CS118 Discussion 1B, Week 5

Zhehui Zhang

HAINES A2 / Friday / 12:00pm-1:50pm

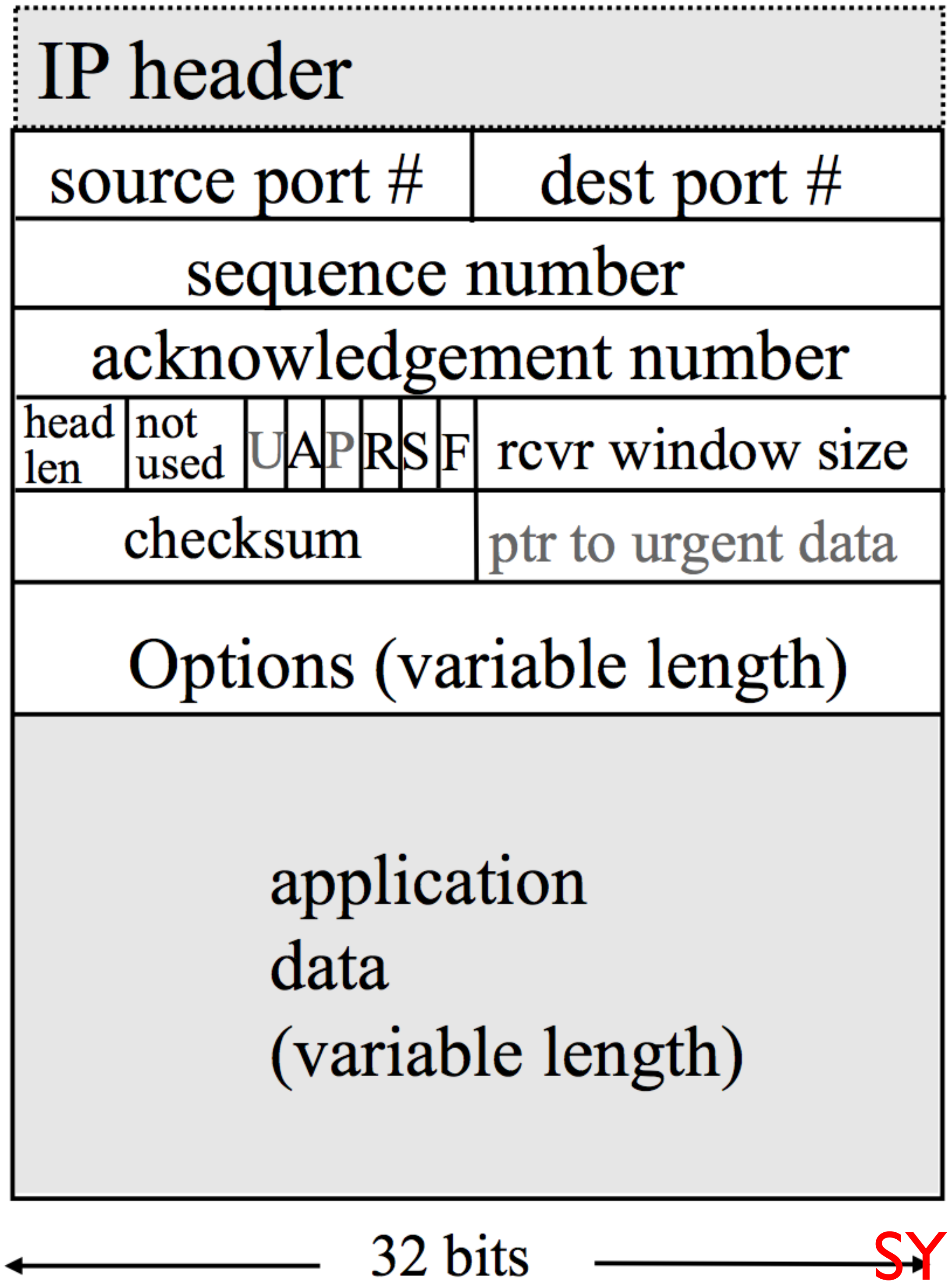
Outline

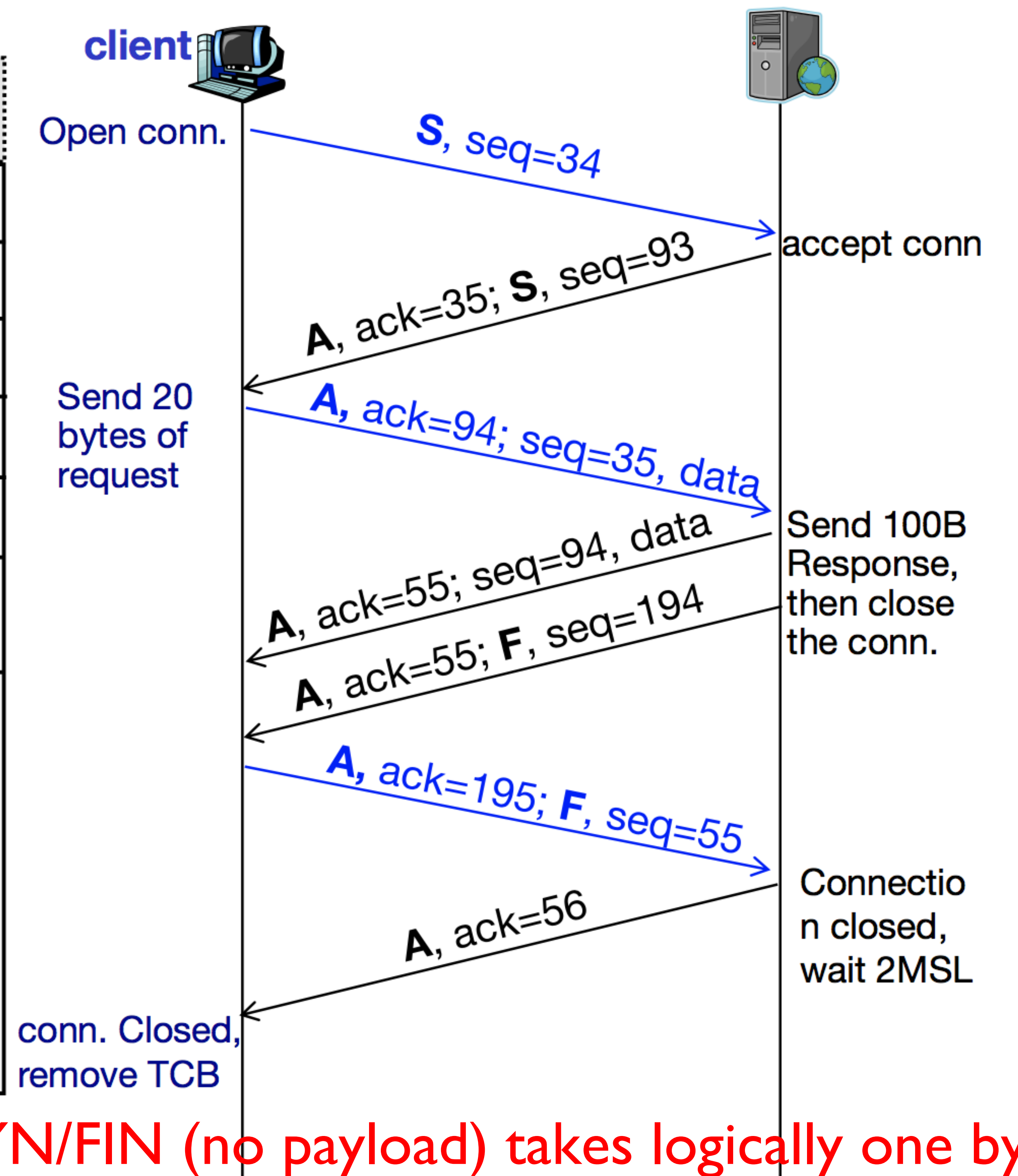
- Lecture review: Clarifications, TCP flow control; congestion control
- Midterm!
  - Email Professor Lu (slu@cs.ucla.edu) if you STRONGLY OPPOSE to deferring the midterm to **May 14th** or **May 16th**.
  - Respond before 6pm tomorrow afternoon (May 3, Friday).
- Project 2 clarification

