Lab 5: Reliable Transport (Week 1)

Checkpoint due: Monday, November 23, 11:59 PM ET

Due: Monday, November 30, 11:59 PM ET

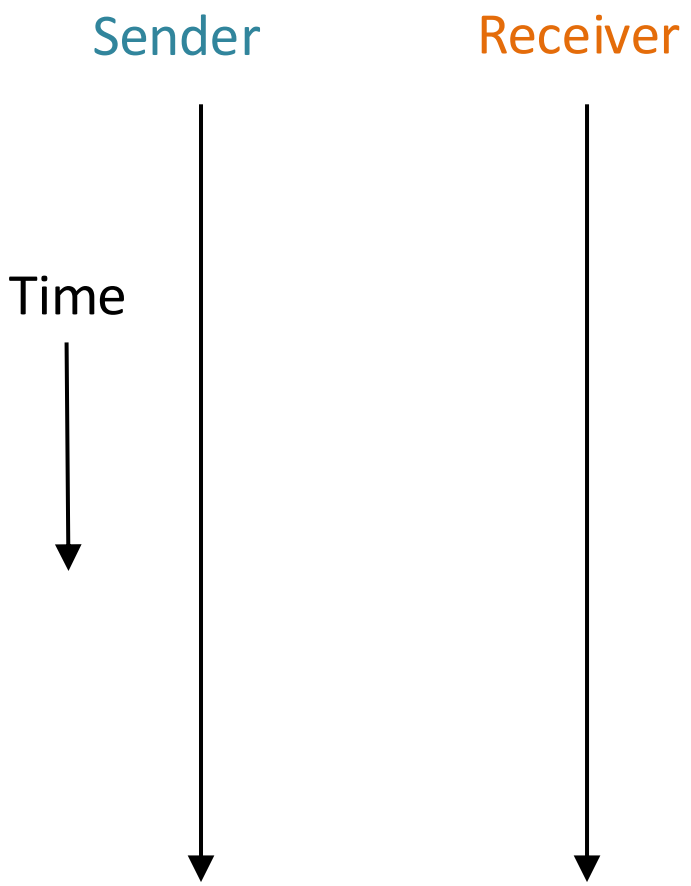
Implementing a stop and wait protocol

- Sending ACKs
- Calculating RTT to set a timeout
- Only one packet in flight

Stop and Wait

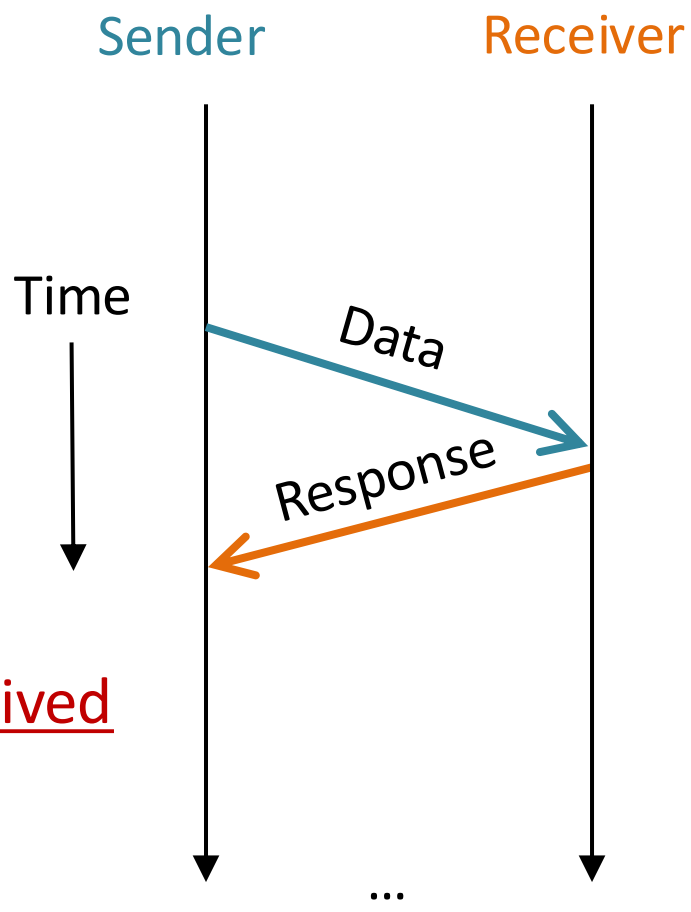
We have:

