TCP Performance

EE 122: Intro to Communication Networks

Fall 2006 (MW 4-5:30 in Donner 155)

Vern Paxson

TAs: Dilip Antony Joseph and Sukun Kim

http://inst.eecs.berkeley.edu/~ee122/

Materials with thanks to Jennifer Rexford, Ion Stoica, and colleagues at Princeton and UC Berkeley

1

Announcements

- Homework #4 should be out tonight
- Next lecture: network performance

2

Goals of Today's Lecture

- A look at a range of TCP mechanisms that influence its behavior & performance:
  - Window limitations
  - Timeout, slow start, exponential backoff
  - Acking policies
  - Fast Retransmit
  - Fast Recovery (full AIMD)
  - Window *scaling*
- Visualized using time-sequence plots

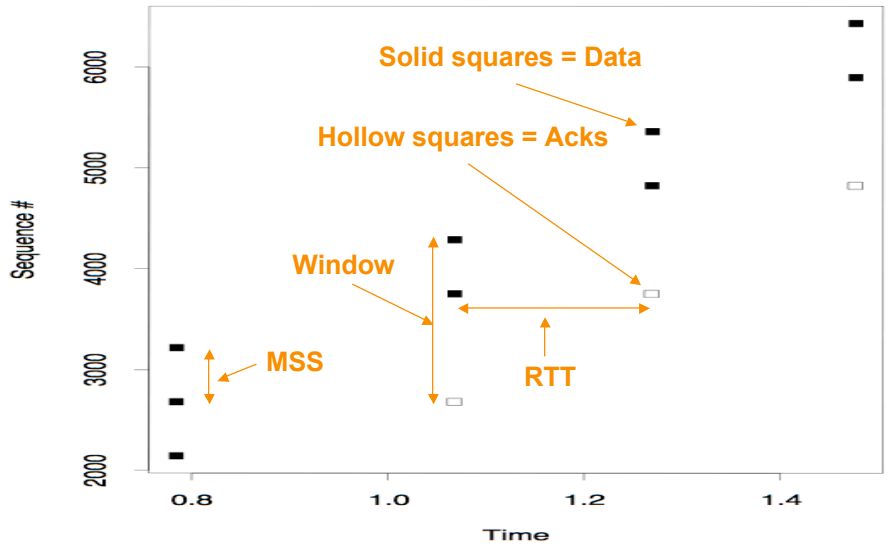
3

Time-Sequence Plots

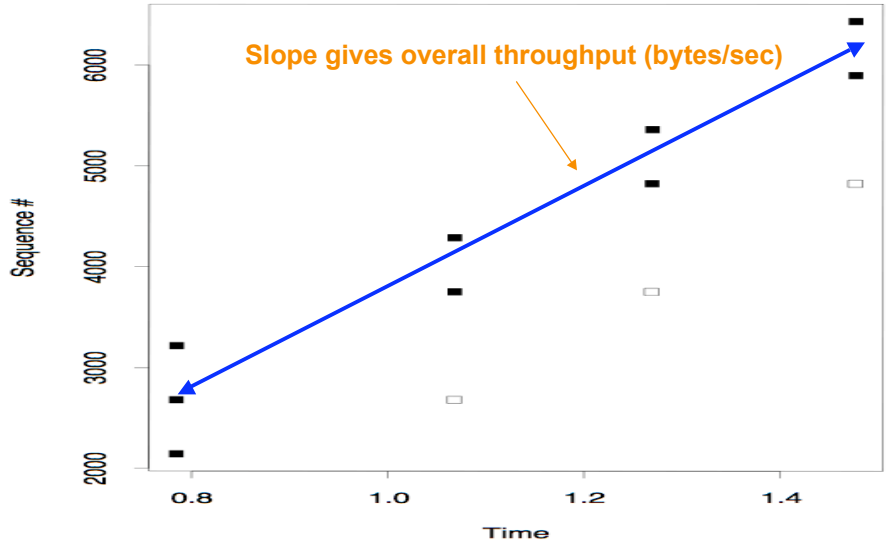
- Powerful tool for visualizing transport protocol dynamics
- Plot sequence number (Y axis) vs. time (X axis)
  - For data packet, sequence number = highest byte carried in packet, so header seqno + payload length
  - For ack, it's just the ack seqno in the header
- Where do you get the data for the plot?
  - Record trace of connection using **tcpdump**
- Illuminates performance achieved & why
  - As well as problems
    - Both network and in endpoint stacks
    - (and also beware measurement errors)

4

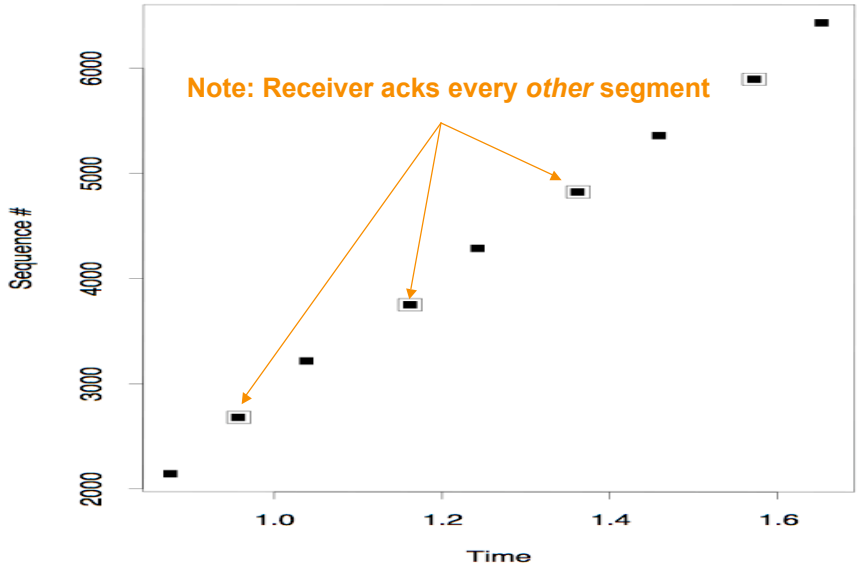
Example of Time-Sequence Plot



Example of Time-Sequence Plot



Same Connection - Why So Different?



