Detecting half-open connections

	TCP A		TCP B
1.	(CRASH)		(send 300, receive 100)
2.	CLOSED		ESTABLISHED
3.	SYN-SENT → <SEQ=400><CTL=SYN>	→	(??)
4.	(!!)	← <SEQ=300><ACK=100><CTL=ACK>	ESTABLISHED
5.	SYN-SENT → <SEQ=100><CTL=RST>	→	(Abort!!)
6.	SYN-SENT		CLOSED
7.	SYN-SENT → <SEQ=400><CTL=SYN>	→	

Observed TCP problems

- Too many small packets
  - Silly window syndrome
  - Nagle's algorithm
- Initial sequence number selection
- Amount of state maintained

Silly window syndrome

- Problem: (Clark, 1982)
  - If receiver advertises small increases in the receive window then the sender may waste time sending lots of small packets
- Solution
  - Receiver must not advertise small window increases
  - Increase window by min(MSS,RecvBuffer/2)

42

Nagle's algorithm

- Small packet problem:
  - Don't want to send a 41 byte packet for each keystroke
  - How long to wait for more data?
- Solution:
  - Allow only one outstanding small (not full sized) segment that has not yet been acknowledged

43

Why is selecting ISN important?

- Suppose machine X selects ISN based on predictable sequence
- Fred has .rhosts to allow login to X from Y
- Evil Ed attacks
  - Disables host Y – denial of service attack
  - Make a bunch of connections to host X
  - Determine ISN pattern and guess next ISN
  - Fake pkt1: [<src Y><dst X>, guessed ISN]
  - Fake pkt2: desired command

44

Time Wait issues

- Web servers not clients close connection first
  - Established → Fin-Waits → Time-Wait → Closed
  - Why would this be a problem?
- Time-Wait state lasts for 2 * MSL
  - MSL is should be 120 seconds (is often 60s)
  - Servers often have order of magnitude more connections in Time-Wait

45

Transport layer: outline

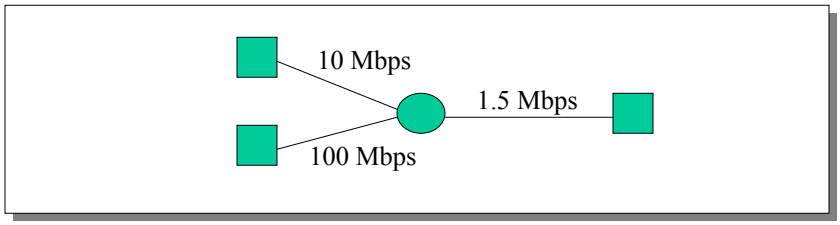
- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
  - Segment structure
  - Reliable data transfer
  - Flow control
  - Connection management
- Principles of congestion control
- TCP congestion control

Principles of congestion control

Congestion:

- Informally: "too many sources sending too much data too fast for *network* to handle"
- Different from flow control!
- Manifestations:
  - Lost packets (buffer overflow at routers)
  - Long delays (queueing in router buffers)
- Another top-10 problem!

Congestion

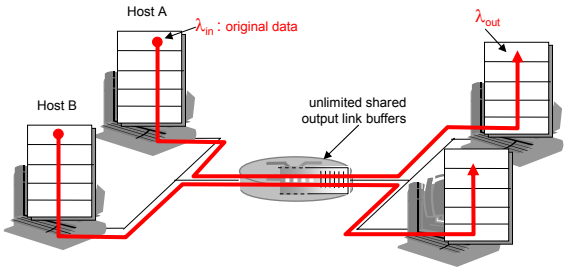


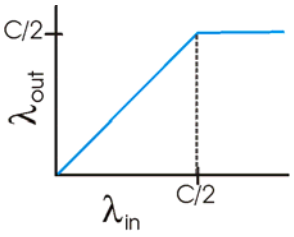
- Different sources compete for resources inside network
- Why is it a problem?
  - Sources are unaware of current state of resource
  - Sources are unaware of each other
  - In many situations will result in < 1.5 Mbps of throughput (congestion collapse)

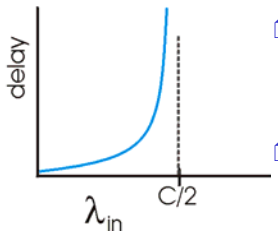
48

Causes/costs of congestion: scenario 1

- Two senders, two receivers
- One router, infinite buffers
- No retransmission





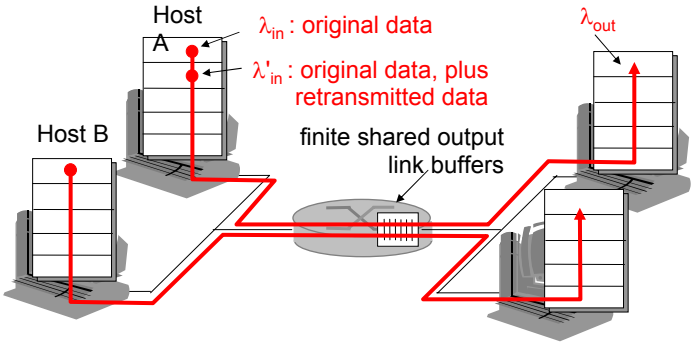


- Maximum achievable throughput
- Large delays when congested

49

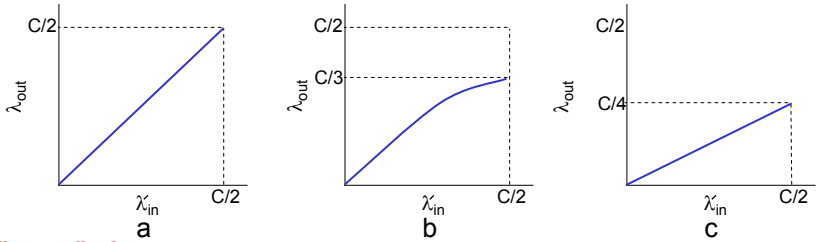
Causes/costs of congestion: scenario 2

- One router, *finite* buffers
- Sender retransmission of lost packet



Causes/costs of congestion: scenario 2

- Always: $\lambda_{in} = \lambda_{out}$ (goodput)
- "Perfect" retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- Retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out}



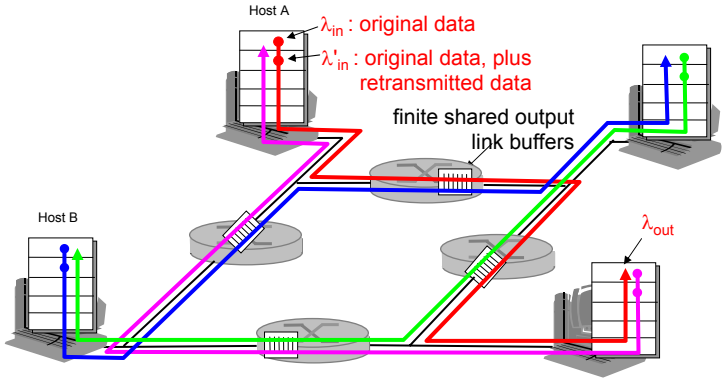
"Costs" of congestion:

