Project 2: Build Your Own Transport Protocol

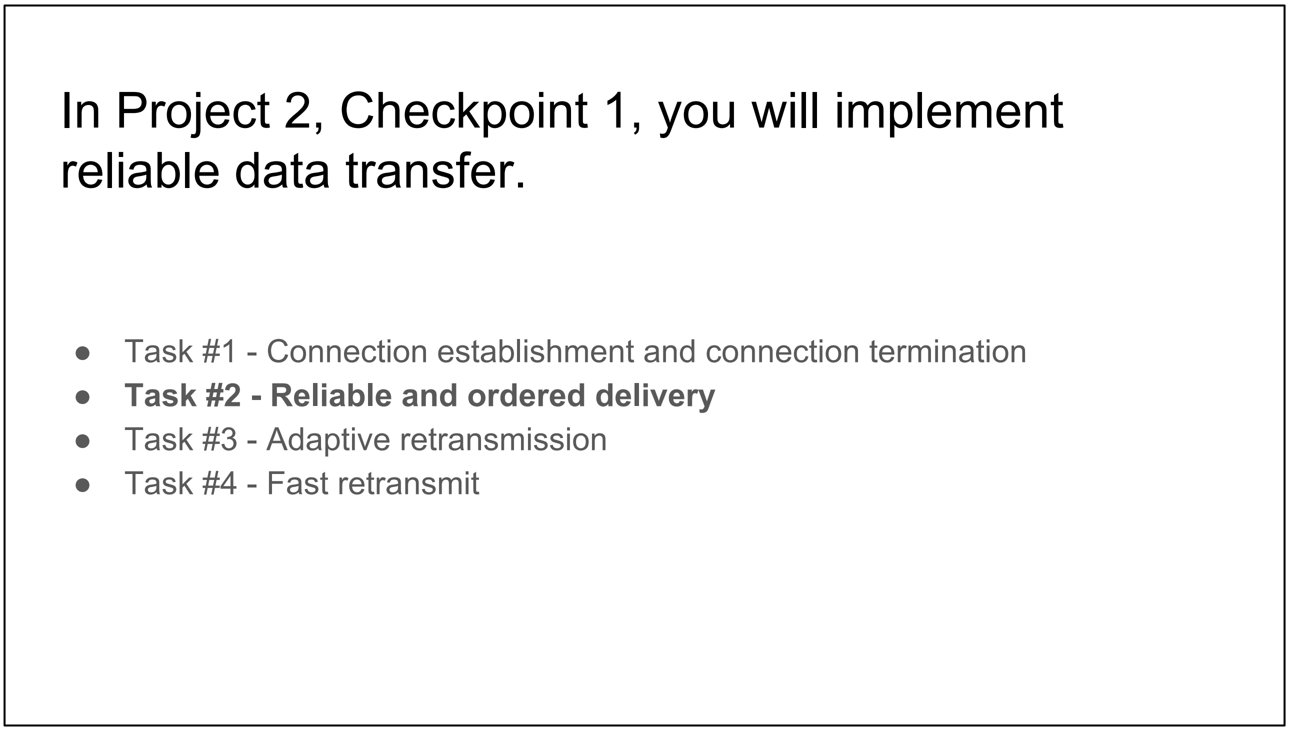
Checkpoint 1: Due Mar 1

15-441

TA

In Project 2, Checkpoint 1, you will implement reliable data transfer.

- Task #1 - Connection establishment and connection termination
- Task #2 - Reliable and ordered delivery
- Task #3 - Adaptive retransmission
- Task #4 - Fast retransmit



In Project 2, Checkpoint 1, you will implement reliable data transfer.

- Task #1 - Connection establishment and connection termination
- **Task #2 - Reliable and ordered delivery**
- Task #3 - Adaptive retransmission
- Task #4 - Fast retransmit

You will likely spend most of your time on Task #2.

Task #1: You will implement logic for connection establishment and connection termination.