Delayed Acknowledgments

- Receiver generally delays sending an ACK
  - Upon receiving a packet, sets a timer
    - Typically, 200 msec; at most, 500 msec
  - If application generates data, go ahead and send
    - And piggyback the acknowledgment
  - If the timer expires, send a (non-piggybacked) ACK
  - If out-of-order segment arrives, immediately ack
  - (if available window changes, send an ACK)
- Limiting the wait
  - Receiver supposed to ACK at least every second full-sized packet ("ack every other")
    - This is the usual case for "streaming" transfers

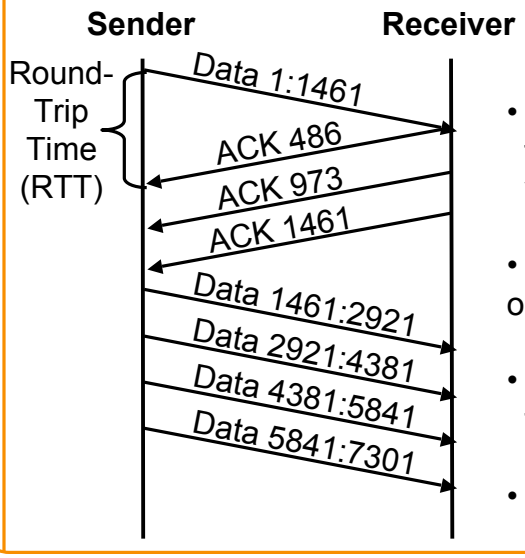
8

Performance Effects of Acking Policies

- How do delayed ACKs affect performance?
  - Increases RTT
  - Window slides a bit later ⇒ throughput a bit lower
- How does ack-every-other affect performance?
  - ***If*** sender adjusts CWND on incoming ACKs, then CWND opens more slowly
    - In slow start, 50% increase/RTT rather than 100%
    - In congestion avoidance, +1 MSS / 2 RTT, not +1 MSS / RTT
- What does this suggest about how a receiver might cheat and speed up a transfer?

9

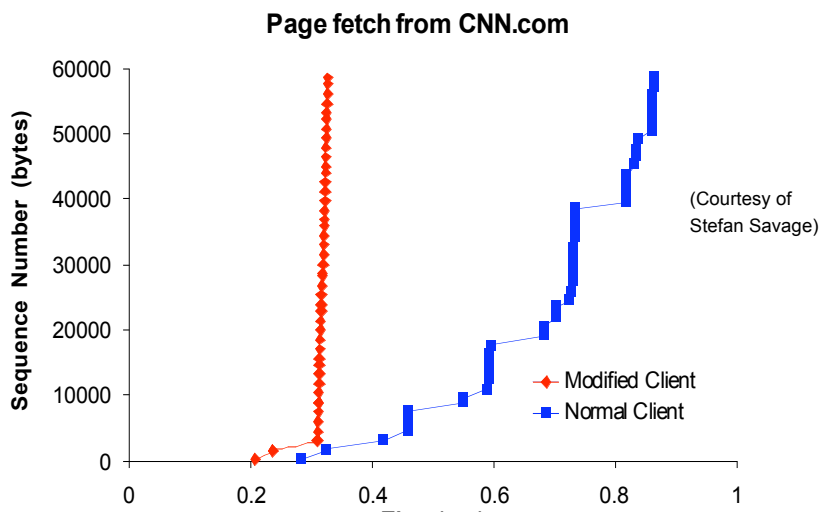
ACK-splitting



- Rule: grow window by one full-sized packet for each valid ACK received
- Send **M** (distinct) ACKs for one packet
- Growth factor proportional to **M**
- What's the fix?

10

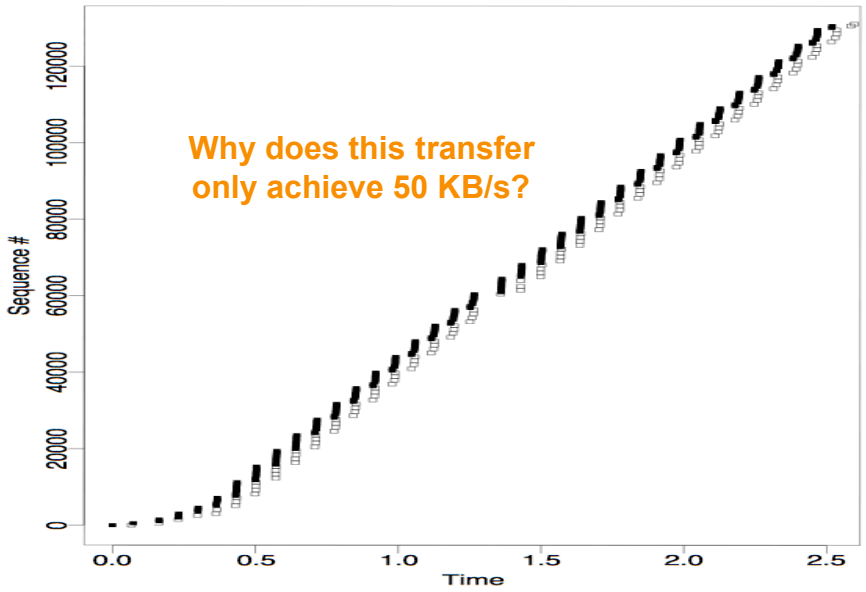
10 line change to Linux TCP



(Courtesy of Stefan Savage)

