ Brno, RTT = 205 ms
  - SC|05: Seattle $\leftrightarrow$ Brno, RTT = 174 ms
- traditional TCP is not suitable for such an environment:
  - 10 Gb/s, RTT = 100 ms, 1500B MTU

    $\Longrightarrow$ sending/outstanding window 83.333 packets

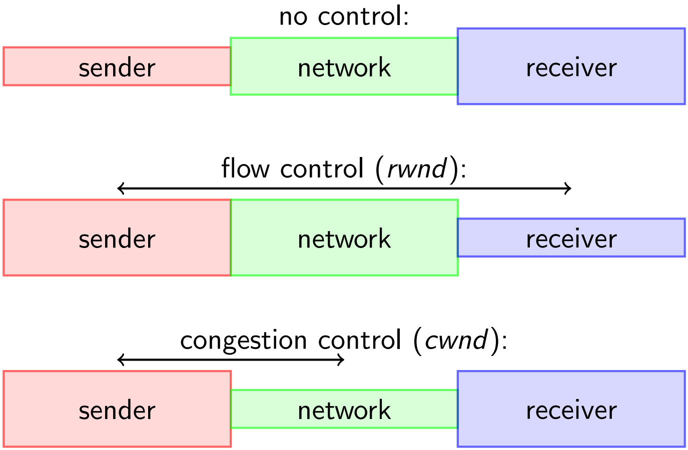
    $\Longrightarrow$ a single packet may be lost in at most 1:36 hour

    1. terribly slow
    2. if errors are more frequent, the maximum throughput cannot be reached

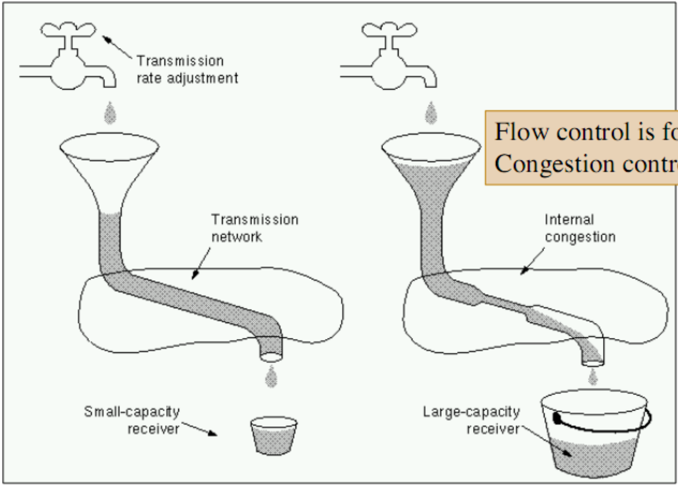
- *How could be a better network utilization achieved?*
- *How could be a reasonable co-existence with traditional TCP ensured?*
- *How could be a gradual deployment of a new protocol ensured?*

Traditional TCP I.

- flow control vs. congestion control



Traditional TCP I.



From Computer Networks, A. Tanenbaum

Flow control is for receivers
Congestion control is for the network

Congestion collapse was first observed in 1986 by V. Jacobson. Congestion control was added to TCP (TCP Tahoe) in 1988.

Traditional TCP II.

- Flow control
  - an explicit feedback from receiver(s) using *rwnd*
  - deterministic
- Congestion control
  - an approximate sender's estimation of available throughput (using *cwnd*)
- the final window used: *ownd*

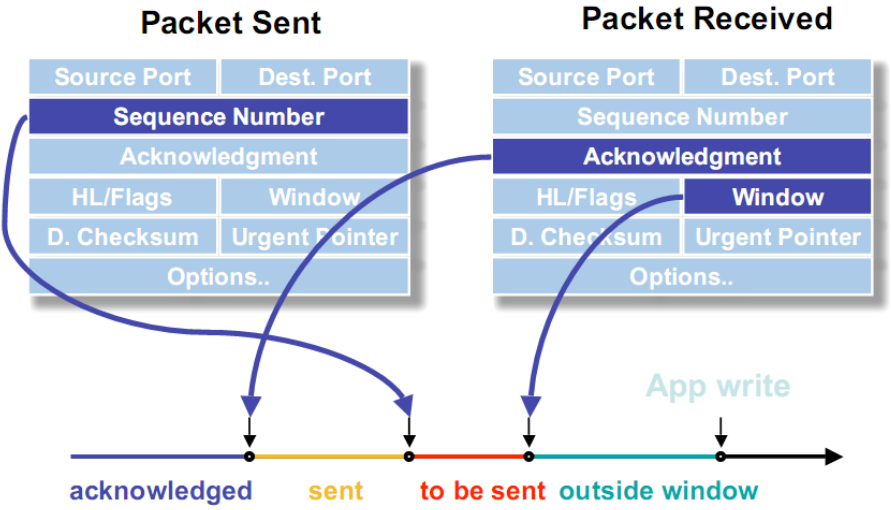
$$ownd = \min\{rwnd, cwnd\}$$

The bandwidth *bw* could be computed as:

$$bw = \frac{8 \cdot \mathrm{MSS} \cdot ownd}{\mathrm{RTT}} \tag{1}$$

Traditional TCP II.

