TCP dynamics : the linux implementation between standards and research

Roberto Innocente

inno@sissa.it

for the LinuxDay 2002 - Trieste

Intro

- Transport protocols are usually devised to optimize movements of large quantities of data and persistent connections
- The most important transport protocol in use today, TCP, is no exception
- Unfortunately or fortunately the Web in the last years has changed traffic patterns. Most of the traffic is now constituted by many small (some KB or tens of KBs) transfers performed in bursts.

TCP flow control

- It is necessary to avoid that that the receiver is flooded by the sender
- TCP as many other protocols uses a *sliding window* mechanism
- in every TCP packet a receiver advertised window (*rwnd*) is communicated and that is the maximum amount of data the sender can send without waiting for an ACK
- originally this was a 16 bit quantity (up to 64k) now an initial option can specify a multiplier up to 2**14

Congestive collapses

As reported by Van Jacobson in his famous paper, in october 1986 the Internet had one of its first *congestive collapses*.

Suddenly after a congestion of a 32 kb/s link, while the traffic remained high because of retransmissions, the valuable data that traversed the link (**goodput**) was reduced by almost a factor of 1/1000 to something like 40 bps.

The study of what happened at that time lead to the development of the tcp congestion control algorithms devised by Van Jacobson at the end of the ’80.

TCP congestion control

- It had an importance similar to that the Watt's Flyball regulator had for the steam engine
- To adapt the protocol to the changing loads of real world

TCP Congestion control

- RFC2581 TCP congestion control, Allman,Paxson, Stevens 1999
- it comprises the following algorithms :
  - Slow start
  - Congestion avoidance
  - Fast retransmit
  - Fast recovery

RTT (Round Trip Time) RTO(Retransmission timeout)

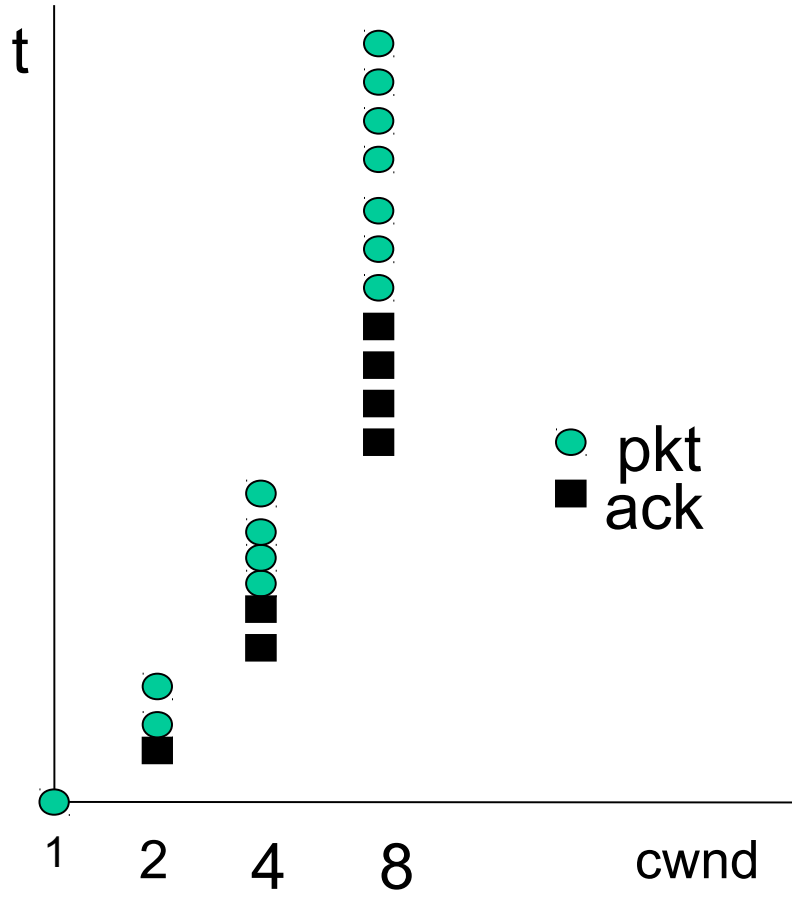
- M = measured RTT
- R = smoothed RTT estimator
- Original specification:
  - R = 0.9 R + 0.1 M, RTO = 2 R
- Jacobson 1988
  - A = average RTT
  - Err = M – A
  - A = A + 0.125 Err
  - D = D + 0.25 (|Err|-D) (mean deviation instead of std deviation to save time on computing sqrt)
  - RTO = A + 4 D, at start A=0 and D= 3 sec and RTO= A+2 D (only for the first SYN pkt)