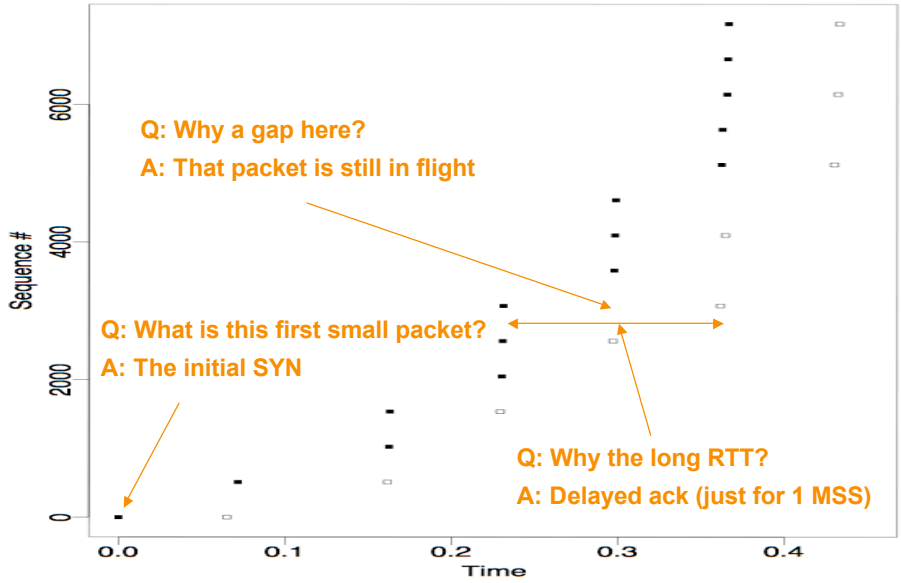
11

Sequence Plot for an Entire Transfer

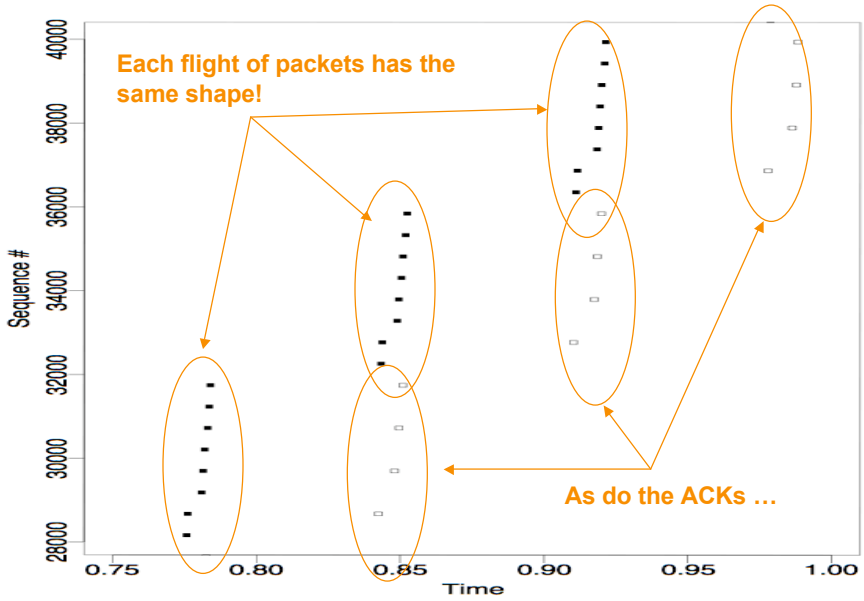


Why does this transfer only achieve 50 KB/s?

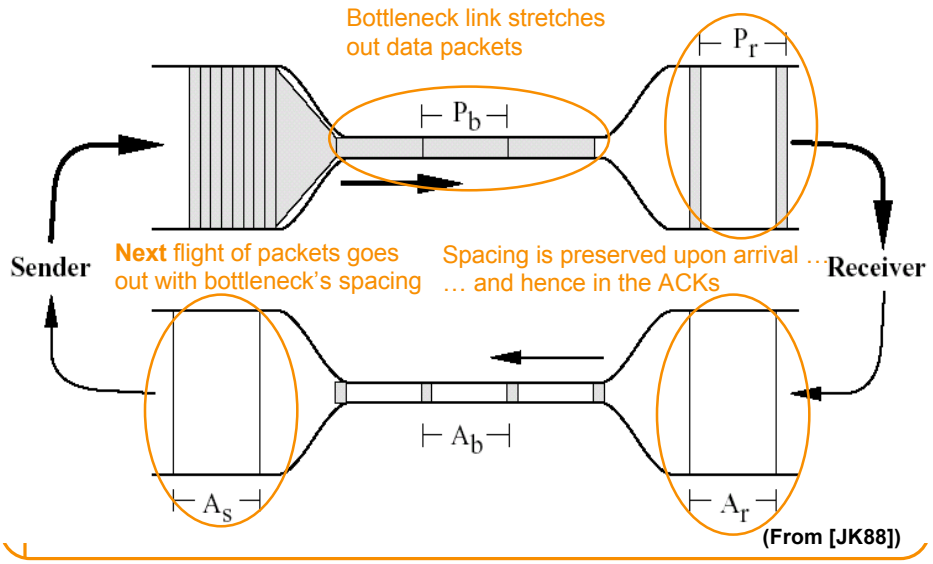
Beginning of Transfer - Slow Start



Mid-Transfer: *Self-Clocking*

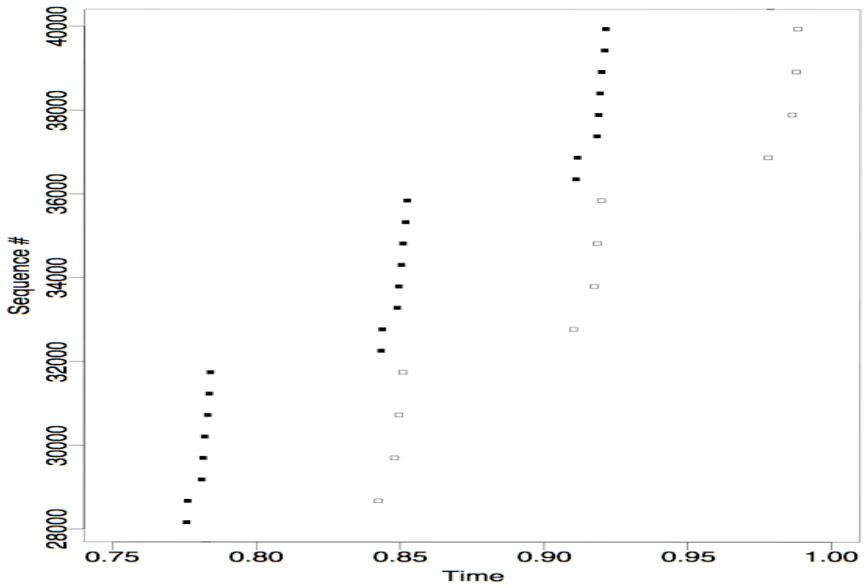


Sliding Window induces *Self-Clocking*

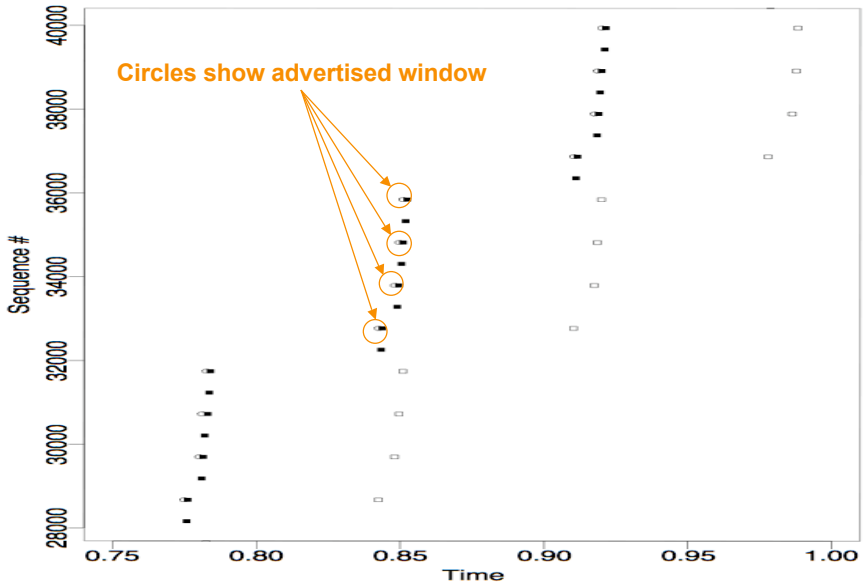


(From [JK88])

