# Reliable Delivery, TCP

CS168 - Spring 2024

## Agenda

- *(new)* Go-Back-N
- Review
- Worksheet

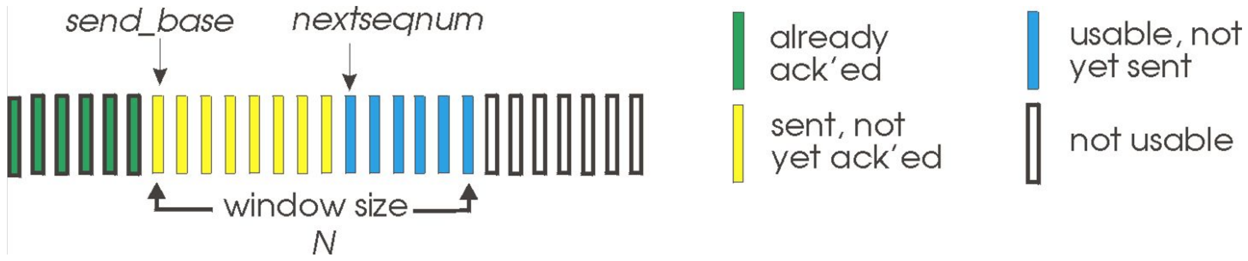
## Reliability

- Best-effort network
  - Need to handle packet loss, corruption, reordering, delays, duplications, etc.
- Building blocks
  - Checksums: detect corruption
  - Feedback: positive/negative feedback from receiver
  - Retransmissions: sender re-sends packets
  - Timeouts: when to resend a packet
  - Sequence numbers: indicate which packets have been received
- Design considerations
  - Window size, nature of feedback, detection of loss, response to loss

# Go-Back-N

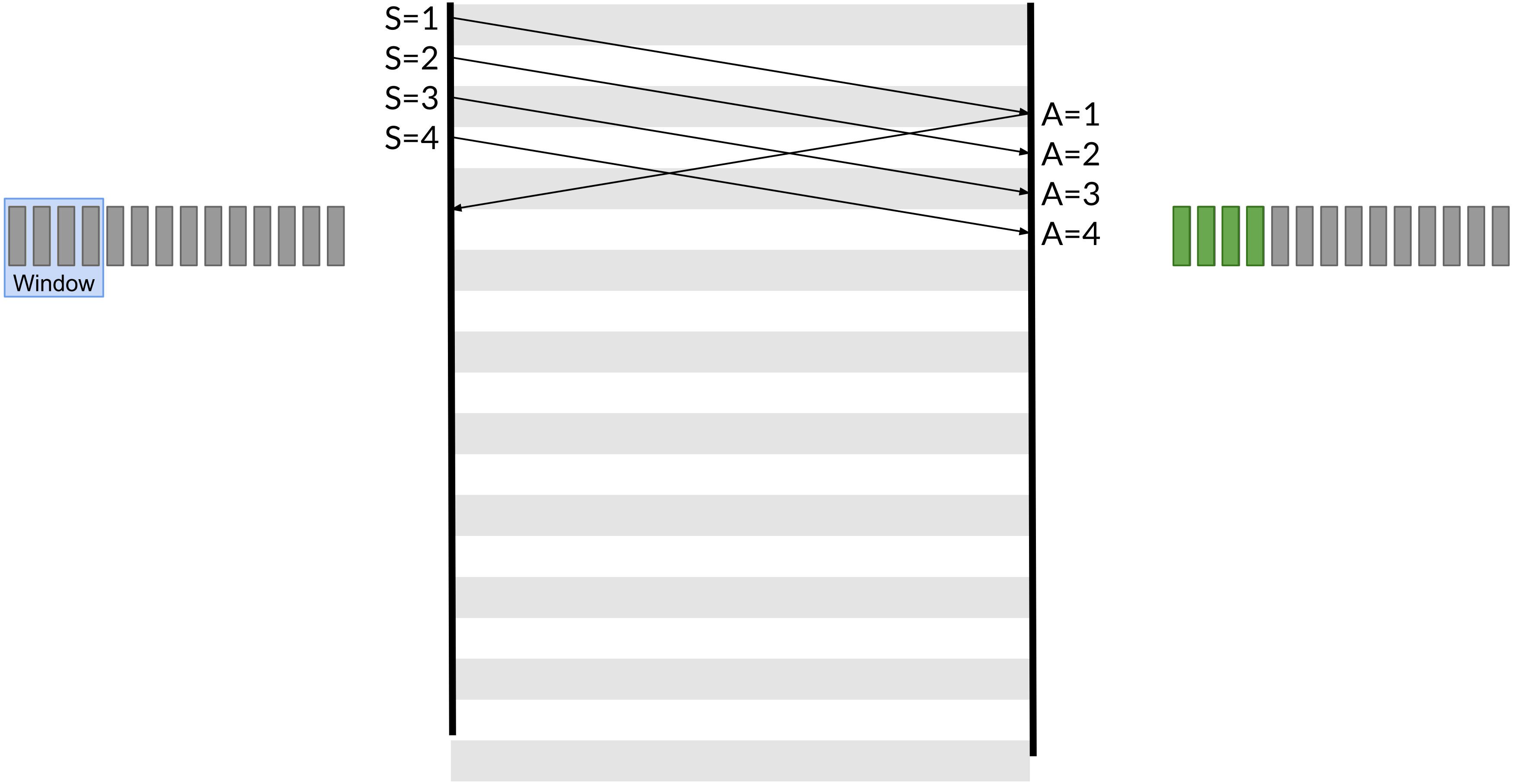
Not what TCP does, but useful as a starting point to figure out something better.

- Simple (though not advisable algorithm)
- “Sliding window” protocol: sender keeps a window of **up to W transmitted but unACKed** packets

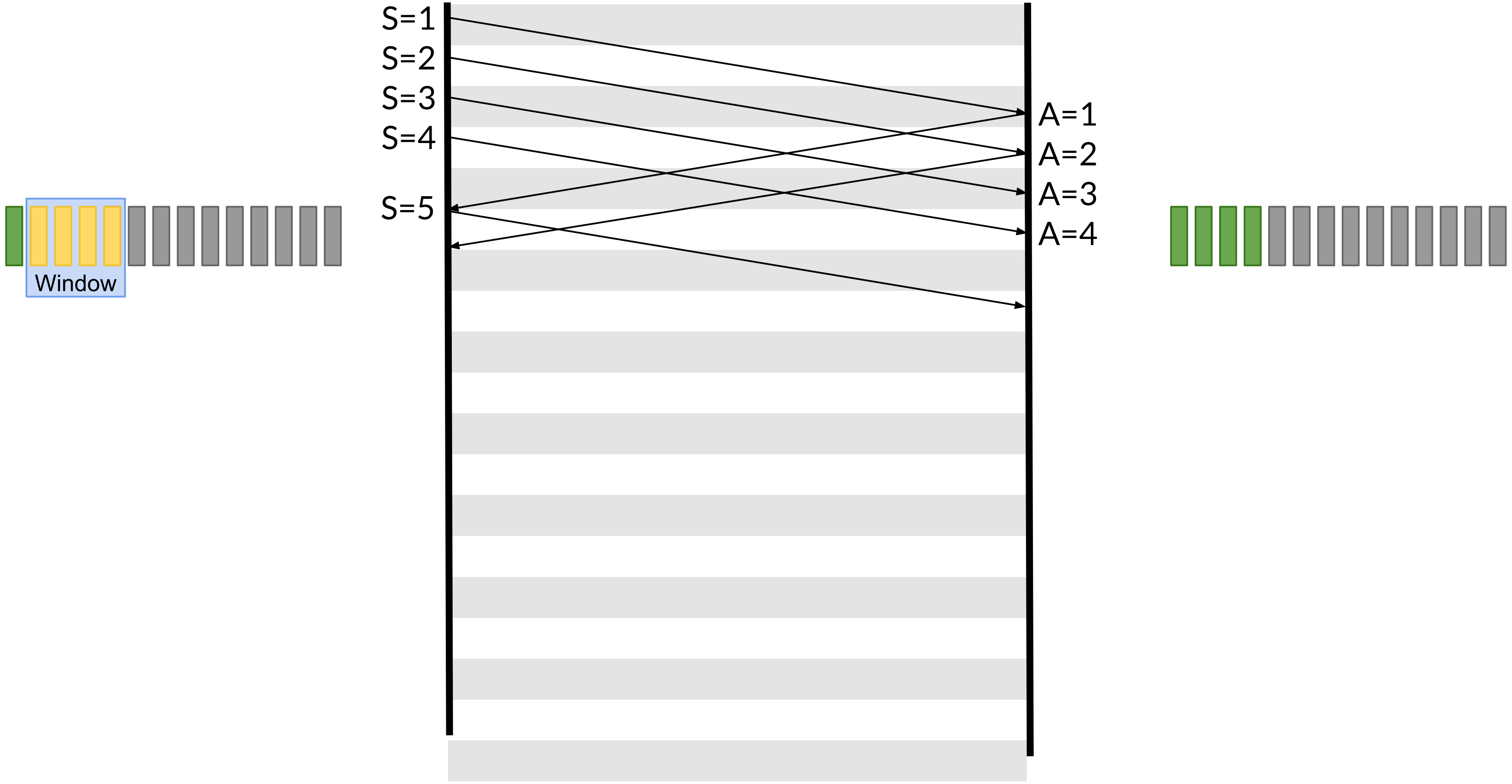


- Timer for oldest in-flight packet
- On timeout, resend all W packets (starting with the lost one)
- Receiver discards out-of-order packets