Congestion control parameters

- *cwnd* congestion window is the maximum number of tcp segments a tcp sender can send in one RTT period, it is the parameter that limits the sending rate
- *ssthresh* slow start threshold, it is a threshold for the *cwnd* parameter below which the slow start algorithm is applied and over which the congestion avoidance algorithm is instead used
- we have already seen that *rwnd* is instead the maximum amount of unack data the sender is willing to accept
- at any time in an RTT period no more than
  - min(rwnd,cwnd)

  packets can be injected into the network

TCP Slow Start



- *ssthresh* is initialized to the advertised window
- Initially *cwnd* is initialized to a constant that RFC2581 suggests to be 1 or at most 2
- Then each RTT *cwnd* is doubled, to obtain an exponential increase of it until a loss or *ssthresh* is reached. Usually this is obtained incrementing *cwnd* by 1 for each ACK received
- When a loss is detected
  - *cwnd* = *cwnd* / 2,
  - *ssthresh* = *cwnd*

TCP acks

- Acknowlegements in TCP are cumulative, they acknowledge all previously received data. Because of this the mechanism is very robust.
- RFC2581 requires to send an ACK at least every second full segment and to implement the *delayed ack* algorithm to try to piggyback the ACK to a data pkt in the other direction. In any case the ACK must be sent within 500ms since the arrival of the data

TCP Congestion Avoidance

- The congestion avoidance algorithm is applied when :
  - *cwnd* >= *ssthresh*

  the congestion window in this case is increased linearly by 1 every RTT.

  In practice this is done increasing *cwnd* by 1/*cwnd* every ACK.

  If a RTO happens then *cwnd* = 1 and slow start is performed

TCP Fast retransmit

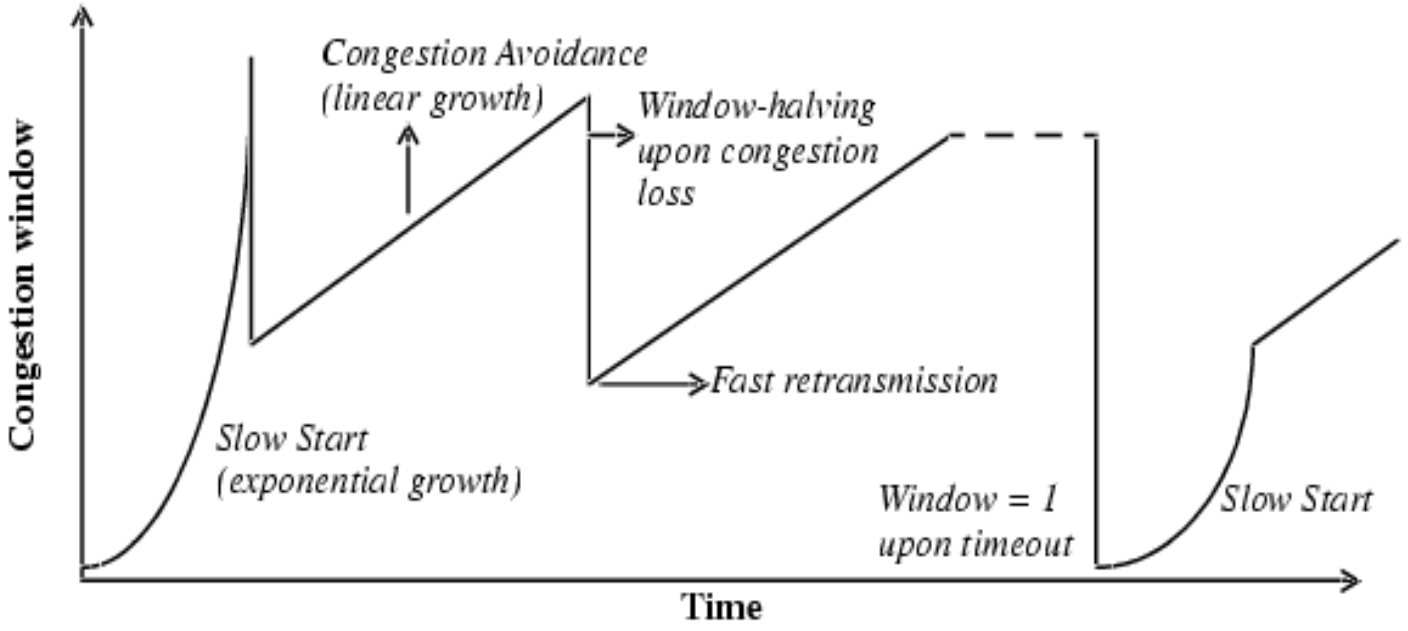
- In the original specification only the expiry of the RTO (Retransmission Time Out) timer induced the retransmission of a segment
- To avoid such a long delay a mechanism was devised to recover from the loss of a single tcp segment as it occurs when probing for congestion
- a TCP receiver should immediately send a duplicate ACK when it receives an out of order segment (this can be due also to some network reordering of segments)
- a TCP sender should detect the arrival of 3 duplicate ACKs (the original + 3 dups) and retransmit the segment immediately following
- the sender sets the
  - *ssthresh* = cwnd/2
  - *cwnd* = *ssthresh* + 3