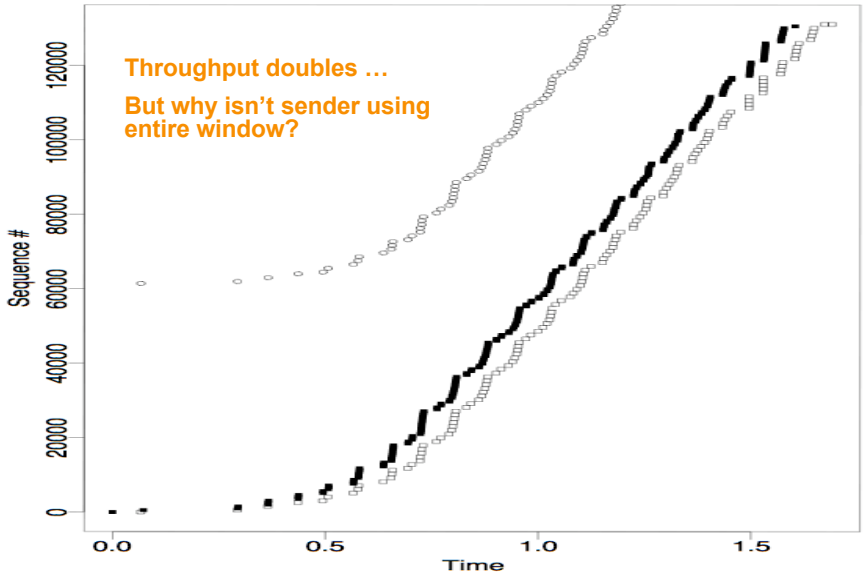
Mid-Transfer: Why Doesn't CWND Grow?



Receiver Window = 8 MSS

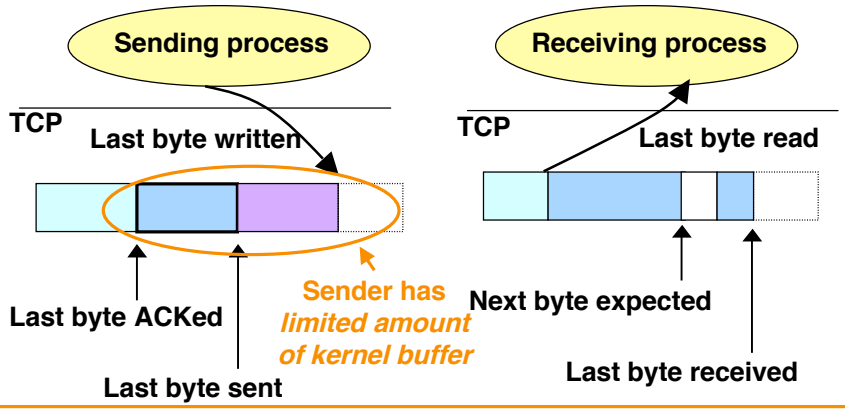


Same Transfer w/ Large Recv. Window



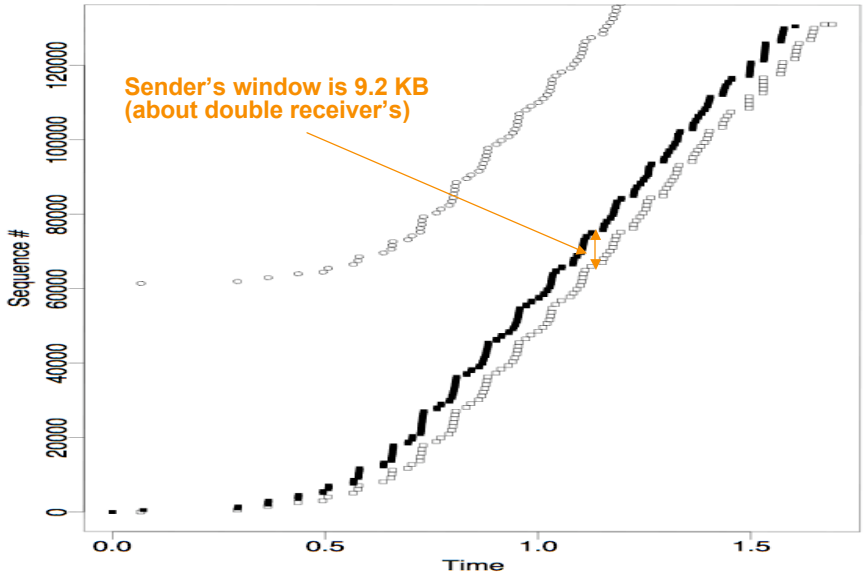
Sliding Window

- Allow a larger amount of data "in flight"
  - Allow sender to get ahead of the receiver
  - … though not *too far* ahead