- More work (retrans) for given "goodput"
- Unneeded retransmissions: link carries multiple copies of pkt

Causes/costs of congestion: scenario 3

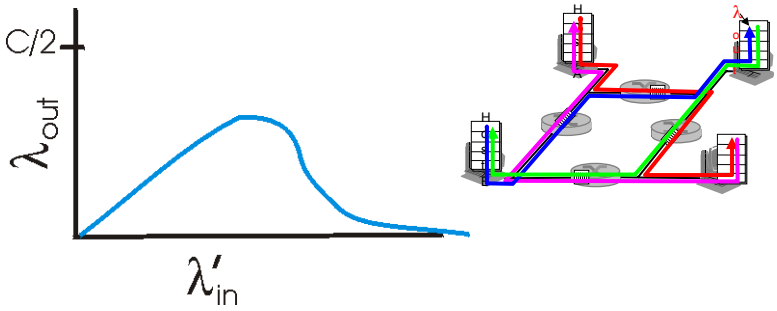
- Four senders
- Multihop paths
- Timeout/retransmit

Q: What happens as λ_{in} and λ'_{in} increase ?



52

Causes/costs of congestion: scenario 3



Another "cost" of congestion:

- When packet dropped, any "upstream" transmission capacity used for that packet was wasted!

53

Congestion collapse

- Definition: *Increase in network load results in decrease of useful work done*
- Many possible causes
  - Spurious retransmissions of packets still in flight
    - Classical congestion collapse
    - How can this happen with packet conservation
    - Solution: better timers and TCP congestion control
  - Undelivered packets
    - Packets consume resources and are dropped elsewhere in network
    - Solution: congestion control for ALL traffic

54

Other congestion collapse causes

- Fragments
  - Mismatch of transmission and retransmission units
  - Solutions
    - Make network drop all fragments of a packet
    - Do path MTU discovery
- Control traffic
  - Large percentage of traffic is for control
    - Headers, routing messages, DNS, etc.
- Stale or unwanted packets
  - Packets that are delayed on long queues
  - "Push" data that is never used

55

Where to prevent collapse?

- Can end hosts prevent problem?
  - Yes, but must trust end hosts to do right thing
  - E.g., sending host must adjust amount of data it puts in the network based on detected congestion
- Can routers prevent collapse?
  - No, not all forms of collapse
  - Doesn't mean they can't help
  - Sending accurate congestion signals
  - Isolating well-behaved from ill-behaved sources

56

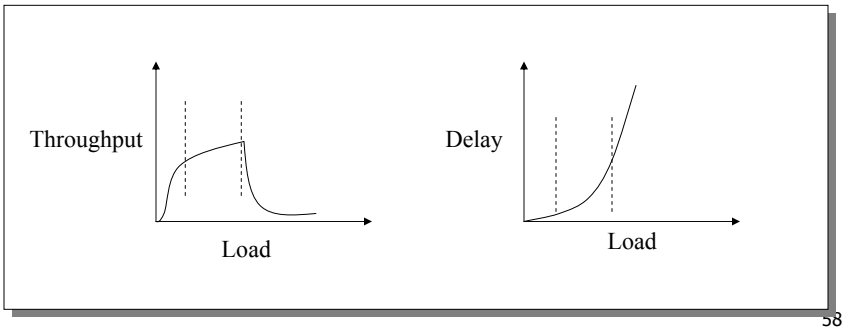
Congestion control and avoidance

- A mechanism which:
  - Uses network resources efficiently
  - Preserves fair network resource allocation
  - Prevents or avoids collapse
- Congestion collapse is not just a theory
  - Has been frequently observed in many networks

57

Congestion control vs. avoidance

- Avoidance keeps the system performing at the knee
- Control kicks in once the system has reached a congested state



58

Approaches towards congestion control

Two broad approaches towards congestion control:

End-end congestion control:

- No explicit feedback from network
- Congestion inferred from end-system observed loss, delay
- Approach taken by TCP

Network-assisted congestion control:

- Routers provide feedback to end systems
  - Choke packet from router to sender
  - Single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - Explicit rate sender should send at

59

Case study: ATM ABR congestion control

ABR: available bit rate:

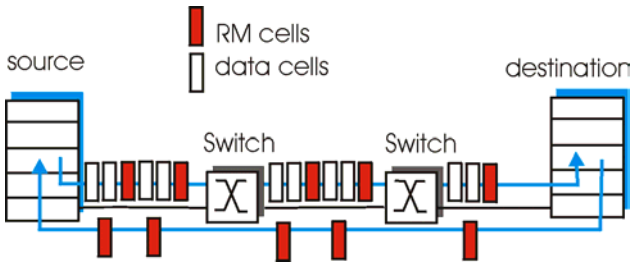
- "Elastic service"
- If sender's path "underloaded":
  - Sender should use available bandwidth
- If sender's path congested:
  - Sender throttled to minimum guaranteed rate

RM (resource management) cells:

- Sent by sender, interspersed with data cells
- Bits in RM cell set by switches ("*network-assisted*")
  - NI bit: no increase in rate (mild congestion)
  - CI bit: congestion indication
- RM cells returned to sender by receiver, with bits intact

60

Case study: ATM ABR congestion control



- Two-byte ER (explicit rate) field in RM cell
  - Congested switch may lower ER value in cell
  - Sender' send rate thus minimum supportable rate on path
- EFCI bit in data cells: set to 1 in congested switch
  - If data cell preceding RM cell has EFCI set, sender sets CI bit in returned RM cell

61

End-to-end congestion control - objectives

- Simple router behavior
- Distributedness
- Efficiency: $X_{knee} = \Sigma x_i(t)$
- Fairness: $(\Sigma x_i)^2/n(\Sigma x_i^2)$
- Power: $(\text{throughput}^{\alpha}/\text{delay})$
- Convergence: control system must be stable

62

Basic control model

- Let's assume window-based control
- Reduce window when congestion is perceived
  - How is congestion signaled?
    - Either mark or drop packets
  - When is a router congested?
    - Drop tail queues – when queue is full
    - Average queue length – at some threshold
- Increase window otherwise
  - Probe for available bandwidth – how?