Flow Control



Traditional TCP – Tahoe and Reno

Congestion control:

- traditionally based on *AIMD – Additive Increase Multiplicative Decrease* approach

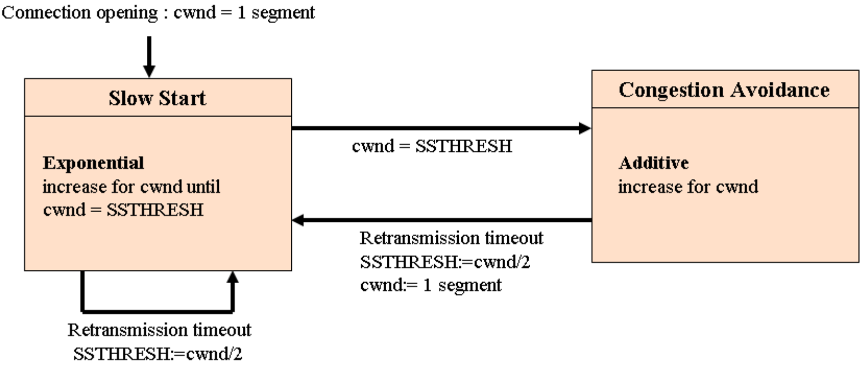
Tahoe [1]

- $cwnd = cwnd + 1$
  ... per RTT without loss (above *sstresh*)
- $sstresh = 0,5cwnd$
  $cwnd = 1$
  ... per every loss

Reno [2] adds

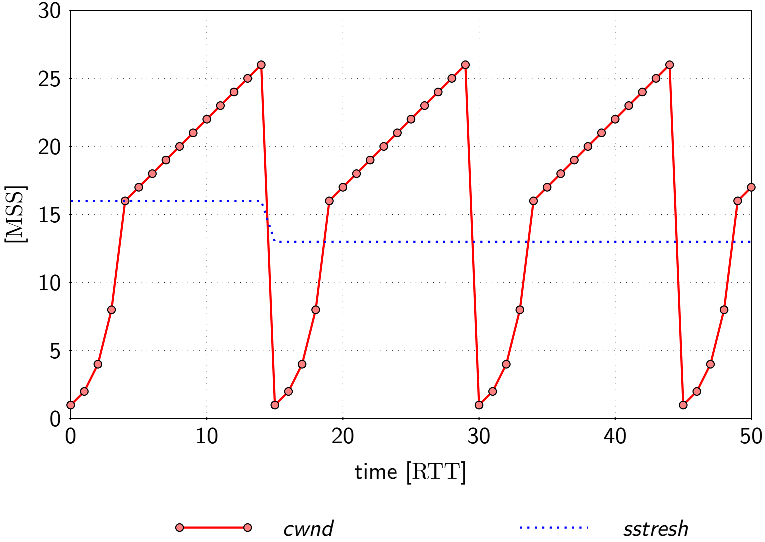
- *fast retransmission*
  - a TCP receiver sends an immediate duplicate ACK when an out-of-order segment arrives
    - all segments after the dropped one trigger duplicate ACKs
  - a loss is indicated by 3 duplicate ACKs (≈ four successive identical ACKs without intervening packets)
    - once received, TCP performs a fast retransmission without waiting for the retransmission timer to expire
- *fast recovery* – slow-start phase not used any more
  $sstresh = cwnd = 0,5cwnd$

Traditional TCP – Tahoe I.

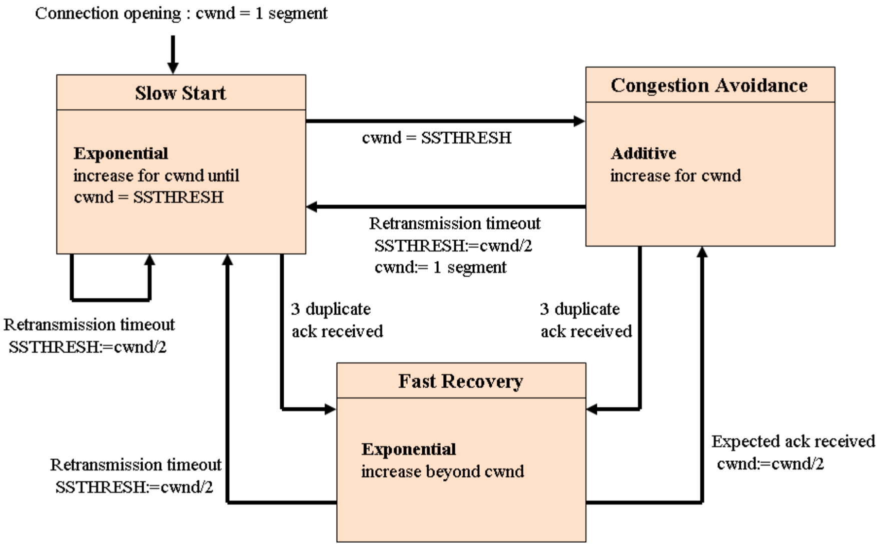


- **Exponential** increase for cwnd :
  for every useful acknowledgment received, cwnd := cwnd + (1 segment size)
- **Additive** increase for cwnd :
  for every useful acknowledgment received, cwnd := cwnd + (segment size)*(segment size) / cwnd
  it takes a full window to increment the window size by one.

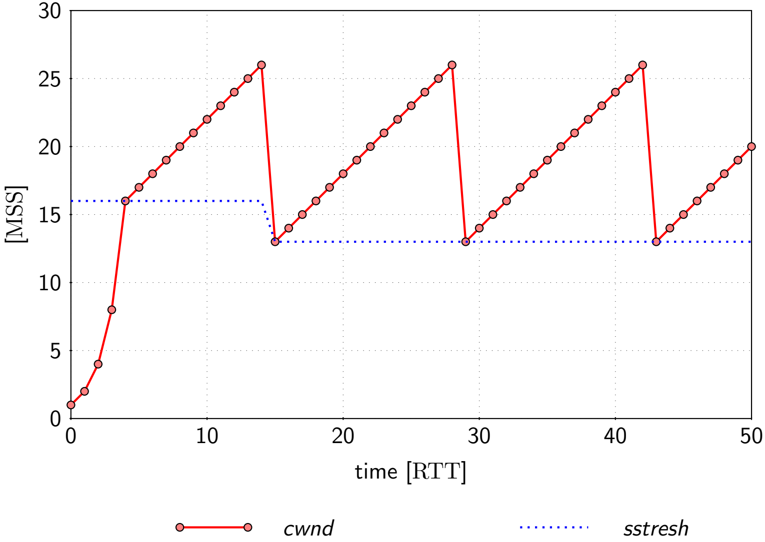
Traditional TCP – Tahoe II.