Selective repeat/TCP clarification

Since TCP adopts the Selective Repeat, we will be using the selective repeat implementation by TCP throughout the course, that is:
(a) ACK and SEQ numbers are counted in *bytes*
(b) ACK numbers are sent in cumulative fashion
(c) ACK number = next expected byte number

An HTTP 1.0 connection example





SYN/FIN (no payload) takes logically one byte

TCP: flow control

- Limits the rate a sender transfers data
- Avoid exceeding the capacity of the receiver to process data
  - LastByteSent - LastByteAcked <= ReceiveWindowAdvertised
  - Receiver specify the receive window
  - The window size announce the number of bytes still free in the receiver buffer

TCP: congestion control

- Why Congestion Control
  - Oct. 1986, Internet had its first congestion collapse (LBL to UC Berkeley)
  - 400 yards, 3 hops, 32 kbps
  - throughput dropped by a factor of 1000 to 40 bps
- 1988, Van Jacobson proposed TCP congestion control
  - Window based with ACK mechanism
  - End-to-end

TCP: congestion control — window-based

- Limit number of packets in network to window size W
  - Source rate allowed (bps) = W×Message Size/RTT
  - Too small W?
  - Too large W?

TCP: congestion control — effects

- Packet loss
- Retransmission and reduced throughput
- Congestion may continue after the overload

TCP: congestion control — basics

- Goals: achieve high utilization without congestion or unfair sharing
- Receiver control (`rwnd`): set by receiver to avoid overloading receiver buffer
- Network control (`cwnd`): set by sender to avoid overloading network
  - `W = min(cwnd, rwnd)`
- Congestion window `cwnd` usually is the bottleneck

TCP: congestion control — main parts

- Slow start
- Congestion Avoidance
- Fast retransmit
- Fast recovery

https://tools.ietf.org/html/rfc5681

TCP: congestion control — slow start

- Start with cwnd = 1 (MSS: max. segment size; abstract as pkt)
- Exponential growth
  - each RTT:
    - `cwnd ← 2 × cwnd`
  - equivalently, each ACK:
    - `cwnd ← cwnd + 1 (MSS)`
- Enter Congestion Avoidance when **`cwnd > ssthresh`**

TCP: congestion control — congestion avoidance

- Start with `cwnd >= ssthresh`
- Linear growth
  - each RTT:
    - `cwnd ← cwnd + 1 (MSS)`
  - equivalently, each ACK:
    - `cwnd ← cwnd + 1/cwnd (MSS^2/cwnd)`