63

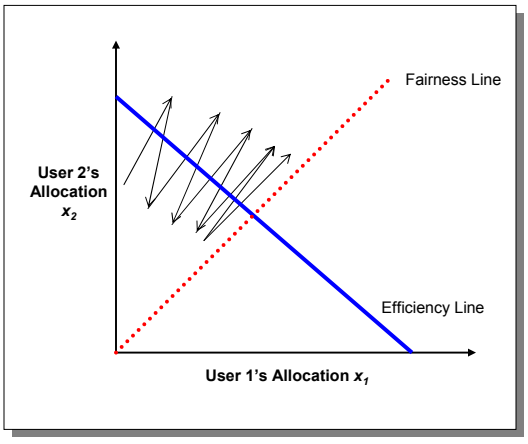
Linear Control

- Many different possibilities for reaction to congestion and probing
  - Examine simple linear controls
  - Window(t + 1) = a + b Window(t)
  - Different a_i/b_i for increase and a_d/b_d for decrease
- Supports various reaction to signals
  - Increase/decrease additively
  - Increased/decrease multiplicatively
  - Which of the four combinations is optimal?

64

Phase plots

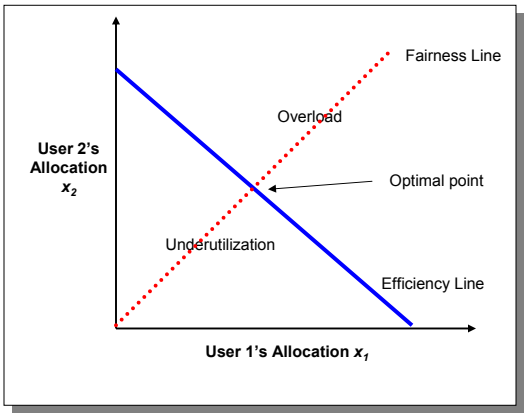
- Simple way to visualize behavior of competing connections over time



65

Phase plots

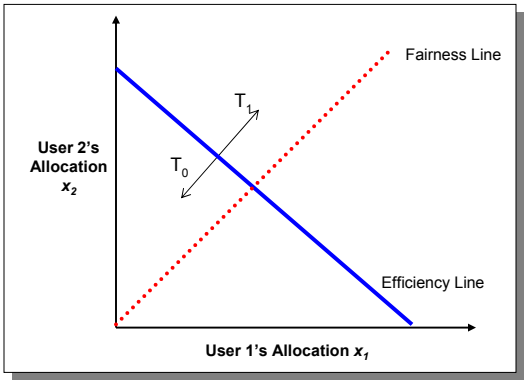
- What are desirable properties?
- What if flows are not equal?



66

Additive increase/decrease

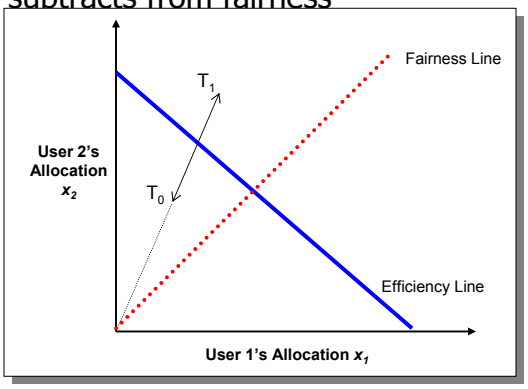
- X_1 and X_2 in-/decrease by the same amount over time
  - Additive increase/decrease – constant fairness



67

Multiplicative increase/decrease

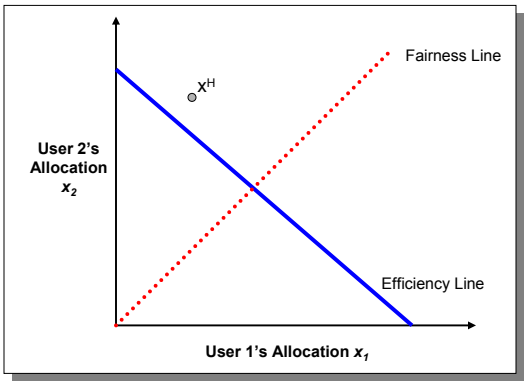
- X_1 and X_2 in-/decrease by the same factor
  - Extension from origin – decrease adds to fairness, increase subtracts from fairness



68

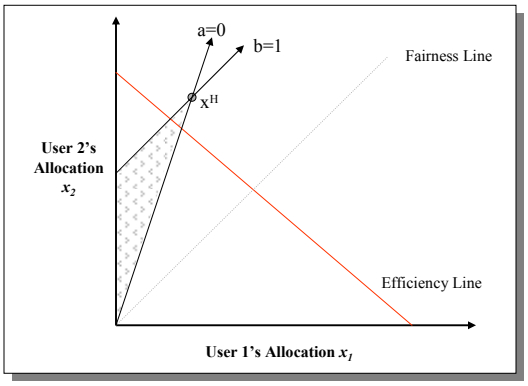
Convergence to efficiency

- Want to converge quickly to intersection of fairness and efficiency lines



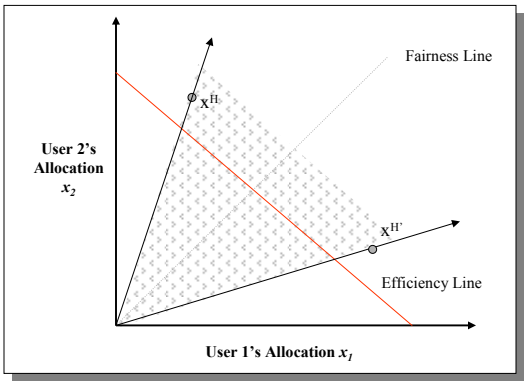
69

Distributed convergence to efficiency



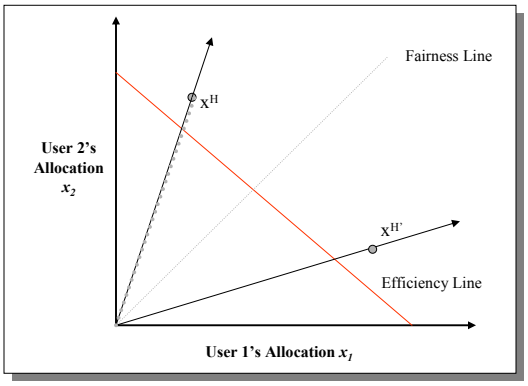
70

Convergence to fairness



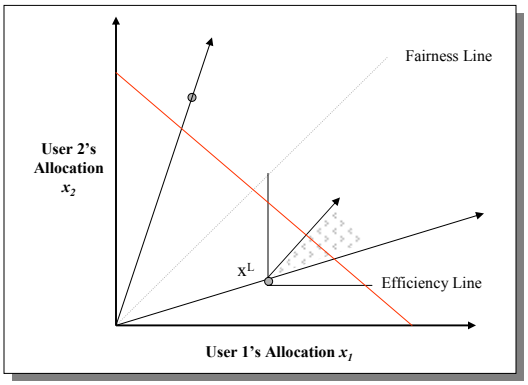
71

Convergence to efficiency & fairness



72

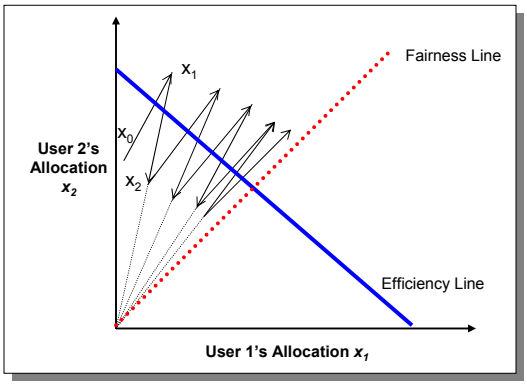
Increase



73

What is the right choice?