  (here there are some little differences with the RFC to simplify the discussion, refer to RFC2581 for details)

TCP Fast Recovery

- This algorithm allows a TCP to continue with the congestion avoidance algorithm after detecting and repairing a single loss with fast retransmit.

Congestion control in TCP



from [Balakrishnan 98]

Restart of Idle connections

- The suggested behaviour after an idle period (more than a RTO without exchanges) is to reset the congestion window to its initial value (usually cwnd=1) and apply slowstart
- Solaris doesnt observe the idle time and doesnt apply ”slow start restart”
- **Rate Based Pacing (RBP)**: *Improving restart of idle TCP connections* ,Viesweswaraiah, Heidemann (1997) suggests to reuse the previous cwnd , but to smoothly pace out pkts, rather than burst them out, using a Vegas like algorithms to estimate the rate

A frequently observed TCP stall

- If a client needs to send more than a single segment to complete a request then :
  - it will send the 1st segment because cwnd=1
  - the receiver will not ack immediately because of the delayed ack argument, and will not reply because the request is not complete. So it will wait something up to 500 ms to send the ACK
  - only then the client will send other segment and updates is *cwnd*

Why the stall is not so frequent ?

- A famous bug in Windows/NT and BSD derived implementations consisted in considering the ACK of the 3-way tcp initial handshake as a signal of data succesfully transferred
- Solaris doesnt use delayed ack during slow start

TCP stacks

- **Tahoe**(4.3BSD,Net/1) : implements Van Jacobson slow start/congestion avoidance and fast retransmit algorithms
- **Reno**(1990,Net/2) : fast recovery, TCP header prediction
- **Net/3**(4.4BSD 1994): multicasting, LFN (Long Fat Networks) mods
- **NewReno** : fast retransmit even with multiple losses (Hoe 1997)
- **Vegas**: experimental stack (Brakmo, Peterson 1995)

TCP Reno

- It is the most widely referenced implementation, basic mechanisms are the same also in Net/3 and NewReno
- It is biased against connections with long delays (large RTT) : in this case the increase of the *cwnd* happens slowly and so they obtain less avg b/w

LFN Optimal Bandwidth and TCP Reno

- LFNs are Long Fat Networks, networks for which the bandwidth delay product is high
- Let's consider a 1 Gb/s WAN connection with a RTT of 100 ms (BW*rtt ~12 MB)
- Optimal congestion window would be about 8.000 segments (1.5k)
- When there will be a loss cwnd will be decreased to 4.000 segments
- It will take 400 seconds (7minutes!) to recover to the optimal b/w. This is awful if the network is not congested !

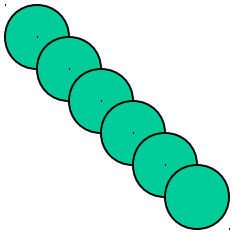
TCP Vegas

- Approaches congestion but tries to avoid it. Tries to estimate the queueing at intermediate routers looking at b/w variations. BaseRTT is the least RTT measured. Then
  - Expected = CongestionWindow/BaseRTT
  - Diff = Actual – Expected
  - Then if Diff < α congestion window is increased else if Diff > β congestion windows is decreased
- It has been shown that it is able to better utilize the available b/w (30% better than Reno)
- It is fair with long delay connections
- Anyway, when mixed with TCP Reno it gets an unfair share (50% less) of the bandwidth