TCP: congestion control — packet loss

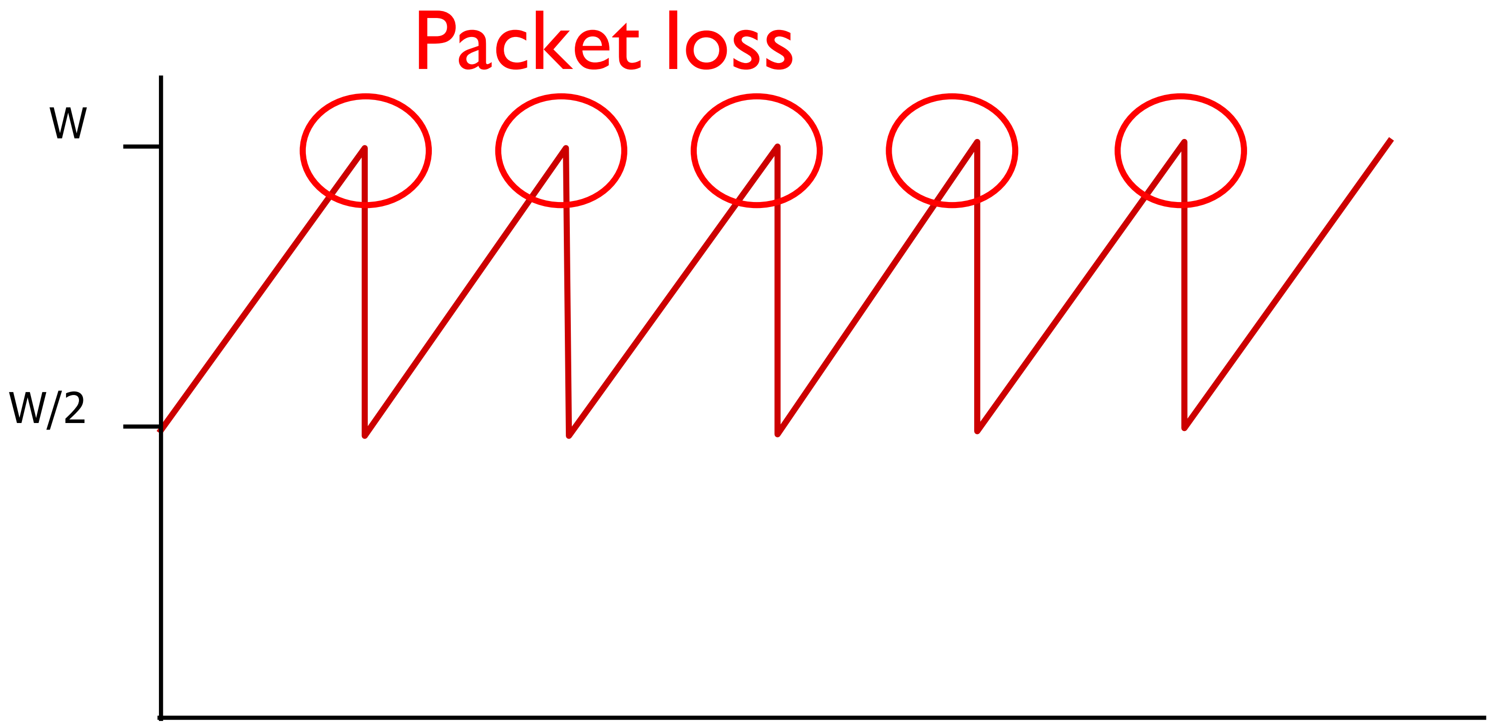
- Assumption: loss indicates congestion
- Packet loss detected by
  - Retransmission Timer Outs (RTO timer)
  - Duplicate ACKs (three)
    - ignore the 1st or 2nd duplicate ACK

TCP: congestion control — fast retx/recovery

- Upon 3rd duplicate ACK:
  - set `ssthresh ← cwnd/2`
  - set `cwnd ← ssthresh + 3 (MSS)`
  - upon additional dup ACK: grow `cwnd` linearly
  - New ACK: `cwnd ← ssthresh`
- Time Out
  - set `ssthresh ← cwnd/2`
  - set `cwnd ← 1 (MSS)`
  - enter slow start