- Constraints limit us to AIMD
  - Can have multiplicative term in increase
  - AIMD moves towards optimal point



74

TCP Congestion Control

- Motivated by ARPANET congestion collapse
- Underlying design principle: packet conservation
  - At equilibrium, inject packet into network only when one is removed
  - Basis for stability of physical systems
- Why was this not working?
  - Connection doesn't reach equilibrium
  - Spurious retransmissions
  - Resource limitations prevent equilibrium

75

TCP congestion control - solutions

- Reaching equilibrium
  - Slow start
- Eliminates spurious retransmissions
  - Accurate RTO estimation
  - Fast retransmit
- Adapting to resource availability
  - Congestion avoidance

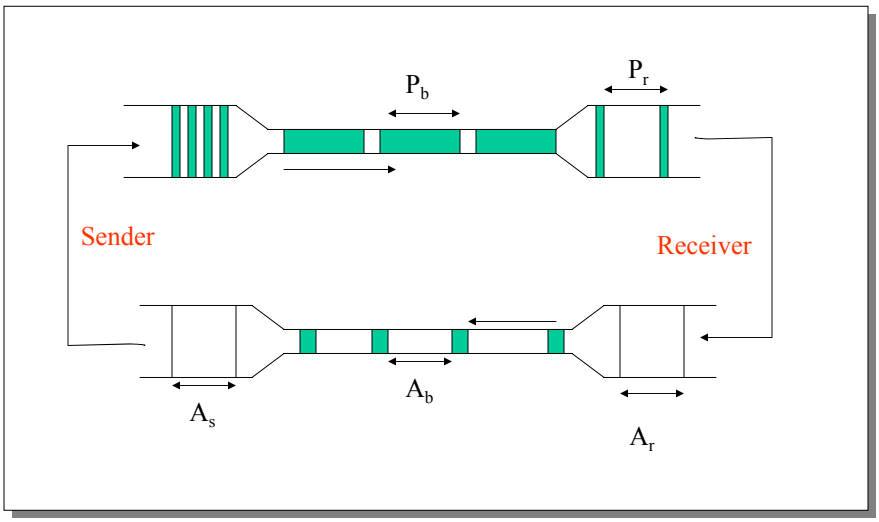
76

TCP congestion control basics

- Keep a congestion window, cwnd
  - Denotes how much network is able to absorb
- Sender's maximum window:
  - Min (advertised receiver window, cwnd)
- Sender's actual window:
  - Max window - unacknowledged segments
- If we have large actual window, should we send data in one shot?
  - No, use acks to clock sending new data

77

Self-clocking



78

TCP congestion control

- End-end control (no network assistance)
- TCP throughput limited by rcvr window (flow control)
- Transmission rate limited by congestion window size, **cwnd**, over segments:

- w segments, each with MSS bytes sent in one RTT:

$$\text{throughput} = \frac{w * MSS}{RTT} \text{ Bytes/sec}$$

79

TCP congestion control:

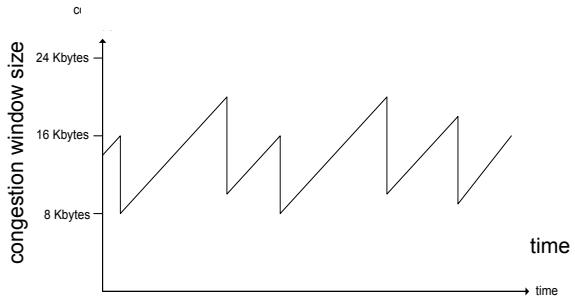
- "Probing" for usable bandwidth:
  - Ideally: transmit as fast as possible (`cwnd` as large as possible) without loss
  - *Increase* `cwnd` until loss (congestion)
  - Loss: *decrease* `cwnd`, then begin probing (increasing) again
- Two "phases"
  - Slow start
  - Congestion avoidance
- Important variables:
  - `cwnd (congwin)`
  - `Threshold:` defines threshold between two slow start phase, congestion control phase

80

TCP congestion control: additive increase, multiplicative decrease

- *Approach:* increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *Additive increase:* increase **cwnd** by 1 MSS every RTT until loss detected
  - *Multiplicative decrease*: cut **cwnd** in half after loss

Saw tooth behavior: probing for bandwidth



81

TCP congestion control: Details

- Sender limits transmission:
  `LastByteSent-LastByteAcked ≤ cwnd`
- Roughly,

$$\text{rate} = \frac{\text{cwnd}}{\text{RTT}} \text{ Bytes/sec}$$

- **`cwnd`** is dynamic, function of perceived network congestion

How does sender perceive congestion?

- Loss event = timeout *or* 3 duplicate acks
- TCP sender reduces rate (**`cwnd`**) after loss event

Three mechanisms:

- AIMD
- Slow start
- Conservative after timeout events

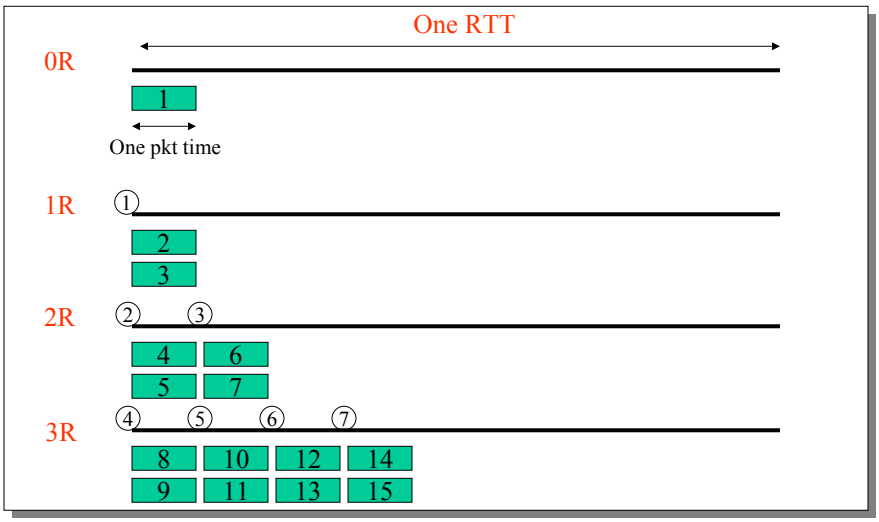
82

TCP Slow start

- How do we get the clocking behavior to start?
  - Initialize cwnd = 1 MSS (typically 1460 bytes)
  - Upon receipt of every ack, cwnd = cwnd + 1 MSS
- Implications
  - Window actually increases to W in RTT * $\log_2(W)$
  - Exponential increase up to first loss event
  - Can overshoot window and cause packet loss
- Summary: initial rate is slow but ramps up exponentially fast