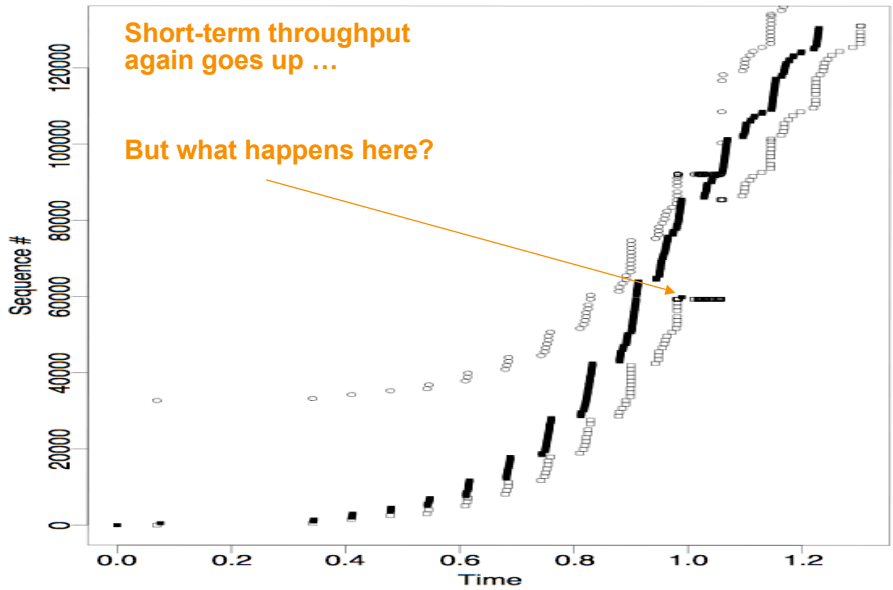


19

Same Transfer w/ Large Recv. Window



Same Transfer w/ Large Snd. Window

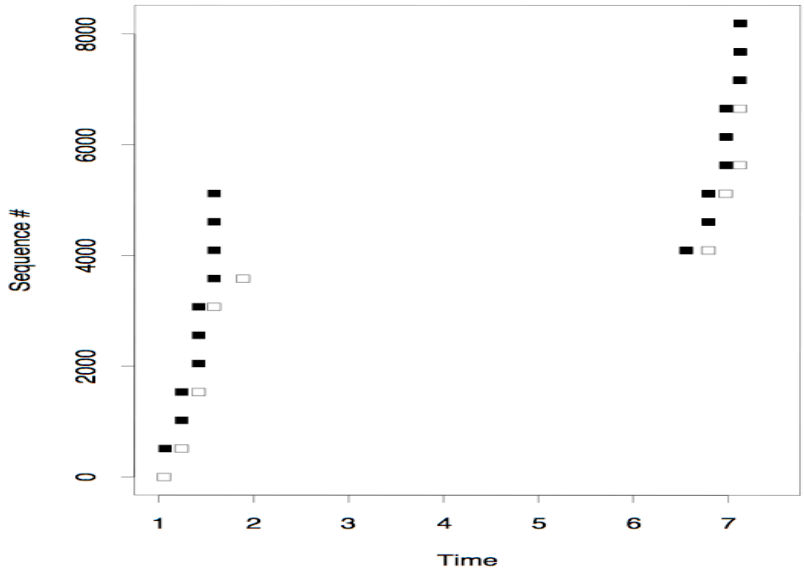


5 Minute Break

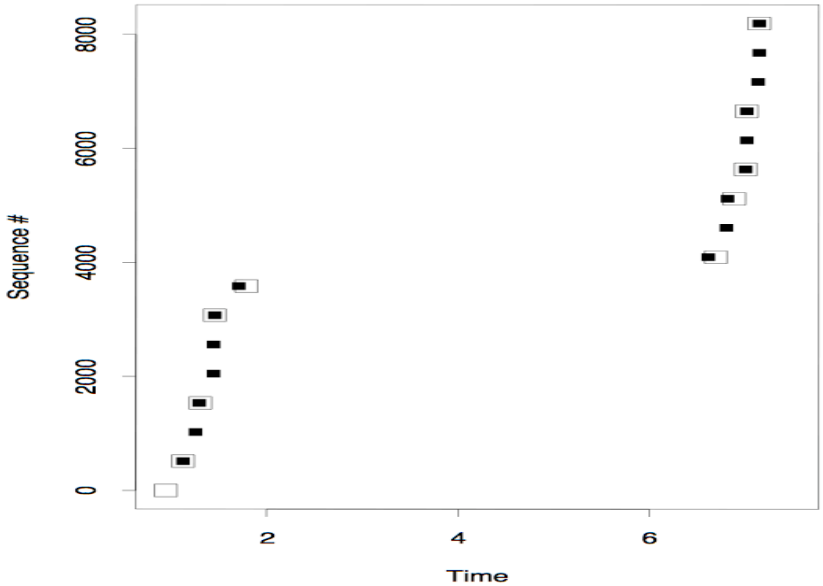
Questions Before We Proceed?

22

Detecting Loss: Timeout



Detecting Loss: Timeout - Recv. View



Timeout and Effect of *SSThresh*

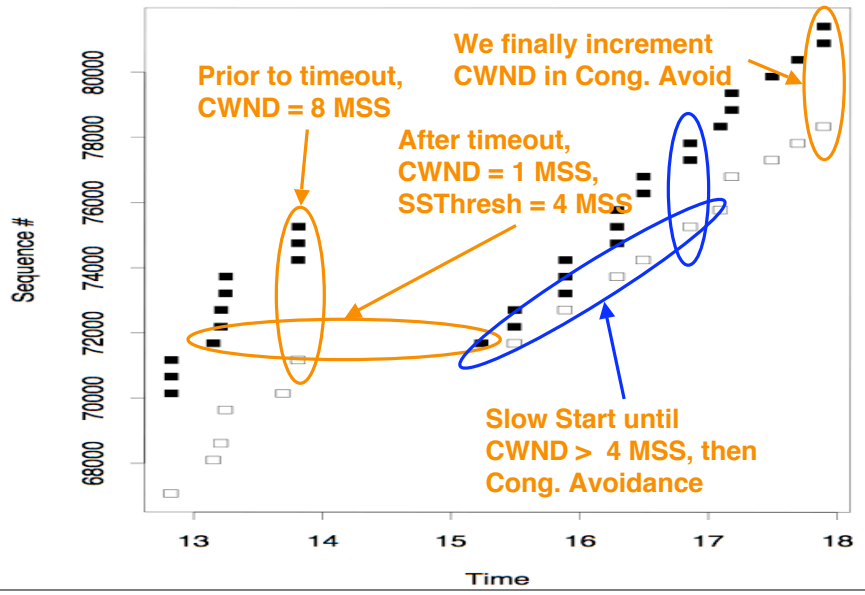
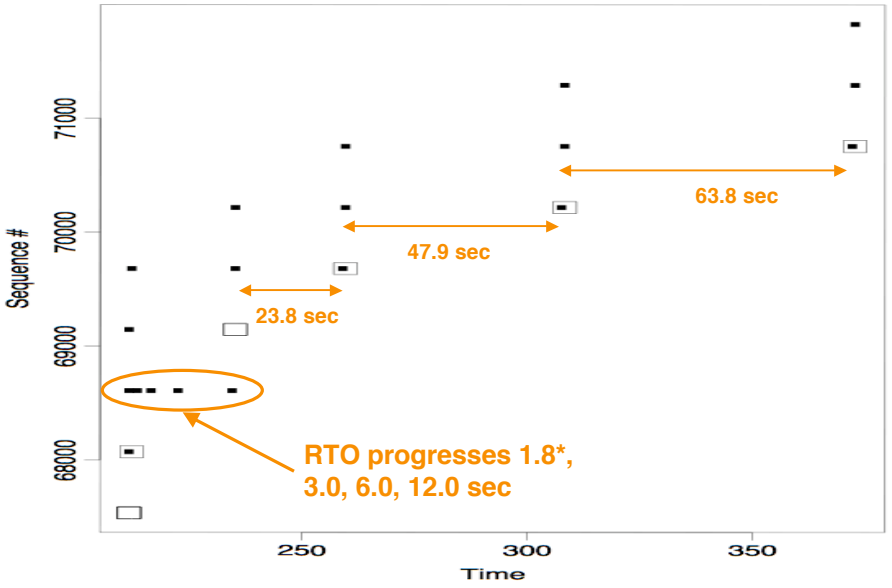
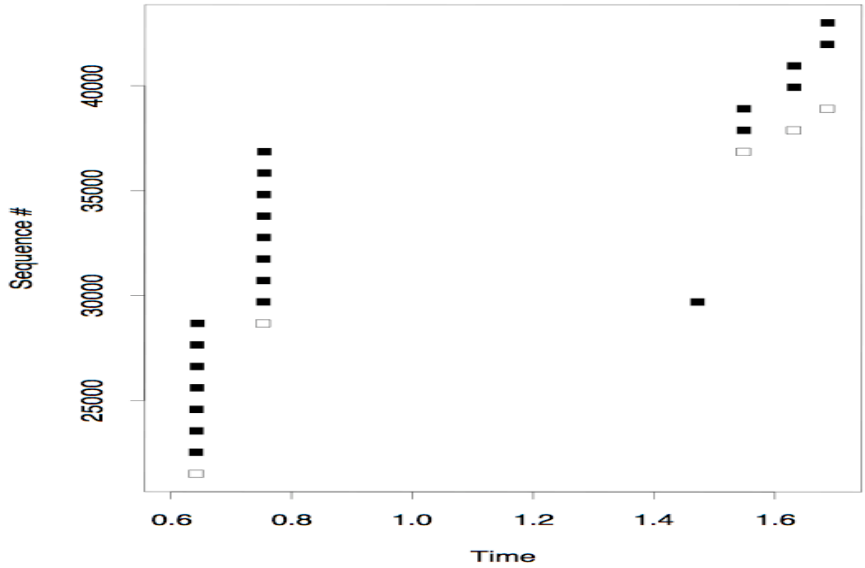