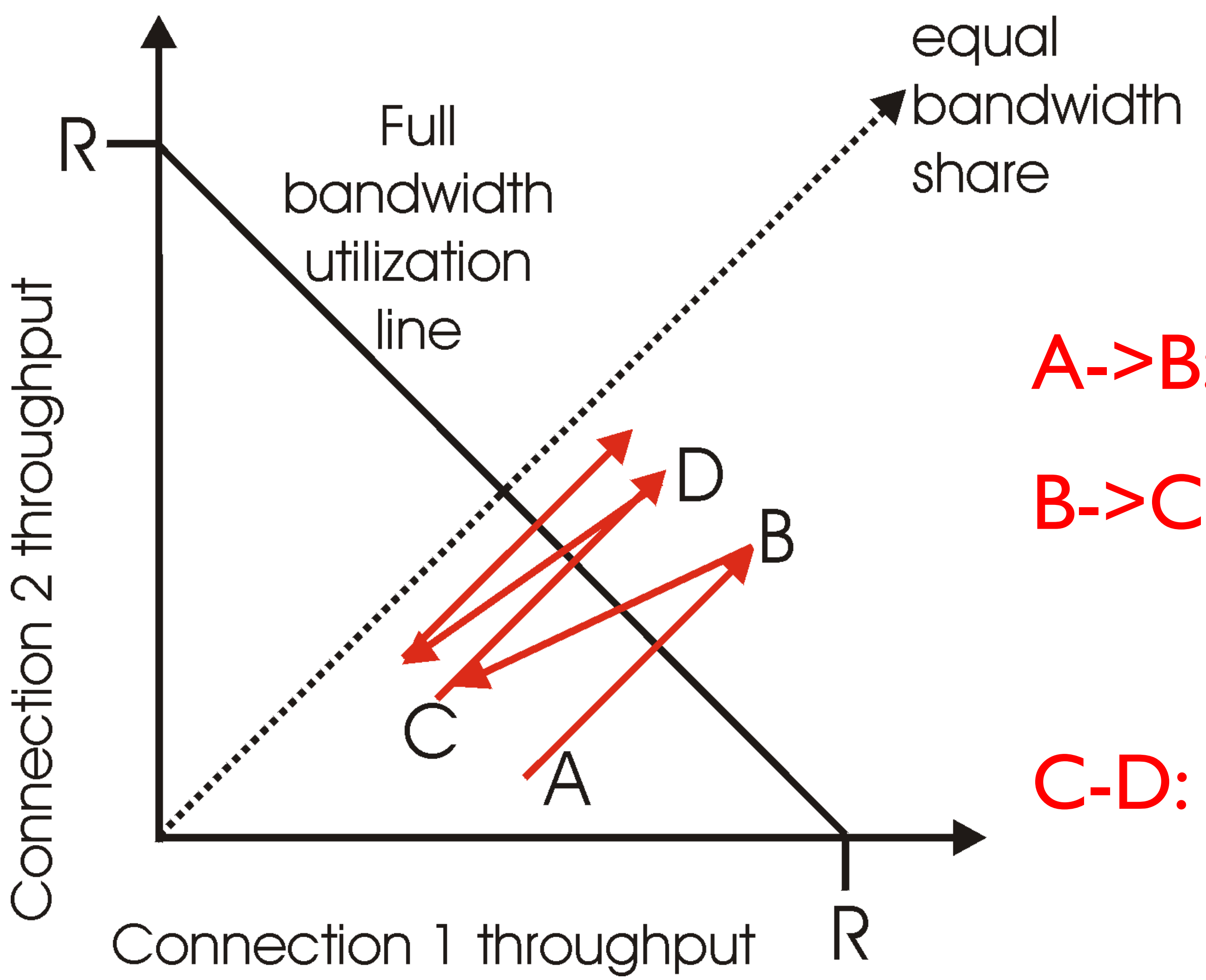
Macroscopic view of TCP throughput

- Additive Increase Multiple Decrease (AIMD)
- Saw tooth behavior: probing for bandwidth



AIMD for fairness

- AIMD-based congestion window adaptation
  - Fairness v.s. efficiency



A->B: Additive increase

B->C: Packet loss, shrink cwnd by half

(Multiple decrease)

C-D: Additive increase

TCP: congestion control — summary

- Slow start for fast convergence
- Congestion is indicated by packet loss
  - Timeout or duplicated loss
  - Congestion control is coupled with reliable transfer
- Fast retransmission/recovery based on duplicated ACK
  - `cwnd ← cwnd/2 + 3MSS`

```
The lost segment starting at SND.UNA MUST be retransmitted and
cwnd set to ssthresh plus 3*SMSS.  This artificially "inflates"
the congestion window by the number of segments (three) that have
left the network and which the receiver has buffered.
```

—RFC 5681

Let’s try it!

