Internet Programming & Protocols
Lecture 11

TCP evolution …

TCP congestion control

TCP Tahoe

www.cs.utk.edu/~dunigan/ipp/

TCP congestion avoidance (1984)

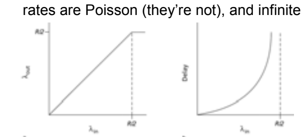
- RFC 896 (1984) noted performance problems with growing Internet
- 1) Excess of small packets (inefficient)
  - Silly window syndrome (Nagle fix)
  - Too many ACKs (delayed ACK fix)
- 2) congestion collapse
  - Interaction of reliable TCP on top of unreliable IP
  - Problems at routers connecting links of widely different bandwidths
  - Queues grow and overflow
  - Senders are retransmitting but not adjusting sending rate, so problem worsens
  - Little new data getting through … network collapse
- Congestion fix ('84):
  - Routers send ICMP source quench when queues start to build
    - This is congestion avoidance
  - When TCP sender receives a source quench, set "effective window" to zero for 10 ACKs or so ?
  - Source quench still allows ACKs and retransmissions

TCP congestion 1988

- The 1984 "recommendations" helped some …
- Problems
  - Traffic bursty – sudden build up of queues and RTT
  - Not all routers would send ICMP source quench
  - Not all senders would respond to source quench with rate reduction
  - At time of congestion when things are real "busy", the router is supposed to figure out who the big senders are and send 'em ICMP messages
    - Takes time away from forwarding operation (draining queue)
    - Actually injects MORE packets into the network
- October '86 (Van Jacobson)
  - Data rate between Internet sites LBL and UC Berkeley (400 yards) dropped by a factor of 1000! Congestion collapse was back.
  - Recommendations (and implemented in 4.3 BSD)
    - Better RTT variance estimation ✓
    - Exponential retransmit timer backoff ✓
    - Slow-start ✓
    - Congestion control (cwnd and ssthresh) (not congestion avoidance)

M/M/1 queues (text App. A)

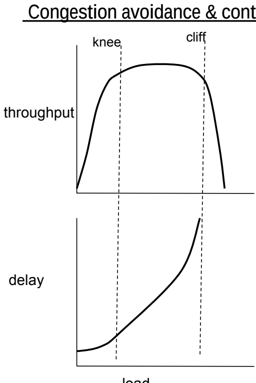
- Network congestion can be viewed as classic queuing problem
- Packets enter router at some arrival rate λ (packets/sec), router tries to forward them on at some (server) rate μ. Queue can build even if λ = μ
  - Server rate == transmission delay, e.g 200kbs link, 40 ms to put 1 KB pkt on wire
  - 10 pkts in queue ahead of you, your RTT increases by 10*40 == 400 ms
- Analytical queuing models allow us to predict queuing times, mean number of packets in the queue, loss rates as function of μ and λ
- For M/M/1 queues, assumptions are service times are exponential, arrival rates are Poisson (they're not), and infinite queues! ☹



© Kurose

But the basic principles apply, throughput increases with the arrival rate, but delay increases as the queues build.

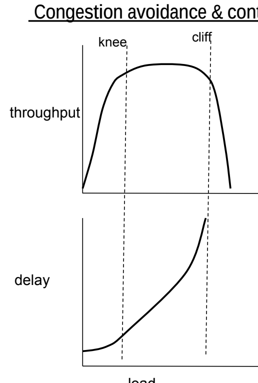
Congestion avoidance & control (Jain '88)



If loads are small the network can keep up. After the load reaches the network capacity, throughput stops increasing and delay (response) gets slower and slower.

If the load continues to grow, congestion occurs and packets are dropped and throughput starts to drop. If senders are retransmitting packets already in the net, very little new (good) data gets through (congestion collapse).

Congestion avoidance & control



At the knee, the increase in throughput is small, but delay (response time) starts to increase dramatically.

At the cliff packets are being dropped and throughput is falling.

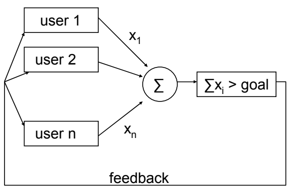
Schemes that allow a network to operate to the left of the knee are congestion avoidance schemes.