Traditional TCP – Reno I.



Traditional TCP – Reno II.



TCP Vegas

- Vegas—a concept of congestion control [3]
    - when a network is congested, the RTT becomes higher
    - RTT is monitored during the transmission
    - when a RTT increase is detected, the congestion's window size is linearly reduced
- a possibility to measure an available network bandwidth using inter-packet spacing/dispersion

Traditional TCP

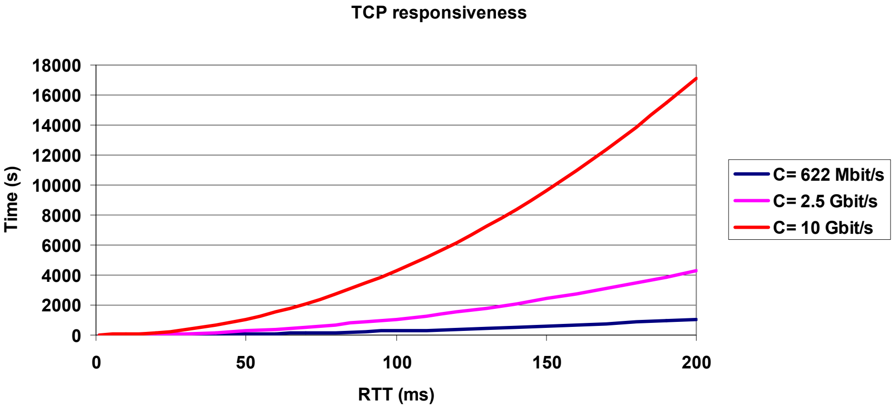
- a reaction to packet loss—retransmission
  - Tahoe: the whole actual window *ownd*
  - Reno: a single segment in the Fast Retransmission mode
  - NewReno: more segments in the Fast Retransmission mode
  - Selective Acknowledgement (SACK): just the lost packets
- fundamental question:
  How could be a sufficient size of cwnd (under real conditions) achieved in the network having high capacity and high RTT?
  . . . without affecting/disallowing the "common" users from using the network?

Traditional TCP – Response Function

- Response Function represents a relation between *bw* and a steady-state packet loss rate *p*
  - $cwnd_{average} \approx \frac{1{,}2}{\sqrt{p}}$ (for MSS-sized segments)
  - using (1): $bw \approx \frac{9{,}6\,\mathrm{MSS}}{\mathrm{RTT}\sqrt{p}}$
- the responsiveness of traditional TCP
  - assuming, that the packet has been lost when $cwnd = bw \cdot \mathrm{RTT}$

$$\varrho = \frac{bw\,\mathrm{RTT}^2}{2\mathrm{MSS}}$$

Traditional TCP – Responsiveness

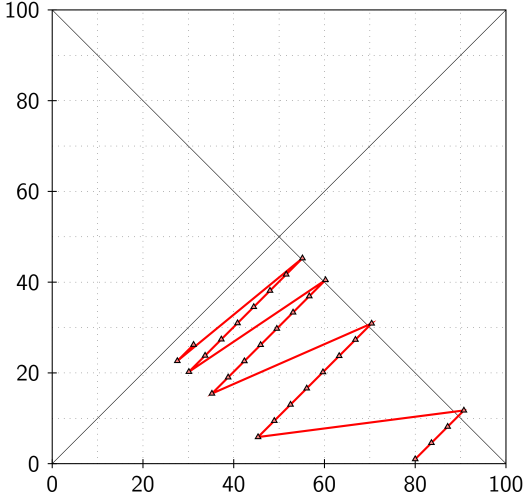


Traditional TCP – Fairness I.

- a fairness in a point of equilibrium
- the fairness is considered for
  - streams with different RTT
  - streams with different MTU
- The speed of convergence to the point of equilibrium DOES matter!

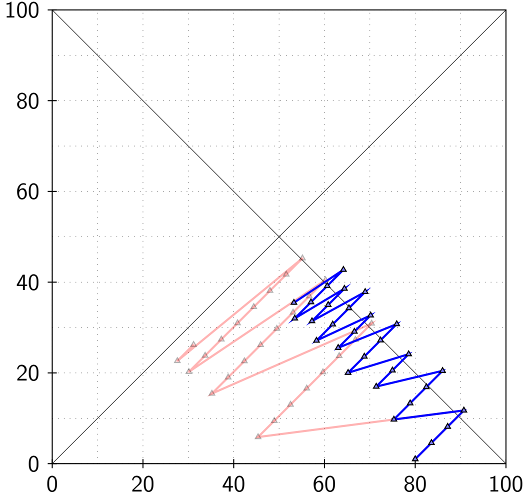
Traditional TCP – Fairness II.

- *cwnd* $+=$ MSS, *cwnd* $*=$ 0,5 (30 steps)



Traditional TCP – Fairness III.

- *cwnd* $+=$ MSS, *cwnd* $*= 0,83$ (30 steps)



Lecture overview

Multi-stream TCP

- assumes multiple TCP streams transferring a single data flow
- in fact, improves the TCP's performace/behavior just in cases of *isolated packet losses*
  - a loss of more packets usually affects more TCP streams
- usually available because of a simple implementation
  - bbftp, GridFTP, Internet Backplane Protocol, ...
- drawbacks:
  - more complicated than traditional TCP (more threads are necessary)
  - the startup is accelerated linearly only
  - leads to *a synchronous overloading of queues and caches in the routers*

TCP implementation tuning I.

- *cooperation with HW*
  - Rx/Tx TCP Checksum Offloading
  - ordinarily available
- *zero copy*
  - accessing the network usually leads to several data copies: user-land $\leftrightarrow$ kernel $\leftrightarrow$ network card
  - page flipping – user-land $\leftrightarrow$ kernel data movement
    - support for, e.g., `sendfile()`
  - implementations for Linux, FreeBSD, Solaris, ...