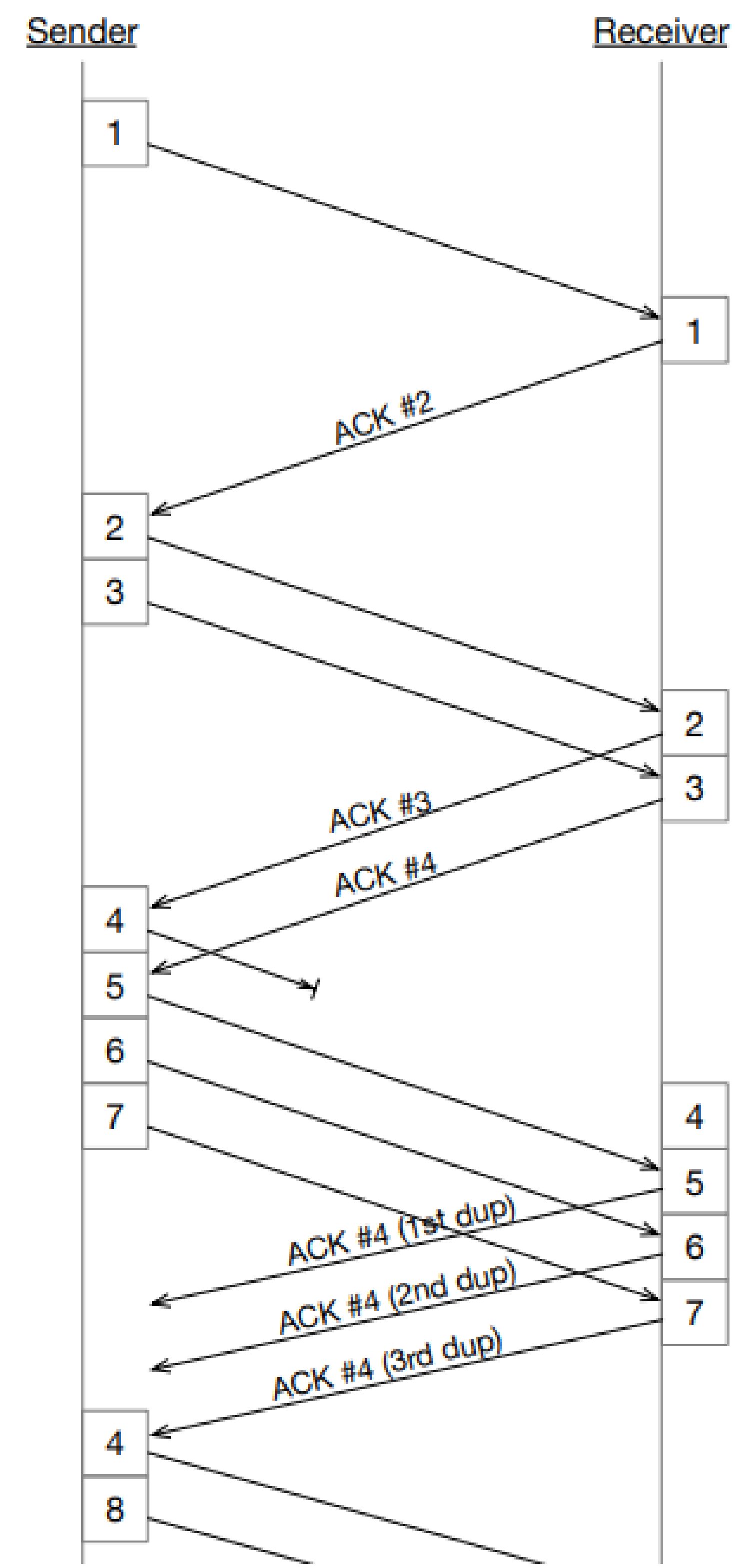
Consider the evolution of a TCP connection with the following characteristics. Assume that all the following algorithms are implemented in TCP congestion control: slow start, congestions avoidance, fast retransmit and fast recovery, and retransmission upon timeout.

- The receiver acknowledges every segment, and the sender always has data available for transmission.
- Initially `ssthresh` at the sender is set to 6. Assume `cwnd` and `ssthresh` are measured in segments, and the transmission time for each segment is negligible. Retransmission timeout (RTO) is initially set to 500ms at the sender and is **unchanged** during the connection lifetime. The RTT is 100ms for all transmissions.
- The connection starts to transmit data at time t = 0, and the initial sequence number starts from 1. **Segment with sequence number 4 is lost once**. No other segments are lost.

How long does it take, in milliseconds, for the sender to receive the ACK for the segment with the sequence number 12? show your intermediate steps or your diagram.