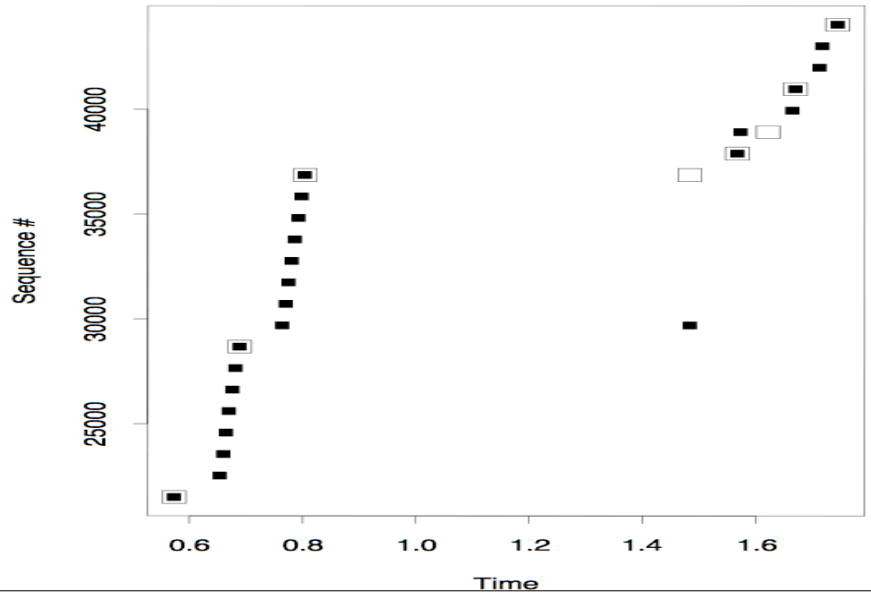
Illustration of Exponential Backoff



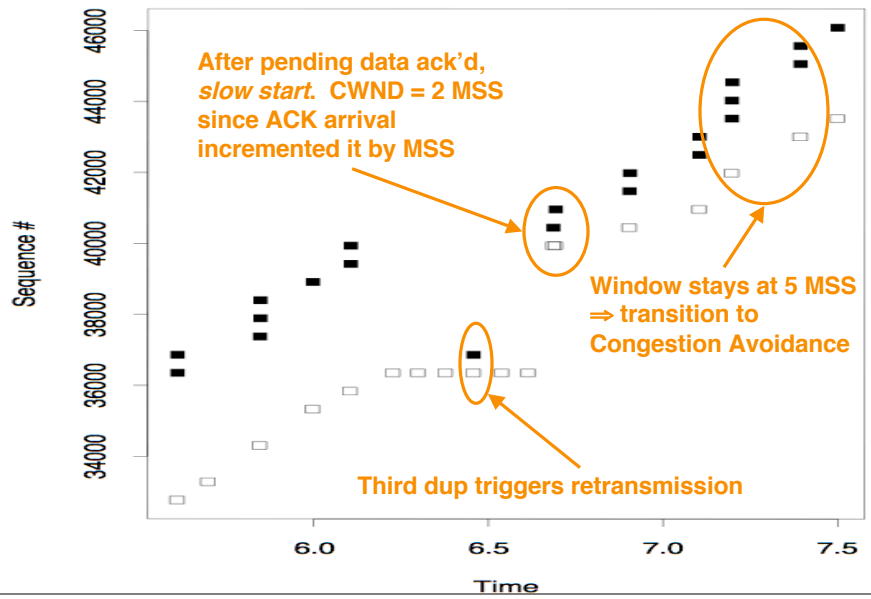
How Many Packets Were Lost?



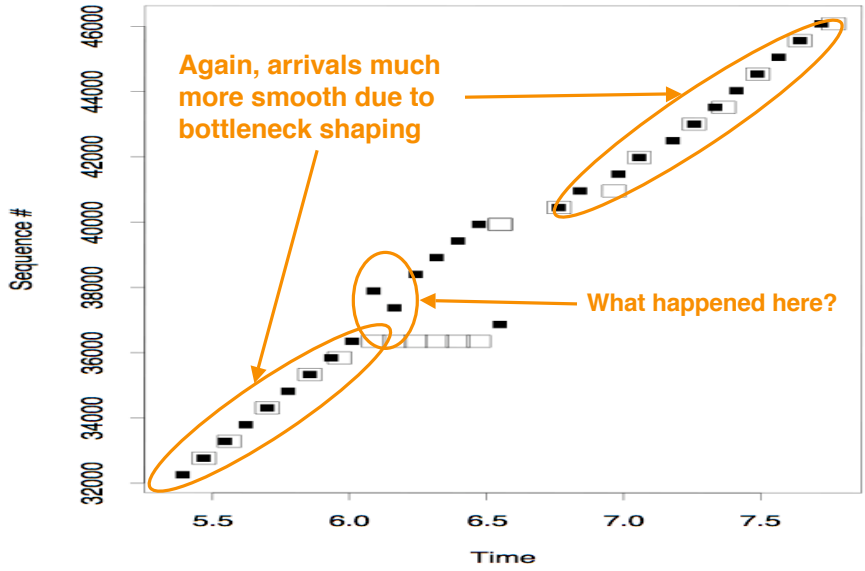
Answer: Zero!



Fast Retransmission



Same Fast Retransmission @ Recv.