TCP implementation tuning II.

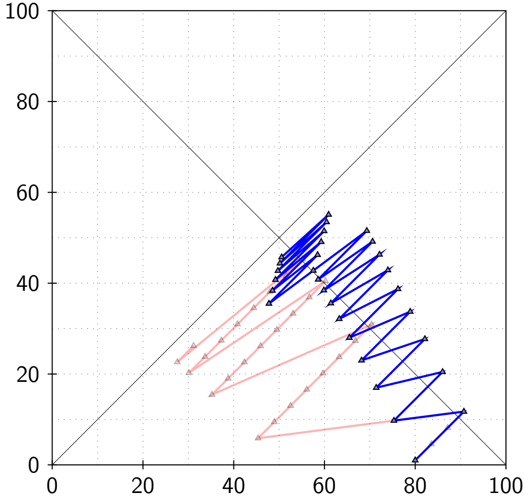
- **Web100** [4, 5]
  - a software that implements instruments in the Linux TCP/IP stack – TCP Kernel Instrumentation Set (TCP-KIS)
    - more than 125 "puls/rods"
    - information available via `/proc`
  - distributed in two pieces:
    - a kernel patch adding the instruments
    - a suite of "userland" libraries and tools for accessing the kernel instrumentation (command-line, GUI)
  - the Web100 software allows:
    - monitoring (extended statistics)
    - instruments' tuning
    - support for auto-tuning

Lecture overview

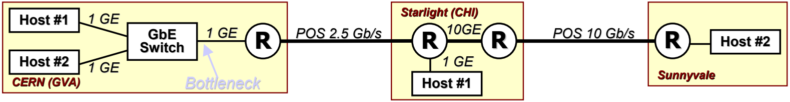
GridDT

- a collection of ad-hoc modifications :(
- correction of *sstresh*
    - faster slowstart
- AIMD's modification for congestion control:
    - $cwnd = cwnd + a$
      ... per RTT without packet loss
    - $cwnd = b\, cwnd$
      ... per packet loss
- just the sender's side has to be modified

GridDT – fairness



GridDT – example

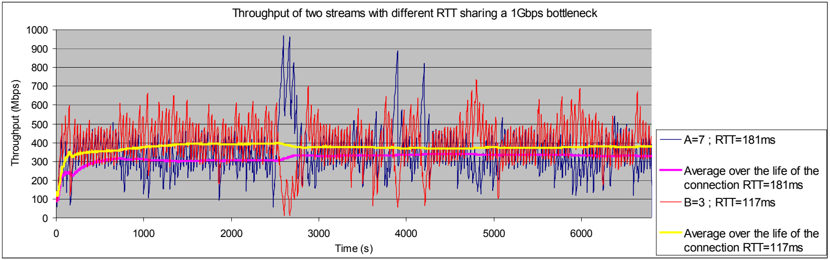


TCP Reno performance (see slide #8):

First stream GVA <-> Sunnyvale : RTT = 181 ms ; Avg. throughput over a period of 7000s = 202 Mb/s
Second stream GVA<->CHI : RTT = 117 ms; Avg. throughput over a period of 7000s = 514 Mb/s
Links utilization 71,6%

Grid DT tuning in order to improve fairness between two TCP streams with different RTT:

First stream GVA <-> Sunnyvale : RTT = 181 ms, Additive increment = A = 7 ; Average throughput = 330 Mb/s
Second stream GVA<->CHI : RTT = 117 ms, Additive increment = B = 3 ; Average throughput = 388 Mb/s
Links utilization 71.8%



Scalable TCP

- proposed by Tom Kelly [1]
- congestion control is not AIMD any more:
  - $cwnd = cwnd + 0,01\, cwnd$
    ... per RTT without packet loss
    $cwnd = cwnd + 0,01$
    ... per ACK
  - $cwnd = 0,875\, cwnd$
    ... per packet loss

$\Longrightarrow$ Multiplicative Increase Multiplicative Decrease (MIMD)

  - for smaller window size and/or higher loss rate in the network the Scalable-TCP switches into AIMD mode

Scalable TCP

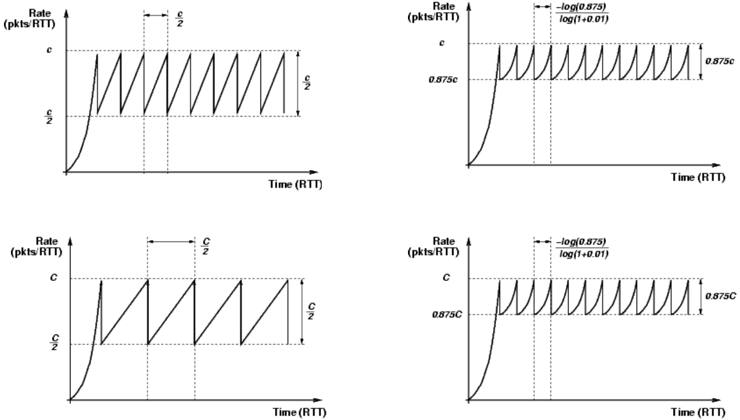
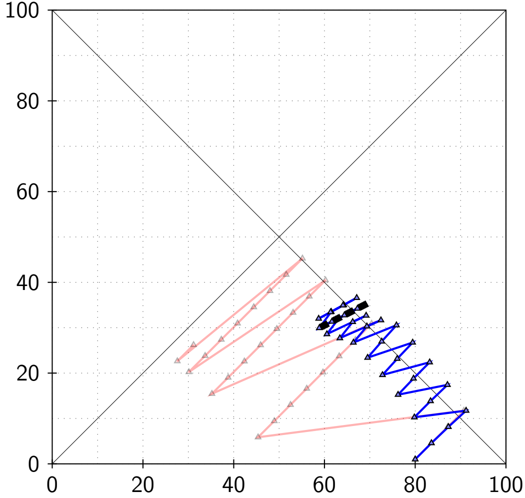