Solution

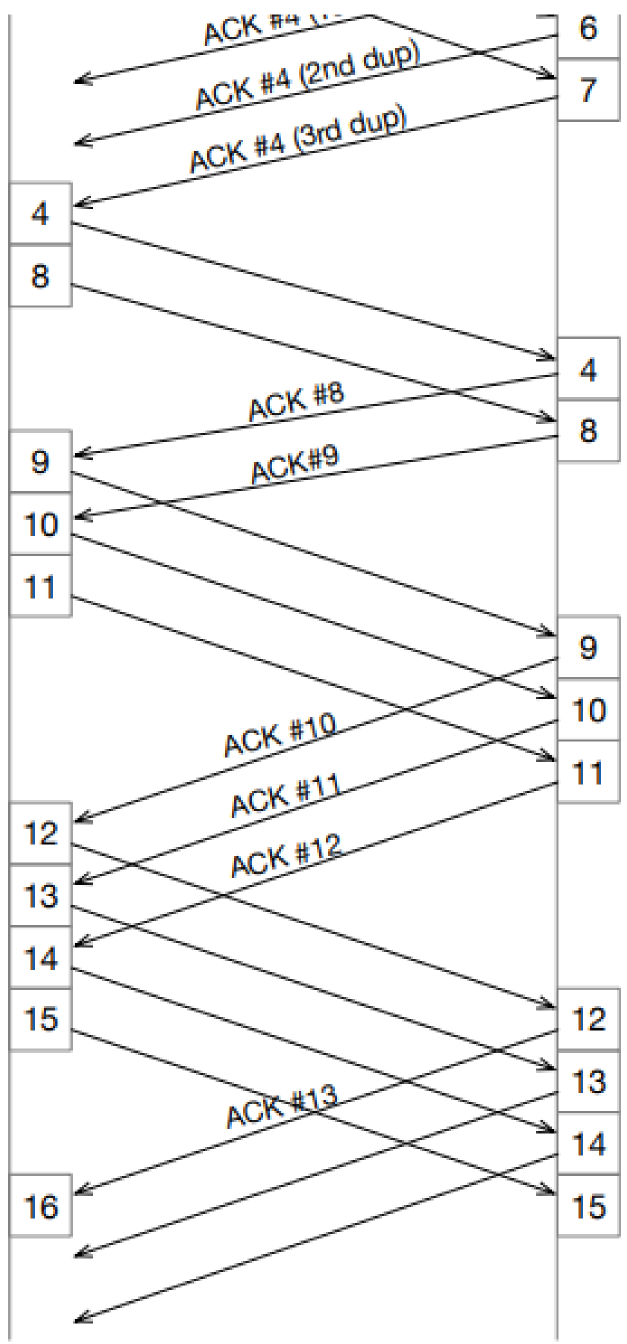
algorithm	ssthresh	cwnd	window
SS	6	1	1
SS	6	1+1=2	3, 2
SS	6	2+1=3	5,4,3
SS	6	3+1=4	7,6,5,4
no action	6	4	7,6,5,4
no action	6	4	7,6,5,4
FR	4/2=2	3+2=5	8,7,6,5,4



Solution

- If `ssthresh` equals to `cwnd`, use the congestion avoidance algorithm in your calculation.

no action	6	4	7,6,5,4
no action	6	4	7,6,5,4
FR	4/2=2	3+2=5	8,7,6,5,4
FR->CA	2	2+1/2=2	9,8
CA	2	2 1/2+1/2=3	11,10,9
CA	2	3+1/3=3	12,11,10
CA	2	3 1/3+1/3=3	13,12,11
CA	2	3 2/3+1/3=4	15,14,13,12
CA	2	4+1/4=4	16,15,14,13