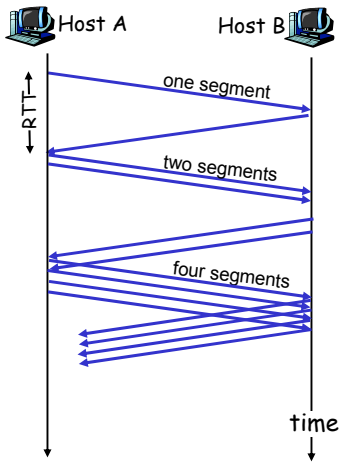
83

Slow start example



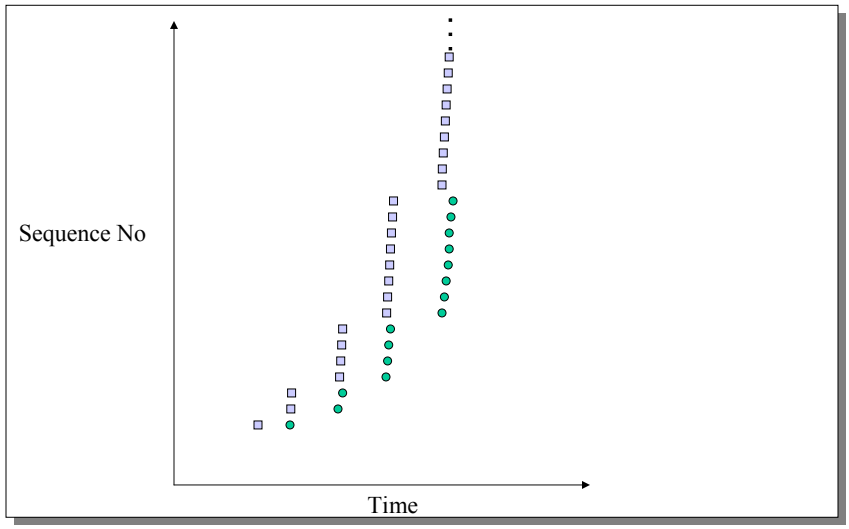
84

Slow start example (cont.)



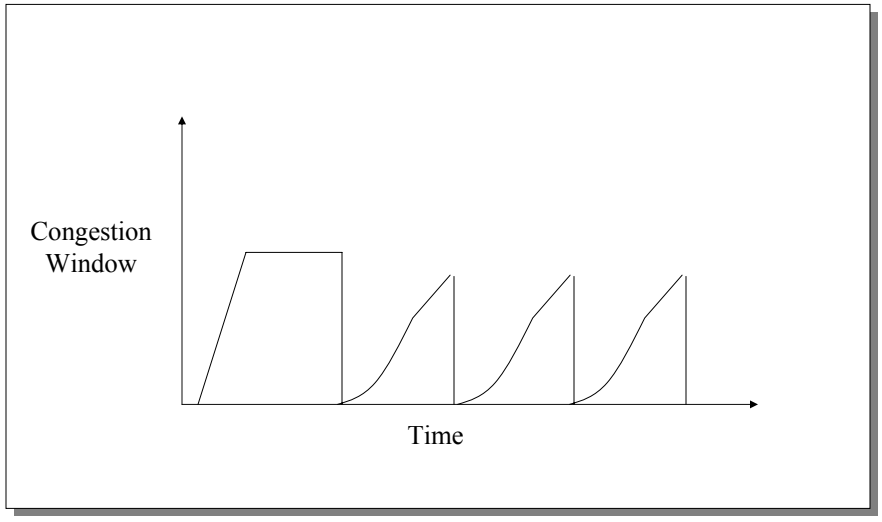
85

Slow start sequence number plot



86

Congestion window



87

Congestion avoidance

- Upon receiving ACK
  - Increase cwnd by 1/cwnd
  - Results in additive increase
- Loss implies congestion – why?
  - Not necessarily true on all link types
- If loss occurs when cwnd = W
  - Network can handle 0.5W ~ W segments
  - Set cwnd to 0.5W (multiplicative decrease)

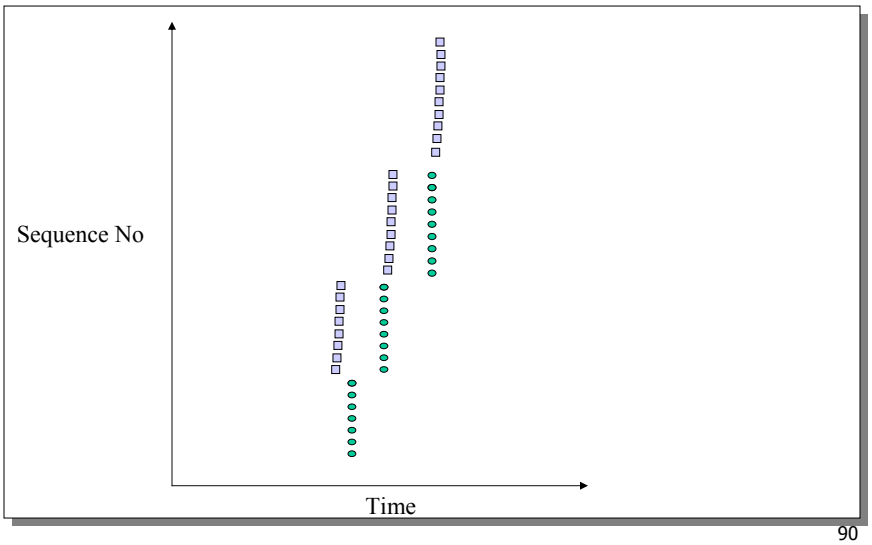
88

Return to slow start

- If packet is lost we lose our self clocking as well
  - Need to implement slow-start and congestion avoidance together
- When timeout occurs set ssthresh to 0.5w
  - If cwnd < ssthresh, use slow start
  - Else use congestion avoidance