Figure: Packet loss recovery times for the traditional TCP (left) are proportional to *cwnd* and RTT. A Scalable TCP connection (right) has packet loss recovery times that are proportional to connection's RTT only. (**Note:** *link capacity* $c < C$)

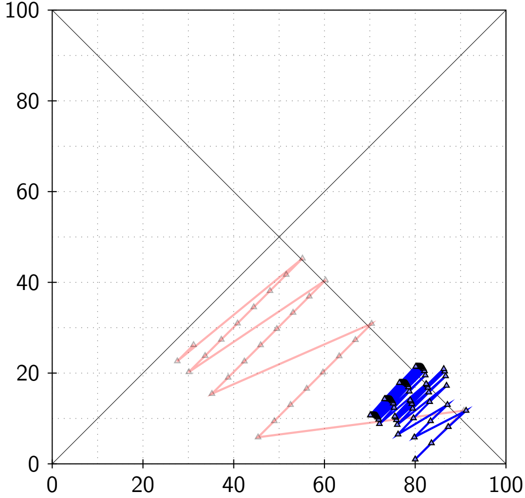
Scalable TCP – fairness I.

Two concurrent Scalable TCP streams, Scalable control switched on when >30Mb/s, twiced number of steps in comparison with previous simulations

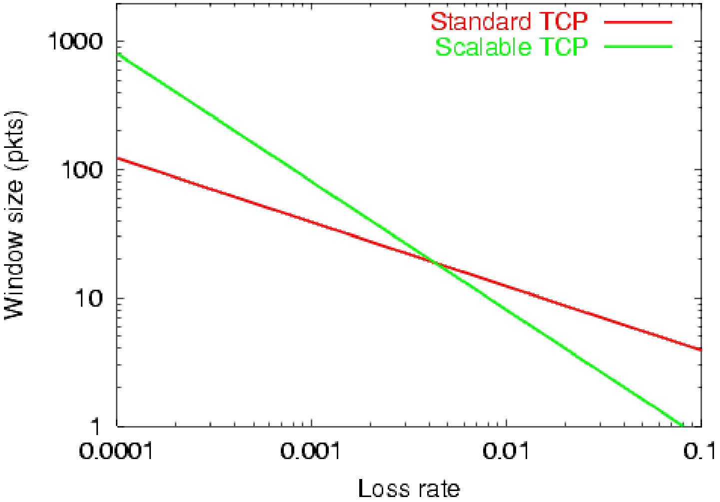


Scalable TCP – fairness II.

Scalable TCP and traditional TCP streams, Scalable control switched on when $>$30Mb/s, twiced number of steps



Scalable TCP – Response curve



High-Speed TCP (HSTCP)

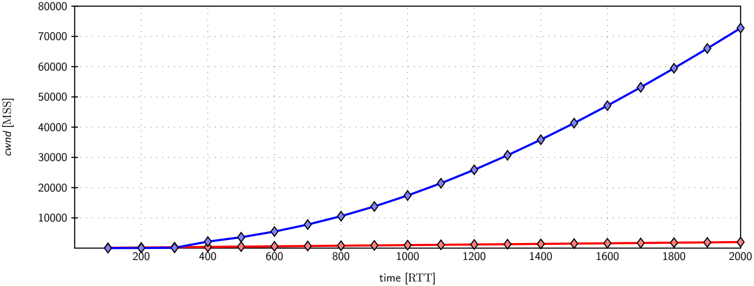
- Sally Floyd, RFC3649, [2]
- congestion control AIMD/MIMD:
  - $cwnd = cwnd + a(cwnd)$
    ... per RTT without loss
    $cwnd = cwnd + \frac{a(cwnd)}{cwnd}$
    ... per ACK
  - $cwnd = b(cwnd)\, cwnd$
    ... per packet loss
- emulates the behavior of traditional TCP for small window sizes and/or higher packet loss rates in the network

High-Speed TCP (HSTCP)

- proposed MIMD parametrization:

$$b(cwnd) = \frac{-0,4(\log(cwnd) - 3,64)}{7,69} + 0,5$$

$$a(cwnd) = \frac{2cwnd^2 b(cwnd)}{12,8(2 - b(cwnd))w^{1,2}}$$



High-Speed TCP (HSTCP)

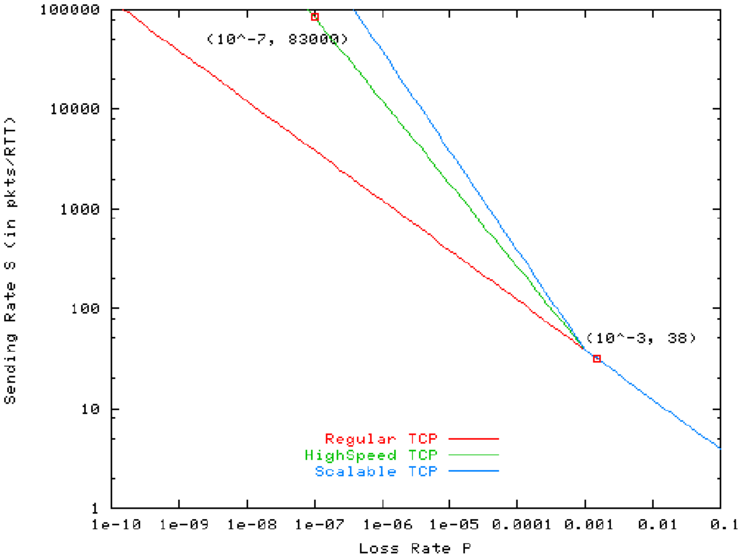
- a parametrization equivalent to the Scalable-TCP is possible: $\Rightarrow$ Linear HSTCP
- a comparison with the Multi-stream TCP

$$N(cwnd) \approx 0,23cwnd^{0,4}$$

  - $N(cwnd)$ – the number of parallel TCP connections emulated by the HighSpeed TCP response function with congestion window *cwnd*

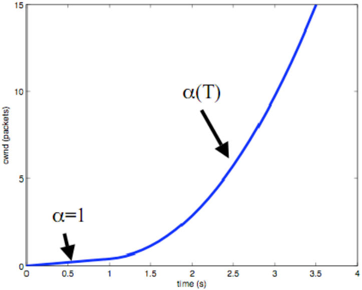
Neither Scalable TCP nor HSTCP (sophistically) deal with the slow-start phase.