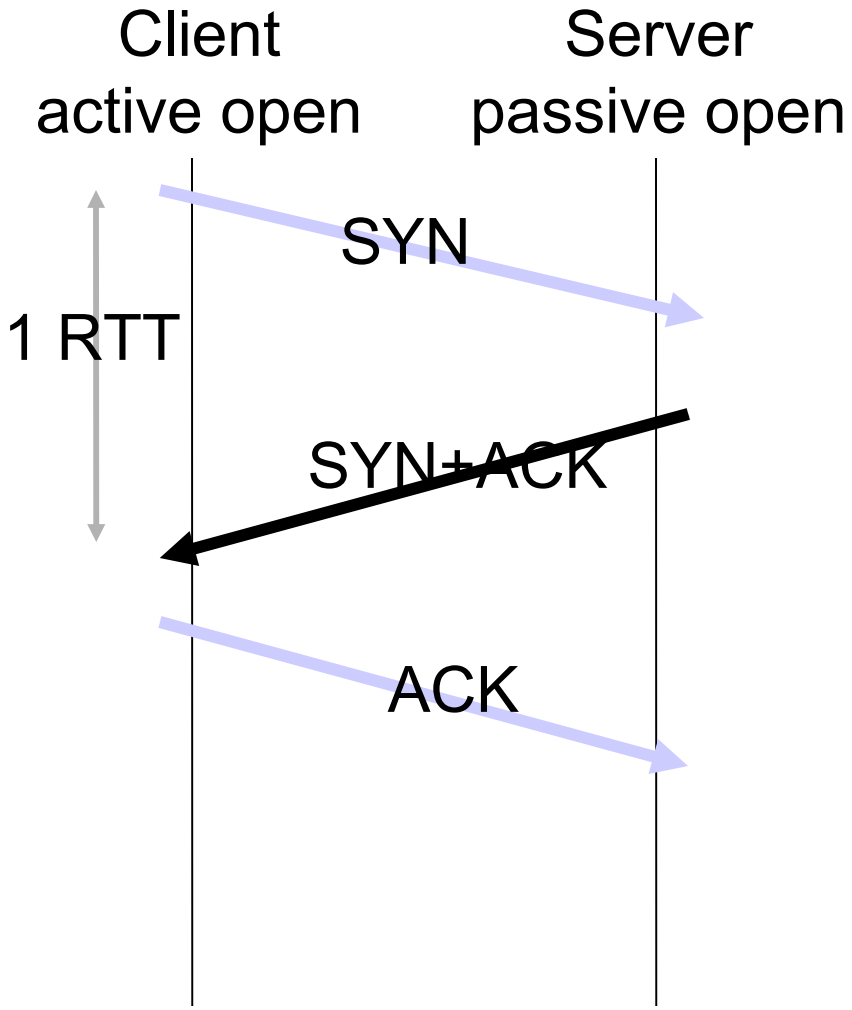
Selective acknowledgment (SACK)

- RFC 2018 TCP Selective Acknowledgment Options, 1996, Mathis, Mahdavi, Floyd
- This is a TCP option that allows the receiver to specify exactly which data it has received. It overcomes the little information conveyed by the std cumulative ACK, giving the possibility to specify up to 3 or 4 contiguous blocks of data received whenever there are gaps in the sequence of bytes received. In this way the sender can selectively retransmit the missing data

SACK /2



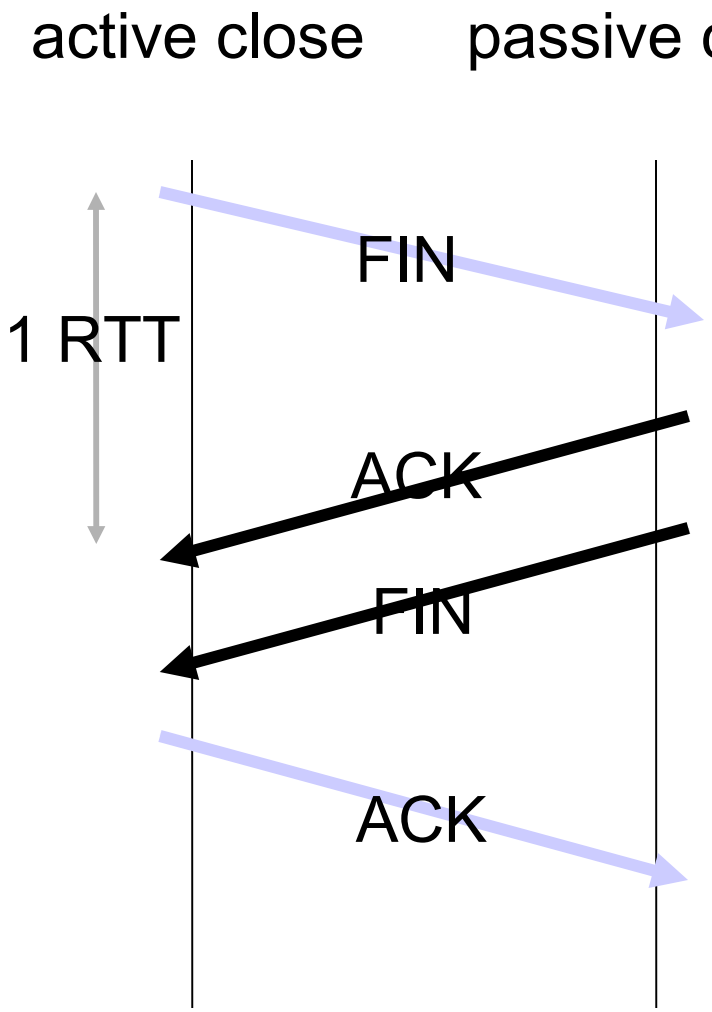
TCP Connection establishment



3-way handshake :