- a sender
- a receiver
- time: represented by downwards arrow



Stop and Wait

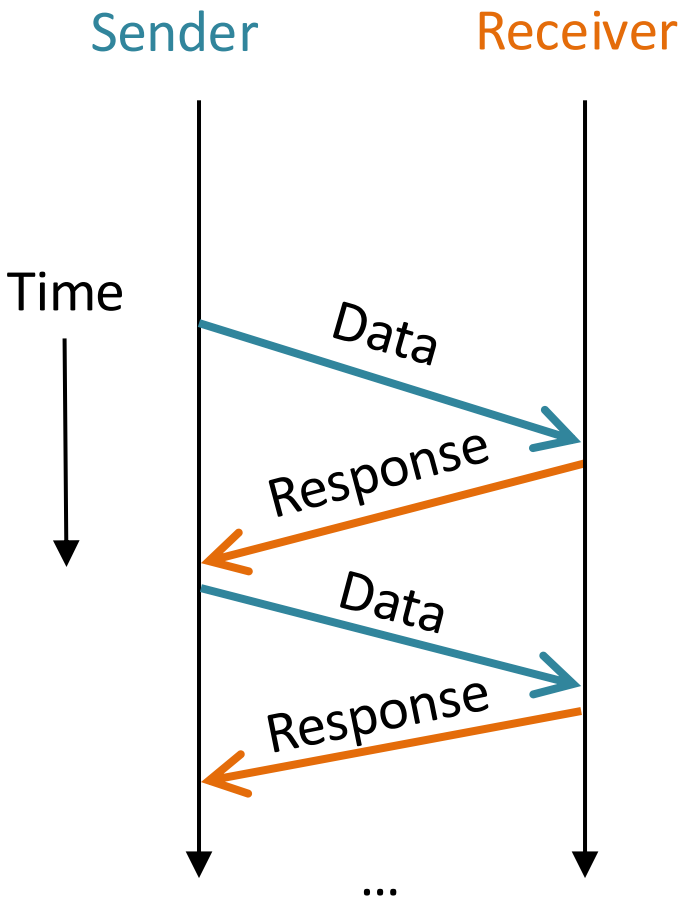
Sender sends data and waits till they get the response message from the receiver.



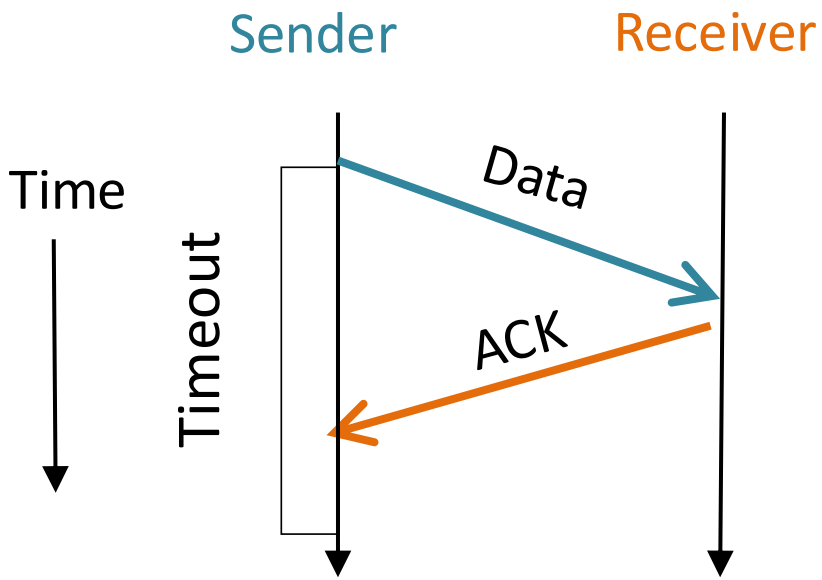
Buffer data, and don't send till response received

Stop and Wait

- Up next: concrete problems and mechanisms to solve them.
- These mechanisms will build upon each other
- Questions?