HSTCP – Response curve



H-TCP I.

- created by researchers at the Hamilton Institute in Ireland
  - a simple change to *cwnd* increase function
- increases its aggressiveness (in particular, the rate of additive increase) as the time since the previous loss (backoff) increases
  - increase rate α is a function of the elapsed time since the last backoff
  - the AIMD mechanism is used
- preserves many of the key properties of standard TCP: fairness, responsiveness, relationship to buffering



H-TCP II.

- Δ ... time elapsed from last congestion experienced
- Δ_L ... for $\Delta \leq \Delta_L$ a TCP's grow is used
- Δ_B ... the bandwidth threshold, above which the TCP fall is used (for significant bandwidth changes the 0.5 fall is used)
- T_{min}, T_{max} ... the minimal resp. maximal RTTs measured
- $B(k)$ ... maximum throughput measurement for the last interval without packet loss

H-TCP III.

- $cwnd = cwnd + \frac{2(1-\beta)\, a(\Delta)}{cwnd}$
  ... per ACK
- $cwnd = b(B)\, cwnd$
  ... per loss

$$a(\Delta) = \begin{cases} 1 & \Delta \leq \Delta_L \\ \max\{a'(\Delta)\, T_{min}; 1\} & \Delta > \Delta_L \end{cases}$$

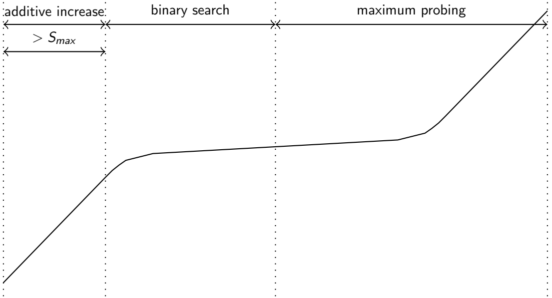
$$b(B) = \begin{cases} 0,5 & \left|\frac{B(k+1)-B(k)}{B(k)}\right| > \Delta_B \\ \min\{\frac{T_{min}}{T_{max}}; 0,8\} & \text{in the other case} \end{cases}$$

$$a'(\Delta) = 1 + 10(\Delta - \Delta_L) + 0,5(\Delta - \Delta_L)^2$$

... quadratic increment function

BIC-TCP

- the default algorithm in Linux kernels (2.6.8 and above)
- uses binary-search algorithm for *cwnd* update [3]
- 4 phases:
  (1) a reaction to a packet loss
  (2) additive increase
  (3) binary search
  (4) maximum probing



BIC-TCP

(1) Packet loss

- BIC-TCP starts from the TCP slow start
- when a loss is detected, it uses multiplicative decrease (as standard TCP) and sets the windows just before and after loss event as:
  - previous window size $\rightarrow W_{max}$ (the size of *cwnd* before the loss)
  - reduced window size $\rightarrow W_{min}$ (the size of *cwnd* after the loss)
- $\implies$ because the loss occured when $cwnd \leq W_{max}$, the point of equilibrium of *cwnd* will be searched in the range $\langle W_{min}; W_{max} \rangle$

(2) Additive increase

- starting the search from $cwnd = \frac{W_{min}+W_{max}}{2}$ might be too challenging for the network
- thus, when $\frac{W_{min}+W_{max}}{2} > W_{min} + S_{max}$, the additive increase takes place $\rightarrow cwnd = W_{min} + S_{max}$
  - the window linearly increases by S_{max} every RTT

BIC-TCP

(3) Binary search

- once the target ($cwnd = \frac{W_{min}+W_{max}}{2}$) is reached, the $W_{min} = cwnd$
  - otherwise (a packet loss happened) $W_{max} = cwnd$
- and the searching continues to the new target (using the additive increase, if necessary) until the change of *cwnd* is less than the S_{min}constant
  - here, $cwnd = W_{max}$ is set

The points (2) and (3) lead to linear (additive) increase, which turns into logarithmic one (binary search).

BIC-TCP

(4) Maximum probing

- inverse process to points (3) and (2)
- first, the inverse binary search takes place (until the *cwnd* growth is greater than S_{max})
- once the *cwnd* growth is greater than S_{max}, the linear growth (by a reasonably large fixed increment) takes place
  - first exponencial growth, then linear growth

Assumed benefits:

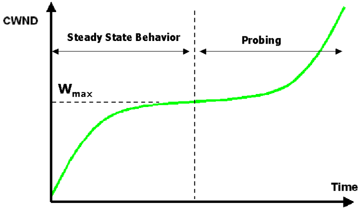
- traditional TCP "friendliness"
  - during the "plateau" (3), the TCP flows are able to grow
  - AIMD behavior (even though faster) during (2) and (4) phases
- more stable window size ⇒ better network utilization
  - most of the time, the BIC-TCP should spend in the "plateau" (3)

CUBIC-TCP

- even though being pretty good scalable, fair, and stable, BIC's growth function is considered to be still aggressive for TCP
    - especially under short RTTs or low speed networks
- *CUBIC-TCP*
    - a new release of BIC, which uses a *cubic function*
    - for the purpose of simplicity in protocol analysis, the number of phases was further reduced

$$W_{cubic} = C(T-K)^3 + W_{max}$$

where C is a scaling constant, T is the time elapsed since last loss event, W_{max} is the window size before loss event, $K = \sqrt[3]{\frac{W_{max}\beta}{C}}$, and β is a constant decrease factor



Lecture overview

Quickstart (QS)/Limited Slowstart I.