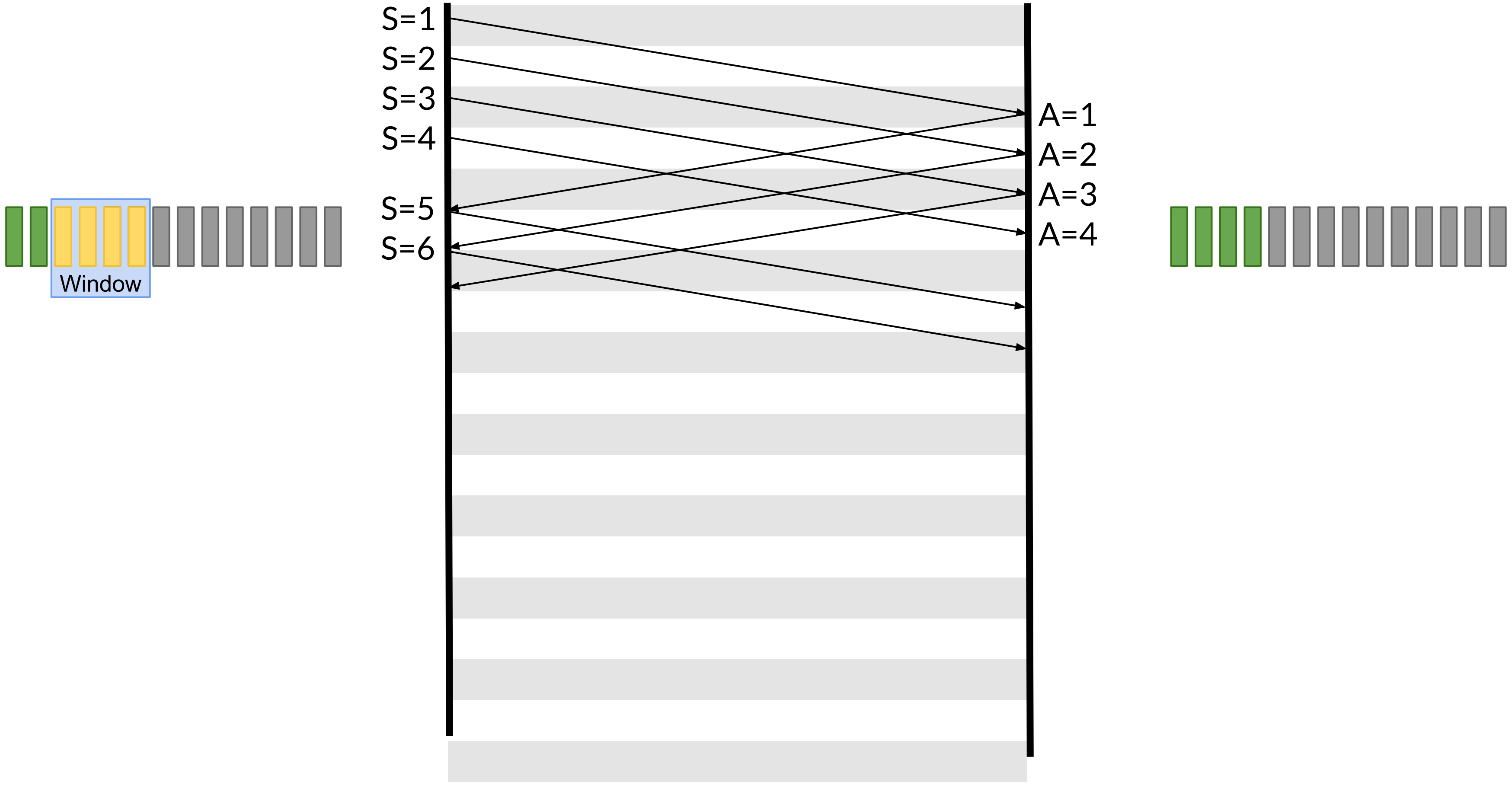
# Go-Back-N w/o Errors



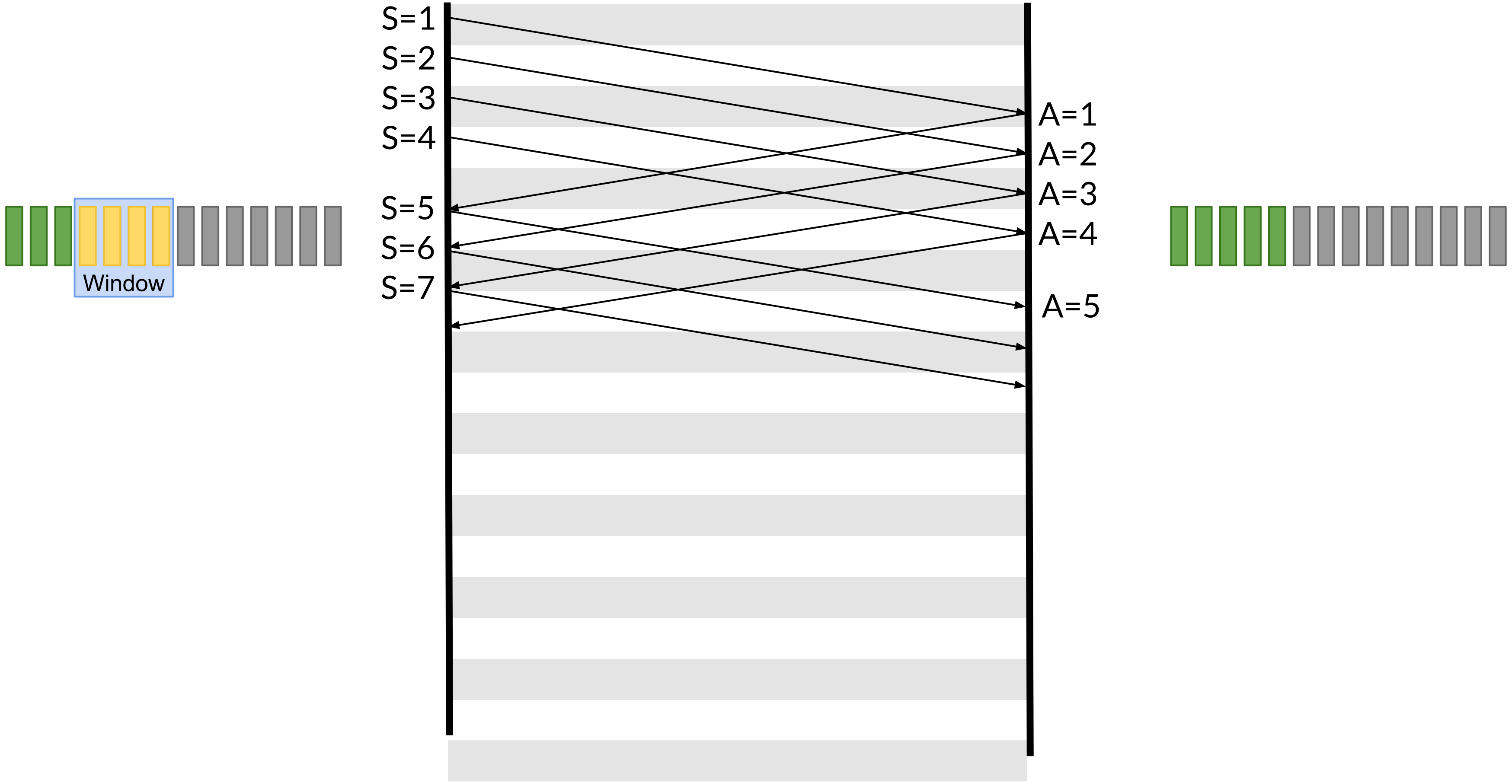
# Go-Back-N w/o Errors



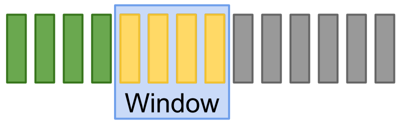
# Go-Back-N w/o Errors

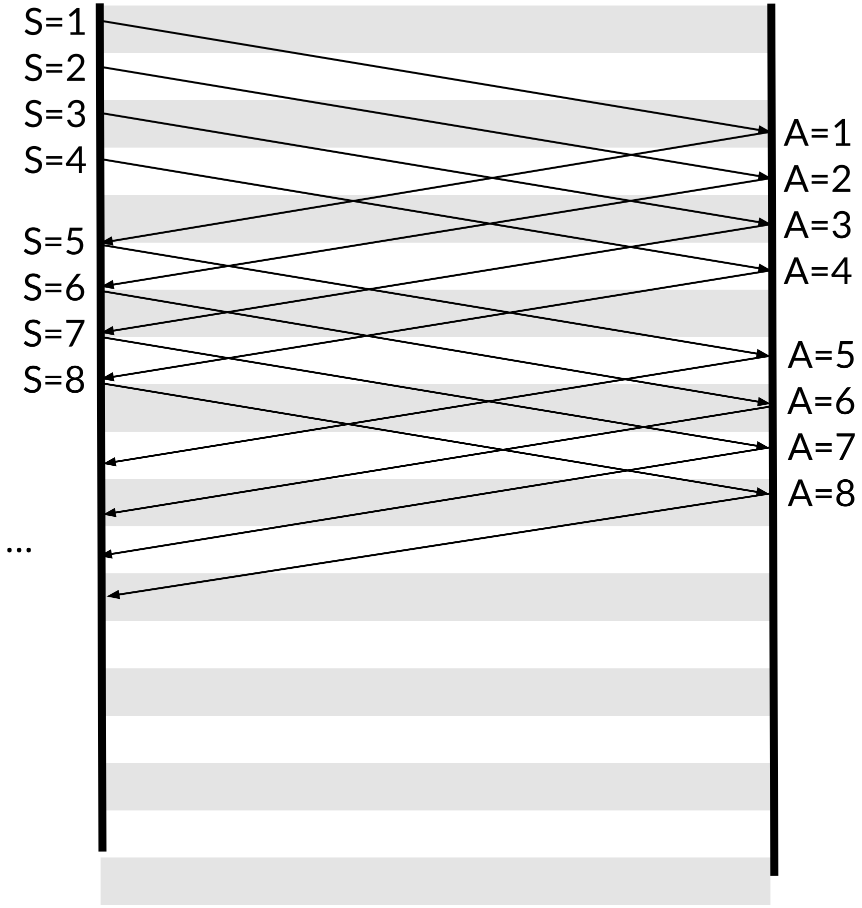


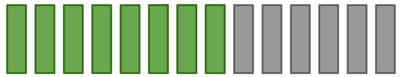
# Go-Back-N w/o Errors



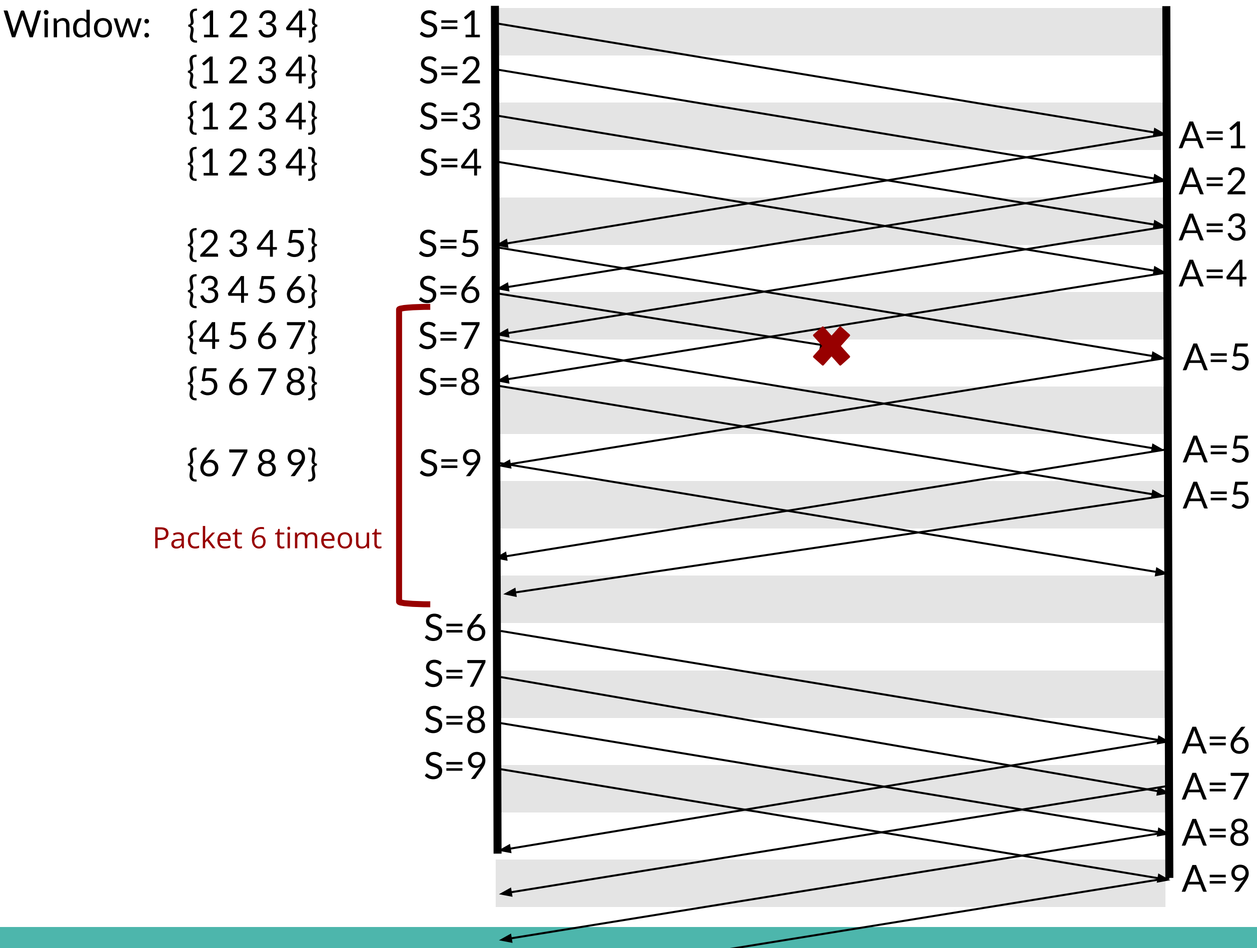
# Go-Back-N w/o Errors







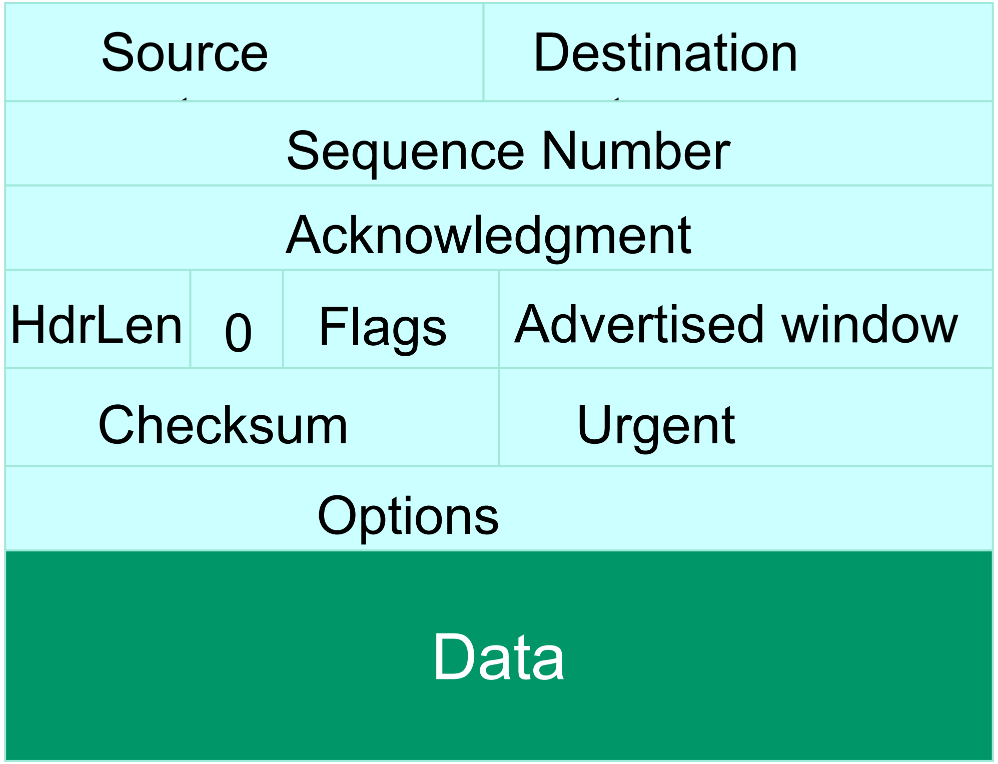
# Go-Back-N w/ Errors



## Review

# TCP

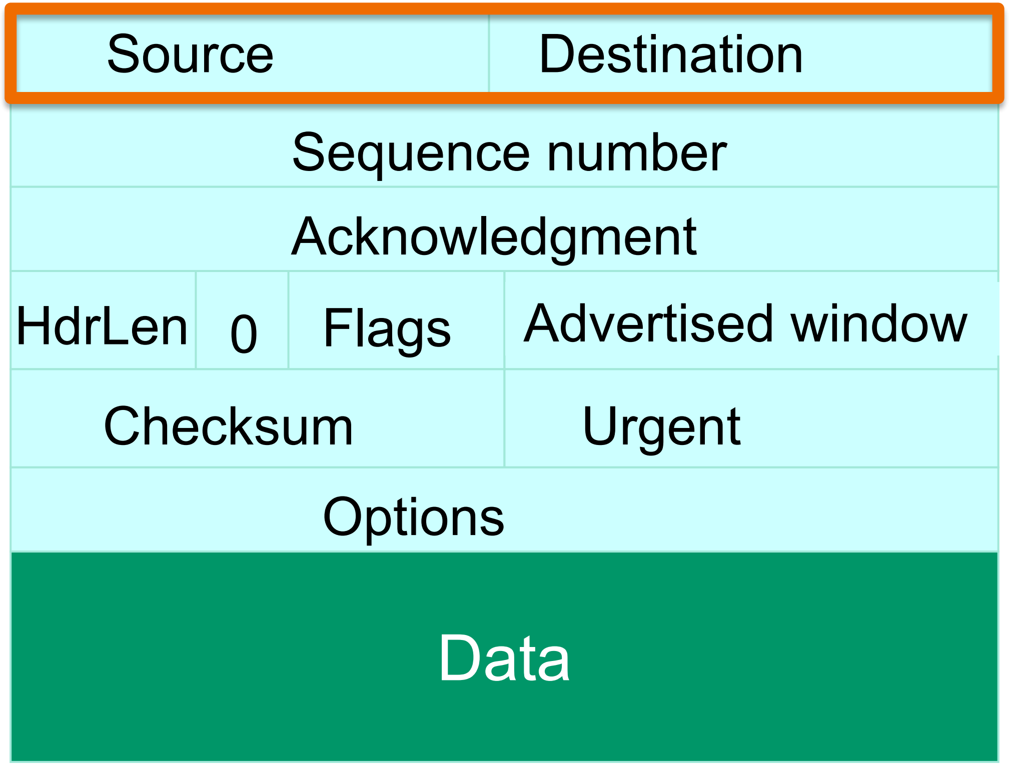
- L4 Protocol (Transport)
- Byte Stream
- Bi-directional
- Reliable
- In-order



# TCP Header

- Host port numbers