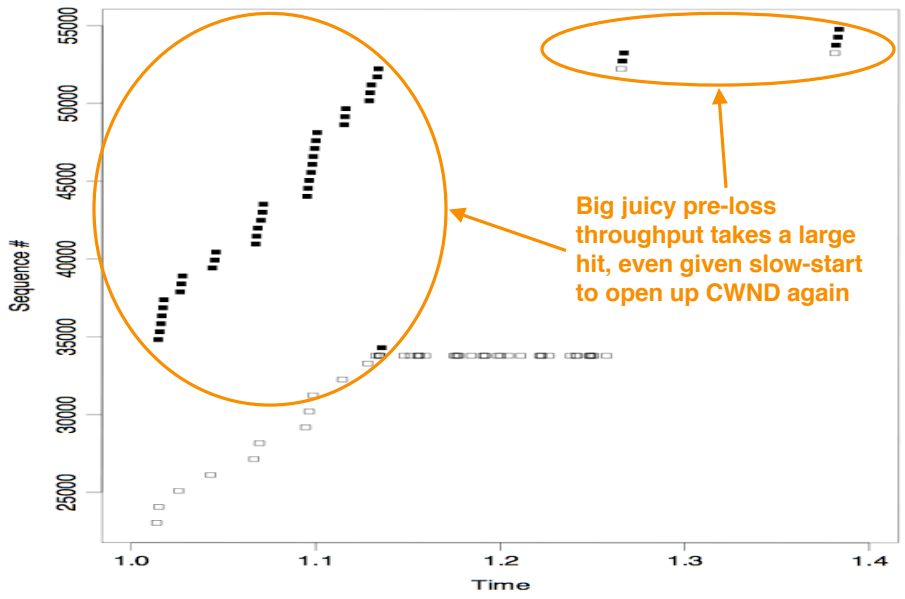


Effectiveness of Fast Retransmit

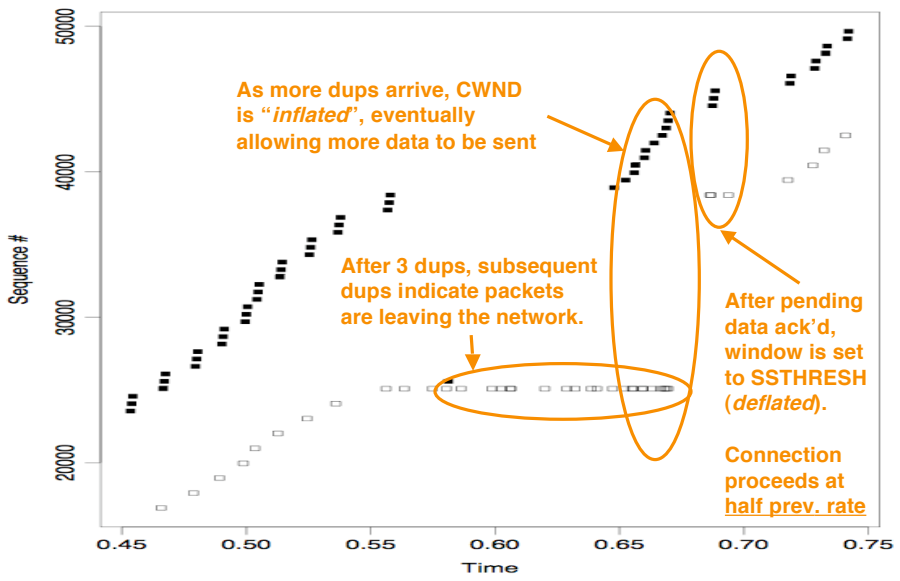
- When does Fast Retransmit work best?
  - Long data transfers
    - High likelihood of many packets in flight
  - High window size
    - High likelihood of many packets in flight
  - Low burstiness in packet losses
    - Higher likelihood that later packets arrive successfully
- Implications for Web traffic
  - Most Web transfers are short (e.g., 10 packets)
    - Short HTML files or small images
  - So, often there aren't many packets in flight
  - … making fast retransmit less likely to "kick in"
  - Forcing users to like "reload" more often…