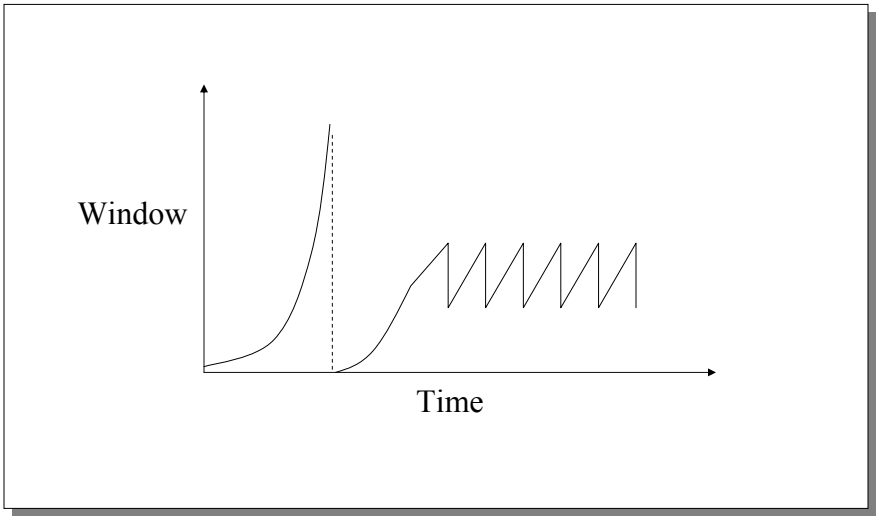
89

Congestion avoidance sequence plot



90

Overall TCP behavior



91

How to change window

- When a loss occurs have W packets outstanding
- New cwnd = 0.5 * cwnd
  - How to get to new state?

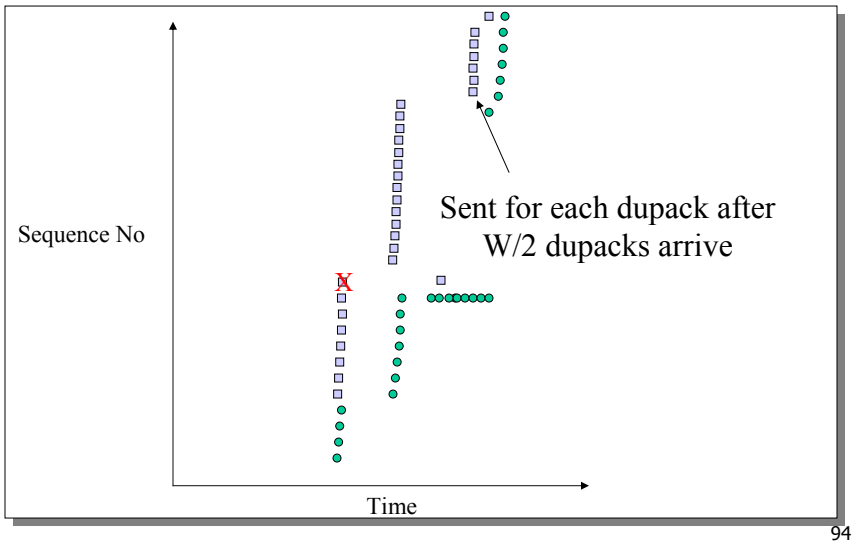
92

Fast Recovery

- Each duplicate ack notifies sender that single packet has cleared network
- When < cwnd packets are outstanding
  - Allow new packets out with each new duplicate acknowledgement
- Behavior
  - Sender is idle for some time – waiting for ½ cwnd worth of dupacks
  - Transmits at original rate after wait
    - Ack clocking rate is same as before loss

93

Fast recovery



94

TCP congestion control: summary

- "Probing" for usable bandwidth:
  - Ideally: transmit as fast as possible (`cwnd` as large as possible) without loss
  - *Increase* `cwnd` until loss (congestion)
  - Loss: *decrease* `cwnd`, then begin probing (increasing) again
- Two "phases"
  - slow start
  - congestion avoidance
- Important variables:
  - `cwnd`
  - `threshold:` defines threshold between two slow start phase, congestion control phase

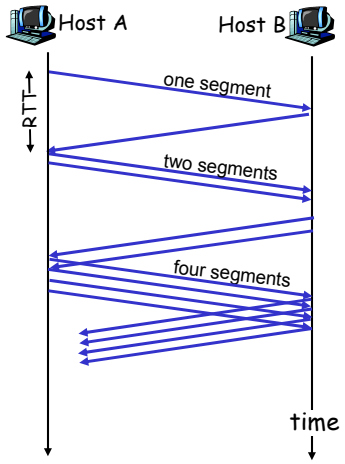
95

TCP slow start

Slowstart algorithm

initialize: **cwnd** = 1 MSS
for (each segment ACKed)
 cwnd += 1MSS
until (loss event OR
 cwnd > threshold)

- Exponential increase (per RTT) in window size (not so slow!)
- Loss event: timeout (Tahoe TCP) and/or or three duplicate ACKs (Reno TCP)

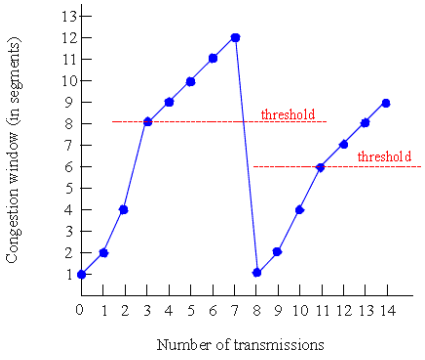


96

TCP congestion avoidance

Congestion avoidance

/* slowstart is over */
/* **cwnd** > threshold */
Until (loss event) {
 every w segments ACKed:
 cwnd += 1 MSS
 }
threshold = **cwnd** /2
cwnd = 1 MSS
perform slow start[1]



1: TCP Reno skips slowstart (fast recovery) after three duplicate ACKs

97

TCP sender congestion control

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	cwnd = cwnd + MSS, If (cwnd > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of cwnd every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	cwnd = cwnd +MSS * (MSS/ cwnd)	Additive increase, resulting in increase of cwnd by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = cwnd /2, cwnd = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. cwnd will not drop below 1 MSS.
SS or CA	Timeout	Threshold = cwnd /2, cwnd = 1 MSS, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	cwnd and Threshold not changed

98

TCP flavors

- Tahoe, Reno, Vegas, SACK
- TCP Tahoe (distributed with 4.3BSD Unix)
  - Original implementation of Van Jacobson's mechanisms (VJ paper)
  - Includes:
    - Slow start
    - Congestion avoidance
    - Fast retransmit