  - Multiplexing and demultiplexing
  - 16 bits

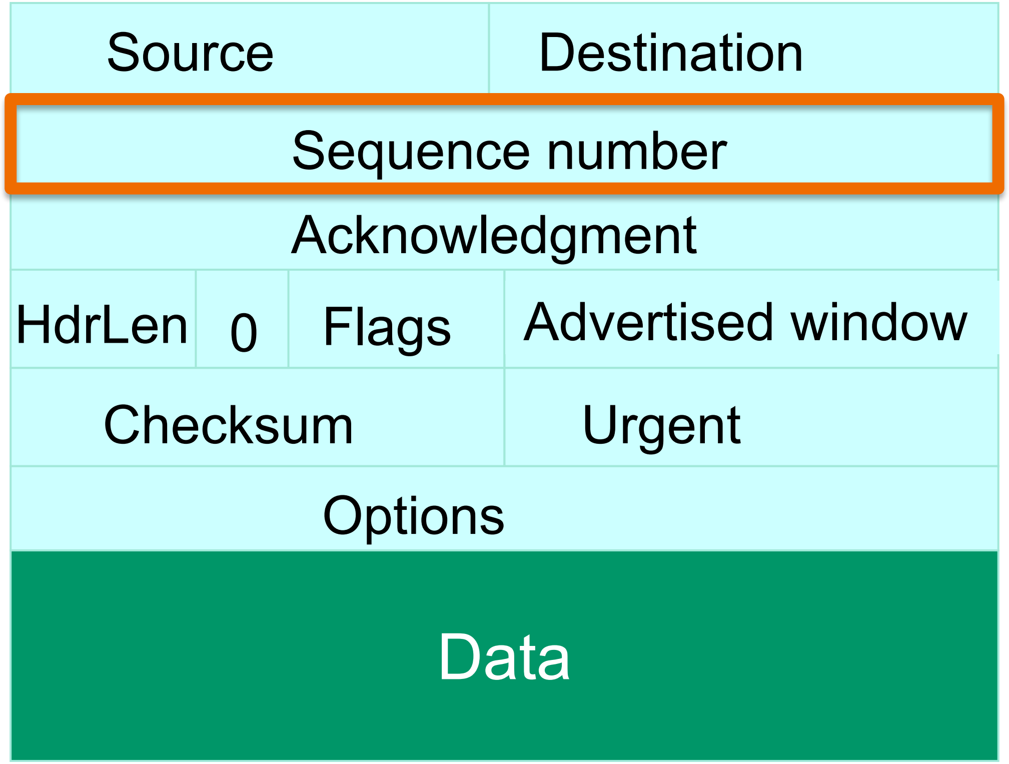
But wait - why no addr?!
- IP header has addr.



# TCP Header

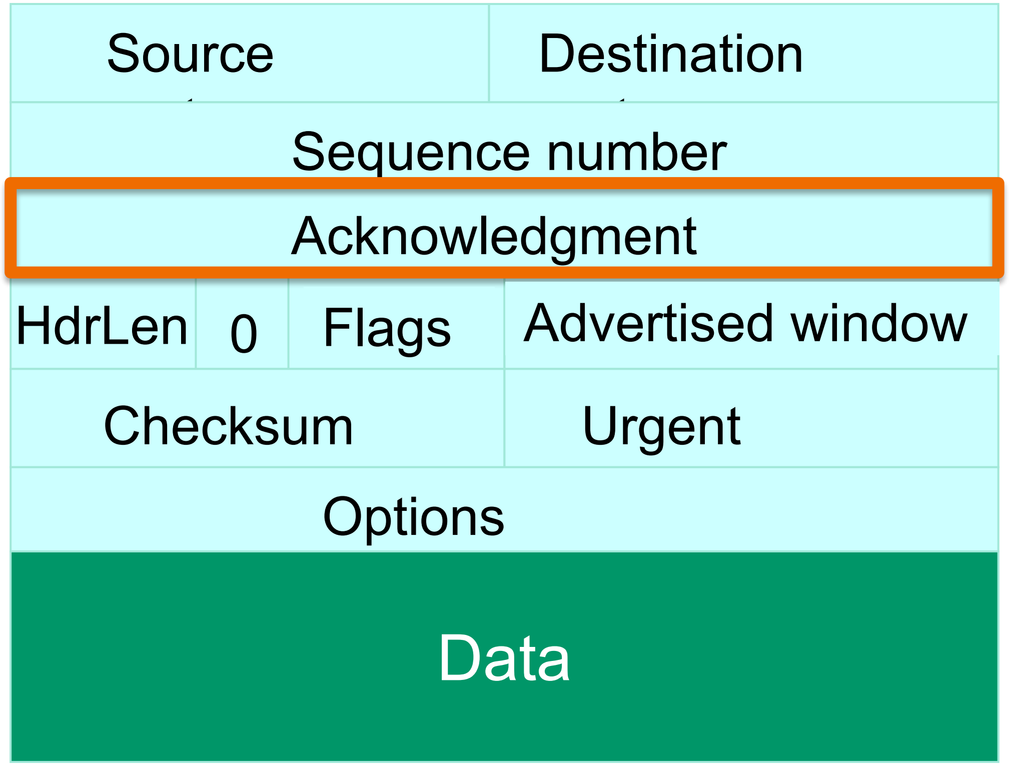
- Byte offset
  - Of first payload byte
  - Initialized randomly

- **Byte Stream Protocol**
  - Seq # refers to **bytes**



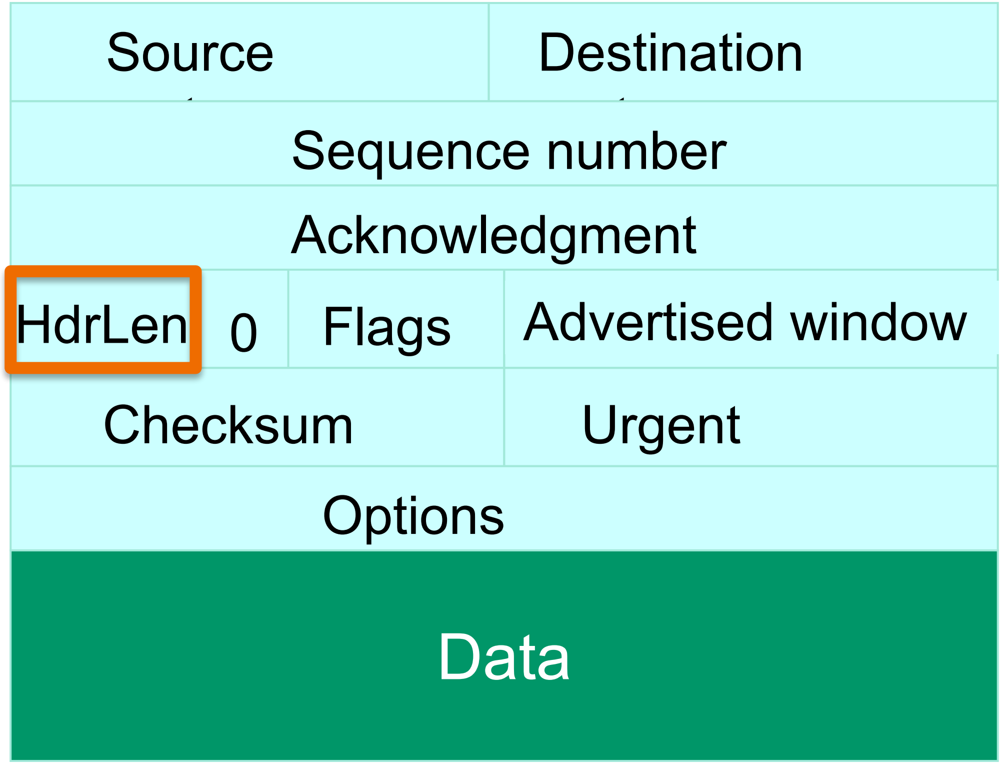
# TCP Header

- Cumulative ACKs
- Seq # of **next byte**
- Data packets carry ACKs



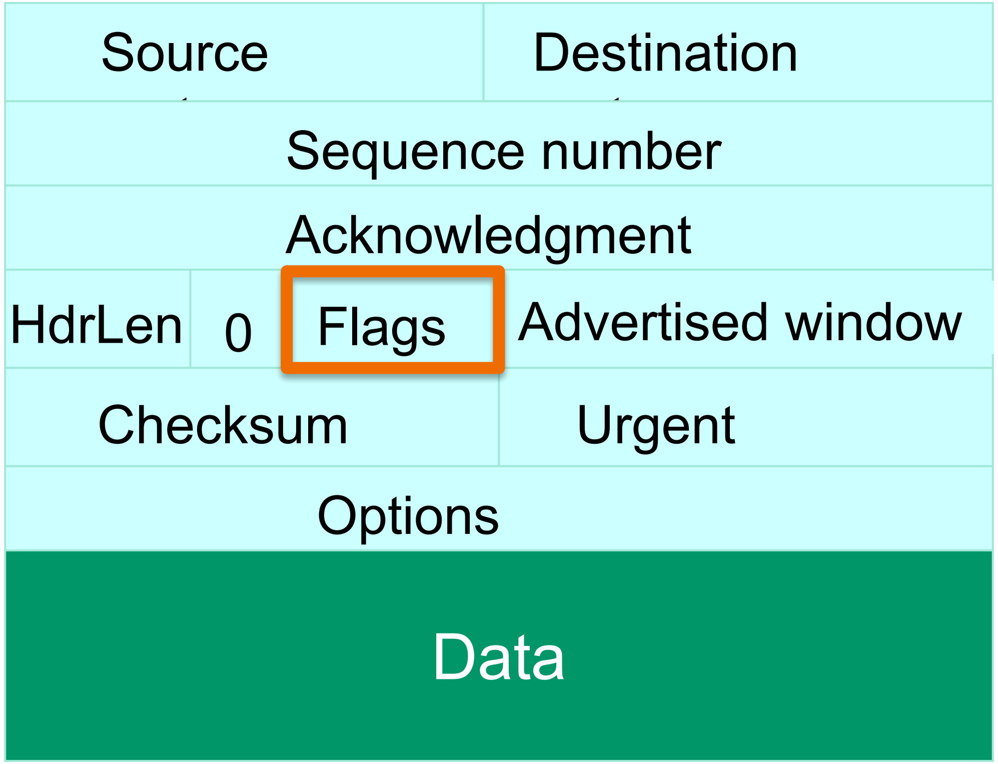
# TCP Header

- Header length
  - 4 bits
  - In 4-byte words
- Minimum 5 words (20B)
- Maximum 15 words
  - Why?