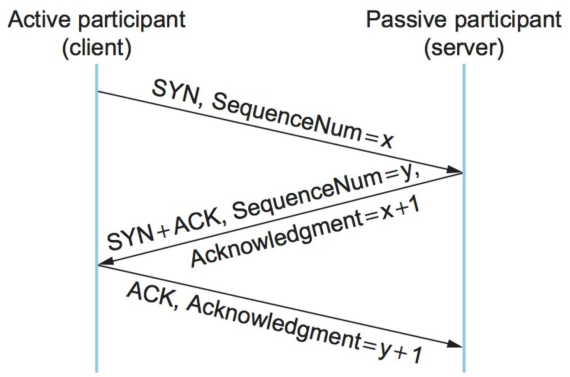
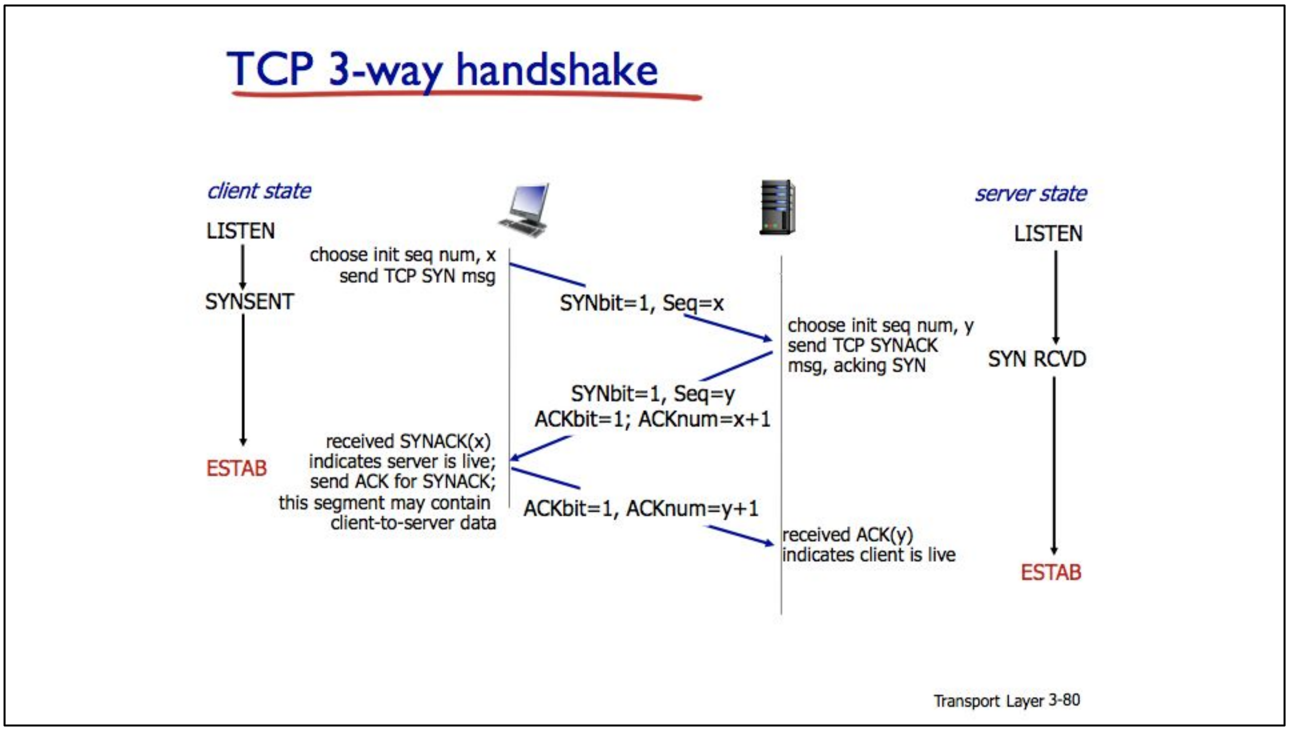

Active participant (client)
Passive participant (server)
SYN, SequenceNum=x
SYN+ACK, SequenceNum=y, Acknowledgment=x+1
ACK, Acknowledgment=y+1