- SYN : client -> server
- SYN-ACK: server -> client
- ACK : client -> server

TCP circuit termination



TCP allows half-open connections.

Each direction is independent.

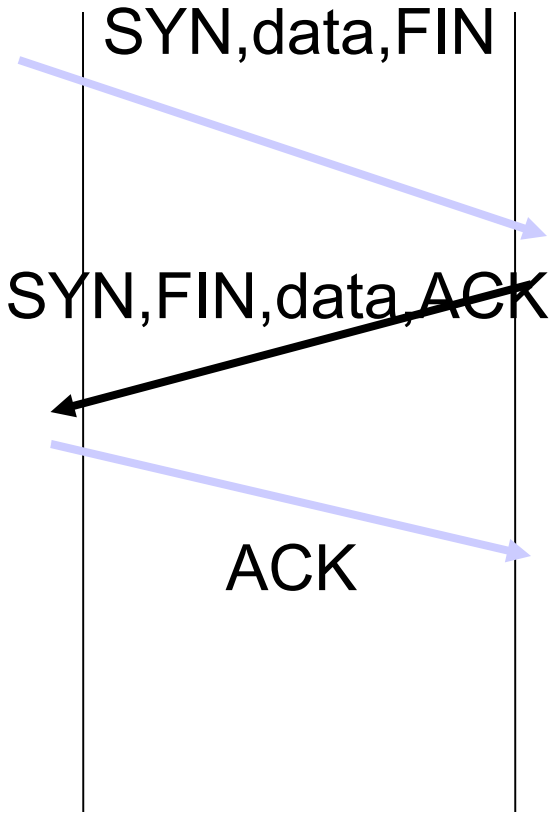
T/TCP (Transaction TCP) /1

- Standard TCP provides a **virtual circuit** service with *connection establishment*, *data transfer*, *circuit termination* phases.
- Some application however are built around a simpler **transaction** service : a client makes a request and the server answers.

T/TCP (Transaction TCP)/2

- For these applications the overhead of the 3-way initial TCP handshake should be avoided
- Then T/TCP introduces a *TCP accelerated open (TAO)*
- This TCP extension avoids 3-way handshake
- Request and reply is sent together with connection messages
- Servers cache a CC (connection count) for clients to detect duplicate requests.
- Latency = RTT + server processing time

T/TCP (Transaction TCP)/3



- There are many concerns about the security of this protocol due to the easy flooding of servers that is possible spoofing the IP address