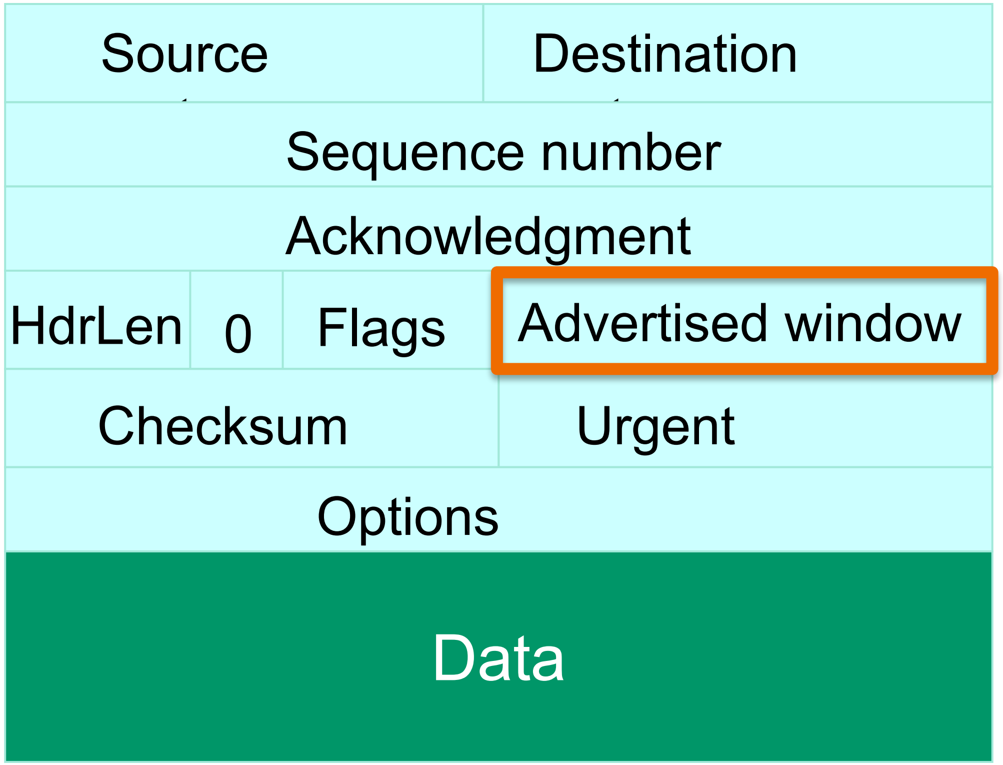
# TCP Header

- SYN
  - SYNchronize initial state
- ACK
  - ACKnowledgement
- FIN
  - No more data
- RST
  - Connection ReSeT



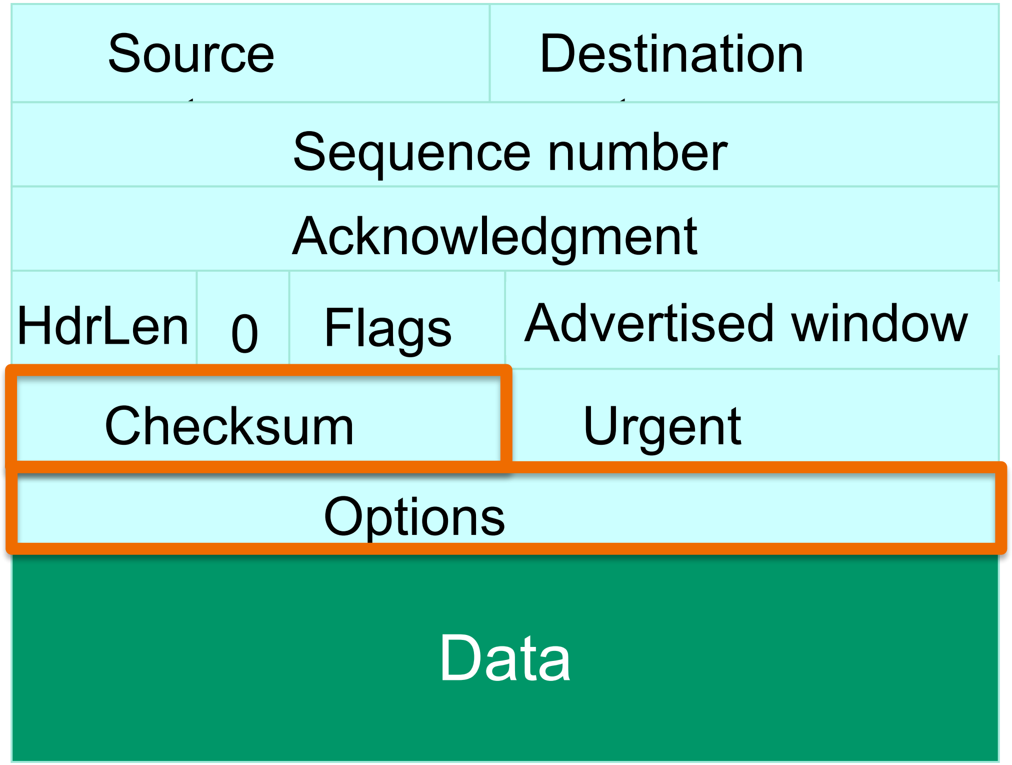
# TCP Header

- Receive Window Size
  - Maximum receiver buffer
- Limits sending rate
  - Don’t send faster than receiver can process



# TCP Header

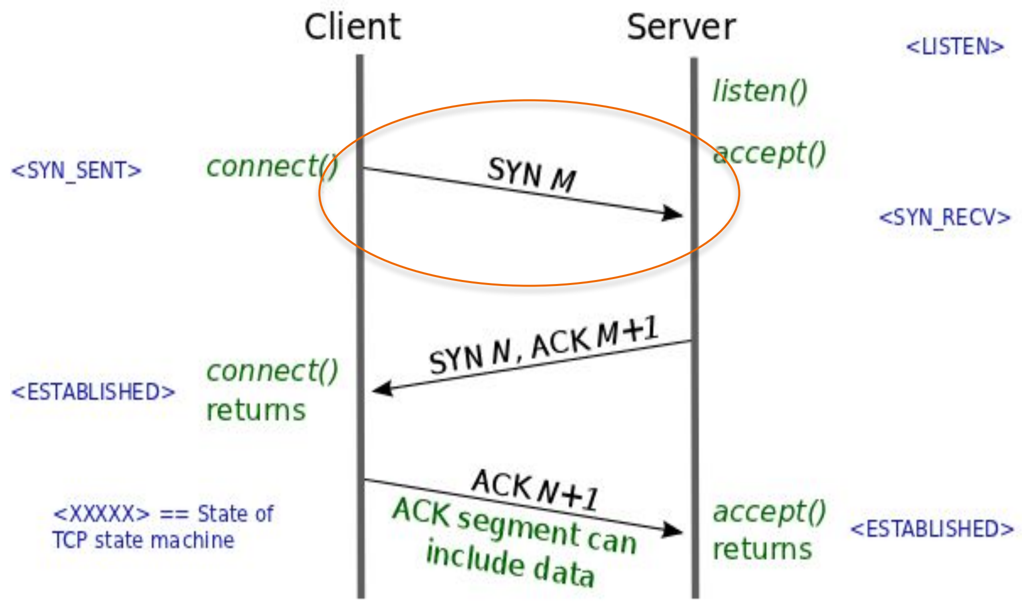
- Checksum
  - Includes header and payload
- Options common
  - Unlike IP
  - Not covered
  - Assume no options unless specified



## Connection Establishment

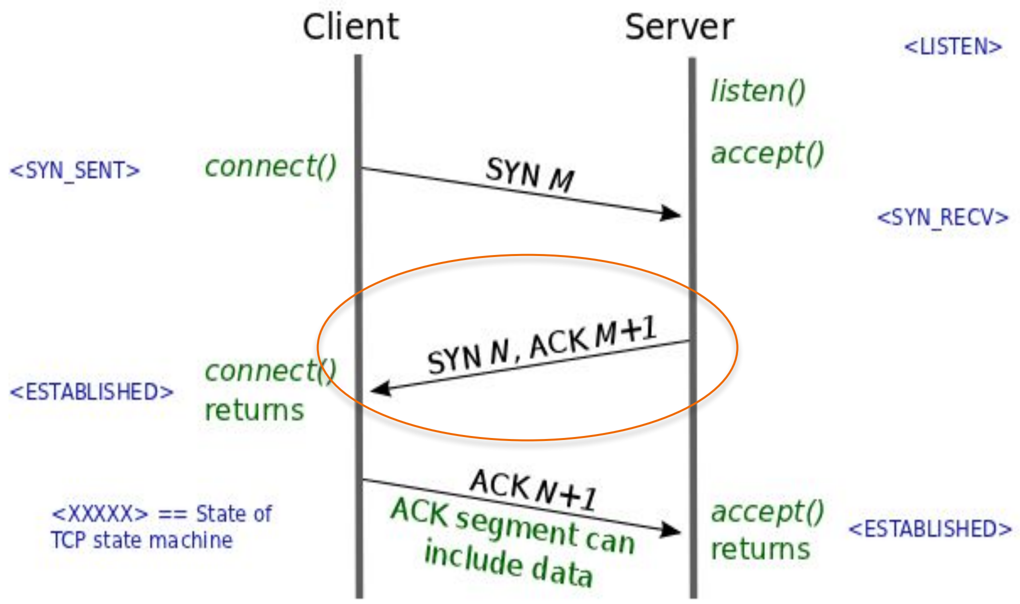
# Three Way Handshake

- Client sends server a SYN
  - A TCP packet with the SYN flag set
- SYN packet carries initial state
  - Receive window
  - Source port number
  - Initial sequence number



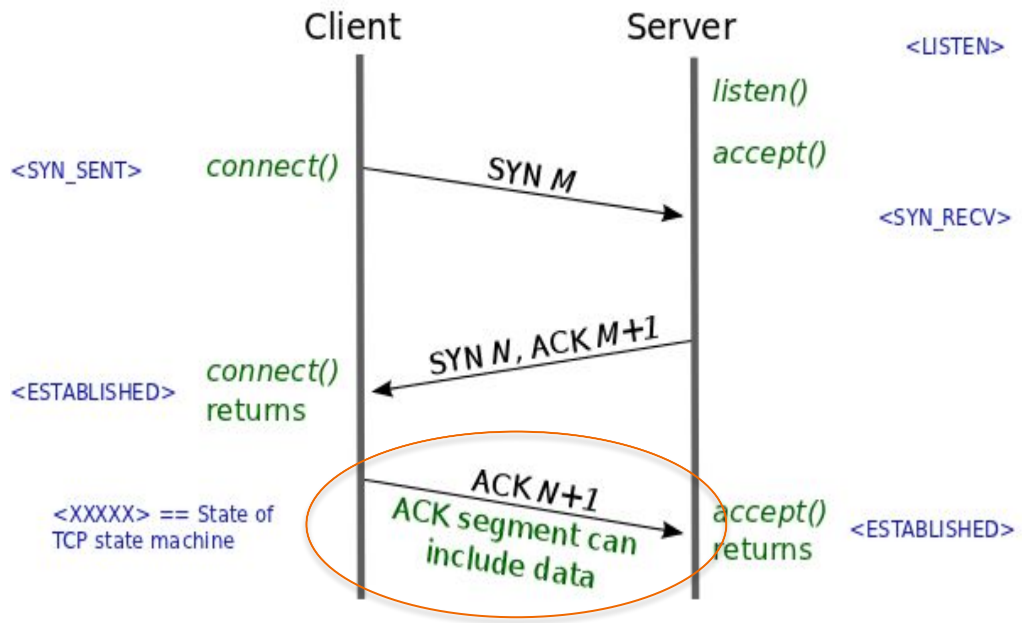
## Three Way Handshake

- Server responds with a SYN-ACK
- Acknowledge the client's SYN
- Respond with the server's own SYN
- Carries server's initial state
  - Receive window
  - Source port number
  - Initial sequence number



# Three Way Handshake

- Client responds with an ACK
- Now the connection is established!
  - Client and server can freely communicate



## Connection Establishment

- Sequence number and acknowledgement number don’t start from zero
- Instead client and server choose a **random** initial seq #
  - Why?
- Must be communicated between client and server

## Connection Establishment

- Client/Server exchange state
  - Initial Sequence numbers
  - Port number
  - TCP Options

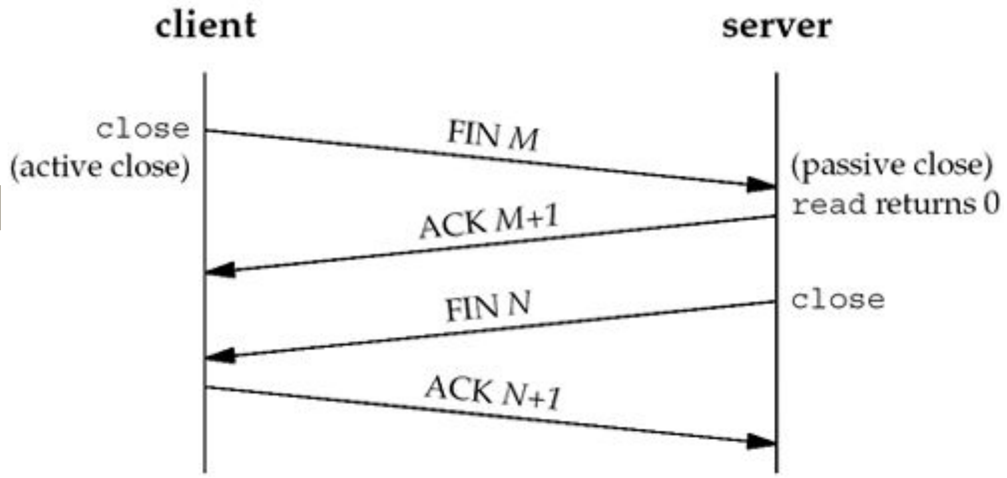
## Teardown

## Two Teardown Methods

- Four way handshake
  - Graceful/normal teardown
- Reset
  - Exceptional Teardown

# Normal Teardown

- One side closes their connection
  - Indicates they will send no more data
  - Sends a FIN
  - Receives an ACK
- Other side may continue transmitting
- Eventually it closes as well
  - Sends a FIN
  - Receives an ACK
- Connection is closed



## TCP Reset

- Reset the connection
  - Most commonly, attempt to SYN on a closed port
- Send RST packet(s) only (no ACK)

## Reliability

## Reliability

- Cumulative ACKs
  - Allow detection of dropped packets
- How do we know a packet was lost?
  - Timeout
  - Duplicate ACKs

## Timeout

- Retransmission Time Out (RTO)
  - Timeout after which packets are retransmitted
  - Based on a constantly updated RTT estimate (and variance)
- Single timer (not per-packet)
  - Each received ACK of new data resets RTO
  - If RTO times out
    - Retransmit packet containing “next byte”

## Timeout

- What if RTO is too large?
- Packet is dropped . . .
  - Wait
  - Keep waiting
  - … Keep waiting
  - Timer goes off
    - Finally retransmit
- Can we do better?