- there is a strong assumption, that the slow-start phase cannot be improved without an interaction with lower network layers
- *a proposal:* 4-byte option in IP header, which comprises of $\mathrm{QS\ TTL}$ and $\mathrm{Initial\ Rate}$ fields
- sender, which wants to use the QS, sets the $\mathrm{QS\ TTL}$ to an arbitrary (but high enough) value and the $\mathrm{Initial\ Rate}$ to requested rate, which it wants to start the sending at, and sends the SYN packet

Quickstart (QS)/Limited Slowstart II.

- each router on the path, which support the QS, decreases the $\mathrm{QS\ TTL}$ by one and decreases the $\mathrm{Initial\ Rate}$, if necessary
- receiver sends the $\mathrm{QS\ TTL}$ and $\mathrm{Initial\ Rate}$ in the SYN/ACK packet to the sender
- sender knows, whether all the routers on the path support the QS (by comparing the $\mathrm{QS\ TTL}$ and the TTL)
- sender sets the appropriate *cwnd* and starts using its congestion control mechanism (e.g., AIMD)
- Requires changes in the IP layer! :-(

E-TCP I.

- *Early Congestion Notification (ECN)*
  - a component of Advanced Queue Management (AQM)
  - a bit, which is set by routers when a congestion of link/buffer/queue is coming
  - ECN flag has to be mirrored by the receiver
  - the TCP should react to the ECN bit being set in the same way as to a packet loss
  - requires the routers' administrators to configure the AQM/ECN :-(

E-TCP II.

- **E-TCP**
  - proposes to mirror the ECN bit just once (for the first time only)
  - freezes the *cwnd* when an ACK having ECN-bit set is received from the receiver
  - requires introducing of small (synthetic) losses to the network in order to perform multiplicative decrease because of fairness
  - requires a change in receivers' behavior to ECN bit :-(

FAST

- Fast AQM Scalable TCP (FAST) [5]
- uses end-to-end delay, ECN and packet losses for congestion detection/avoidance
  - if too few packets are queued in the routers (detected by RTT monitoring), the sending rate is increased
- differences from the TCP Vegas:
  - TCP Vegas makes fixed size adjustments to the rate, independent of how far the current rate is from the target rate
  - FAST TCP makes larger steps when the system is further from equilibrium and smaller steps near equilibrium
  - if the ECN is available in the network, FAST TCP can be extended to use ECN marking to replace/supplement queueing delay and packet loss as the congestion measure

Lecture overview

tsunami

- TCP connection for out-of-band control channel
  - connection parameters negitiation
  - requirements for retransmissions – uses NACKs instead of ACKs
  - connection termination negotiation
- UDP channel for data transmission
  - MIMD congestion control
  - highly configurable/customizable
    - MIMD parameters, losses threshold, maximum size of the queue for retransmissions, the interval of sending the retransmissions' requests, etc.

Reliable Blast UDP – RBUDP

- similar to *tsunami* – out-of-band TCP channel for control, UDP for data transmission
- proposed for disk-to-disk transmissions, resp. the transmissions where the complete transmitted data could be saved in the sender's memory
- sends data in a user-defined rate
  - `app_perf` (a clon of `iperf`) is used for an estimation of networks'/receivers' capacity

Reliable Blast UDP – RBUDP

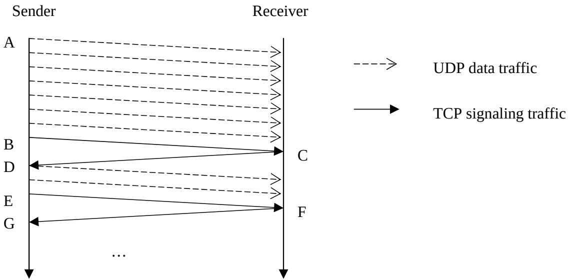


Figure 1. The Time Sequence Diagram of RBUDP

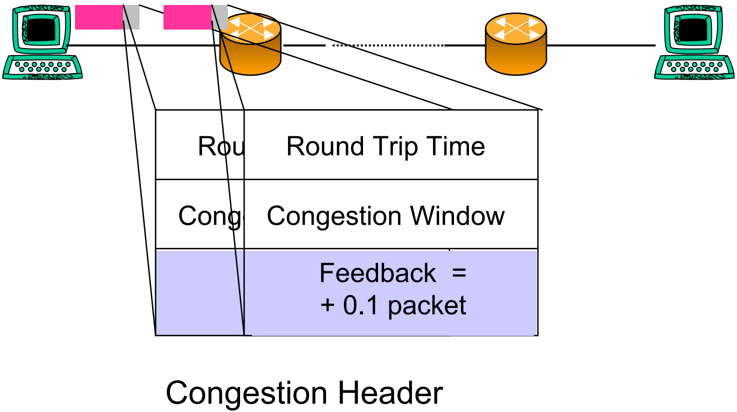
Source: E. He, J. Leigh, O. Yu, T. A. DeFanti, "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer," *IEEE Cluster Computing 2002*, Chicago, Illinois, Sept, 2002.

- A start of the transmission (using pre-defined rate)
- B end of the transmission
- C sending the DONE signal via the control channel; the receiver responses with a mask of data, that had arrived
- D re-sending of missing data
- E-F-G end of transmission

The steps C and D repeat until all the data are delivered.