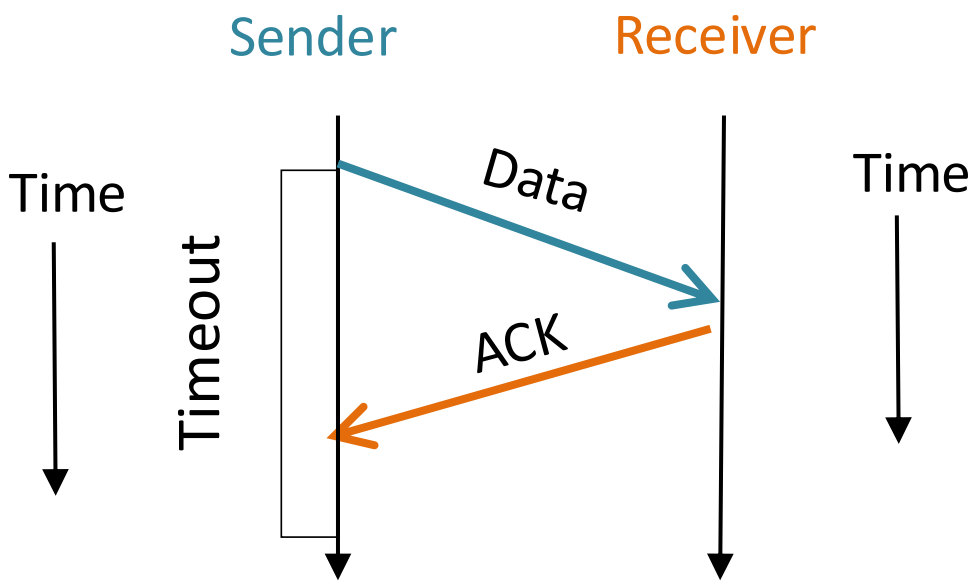


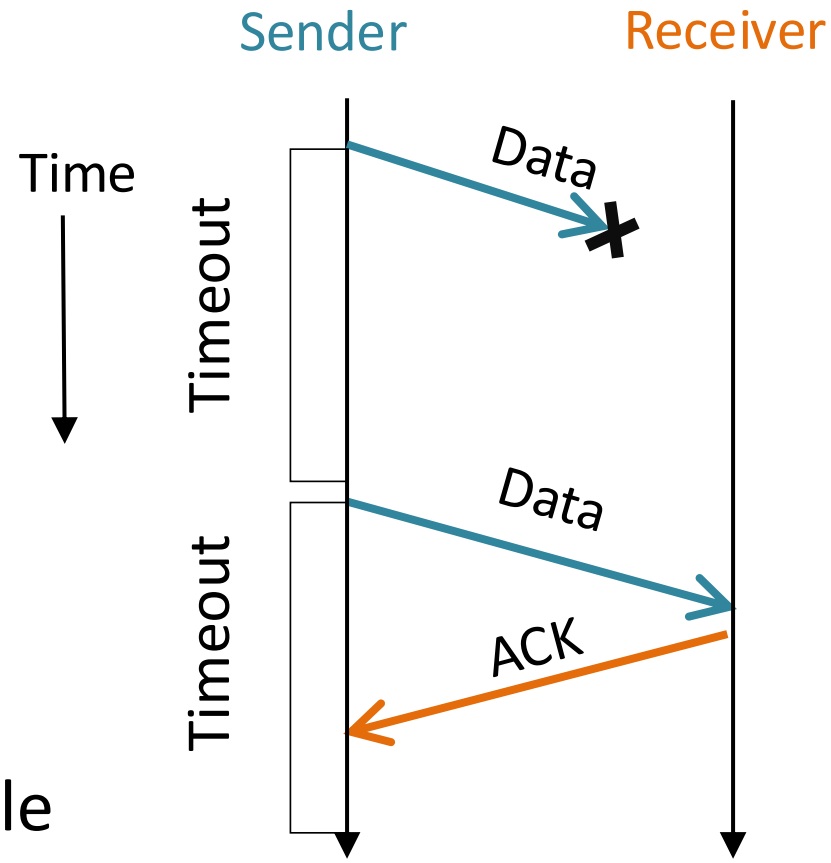
Timeouts and Losses



- Sender starts a clock. If no response, retry.