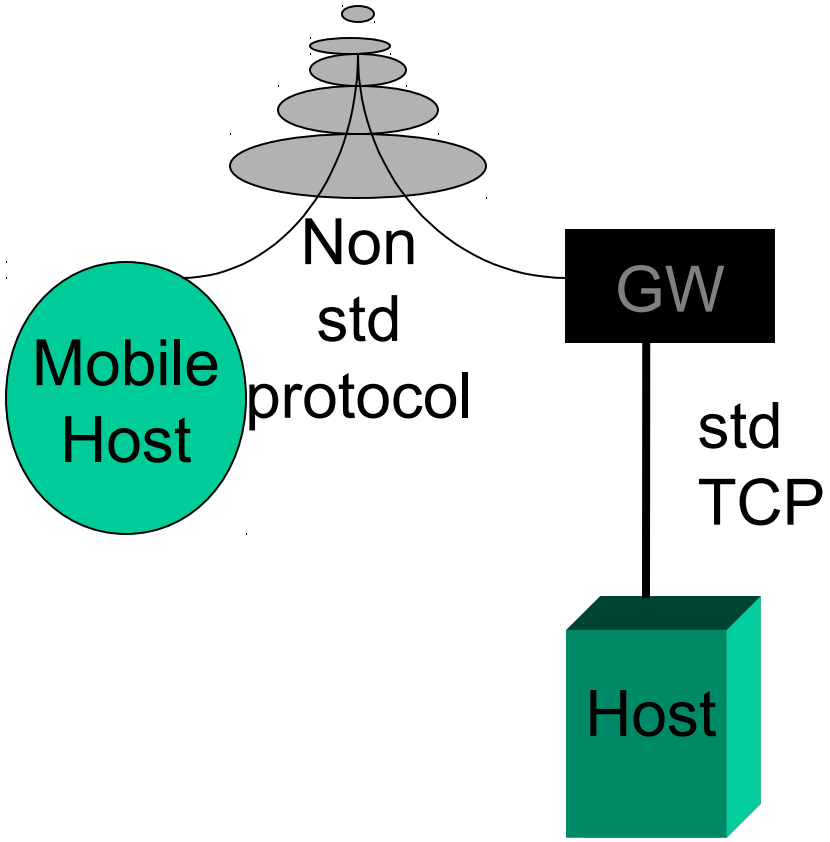
T/TCP (Transaction TCP) /4

- A further drawback of T/TCP is that it requires an application to use *sendto()* or *sendmsg()* calls, while most applications use the *connect()* and then *send()* or *write()* calls.
- An implementation at the alpha/beta stage is available as a kernel patch to Linux 2.4.2 :
  - http://ttcplinux.sourceforge.net/

Mobile and Satellite Links

- As we have seen TCP interprets packet losses as a signal of congestion and therefore halves its sending rate.
- This is frequently inappropriate for lossy channels as satellite links or wireless links.
- There are essentially 2 kinds of proposals to overcome this problem :
  - to use a link layer protocol as ARQ that hides the real behaviour of the link making it as reliable as ordinary links : snoop
  - to split the tcp connection between the 2 different networks (using on the wireless network a special implementation or an ad hoc protocol) : Indirect TCP: I-TCP

Split TCP/ I-TCP (Indirect TCP)



- First proposed in the mid ’90
- Based on the idea that a connection can be established over 2 completely different networks : wired and wireless
- ACKs are not end-to-end and this breaks the end-2-end property of TCP

HTTP

- ver. 1.0 opens a TCP conection for each URI retrieved : retrieves first URI and then tries to get all connected URI using simultaneous separated TCP connections. Creates bursts on the net, incurs startup overhead
- ver. 1.1 persistent connections and pipelining : good because only one transport connection and so it can be trained, but slow because it serializes transfers

Session Control Protocol (SCP)

- devised by Simon Spero UNC
- several common applications like ftp, http use for every transaction a separate TCP connection, while in fact many requests are done to the same server. This is inefficient and the applications incur all the startup costs
- SCP is a simple protocol that lets a server and a client to have multiple conversations over a single TCP connection.
- has a fixed overhead of 8 bytes per segment

Simple Multiplexing Protocol

- Frequently referred as MUX (Gettys,Nielsen 1997) many ideas derived from SCP (session control protocol)
- It separates the transport protocol as for example TCP from the upper level application protocol inserting a lightweight session multiplexing protocol (can manage up to 2**8 sessions) by which an application can have multiple ongoing (e.g. HTTP) transactions over a single transport level connection
- Differently from SCP can have only 4 bytes of overhead, and can multiplex different protocols

TCB (TCP control blocks) interdependence

- proposal to share TCP control block between the different connections to the same host, such that for example congestion windows, RTTs, ssthresh dont need to be recalculated and startup times can be avoided

Connection managers

- There have been proposals to share TCP parameters (RTT, b/w available, ..) between more hosts (e.g. a company) setting up a server that caches those values for the different networks

AIMD / AIPD Congestion control

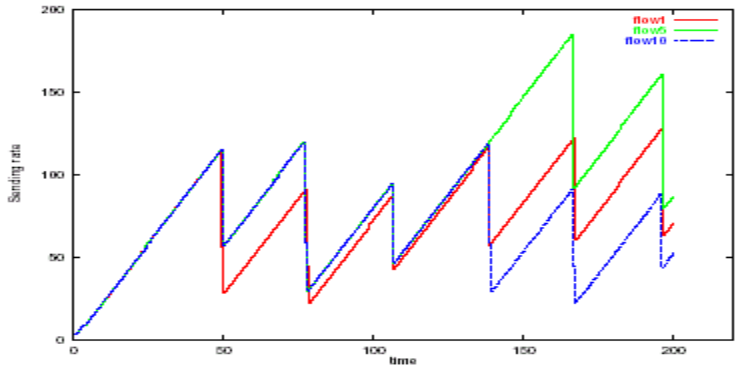
- AIMD (Additive Increase Multiplicative Decrease) :
  - W = W + a (no loss)
  - W = W * (1 – b)