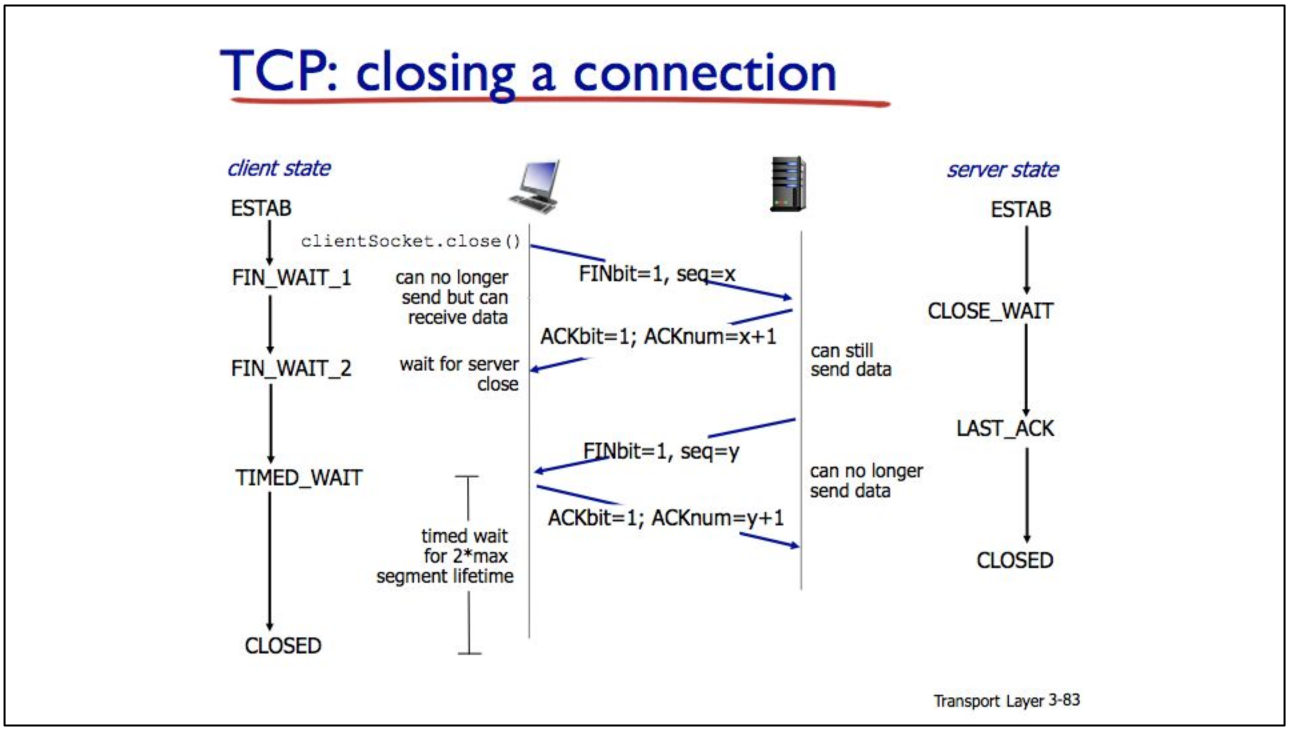
Figure 4. Timeline for three-way handshake algorithm.

- For Task #1, you should refer to the “Connection Establishment and Termination” section of *Systems Approach* https://book.systemsapproach.org/e2e/tcp.html#connection-establishment-and-termination (page 214)



TCP 3-way handshake

- Here are the typical state transitions the client and server will go through when establishing a connection. For complete state machine refer to the handout.



TCP: closing a connection

Transport Layer 3-83

- Here are the typical state transitions the client and server will go through when closing a connection. For complete state machine refer to the handout.

Task #2: You will implement reliable and ordered delivery using TCP's sliding window algorithm.

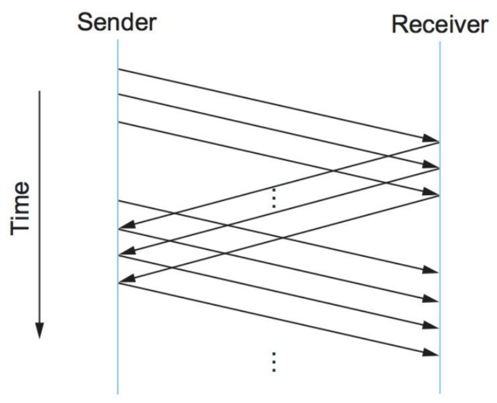
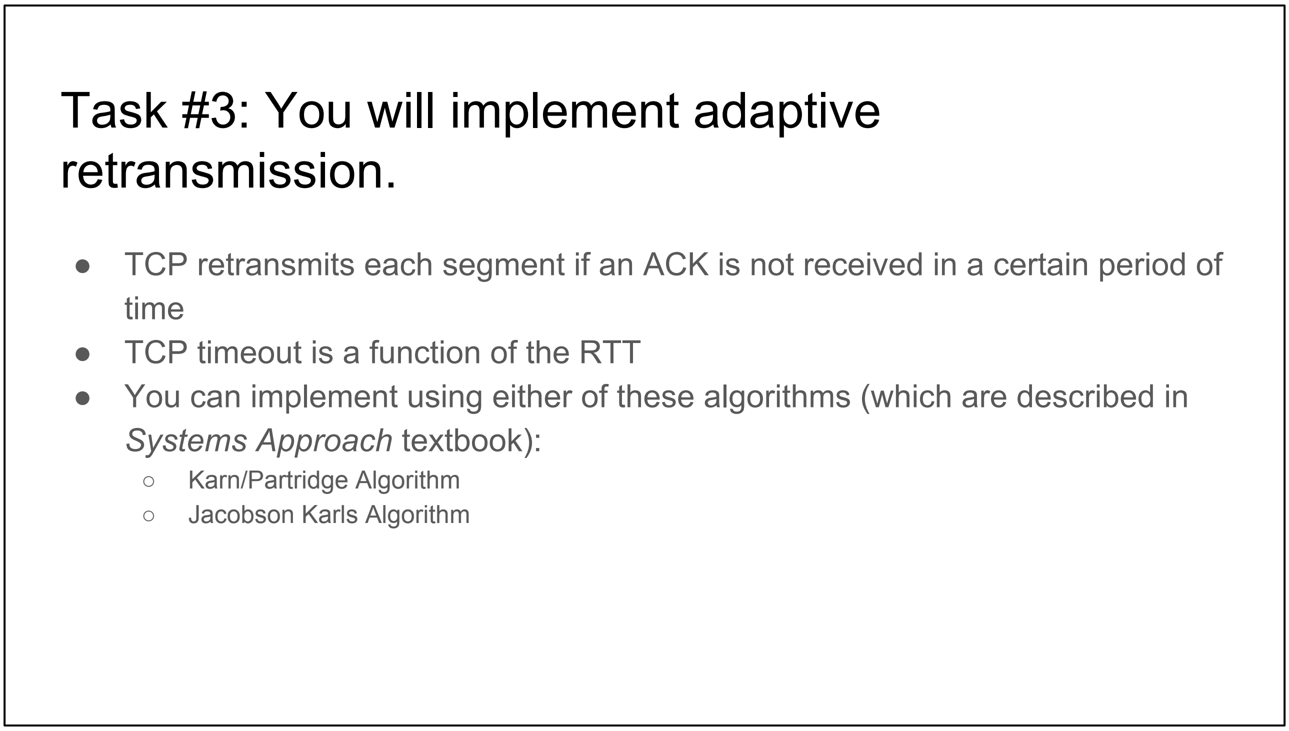