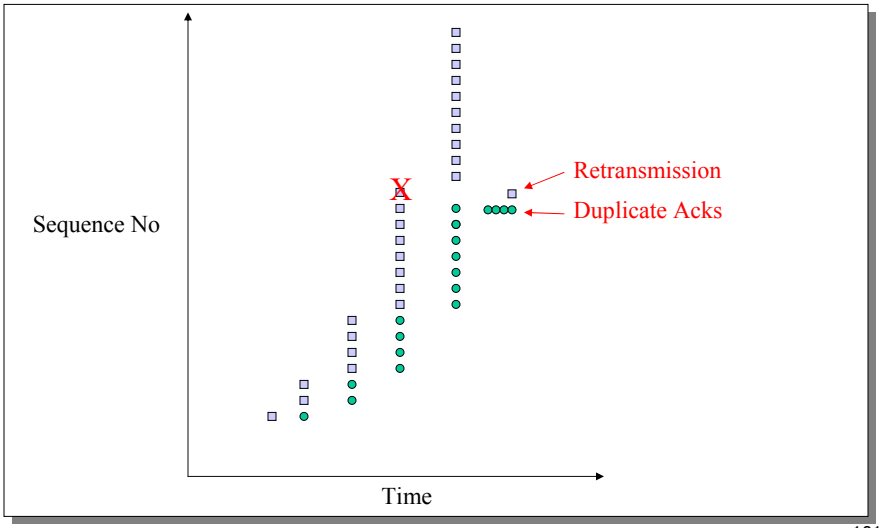
99

Fast retransmit

- What are duplicate acks (dupacks)?
  - Repeated acks for the same sequence
- When can duplicate acks occur?
  - Loss
  - Packet re-ordering
  - Window update – advertisement of new flow control window
- Assume re-ordering is infrequent and not of large magnitude
  - Use receipt of 3 or more duplicate acks as indication of loss
  - Don't wait for timeout to retransmit packet

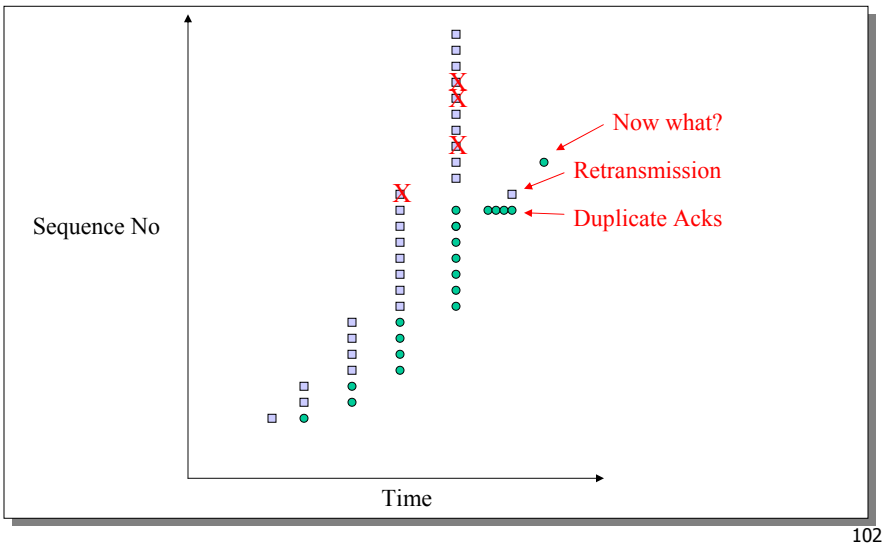
100

Fast retransmit



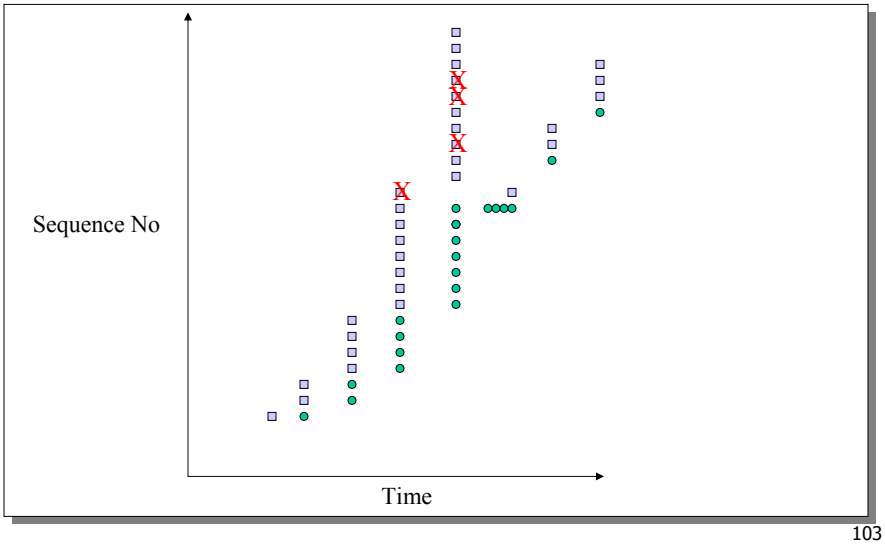
101

Multiple losses



102

Tahoe



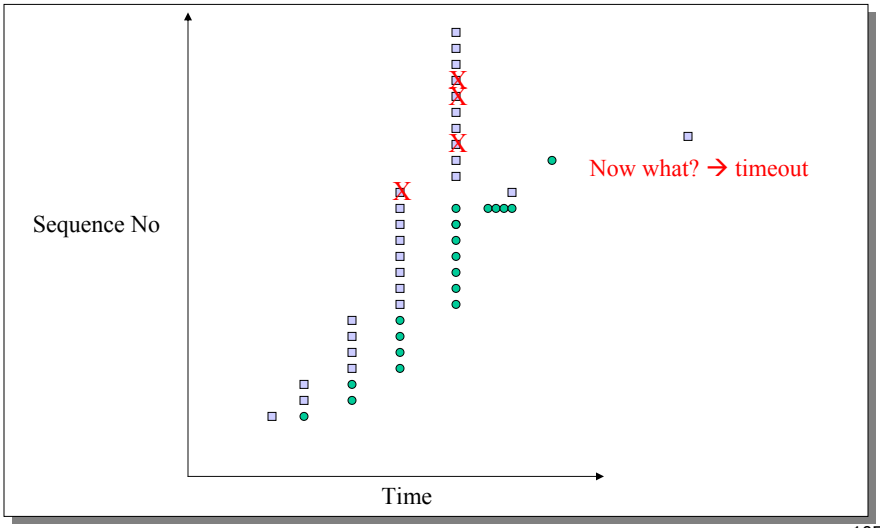
103

TCP Reno (1990)

- All mechanisms in Tahoe
- Addition of fast-recovery
  - Opening up congestion window after fast retransmit
- Delayed acks
- Header prediction
  - Implementation designed to improve performance
  - Has common case code inlined
- With multiple losses, Reno typically timeouts because it does not see duplicate acknowledgements