## Duplicate ACKs

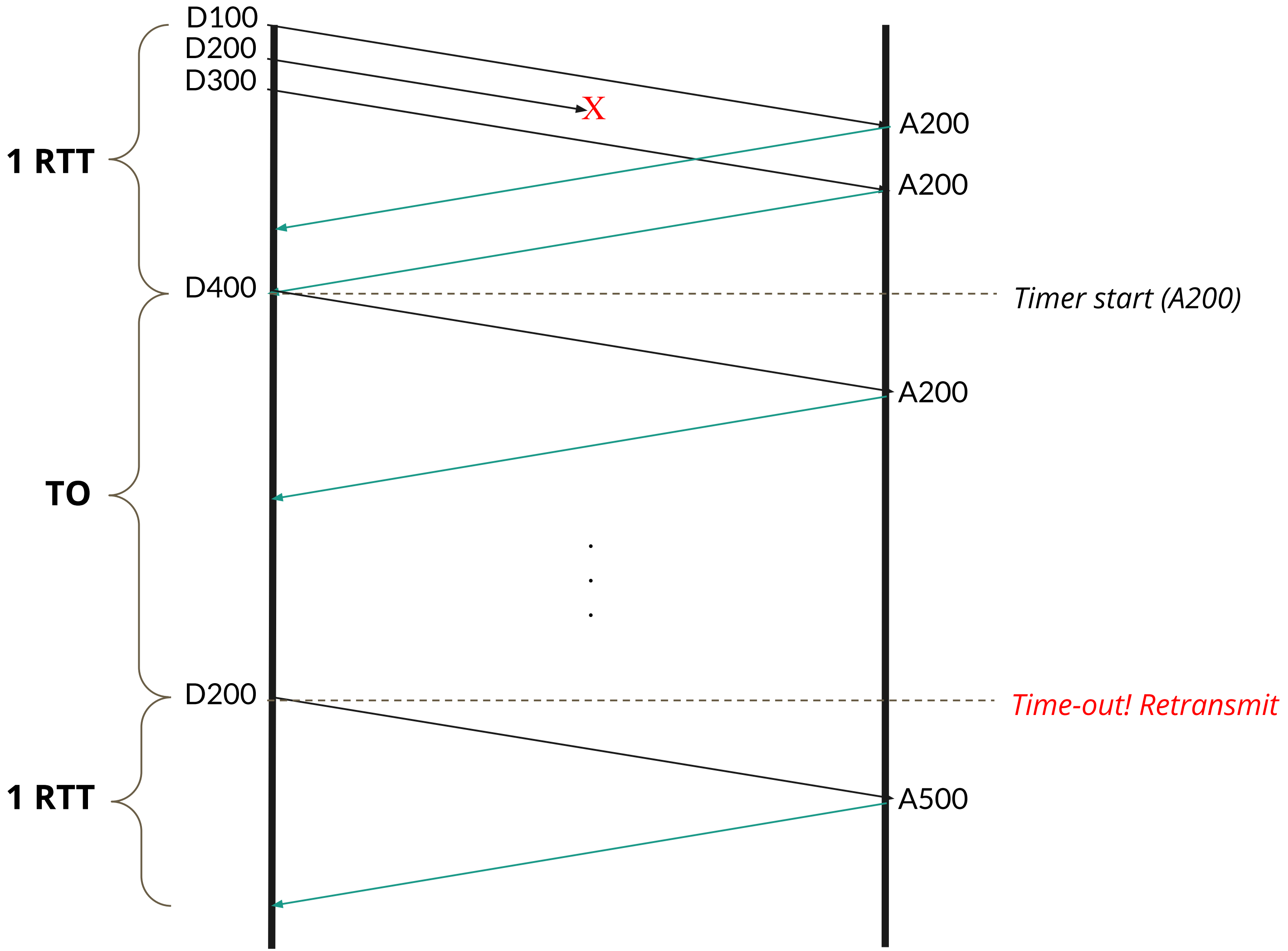
- Transmit
  - Seq 1000
  - Seq 2000
  - ***Seq 3000 Dropped in flight***
  - Seq 4000
  - Seq 5000
  - Seq 6000
- What ACKs do we receive?
  - ACK 2000
  - ACK 3000
  - ACK 3000
  - ACK 3000
  - ACK 3000
- Duplicate ACKs indicate packet loss/reorder

## Worksheet

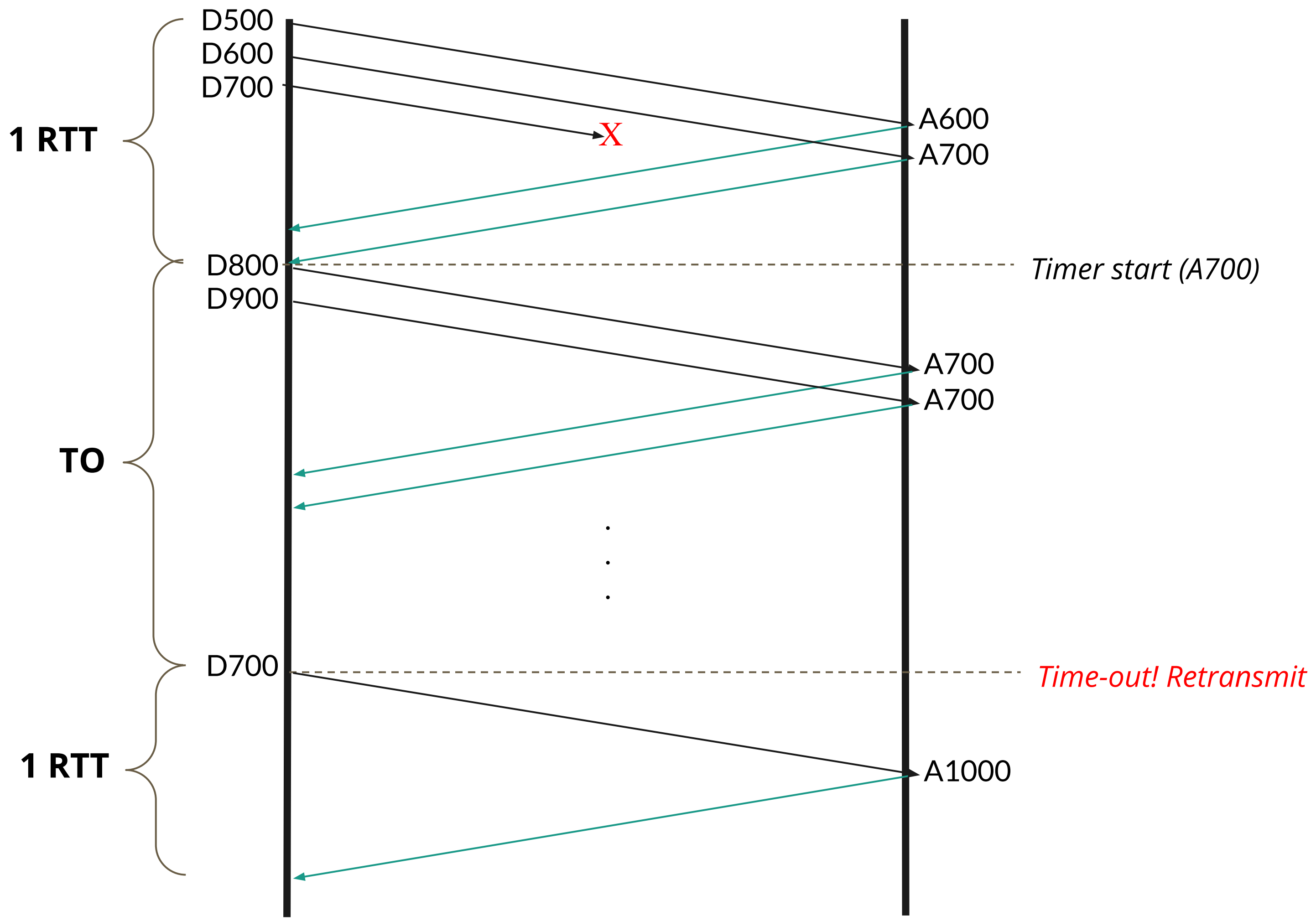
## Q1.1

| # | Packet # Sent | Sent on timeout/3rd duplicate ACK? | Dropped? | *Cumulative* ACK |
|---|---|---|---|---|
| 1 | D100 | | | A200 |
| 2 | D200 | | X | |
| 3 | D300 | | | A200 |
| 4 | D400 | | | A200 |
| 5 | D200 | X | | A500 |
| 6 | D500 | | | A600 |
| 7 | D600 | | | A700 |
| 8 | D700 | | X | |
| 9 | D800 | | | A700 |
| 10 | D900 | | | A700 |
| 11 | D700 | X | | A1000 |
| 12 | D1000 | | | A1100 |
| 13 | | | | |
| 14 | | | | |

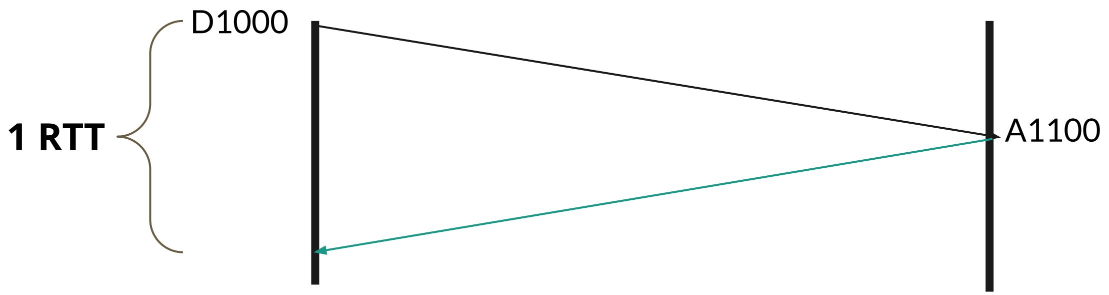
# Q1.2


D100
D200
D300
X
A200
A200
1 RTT
D400
Timer start (A200)
A200
TO
D200
Time-out! Retransmit
A500
1 RTT

## Q1.2


D500
D600
D700
X
A600
A700
1 RTT
D800
D900
Timer start (A700)
A700
A700
TO
D700
Time-out! Retransmit
A1000
1 RTT

## Q1.2



Total Time = RTT + TO + RTT + RTT + TO + RTT + RTT

= 5 * RTT + 2 * TO

= 6.05 seconds

## Q2.1 & 2.2

1. Suppose two hosts are about to open a TCP connection. The TCP headers used in the communication are only 20 bytes long and regular (no-options) IPv4 is being used for Layer 3. If the MTU of the link is 1260 bytes, what is the MSS?

2. When this connection starts, the sender starts with an ISN 19. The initial window for the sender is set to 10 packets. Given the previously calculated MSS, what ACK does the sender receive as part of the TCP handshake? After that, what is the first and last ACK the sender receives for this initial window? (Assume no packets were lost or reordered).

## Q2.1 & 2.2

1. Suppose two hosts are about to open a TCP connection. The TCP headers used in the communication are only 20 bytes long and regular (no-options) IPv4 is being used for Layer 3. If the MTU of the link is 1260 bytes, what is the MSS?

   **Solution:** 1220 bytes = 1260 bytes - 20 bytes (TCP) - 20 bytes (IP)

2. When this connection starts, the sender starts with an ISN 19. The initial window for the sender is set to 10 packets. Given the previously calculated MSS, what ACK does the sender receive as part of the TCP handshake? After that, what is the first and last ACK the sender receives for this initial window? (Assume no packets were lost or reordered).

   **Solution:** The ACK as part of the TCP handshake will be 20, because the receiver is ACKing having received the ISN. After that, the first ACK for real data sent by the sender will be 1240 which is the MSS (1220, calculated in the last problem) + the current sequence number, which is 20. The last last ACK will be 12220, which is the next sequence number after all data is sent, 12200 + ISN = 12200 + 20 = 12220.

## Q2.3

(3) In this part of the question we will determine the estimated RTT using some parameters of TCP. The following equations *may* be useful. Assume that connections have been open and that there are **no** dropped packets.

$$ETO = Estimated\_RTT + 4 \cdot Estimated\_Deviation$$
$$Estimated\_Deviation = |Estimated\_RTT - Sample\_RTT|$$
$$Estimated\_RTT = \alpha \cdot Estimated\_RTT + (1 - \alpha) \cdot Sampled\_RTT$$

a. The sender receives the current ACK and proceeds to update its various estimations. The time from when the packet was sent to when the ACK was received was $10msec$. If the previous $Estimated\_RTT$ was $70msec$, what will the new $Estimated_RTT$ be ($\alpha$ is 0.5)?

b. If the previous $Estimated\_Deviation$ was $50msec$, what is the new $Estimated\_Deviation$ ($\alpha$ is still 0.5)?

c. Based on the previous 2 questions, what will the $ETO$ be now?