Congestion control schemes try to keep the network operating in the region to the left of the cliff. (TCP)

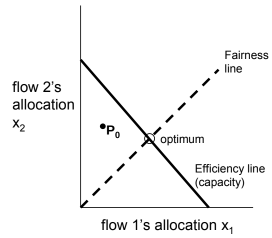
Such schemes are "system control problems" where the system senses its state and provides feedback to the senders.

Control system model of a network

- N users sharing resource
- Each user presents a load (x_i) e.g, packets/sec
- Network provides some sort of feedback so users can adjust (increase or decrease) their offered load over time to achieve operating goal



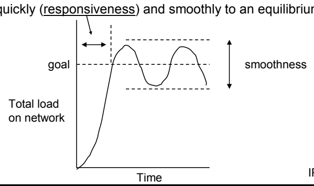
Two user case



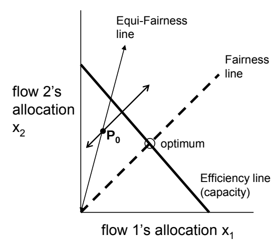
- Flows are in pkts/sec or bytes/sec
  - Sending rate (offered load)
- Blue line is link capacity
  - Above blue line, over utilized
  - Below blue line, under utilized
- Points along Fairness line mean both flows have equal amounts
  - Left of fairness line – x_2 has more
  - Right of fairness line -- x_1 has more
- Question: how to converge to optimum?

Selecting a rate adjustment algorithm

- linear vs non-linear
  - $x_i(t+1) = a\, x_i(t)+b$
  - $x_i(t+1) = x_i(t) + a\, x_i(t)^k$
- Critera
  - Efficient – operating just under capacity line
  - Fair
    - Roughly, N users should each get 1/N of the capacity
    - $(\sum x_i)^2/(n(\sum x_i^2)) = 1$ fair
  - Converges quickly (responsiveness) and smoothly to an equilibrium

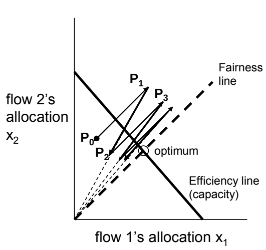


Linear adjustments



- Linear rate adjustment ax +b
  - a_D and b_D for decrease
  - a_I and b_I for increase
- Four possibilities
  - Additive only (a=0)
  - Multiplicative only (b=0)
  - Additive increase, multiplicative decrease (AIMD)
  - Multiplicative increase, additive decrease (MIAD)
- Additive only (magenta) (x+k,y+k)
  - Doesn't converge to optimum
- Multiplicative only (green) (kx,ky)
  - Doesn't converge to optimum
- MIAD
  - Doesn't converge to optimum

Additive increase, multiplicative decrease (AIMD)



- If over-utilized, decrease rapidly (conservative)
- If under-utilized, increase gradually
- Converges to optimum!
- Jain ('87) (DECnet) suggested
  - Multiplicative decrease 7/8
  - Additive increase 1
- If net congested, decrease rate multiplicatively, otherwise increase rate additively
- How do we know net is congested? What is the feedback mechanism?

Congestion control feedback

Network assisted (explicit)

- Routers provide feedback
  - ICMP source quench ☹
  - Congestion bit (DECnet, SNA, ECN)
  - Rate sender should send at

End to end (implicit)

- No explicit feedback from network
- End node infers congestion from
  - increased delay (RTT) – Vegas
    - knee
  - or packet loss (TCP)
    - cliff

TCP originally had network assist (source quench). Today TCP uses packet loss. Packet loss is ambiguous (loss may be due to something other than congestion, bit error), so today there are proposals for network assist through Explicit Congestion Notification (ECN bit) and more Active Queue Management (AQM) in the routers (more later).