Timeouts and Losses

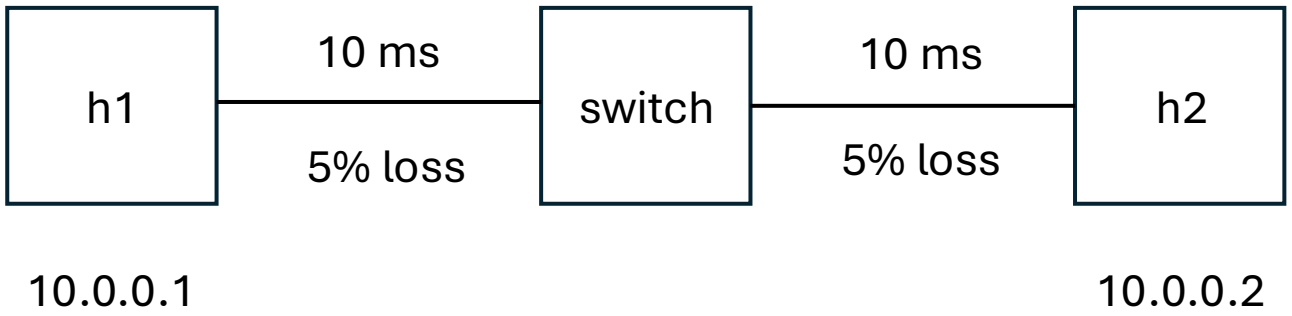




- Timeouts help us handle message losses too!

Using mininet to emulate a slow, lossy network

- Setup a virtual machine so you can emulate a network
- Can setup a network and it will create a network that



Links go in both directions. So total RTT is 40ms.

Using mininet to emulate a slow, lossy network

To setup network:
sudo -E mn --link tc,delay='10ms',loss=5,max_queue_size=2 –x

Only need h1 and h2 window (close "controller” and “switch” window)

Using virtual machine

CS43-MininetVM-S22-rware

- Click start to boot up this machine
- ssh into the VM

ssh –Y localhost –p 2222 –l mininet

Starter code

- Your implementation goes here: lab5.c and lab5.h
- **Do not** change lab5_sender or lab5_receiver. (If you want to add print statements for debugging, that's fine.) These are the applications that are going to call your implementation of reliable transport
- Need to implement my_connect, my_send, and my_recv, that lab5_sender and lab5_receiver will call

Starter code

my_socket

- Intialize your state
- Calling regular sock function to create UDP socket

my_connect

- Don’t need to add anything here
- By calling connect on the socket, we are telling underlying socket that we are only going to send to one receiver and can call send() on the socket instead of sendto()

Starter code

my_rtt

- Use to keep track of your RTT estimate is
- You'll need the RTT estimate to set the timeout

my_send

- Take in data from the user, a buffer, and how many bytes they want to send
- After sending want to wait until either you get a reply or if there is a timeout. Implement this using the select function with a timeout (see Setting Timeout section of the lab writeup)

Starter code

my_recv

- If it is correct data, pass up to user, but if it is the wrong data need to probably send something so that data can be retransmitted

my_close

- Will go over this next week

By the end of the first week you should be able to…

- In my_send: attach transport headers application data and send it
- in my_recv: receive data, remove transport headers, and pass data from the body to the application

It’s ok if your sends and receives aren’t fully reliable yet, but you need to make non-trivial progress towards reliability

Setting up virtual machine

- **You cannot store your VM in your home directory! It must be stored in /local**
- Make sure you push things to Github because this is only stored on one disk, unlike your home directory which is backed up

Setting up virtual machine

1. Open VirtualBox
2. File -> Preferences
   - Change: Default Machine Folder: /local
3. File -> Import Appliance
   - Type: /local and click Folder button and click CS43-MininetVM-S22.ova
   - Click Next
   - Double Click Name and and your name on the end
   - Click the Import button
4. Click Start and Click enter on Ubuntu
5. Can Ignore VB at this point. Open a terminal to open mininet

Getting started…

- See if you can run the lab5_sender and the lab5_receiver it will send stuff but stuff will get lost and that is not handled
- You don’t need to work on RTT estimation. Just set the RTT estimate/timeout to a fixed value (something like a twice the size of the actual 2*actual RTT or so should work fine for testing)
- Work on updating my_send and my_recv

Some important reminders…

- Your code does not need to implement the full TCP header. The state provided should be mostly all you need
- You are not required to implement the TCP behavior that allows each side to shutdown independently
  - You don’t need to implement handshake and connection setup
- When testing, printf to stderr instead of stdout so it won’t interfere with the results the receiver outputs
- When testing, can initially test with no loss rate just to see that anything is sending at all
- When testing you can redirect all output to a file on the receiver and compare the md5sum to the md5sum of 1000lines
  ./lab5_receiver 5000 > 1000lines.received
  md5sum 1000lines 1000lines.received

Using VSCode

- Weird stuff can happen if you’re not careful about using sshfs to mount directories in the right place and unmounting before you shutdown the VM
- Make sure VSCode is closed before you try to unmount