104

Reno



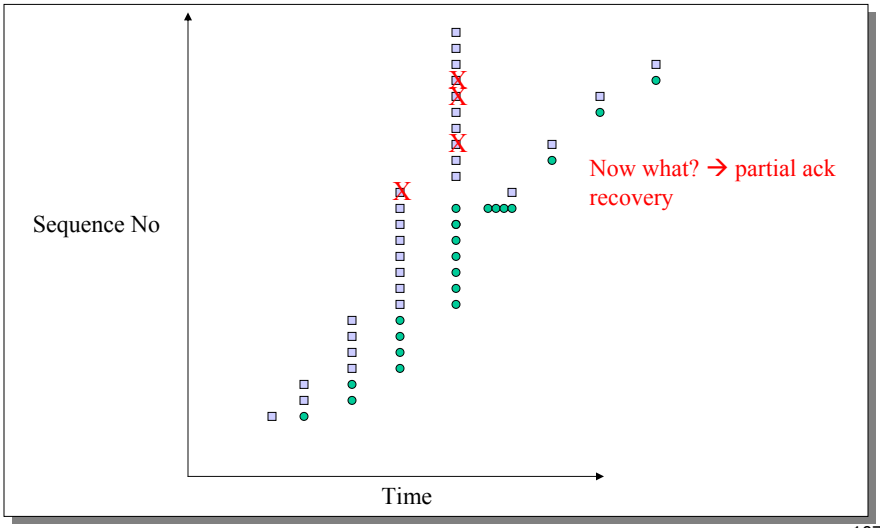
105

NewReno

- The ack that arrives after retransmission (partial ack) should indicate that a second loss occurred
- When does NewReno timeout?
  - When there are fewer than three dupacks for first loss
  - When partial ack is lost
- How fast does it recover losses?
  - One per RTT

106

NewReno



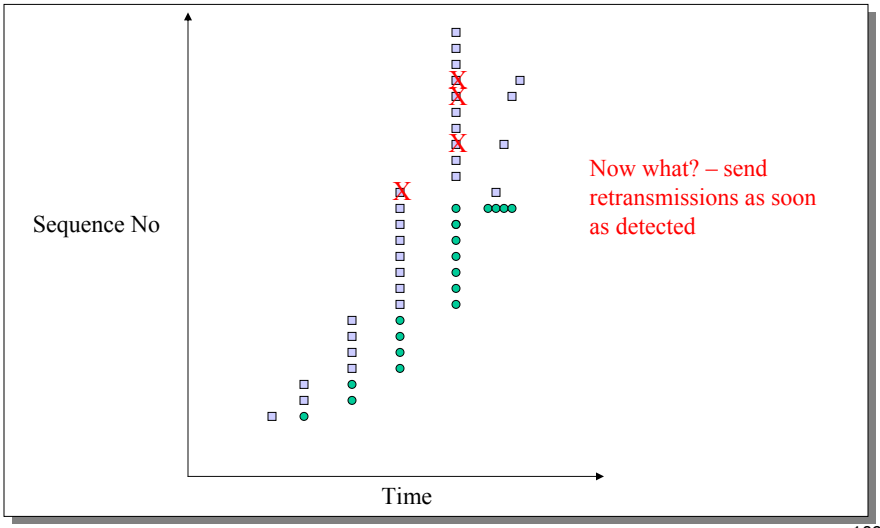
107

SACK

- Basic problem is that cumulative acks only provide little information
  - Ack for just the packet received
    - What if acks are lost? → carry cumulative also
    - Not used
  - Bitmask of packets received
    - Selective acknowledgement (SACK)
- How to deal with reordering

108

SACK



109

Performance issues

- Timeout >> fast rexmit
  - Need 3 dupacks/sacks
  - Not great for small transfers
    - Don't have 3 packets outstanding
  - What are real loss patterns like?
- Right edge recovery
  - Allow packets to be sent on arrival of first and second duplicate ack
  - Helps recovery for small windows
- How to deal with reordering?

110

TCP extensions

- Implemented using TCP options
  - Timestamp
  - Protection from sequence number wraparound
  - Large windows

111

Protection from wraparound

- Wraparound time vs. link speed
  - 1.5Mbps: 6.4 hours
  - 10Mbps: 57 minutes
  - 45Mbps: 13 minutes
  - 100Mbps: 6 minutes
  - 622Mbps: 55 seconds → < MSL!
  - 1.2Gbps: 28 seconds
- Use timestamp to distinguish sequence number wraparound

112

Large windows

- Delay-bandwidth product for 100ms delay
  - 1.5Mbps: 18KB
  - 10Mbps: 122KB > max 16bit window
  - 45Mbps: 549KB
  - 100Mbps: 1.2MB
  - 622Mbps: 7.4MB
  - 1.2Gbps: 14.8MB
- Scaling factor on advertised window
  - Specifies how many bits window must be shifted to the left
  - Scaling factor exchanged during connection setup

113

Maximum segment size (MSS)

- Exchanged at connection setup
  - Typically pick MTU of local link
- What all does this effect?
  - Efficiency
  - Congestion control
  - Retransmission
- Path MTU discovery
  - Why should MTU match MSS?

114

Effects of TCP latencies

Q: client latency from object request from WWW server to receipt?

- TCP connection establishment
- data transfer delay

Notation, assumptions:

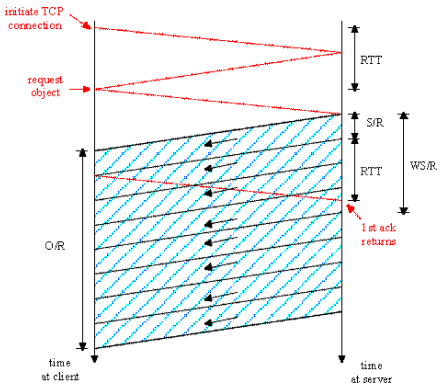
- Assume: fixed congestion window, W, giving throughput of R bps
- S: MSS (bits)
- O: oject size (bits)
- no retransmissions (no loss, no corruption)

Two cases to consider:

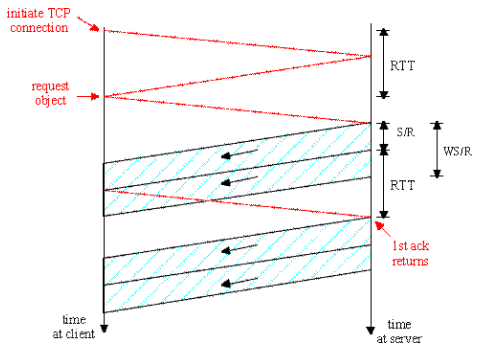
- WS/R > RTT + S/R: ACK for first segment in window before window's worth of data sent
- WS/R < RTT + S/R: wait for ACK after sending window's worth of data sent

115

Effects of TCP latencies



Case 1: latency = 2RTT + O/R



Case 2: latency = 2RTT + O/R + (K-1)[S/R + RTT - WS/R]

Transport layer: Summary

- Principles behind transport layer services:
  - Multiplexing/demultiplexing
  - Reliable data transfer
  - Flow control
  - Congestion control
- Instantiation and implementation in the Internet
  - UDP
  - TCP

Next:

- Leaving the network "edge" (application transport layer)
- Into the network "core"