TCP congestion control '88

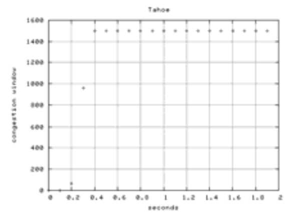
- Based on Jain work, Van Jacobson proposed dynamic window sizing upon packet loss in TCP (sender rate adjustment)
- Implemented in 4.3 BSD (Tahoe) combined with slow-start
- TCP sender maintains two new state variables
  - Congestion window (**cwnd**)
  - Slow-start threshold (**ssthresh**)
- cwnd starts at 1, during slow-start, incremented by 1 for every ACK received. In steady state, grows to min(SNDBUF, advertised window)
- On a timeout, record half the cwnd in ssthresh (multiplicative decrease) and set cwnd to 1 and begin slow-start. When cwnd reaches ssthresh switch to additive increase (add 1 to cwnd every RTT), the congestion avoidance phase.

AIMD
a = 0.5
b = 1

```
/* ACK arrived */
if (cwnd < ssthresh) cwnd += 1;  /* slow-start, exponential */
  else cwnd += 1/cwnd;   /* congestion avoidance */
```


TCP tahoe – no packet loss

- If there is no packet loss, congestion window (cwnd) grows in slow-start til it reaches min(sender's SNBUF, receiver's RCVBUF)
- cwnd is just the amount of data to send in one RTT



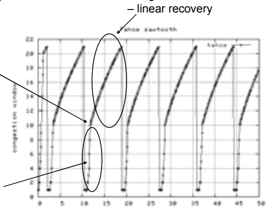
- Not very interesting (trace from ns simulation)

TCP Tahoe

- AIMD (1,0.5)
  - Data rate is cut in half on a timeout (Jain said cut it only by 1/8)
- Sender can not send more data than min(cwnd, SNDBUF, adv. window)
- ssthresh usually initialized to infinity or receiver's advertised window
- TCP detects link capacity by increasing cwnd til there is a packet loss!
- With a bottleneck link (router drops), you get a sawtooth like pattern
  - SNDBUF too big

Congestion control – linear recovery

ssthresh

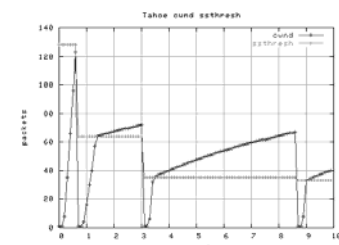


slow-start

cwnd to 21, packet loss sets ssthresh to cwnd/2 (10), and then cwnd← 1

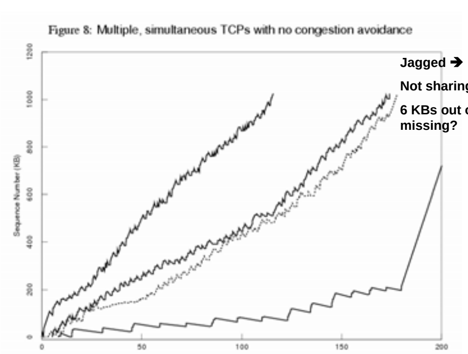
Tahoe with multiple losses

- If another loss occurs during recovery, cwnd is cut in half again …
- ssthresh is set to current cwnd/2



As load increases on net, cwnd for your flow decreases. If load decreases, your cwnd will slowly increase.