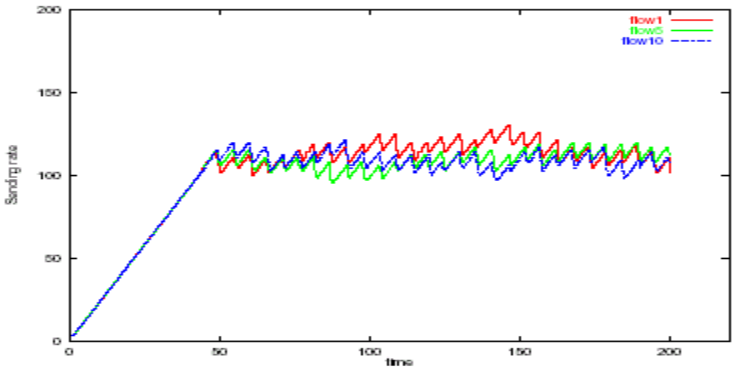
    (L > 0 losses)

  it achieves a b/w ∝1/sqrt(L)
- AIPD (Additive Increase Proportional Decrease):
  - W = W + a (no loss)
  - W = W * (1 –b *L)

    (L > 0 losses)

  it achieves a b/w ∝1/L





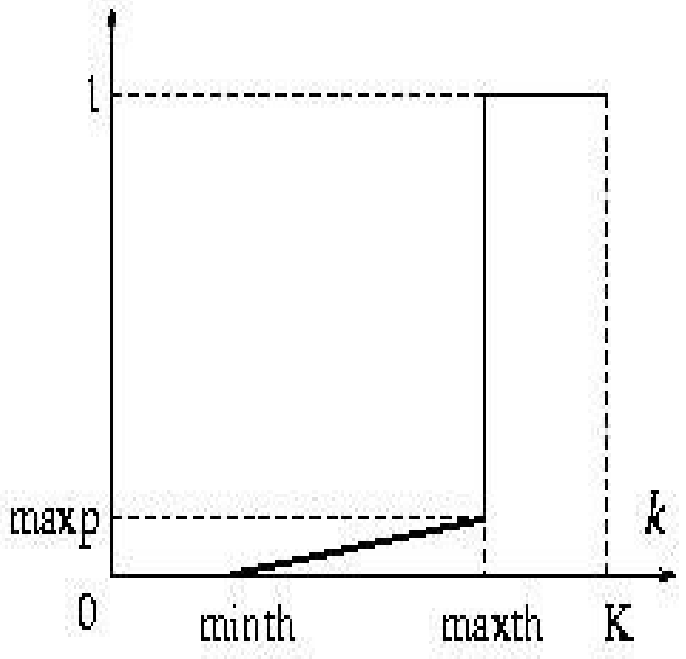
(ns2 plot of 3 flows,source Lee 2001)

TCP buffer tuning

- The optimal TCP flow control window size is the bandwidth delay product for the link. In these years as network speeds have increased, o.s. have adjusted the default buffer size from 8KB to 64KB. This is far too small for hpn. An OC12 link with 50 ms rtt delay requires at least 3.75 MB of buffers ! There is a lot of reseach on mechanisms to automatically set an optimal flow control window and buffer size, just some examples :
  - **Auto-tuning** (Semke,Mahdavi,Mathis 1998)
  - **Dynamic Right Sizing**(DRS) :
    - Weigl, Feng 2001 : *Dynamic Right-Sizing: A Simulation study* .
  - **Enable** (Tierney et al LBNL 2001) : database of BW-delay products
  - Linux 2.4 **autotuning**/connection **caching**: controlled by the new kernel variables net.ipv4.tcp_wmem/tcp_rmem, the advertised receive window starts small and grows with each segment from the transmitter; tcp control info for a destination are cached for 10 minutes(cwnd,rtt,rttvar,sshthresh)

RED /1
(Random Early Detection)