Figure 3. Timeline for the sliding window algorithm.

- The starter code you will get will implement Stop-and-Go which will send a packet and then wait for an ACK before sending another packet
- That's super slow though, so in practice TCP uses a sliding window algorithm to try and keep the pipe full of packets.
- For Task #2, you should refer to the "Sliding Window Algorithm" and "Sliding Window Revisited" sections of *Systems Approach* . There is lots of sudo code here!
  https://book.systemsapproach.org/direct/reliable.html#the-sliding-window-algorithm
  https://book.systemsapproach.org/e2e/tcp.html#sliding-window-revisited

Here is a cool animation of sliding window:
http://www2.rad.com/networks/2004/sliding_window/

- We illustrate how TCP sliding window works using an animation
- Run this animation with loss = 0% to see how it works without any loss
- Then run this animation with loss = 10% to see how it works when there is loss
    - Important things to note:
        - Receiver and sender each have a fixed window size
        - When packets arrive out of order you should still buffer them, not drop them
        - Always ACK highest in-order packet received (# of bytes)
        - Sender doesn't realize a packet was lost until a timeout (implement adaptive timeout in #3)
        - Duplicate ACKs indicate that a packet was probably lost (implement fast retransmit in #4)
    - **Not illustrated in this animation:**
        - For flow control, the receiver will throttle the sender by advertising a window that is no larger than the amount of data that it can buffer. See "Flow Control" in *Systems Approach*: https://book.systemsapproach.org/e2e/tcp.html#flow-control
        - The animation shows the sender and receiver have fixed sized windows, when really the sender is going to compute an "effective window" size to determine when it can actually send data without overloading the receiver.



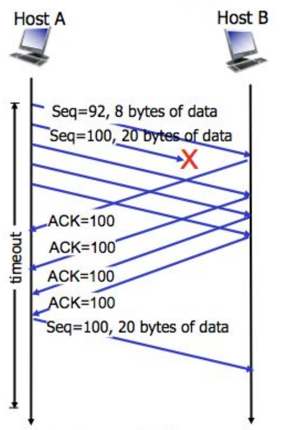
Task #3: You will implement adaptive retransmission.

- TCP retransmits each segment if an ACK is not received in a certain period of time
- TCP timeout is a function of the RTT
- You can implement using either of these algorithms (which are described in *Systems Approach* textbook):
  - Karn/Partridge Algorithm
  - Jacobson Karls Algorithm

- In Task #2, you will use a hard-coded limit of 3 seconds for deciding when to timeout. In Task #2, you need to modify this timeout to be a function of an estimated RTT.
- For Task #3, you should refer to the "Adaptive Retransmission" sections of *Systems Approach* . https://book.systemsapproach.org/e2e/tcp.html#adaptive-retransmission

Task #4: You will implement fast retransmit.

TCP fast retransmit



fast retransmit after sender receipt of triple duplicate ACK

- We noticed in sliding window animation that the receiver will send duplicate ACKs when packets are lost.
- So, in Task #4, instead of relying on timeouts for retransmissions, you may also retransmit data if you receive duplicate ACKs.
- No book/handout reference here because it should be straightforward to implement this

You do not have to implement Nagle’s algorithm.

There is a description of Nagle’s algorithm in the *Systems Approach* book (handout) but you will not have to implement this.

In Checkpoint 2, you will implement congestion control. Don’t worry about that yet. :)