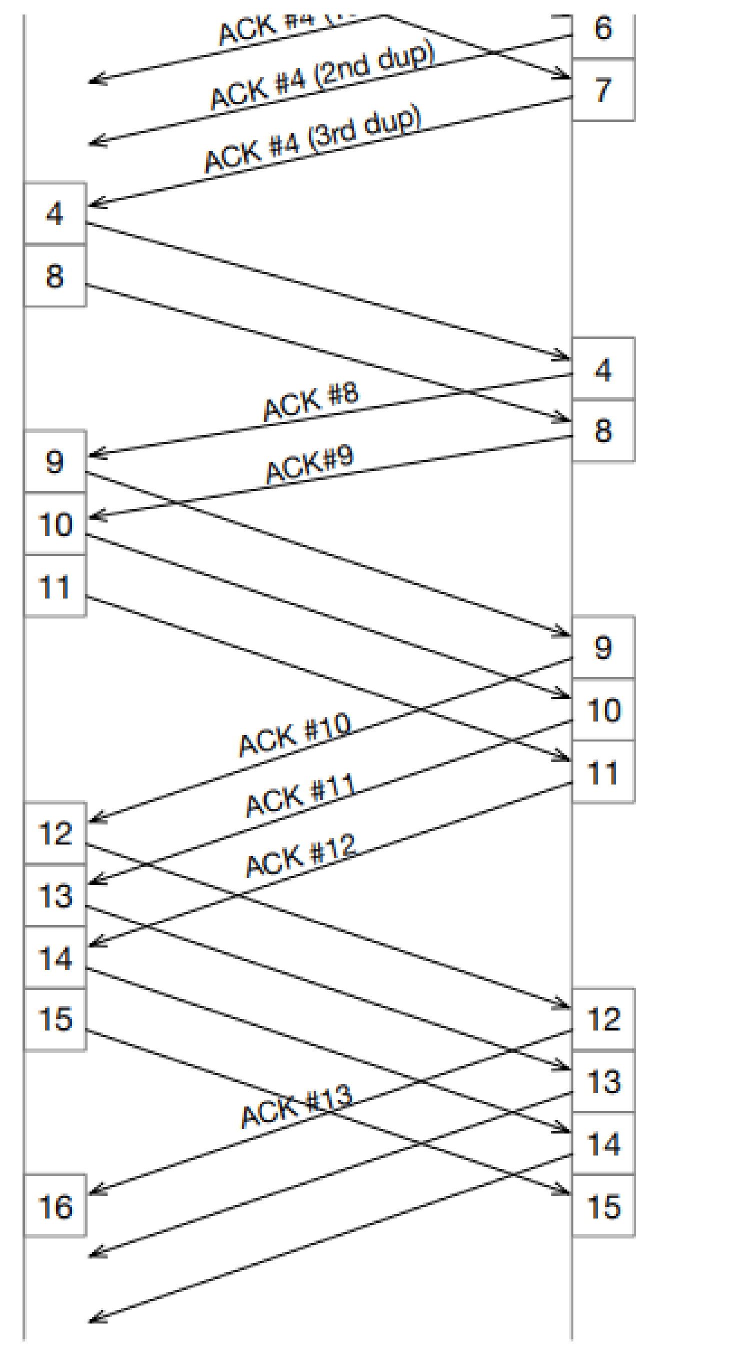


6 RTTs

Solution

- If `ssthresh` equals to `cwnd`, use the slow start algorithm in your calculation.

no action	6	4	7,6,5,4
no action	6	4	7,6,5,4
FR	4/2=2	3+2=5	8,7,6,5,4
FR->SS	2	2+1=3	10,9,8
CA	2	[3+1/3]=3	11,10,9
CA	2	[3 1/3 + 1/3] = 3	12,11,10
CA	2	[3 2/3 + 1/3] = 4	14,13,12,11
CA	2	[4 + 1/4] = 4	15,14,13,12
CA	2	[4 1/4 + 1/4] = 4	16,15,14,13



6 RTTs

Project 2

- The maximum UDP packet size is 524 bytes including a header (512 bytes for the payload)
- FIN should take logically 1 byte of the data stream
- Handling edge cases when receiving files
- The maximum congestion window size (cwnd) should be 10240
- CA: If cwnd ≥ ssthresh : cwnd += (512 * 512) / cwnd
- output MUST be written to standard output (std::cout) in the **EXACT** format defined.
- Due June 7th, 2019 at **23:59 PT**