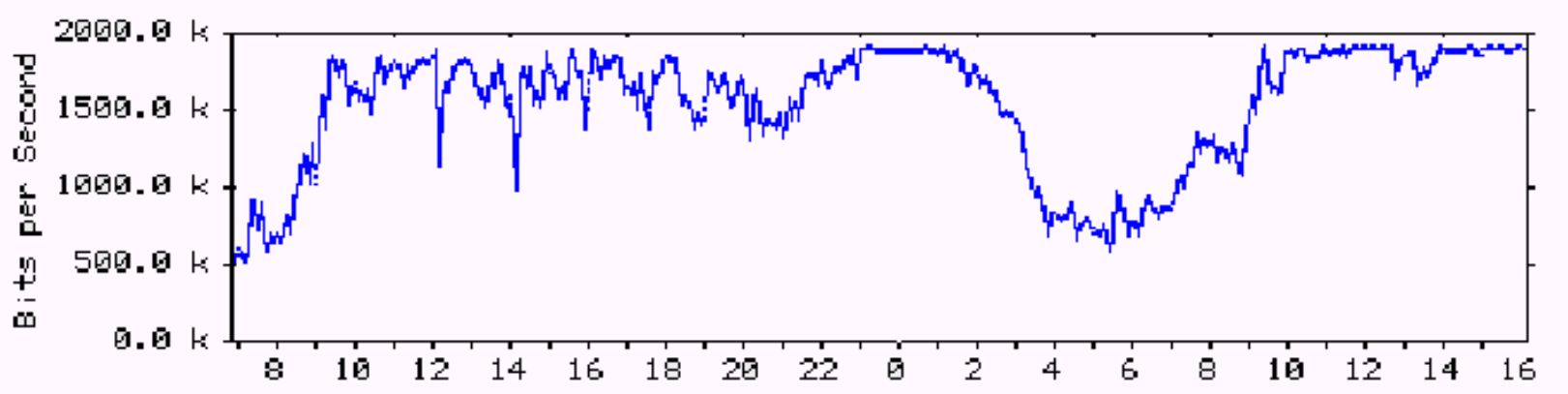
- Floyd, V.Jacobson 1993
- Detects incipient congestion and drops packets with a certain probability depending on the queue length. The drop probability increases linearly from 0 to *maxp* as the queue length increases from *minth* to *maxth*. When the queue length goes above *maxth (maximum threshold)* all pkts are dropped.
- Used together with ECN this algorithm can just signal ECN capable connections, instead of dropping packets



RED /2

- It is now widely deployed
- There are very good performance reports
- Still an active area of research for possible modifications
- Cisco implements its own WRED (Weighted Random Early Detection) that discards packets based also on precedence (provides separate threshold and weights for different IP precedences)

RED /3



- (from Van Jacobson, 1998) Traffic on a busy E1 (2 Mbit/s) Internet link. RED was turned on at 10.00 of the 2nd day, and from then utilization rised up to 100% and remained there steady.

ECN (Explicit Congestion Notification) /1

- RFC 3168 Ramakrishnan, Floyd, Black :

  The addition of Explicit Congestion Notification (ECN) to IP (Sep 2001)
- Uses 2 bits of the IPv4 Tos (Now and in IPv6 reserved as a DiffServ codepoint). These 2 bits encode the states :
  - ECN-Capable Transport : ECT(0) and ECT(1)
  - Congestion Experienced (CE)
- If the transport is TCP, uses 2 bits of the TCP header, next to the Urgent flag :
  - ECN-Echo(ECE) set in the first ACK after receiving a CE pkt
  - Congestion Window Reduced (CWR) set in the first pkt after having reduced cwnd in response to an ECE pkt
- With TCP the ECN is initialized sending a SYN pkt with ECE and CWR on, and receiving a SYN+ACK pkt with ECE on

ECN /2

How it works:

- senders set ECT(0) or ECT(1) to indicate that the end-nodes of the transport protocol are ECN capable
- a router experiencing congestion, sets the 2 bits of ECT pkts to the CE state (instead of dropping the pkt)
- the receiver of the pkt signals back the condition to the other end
- the transports behave as in the case of a pkt drop (no more than once in a RTT)

Linux adopted it on 2.4 ... many connection troubles (the default now is to be off, can be turned on/off using : */proc/sys/net/ipv4/tcp_ecn*). Major sites are using firewalls or load balancers that refuse connections from ECT (Cisco PIX and Load Director, etc).

Bibliography /1

- Van Jacobson,M.Karel : Congestion avoidance and control, 1988
- R.Stevens: TCP/IP Illustrated vol.1 The protocols, 1994
- G.Wright, R.Stevens: TCP/IP Illustrated vol. 2 The Implementation, 1995
- RFC2581 TCP Congestion Control,Allman, Paxson, Stevens (1999)
- RFC2018 TCP Selective Acknowledgements Options, Mathis,Mahdavi, Floyd 1996

Bibliography /2

- Brakmo, Peterson: TCP Vegas End to End congestion avoidance on a Global Internet, 1995
- M.Allman, On the generation and use of TCP Acknowledgments, 1999
- L.Peterson,S.Davies : Computer networks,2000
- Balakrishnan,Seshan : Improving TCP/IP performance over wireless networks, 1995
- Bakre,Badrinath: I-TCP: Indirect TCP for mobile host, 1994