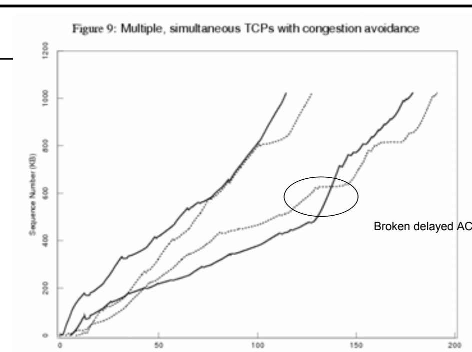
Van jacobson figures



Jagged → losses

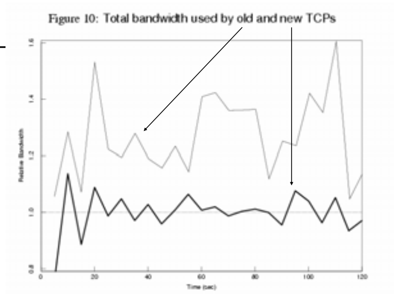
Not sharing equally

6 KBs out of 25 KBs missing?



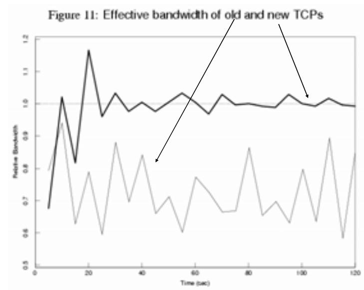
Broken delayed ACK

- With modified TCP (Tahoe), fewer losses and fairer



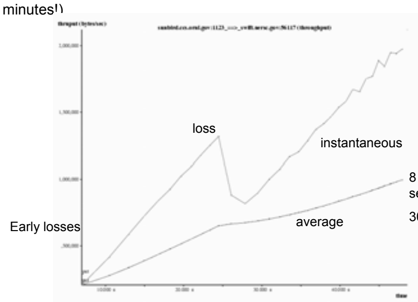
- Old senders send about 25% more than will fit on the wire

- Old senders are using (goodput) only 75% of link, the remainder is being used (wasted) for retransmissions of packets that didn't need to be retransmitted



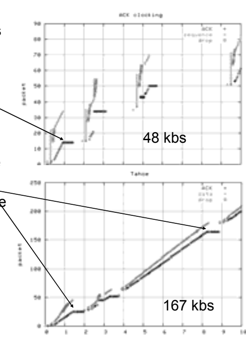
Long fat pipe nasty

- Packet loss in slow-start – linear recovery!!
- If RTT is 160 ms, window of 1550 packets, loss when cwnd=50, then cwnd+1 every RTT. That will take 1500 RTT's or 240 seconds (4 minutes!)



Fast retransmit

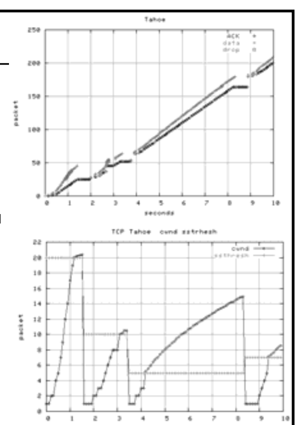
- Observe all the dup ACK's when packet is lost, then the timeout
- Dup ACK's tell us a packet is missing (or out of sequence) AND that other packets seem to be reaching the destination
- Jacobson ('88, Tahoe) recommends retransmitting the missing segment if three consecutive duplicate ACK's arrive – "fast retransmit" and enter congestion avoidance
  - ssthresh ← cwnd/2
  - cwnd ← 1
    - Means wait til all inflight data ACK'd
- This avoids waiting for timeout, improves performance! (not as net friendly as those big timeout pauses ☺)



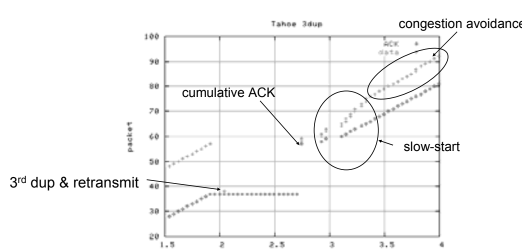
No more window blasts, use slow-start
Avoid timeouts with 3 dup ACKs

Tahoe fast retransmit

- OS keeps a dup ACK counter
- If counter == threshold enter SS
- If "advancing" ACK arrives, clear counter
- Dup threshold is 3 on most OS
  - Some have configurable threshold
  - Linux dynamically adjusts
- Dup threshold too small -- have unneeded retransmits, add to net congestion – packet out of order
- Threshold too big -- don't respond quickly enough or end up timing out (poorer performance)
- If retransmitted packet is also lost, timeout will eventually occur