# Q2.3

$$\begin{aligned} ETO &= Estimated\_RTT + 4 \cdot Estimated\_Deviation \\ Estimated\_Deviation &= |Estimated\_RTT - Sample\_RTT| \\ Estimated\_RTT &= \alpha \cdot Estimated\_RTT + (1-\alpha) \cdot Sampled\_RTT \end{aligned}$$

a. The sender receives the current ACK and proceeds to update its various estimations. The time from when the packet was sent to when the ACK was received was $10msec$. If the previous $Estimated\_RTT$ was $70msec$, what will the new $Estimated_RTT$ be ($\alpha$ is 0.5)?

**Solution:** $Estimated\_RTT = 0.5 \cdot 70msec + 0.5 + 10msec = 40msec$

b. If the previous $Estimated\_Deviation$ was $50msec$, what is the new $Estimated\_Deviation$ ($\alpha$ is still 0.5)?

**Solution:**

$$\begin{aligned} \text{Estimated Deviation} &= 0.5 \cdot \text{Estimated Deviation} + 0.5 \cdot |\text{Estimated RTT - Sample RTT}| \\ &= 0.5 \cdot 50msec + 0.5 \cdot (40msec - 10msec) \\ &= 40msec \end{aligned}$$

c. Based on the previous 2 questions, what will the $ETO$ be now?

**Solution:** $ETO = 40msec + 4 \cdot 40msec = 200msec$

## Q2.4 & 2.5

(4) What is the maximum theoretical rate of data transfer for this window size if the $Estimated\_RTT$ is what was previously estimated?

(5) Assume $40msec$ is the $Estimated\_RTT$. If the lowest bandwidth across this connection is $76.25MBps$, what is the smallest window that optimizes the question?

## Q2.4 & 2.5

(4) What is the maximum theoretical rate of data transfer for this window size if the $Estimated\_RTT$ is what was previously estimated?

**Solution:**

$$Window\_Size(\text{bytes}) = Window\_Size(Packets) \cdot 1220\text{bytes} = 12200\text{bytes}$$

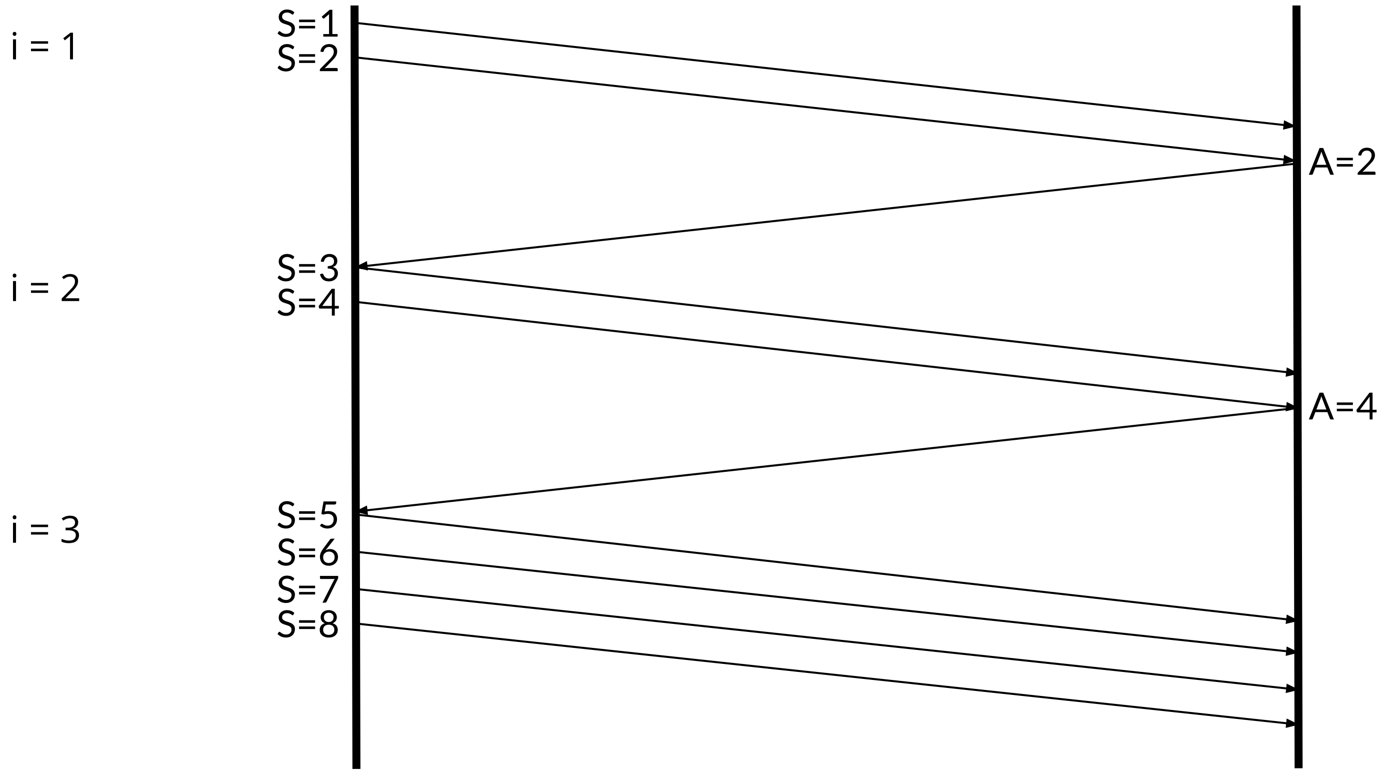
$$\text{Bandwidth} = \frac{12200\text{bytes}}{40msec} = 30500\frac{\text{bytes}}{sec} = 305KBps$$

(5) Assume $40msec$ is the $Estimated\_RTT$. If the lowest bandwidth across this connection is $76.25MBps$, what is the smallest window that optimizes the question?

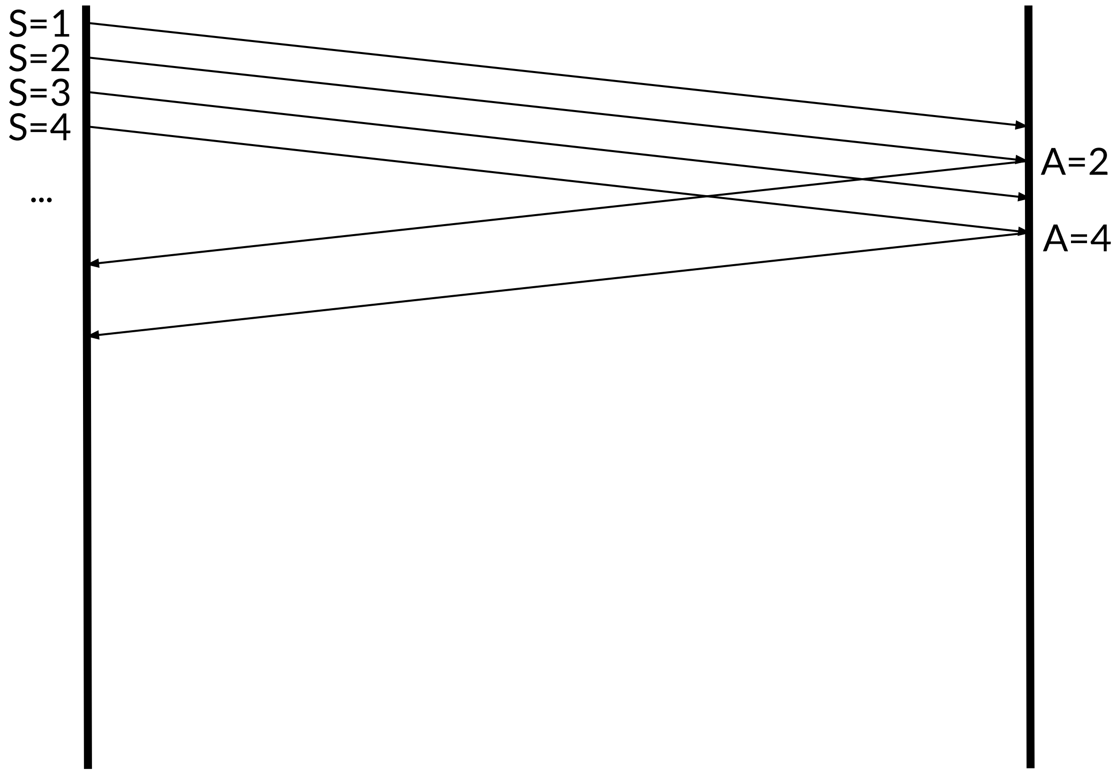
**Solution:** $\frac{76.25\times10^6\text{bytes}}{sec} \times 40msec = 3.05MB$

## Question 3: Reliability ?

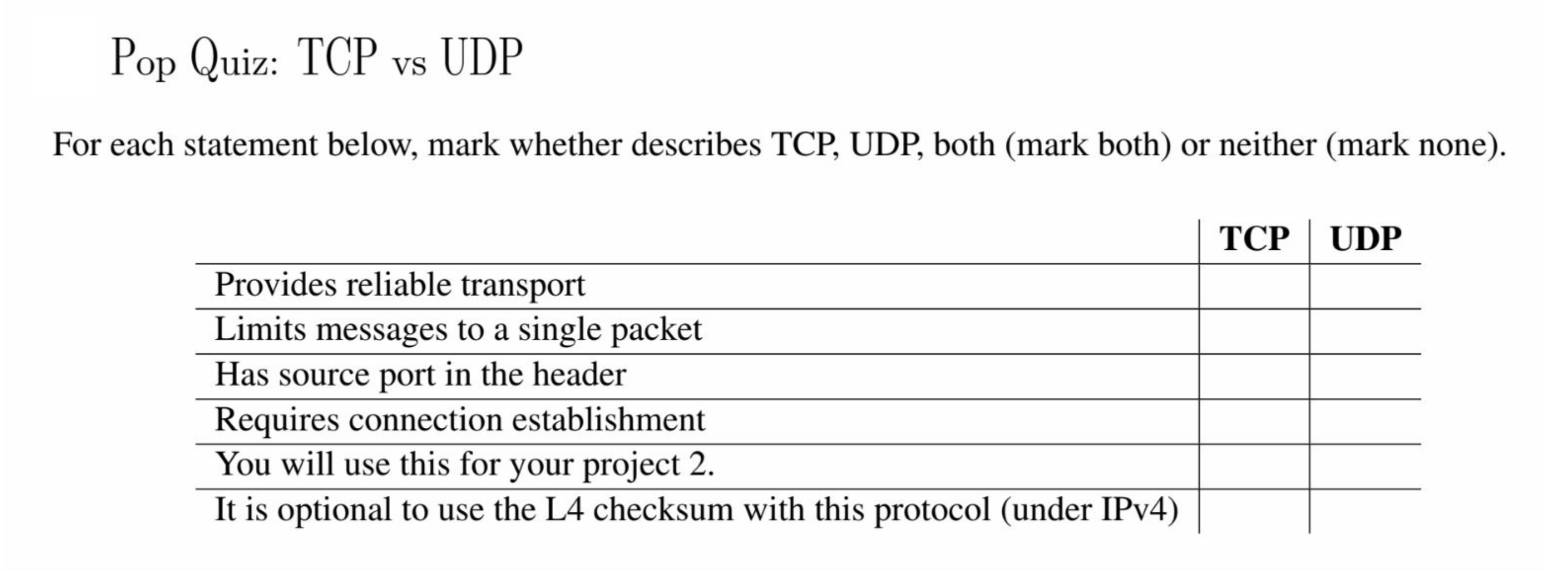
## Q3: Bob's Idea



## Q3: Alice's Idea



# Q4

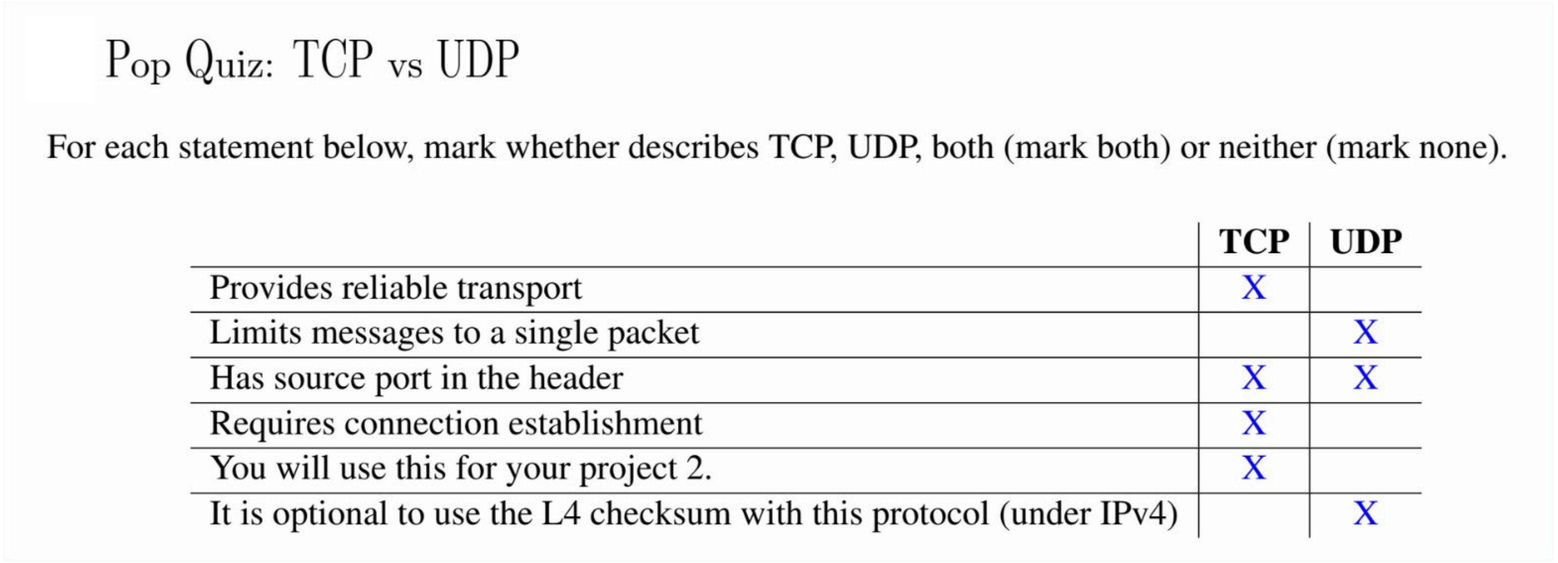


## Pop Quiz: TCP vs UDP

For each statement below, mark whether describes TCP, UDP, both (mark both) or neither (mark none).