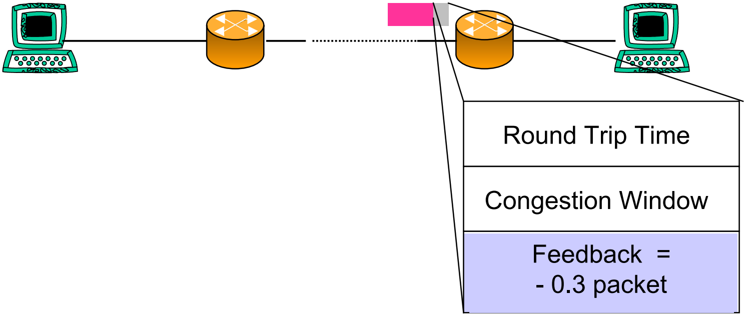
eXplicit Control Protocol – XPC

- uses a feedback from routers per paket



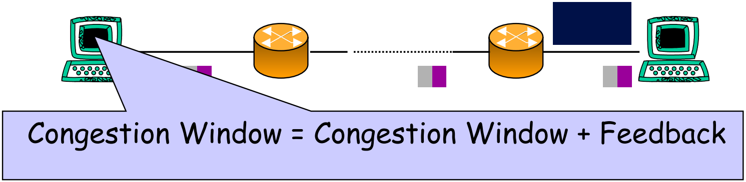
eXplicit Control Protocol – XPC

- uses a feedback from routers per paket



eXplicit Control Protocol – XPC

- uses a feedback from routers per paket



Different approaches I.

- SCTP
  - multi-stream, multi-homed transport (end node might have several IP addresses)
  - message-oriented like UDP, ensures reliable, in-sequence transport of messages with congestion control like TCP
  - `http://www.sctp.org/`
- DCCP
  - non-reliable protocol (UDP) with a congestion control compatible with the TCP
  - `http://www.ietf.org/html.charters/dccp-charter.html`
  - `http://www.icir.org/kohler/dcp/`

Different approaches II.

- STP
  - based on CTS/RTS
  - a simple protocol designed for a simple implementation in HW
  - without any sophisticated congestion control mechanism
  - `http://lwn.net/2001/features/OLS/pdf/pdf/stlinux.pdf`
- Reliable UDP
  - ensures reliable and in-order delivery (up to the maximum number of retransmissions)
  - RFC908 a RFC1151
  - originally proposed for IP telephony
  - connection parameters can be set per-connection
  - `http://www.javvin.com/protocolRUDP.html`
- XTP (Xpress Transfer Protocol), . . .

Lecture overview

Conclusions I.

- Current state:
  - multi-stream TCP is intensively used (e.g., Grid applications)
  - looking for a way which will allow safe (i.e., backward compatible) development/deployment of post-TCP protocols
  - aggressive protocols are used on private/dedicated networks/circuits (e.g., λ-networks CzechLight/CESNET2, SurfNet, CaNET*4, ...)
  - implementation SCTP under FreeBSD 7.0
  - implementation DCCP under Linux

Conclusions II.

- interaction with L3 (IP)
- interaction with data link layer
  - variable delay and throughput in wireless networks
  - optical burst switching
- specific per-flow states in routers:
  - e.g., per-flow setting for packet loss generation ($\rightarrow$ E-TCP)
  - may help short-term flows with high capacity demands (macro-bursts)
  - problem with scalability and cost :-(

Lecture overview

Literature

- Jacobson V. "Congestion Avoidance and Control", Proceedings of ACM SIGCOMM'88 (Standford, CA, Aug. 1988), pp. 314–329.
  ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z
- Allman M., Paxson V., Stevens W. "TCP Congestion Control", RFC2581, Apr. 1999.
  http://www.rfc-editor.org/rfc/rfc2581.txt
- Brakmo L., Peterson L. "TCP Vegas: End to End Congestion Avoidance on a Global Internet", IEEE Journal of Selected Areas in Communication, Vol. 13, No. 8, pp. 1465–1480, Oct. 1995. ftp://ftp.cs.arizona.edu/xkernel/Papers/jsac.ps
- http://www.web100.org
- Hacker T. J., Athey B. D., Sommerfield J. "Experiences Using Web100 for End-To-End Network Performance Tuning"
  http://www.web100.org/docs/ExperiencesUsingWeb100forHostTuning.pdf

Literature

- Kelly T. "Scalable TCP: Improving Performance in Highspeed Wide Area Networks", PFLDnet 2003,
  http://datatag.web.cern.ch/datatag/pfldnet2003/papers/kelly.pdf,
  http://wwwlce.eng.cam.ac.uk/~ctk21/scalable/
- Floyd S. "HighSpeed TCP for Large Congestion Windows", 2003,
  http://www.potaroo.net/ietf/all-ids/draft-floyd-tcp-highspeed-03.txt
- BIC-TCP, http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/
- Floyd S., Allman M., Jain A., Sarolahti P. "Quick-Start for TCP and IP", 2006,
  http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-quickstart-02.txt
- Jin C., Wei D., Low S. H., Buhrmaster G., Bunn J., Choe D. H., Cottrell R. L. A., Doyle J. C., Newman H., Paganini F., Ravot S., Singh S. "FAST – Fast AQM Scalable TCP."
  http://netlab.caltech.edu/FAST/
  http://netlab.caltech.edu/pub/papers/FAST-infocom2004.pdf
- tsunami, http://www.anml.iu.edu/anmlresearch.html
- Zdroj: E. He, J. Leigh, O. Yu, T. A. DeFanti, "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer," *IEEE Cluster Computing 2002*, Chicago, Illinois, Sept, 2002.

Further materials

- Workshops PFLDnet 2003–2010
  - http://datatag.web.cern.ch/datatag/pfldnet2003/program.html
  - http://www-didc.lbl.gov/PFLDnet2004/
  - http://www.ens-lyon.fr/LIP/RESO/pfldnet2005/
  - http://www.hpcc.jp/pfldnet2006/
  - http://wil.cs.caltech.edu/pfldnet2007/
- prof. Sally Floyd's pages:
  - http://www.icir.org/floyd/papers.html
- RFC3426 – "General Architectural and Policy Considerations"

http://www.hamilton.ie/net/eval/results_HI2005.pdf