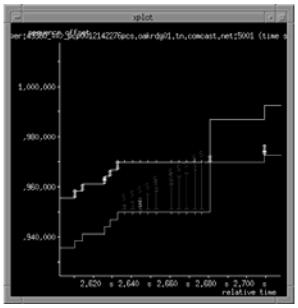


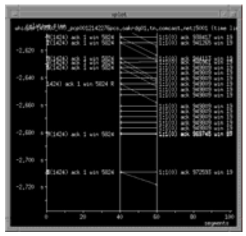
Tahoe fast retransmit



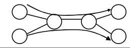
Fast retransmit (tcptrace)

- *tcptrace –rl tmp.dmp* reports ACKs, duplicate ACKs, triple dupACKs
- *xplot* shows tripledup retransmission as green 3

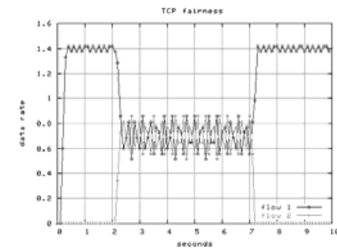




TCP fairness

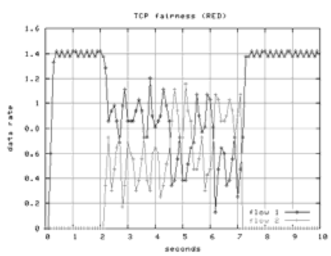


- First flow using all of link capacity, then 2^{nd} flow starts, later stops
- With adequate buffers at router, no losses and fair sharing induced by ACK clocking ANIMATION



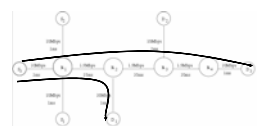
TCP fairness with loss

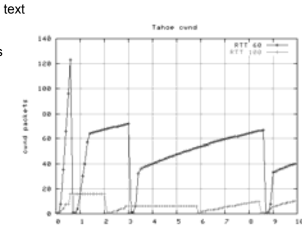
- With smaller queue at router, both flows experience losses at different times, but still pretty fair (worse for droptail queues)



Tahoe and RTT fairness

- Loss recovery is sensitive to RTT
  - Slow-start doubles cwnd (data rate) every RTT
  - Linear recovery increments cwnd by one segment (MSS) every RTT
- Nearby host will recover faster than distant host (droptail queue)
  - Example chap. 11 text
  - Congestion
  - Red flow 1349 kbs
  - Green 60 kbs



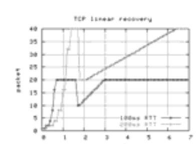


Recovery speeds

- Slow-start data rate (exponential)
  - cwnd ←1 means we have to wait one RTT til all inflight data is ACK'd before we send any new data
  - cwnd (number of segments) doubles every RTT
  - After k RTT's, instantaneous data rate is $(2^{k+1} – 1)$MSS/RTT
  - If available window is N segments, takes $\log_2(N)$ RTTs
- Linear recovery data rate
  - cwnd + 1 every RTT, i.e., data rate increases by MSS/RTT
    - In one second we will add (1/RTT) segments
    - So at end of that second we will have sped up by **MSS/RTT²** bits/sec
    - If you double the RTT, it will take 4 times as long to reach data rate
  - RTT = 100 ms, MSS =1460 Bytes, throughput is increasing by only $1460*8/(0.1)^2$ = 1.168 Mbits/sec
  - If we start at 50 Mbs (cwnd/2), it will take 100/1.168 = 43 seconds to reach 100 Mbs
  - Alternatively, bandwidth-delay window for the path is 856 segments (1460B), so cwnd/2 is 428 segments, @ 1 segment/RTT = 428/0.1 = 42.8 seconds to open window back to 856

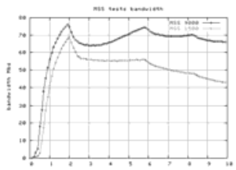
The case for bigger MTU's

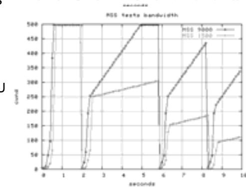
- Recovery rates sensitive to RTT (can't fix that) and MSS
- RTT
  - Double RTT, double slow-start duration
  - But linear recovery takes 4 times longer
    - Window needs to be twice as big
    - RTT twice as long
    - hence square term (MSS/RTT²)
- MSS
  - Ethernet MSS is MTU-headers= 1500 – 40 = 1460
  - Jumbo frame 9180 bytes
    - Reduces slow-start by a couple of RTTs
    - improves linear recovery rate by a factor of 6
  - Vote for bigger MTUs !