| | **TCP** | **UDP** |
|---|---|---|
| Provides reliable transport | | |
| Limits messages to a single packet | | |
| Has source port in the header | | |
| Requires connection establishment | | |
| You will use this for your project 2. | | |
| It is optional to use the L4 checksum with this protocol (under IPv4) | | |

# Q4



## Pop Quiz: TCP vs UDP

For each statement below, mark whether describes TCP, UDP, both (mark both) or neither (mark none).

| | **TCP** | **UDP** |
|---|---|---|
| Provides reliable transport | X | |
| Limits messages to a single packet | | X |
| Has source port in the header | X | X |
| Requires connection establishment | X | |
| You will use this for your project 2. | X | |
| It is optional to use the L4 checksum with this protocol (under IPv4) | | X |

## Q5. TCP Short Questions

Discuss with a person next to you.

## Q5. TCP Short Questions

1. The purpose of a TCP handshake is: (circle all the apply)

   (a) Connection establishment

   (b) Exchange of initial sequence number

   (c) Transfer of client's request data

   (d) To indicate a packet loss

   **Solution:** (a) Connection establishment, (b) Exchange of initial sequence number

2. Which of the following is provided by TCP, but not by UDP? (circle all that apply)

   (a) Reliable transfer

   (b) Congestion control

   (c) Mux and demux from/to application processes

   (d) Byte-stream abstraction

   **Solution:** (a) Reliable transfer, (b) Congestion control, and (d) Byte-stream abstraction. These are all true by definition of what TCP is. Mux/demux from/to application processes is done using port numbers, which both TCP and UDP use.

## Q5. TCP Short Questions

3. What flag is usually set on the first packet in a TCP exchange?

   (a) RST

   (b) SYN

   (c) IETF

   (d) FIN

   **Solution:** (a) SYN.

4. What message type does TCP send to abruptly terminate a connection?

   (a) RST

   (b) SYN

   (c) IETF

   (d) FIN

   **Solution:** (a) RST. SYN packets are used for handshakes, IETF isn't a type of TCP packet, and FIN packet are indeed used by TCP, but for *graceful* termination of a connection.

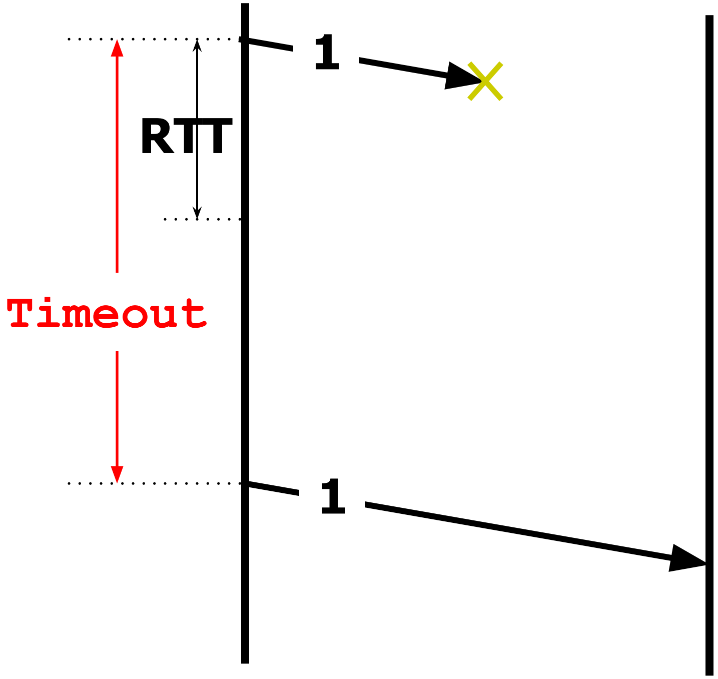
# Extra Slides

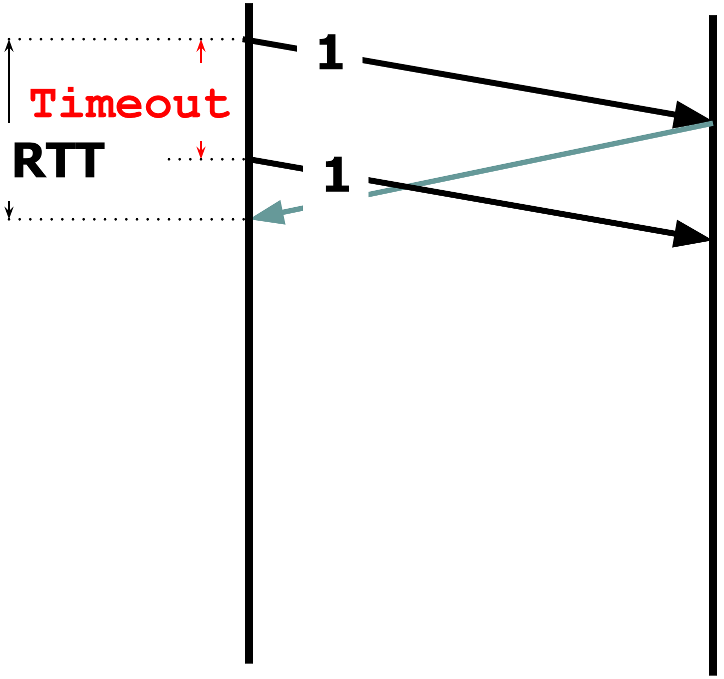
## Estimating RTO

## Timeouts and Retransmissions

- Reliability requires retransmitting lost data
- Involves setting timers and retransmitting on timeouts
- TCP only has a single timer
- TCP resets timer whenever new data is ACKed
- Retx packet containing “next byte” when timer expires
- RTO (Retransmit Time Out) is basic timeout value

# Setting the Timeout Value (RTO)





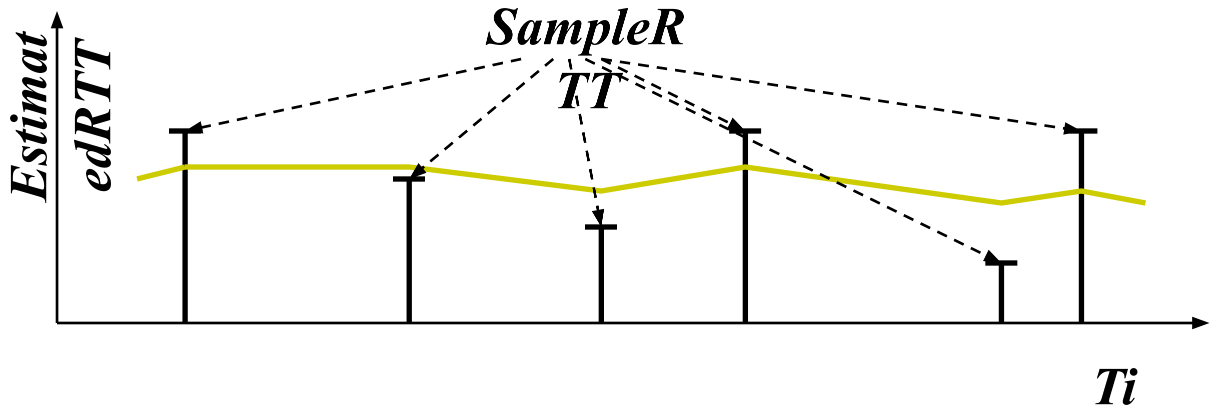
Timeout too long → inefficient

Timeout too short → duplicate packets

# Could Base RTO on RTT Estimation

- Use exponential averaging of RTT samples

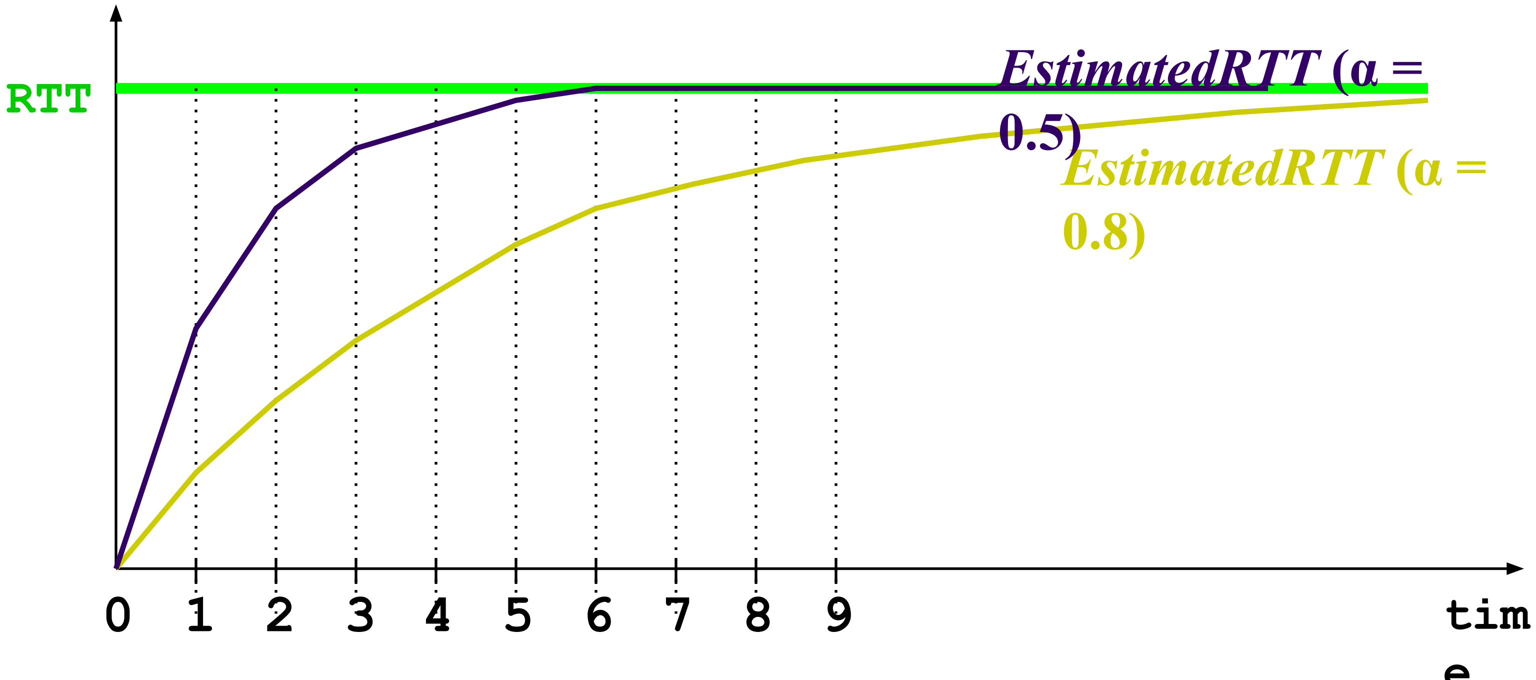
$$SampleRTT = AckRcvdTime - SendPacketTime$$
$$EstimatedRTT = \alpha \times EstimatedRTT + (1-\alpha) \times SampleRTT$$
$$0 < \alpha \le 1$$