31

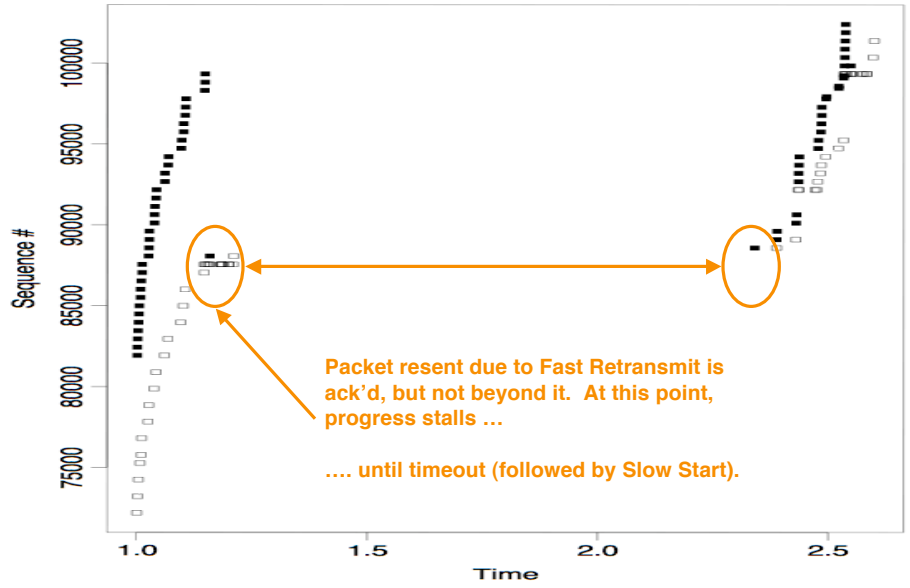
Fast Retransmit & Loss of Performance



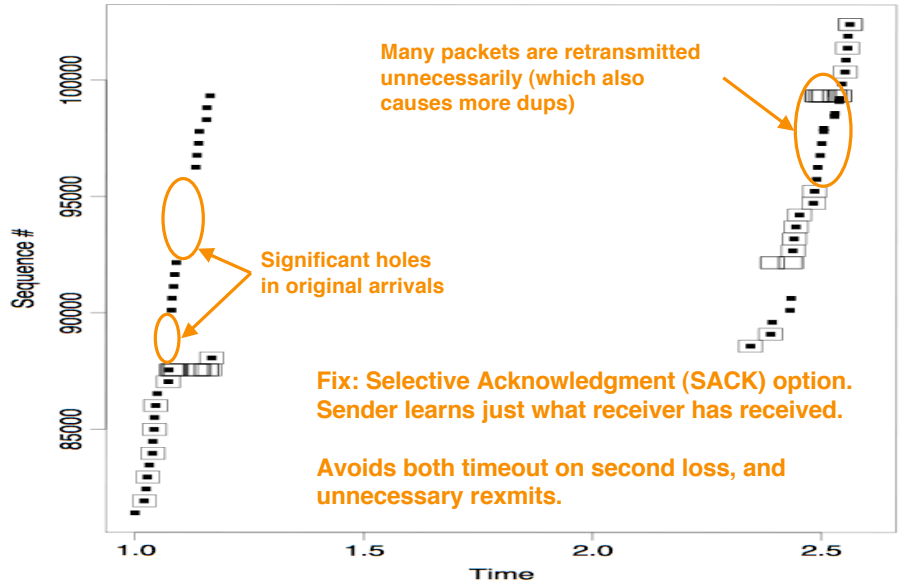
Fast Recovery Algorithm



Fast Rexmit/Recovery with > 1 Loss



Same From Receiver Perspective

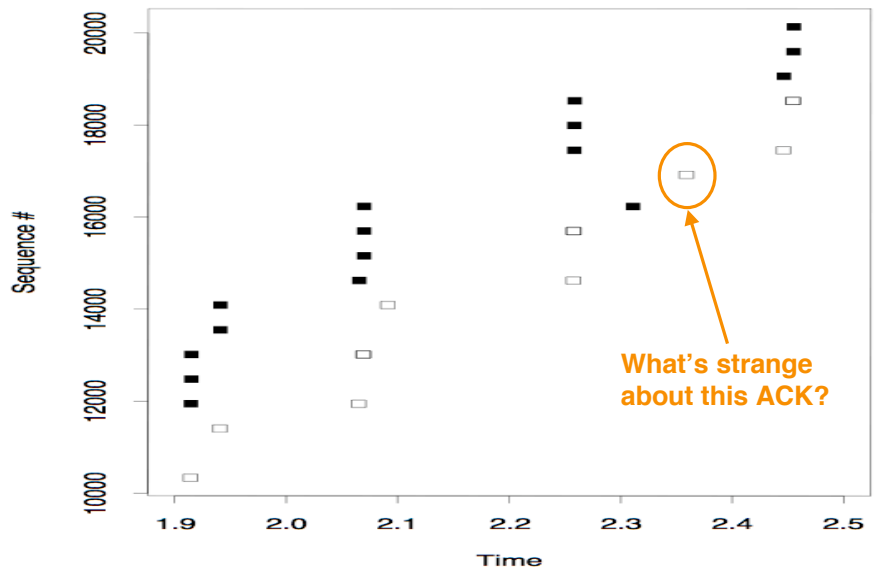


Summary of TCP Mechanisms

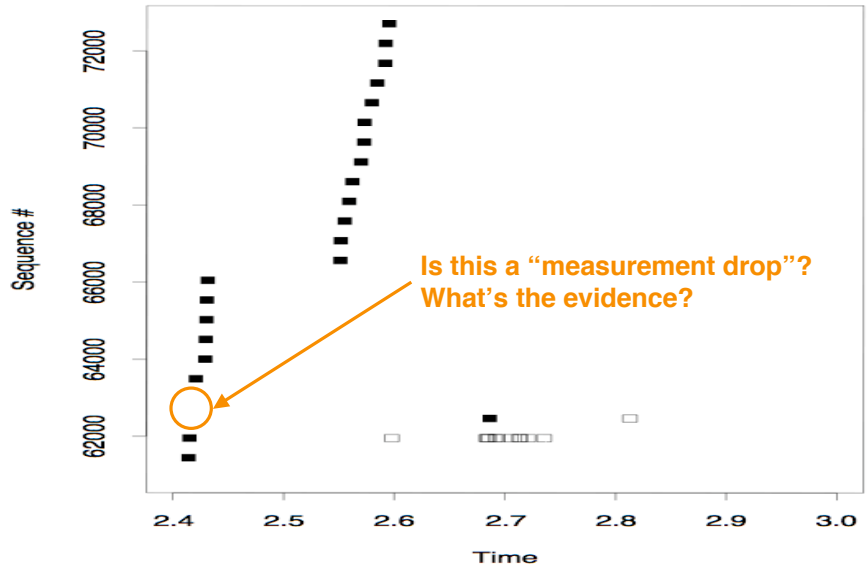
- Delayed Acknowledgment
  - Lessens overhead (40 bytes per ACK)
  - But can cause CWND to grow more slowly
- Fast Retransmit
  - NACK-based loss detection in 1 RTT
  - Avoids timeout delay
  - AIMD after subsequent Slow Start reaches SSTHRESH
- Fast Recovery
  - Avoids needing to Slow Start after Fast Retransmit
  - True AIMD
- SACK
  - Both speeds recovery and avoids unnecessary rexmit.

36

A Precautionary Tale



<u>Different</u> Problem. What's up now?



Going Fast

- Q: on a path with RTT = 100 msec, what's the absolute fastest rate that TCP can achieve?
- Q: what's the absolute largest sliding window that TCP can use?
  - A: advertised window is 16 bits ⇒ 65,535 bytes
  - Thus: max speed = 65,535 bytes / 100 msec = 655 KB/s
- Q: how can we fix this problem?
  - A: we need a larger window
- Q: how do we make the window larger?
  - A: using a TCP option

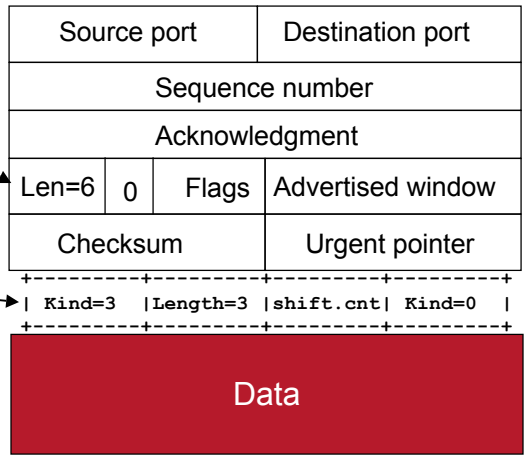
39

Window Scaling Option (RFC 1323)

HdrLen specifies 4 bytes of options.

Kind=3 indicates "Window Scaling".

Kind=0 indicates "end of options".



40

Window Scaling, con't

- Sent in initial SYN
- If server's SYN-ACK also includes a Window Scaling option, then scaling is in effect
  - The server including the option confirms its use
- **`shift.cnt`** specifies scaling factor for units used in window advertisement
  - E.g., **`shift.cnt`** = 5 ⇒ advertised window is 2^5 = 32-byte units

41

Window Scaling, con't

- Q: Now how large can the window be?
- A: Clearly, must not exceed 2^{32} …
  - If it does, then can't disambiguate data in flight
  - So, scaling ≤ 16
- In fact, somewhat subtle requirements limit window to 2^{30} to allow receiver to determine whether data fits in the offered window
  - So, scaling ≤ 14

42

Window Scaling, con't

- Now we can go fast. Suppose in a high-speed LAN with RTT = 1 msec we can transmit at 1 GB/s.
- What problem arises if packets are occasionally delayed in the network for 10 msec?
- Sequence number wrap: can't tell earlier, delayed segments from later instances.
- Fix: another TCP option to associate (high-res) timestamps with TCP segments
  - Essentially, adds more bits to sequence space
  - (Side effect: no more need for Karn/Partridge restriction not to compute RTT for ACKs of retransmitted packets)

43

Summary

- Time-Sequence plots provide a powerful tool for visualizing TCP behavior & performance
- Spectrum of TCP mechanisms influence performance
  - Advertised window, sender window
  - Timeout, slow start, exponential backoff
  - Acking policy (delayed; ack-splitting)
  - Fast Retransmit (avoid RTO stall)
  - Fast Recovery (full AIMD)
  - Window scaling (required for large bandwidth-delay product)
- Next lecture: a broader view of performance

44