1500 Byte MTU vs 9000 byte MTU

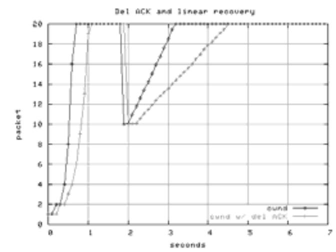
- 100 mbs link, 60 ms RTT, Tahoe
- Jumbo frame slow-start is faster
- Packet drops at 2 s, 6 s, 8 s
  - Jumbo slow-start slightly faster
  - Linear phase 6x faster
- Jumbo frames across the wide area will only work if all intervening routers have jumbo MTUs (OK on Internet2 and ESnet)
- Jumbo speeds up LAN performance
  - TCP data rate 700 mbs @ 1500 MTU but 980 mbs @ 9180 MTU
  - Less packet processing overhead
  - 6x fewer interrupts





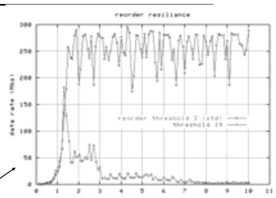
Recovery speed – delayed ACK

- As noted earlier, slow-start doubles once per ACK
- Similarly, linear phase adds 1/cwnd for each ACK
- Delayed ACK can slow both slow-start and linear phase



Packet reordering

- Out of order packets are not uncommon
  - Version of Juniper router under load would reorder packets by 8, e.g., {1,2,3,4,5,6,7,8} would go out {2,3,4,5,6,7,8,1}
- If out of order more than 3, TCP goes into congestion avoidance (cwnd/2), even though there was really no packet loss ☹ (LBL example, 289 unneeded retransmits)



Linux TCP actually "adapts" to flows with packet reordering.

if dupcounter == dupthreshold go into SS/CA

if ACK arrives "soon" (e.g., not lost, just out of order),

"cancel" SS/CA and increment dupthreshold for this flow

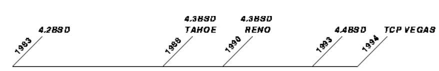
Linux saves last dupthreshold for this target host in local routing table, so next connection to that target will use the larger threshold!

iperf –u –s (UDP) reports packet reordering D-SACK can help.

Do over

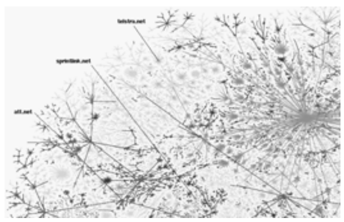
TCP Tahoe summary

- Van Jacobson's tweaks to TCP in 4.3 BSD ('88)
- Exponential backoff on timeouts
- Improved RTT estimator
- Slow-start (startup, packet loss, idle)
- Congestion management (AIMD) cwnd/ssthresh
  - Sender can't send more than min(cwnd, his SNDBUF, receiver's adv. window)
  - If packet loss, cut sending rate in half, then slowly increase
- Fast retransmit (3 dup ACKs), avoid timeouts



TCP control system

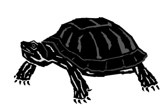
- TCP adjusts sending rate based on feedback (packet loss) from the network
- Adjustment is linear rate control (Additive Increase Multiplicative Decrease)
- The overall system formed by the total number of TCP flows operating across the Internet is one of the largest man-made control systems ever achieved in terms of both geographic scale and the number of inputs and outputs!



Things that slow TCP down

- SNDBUF limits
- RCVBUF limits
- NIC speed or bottleneck link speed
- Packet loss
- Packet reordering
- Delayed ACKs, Nagle
- TCP congestion response
- Application "protocol"

- Recovery rate sensitive to RTT and MSS



Next time …

- More TCP evolution
  - TCP fast recovery, TCP Reno
  - TCP NewReno
  - TCP SACK, D-SACK

assignment 5 and 6