# Exponential Averaging Example

*EstimatedRTT* = α**EstimatedRTT* + (1 – α)**SampleRTT*

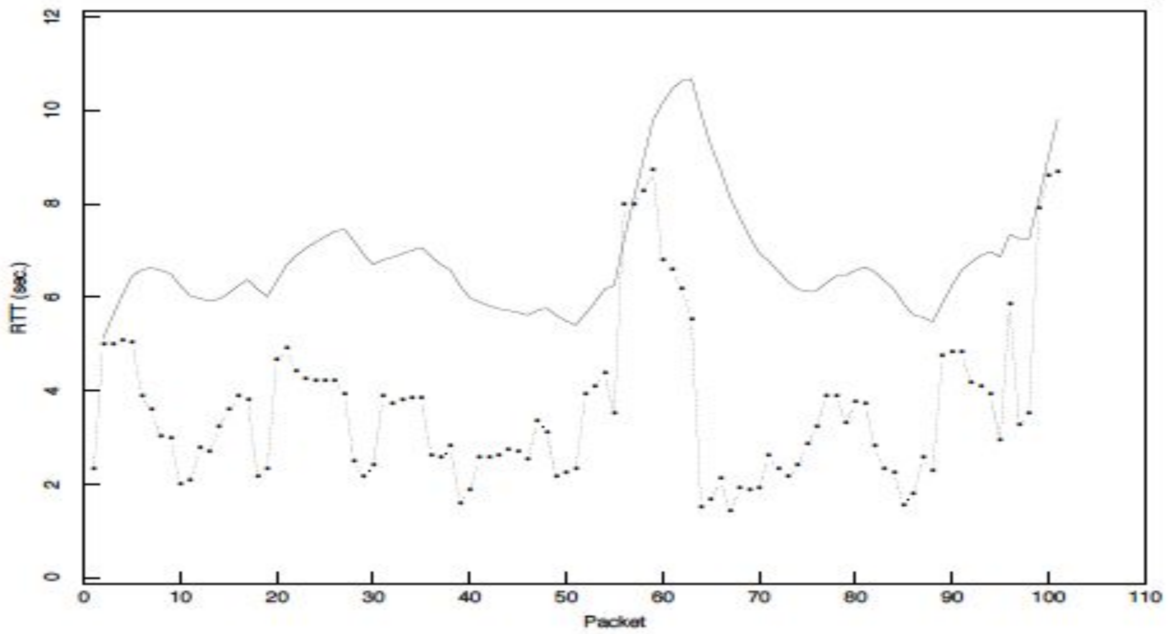
Assume RTT is constant → *SampleRTT* = RTT



# Exponential Averaging in Action

## Set Timeout estimate (ETO) = 2 × *EstimatedRTT*

Figure 5: Performance of an RFC793 retransmit timer



from Jacobson and Karels, SIGCOMM 1988

# Jacobson/Karels Algorithm

- Problem: need to better capture variability in RTT
  - Directly measure deviation

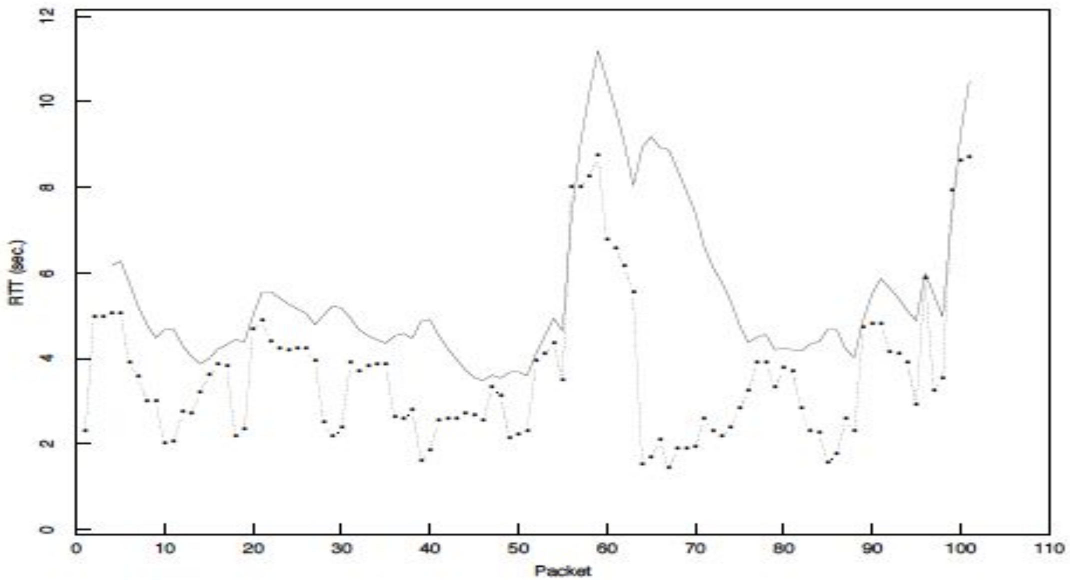
- Deviation = | SampleRTT – EstimatedRTT |

- EstimatedDeviation: exponential average of Deviation

- ETO = EstimatedRTT + 4 x EstimatedDeviation

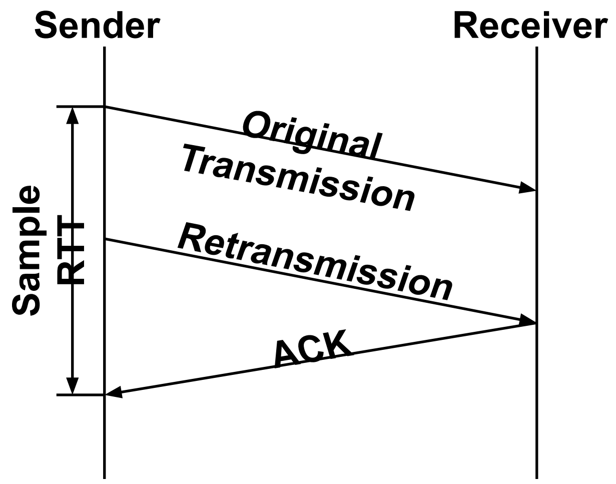
# With Jacobson/Karels

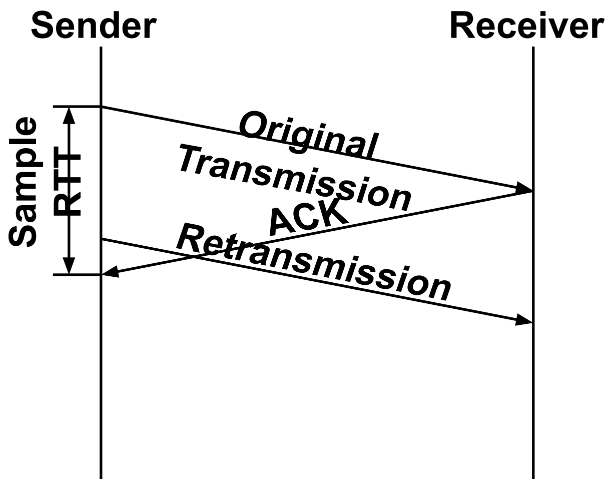
Figure 6: Performance of a Mean+Variance retransmit timer



## Problem: Ambiguous Measurements

- How do we differentiate between the real ACK, and ACK of the retransmitted packet?





## The Following TCP Rules….

- Do not describe current implementations….
- ...but will be considered the “Truth” in this class

## TCP Timers

- Two important quantities:
  - RTO: value you set timer to for timeouts
  - ETO: current estimate of appropriate “raw” timeout
- Use exponential averaging to estimate:
  - RTT
  - Deviation = | EstimatedRTT – SampleRTT|
- ETO = EstimatedRTT + 4 x EstimatedDeviation

## Use Only “Clean” Samples for ETO

- Only update ETO when you get a clean sample
- Where clean means ACK includes no retransmitted segments

# Example

- Send 100, 200, 300
  - 100 means packet whose first byte is 100, last byte is 199
- Receive A200:
  - A200 means bytes up to 199 rec’d, expecting 200 next.
  - **Clean sample**
- 200 times out, resend 200, receive A300
  - No clean samples
- Send 400, 500, receive A600
  - **Clean samples**

## Setting RTO

- Every time RTO timer expires, set RTO $\leftarrow$ 2·RTO
  - (Up  to maximum ≥ 60 sec)

- Every time clean sample arrives set RTO to ETO

## Example

- First arriving ACK expects 100 (adv. window=500)
  - Initialize ETO; RTO = ETO
  - Restart timer for RTO seconds (new data ACKed)
    - Remember, TCP only has one timer, not timer per packet
  - Send packets 100, 200, 300, 400, 500
- Arriving ACK expects 300 (A300)
  - Update ETO; RTO = ETO
  - Restart timer for RTO seconds (new data ACKed)
  - Send packets 600, 700
- Arriving ACK expects 300 (A300)

# Example (Cont’d)

- Timer goes off
  - RTO = 2*RTO (back off the timer)
  - Restart timer for RTO seconds (it had expired!)
  - Resend packet 300
- Arriving ACK expects 800
  - Don't update ETO (ACK includes a retransmission)
  - Restart timer for RTO seconds (new data ACKed)
  - Send packets 800, 900, 1000, 1100, 1200

## Example (Cont'd)

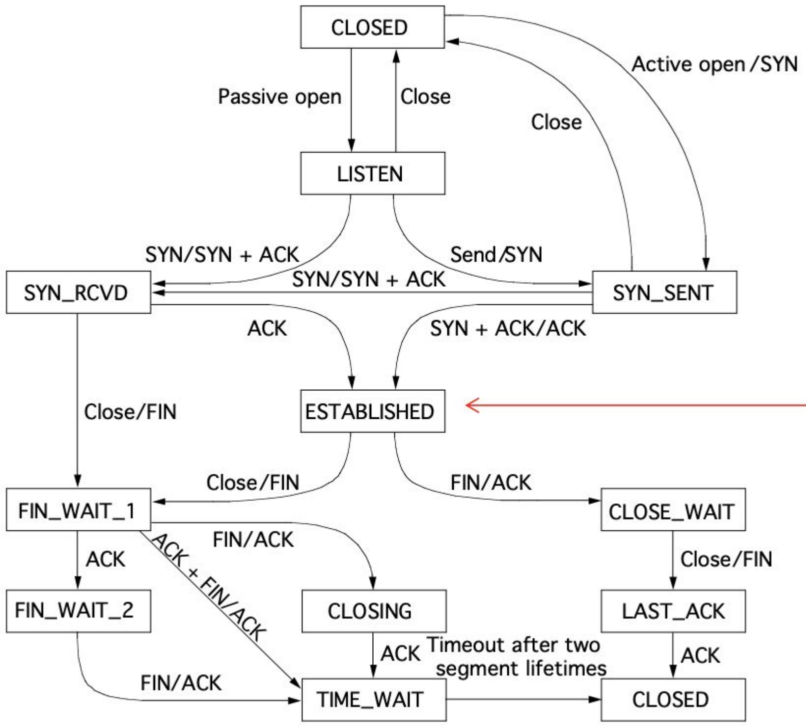
- Arriving ACK expects 1000
  - Update ETO; RTO = ETO
  - Restart timer for RTO seconds (new data ACKed)
  - Send packets 1300, 1400

- … Connection continues …

## This is all very interesting, but…..

- Implementations often use a coarse-grained timer
  - 200 msec is typical (depends on implementation)
- So what?
  - Above algorithms are largely irrelevant
  - **Incurring a timeout is expensive**
- So we rely on duplicate ACKs

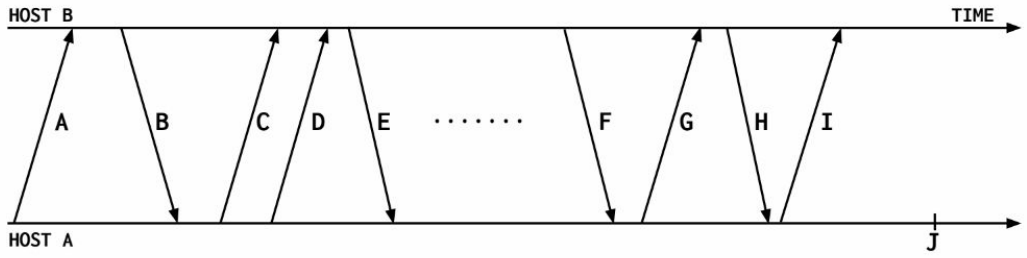
# TCP State Transitions



Data, ACK exchanges are in here

# Bonus Question: TCP Connection Life Cycle

## 2 Flags



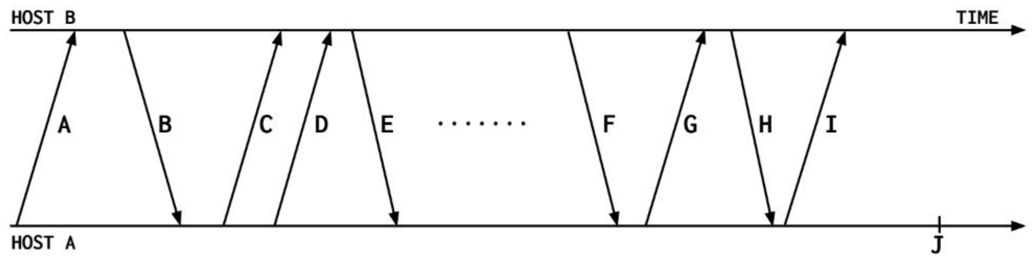
The above figure shows the life cycle of a TCP connection with normal termination - that is, connection establishment, data exchange, and teardown.

(1) For each of the arrows, choose whether it is a SYN, ACK, data, FIN or RST packet. A single arrow might have more than one of these flags set.

| | | |
|---|---|---|
| A: ____________ | D: ____________ | G: ____________ |
| B: ____________ | E: ____________ | H: ____________ |
| C: ____________ | F: ____________ | I: ____________ |

# Bonus Question: TCP Connection Life Cycle

## 2 Flags



The above figure shows the life cycle of a TCP connection with normal termination - that is, connection establishment, data exchange, and teardown.

(1) For each of the arrows, choose whether it is a SYN, ACK, data, FIN or RST packet. A single arrow might have more than one of these flags set.

| | | | | | |
|---|---|---|---|---|---|
| A: | SYN | D: | data | G: | FIN |
| B: | SYN + ACK | E: | ACK | H: | FIN + ACK |
| C: | ACK | F: | ACK | I: | ACK |