

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

KennyStockmanPhotography



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 Programming
- Threading Models
- Understanding Performance
- Communication Frameworks
- High Performance Middleboxes

How will we do it?

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

Part 2: Distributed Systems

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

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 student gets \$100 credit towards cloud resources

Grading

(To be determined)

Attendance and Participation

Group Projects

Midterm and Final Exam

Networking Basics

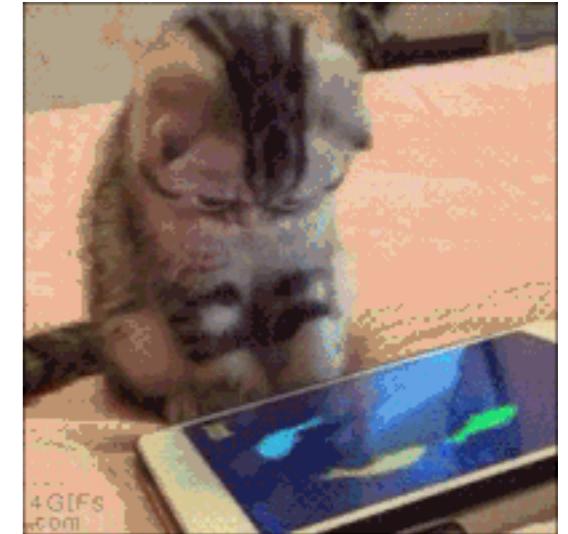


How to watch a cat video?



Me

—????????????????→

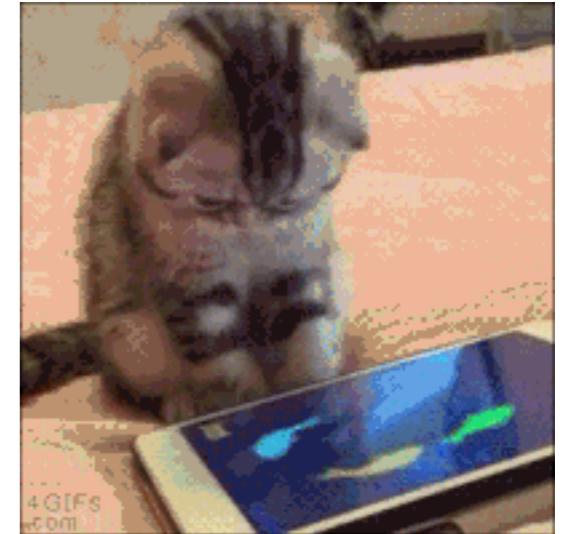
A thick green horizontal line starts at the bottom of the smartphone and extends to the right, ending in a green arrowhead pointing towards the cat image.

a cat

How to watch a cat video?



— catvids.org/fishes.gif →



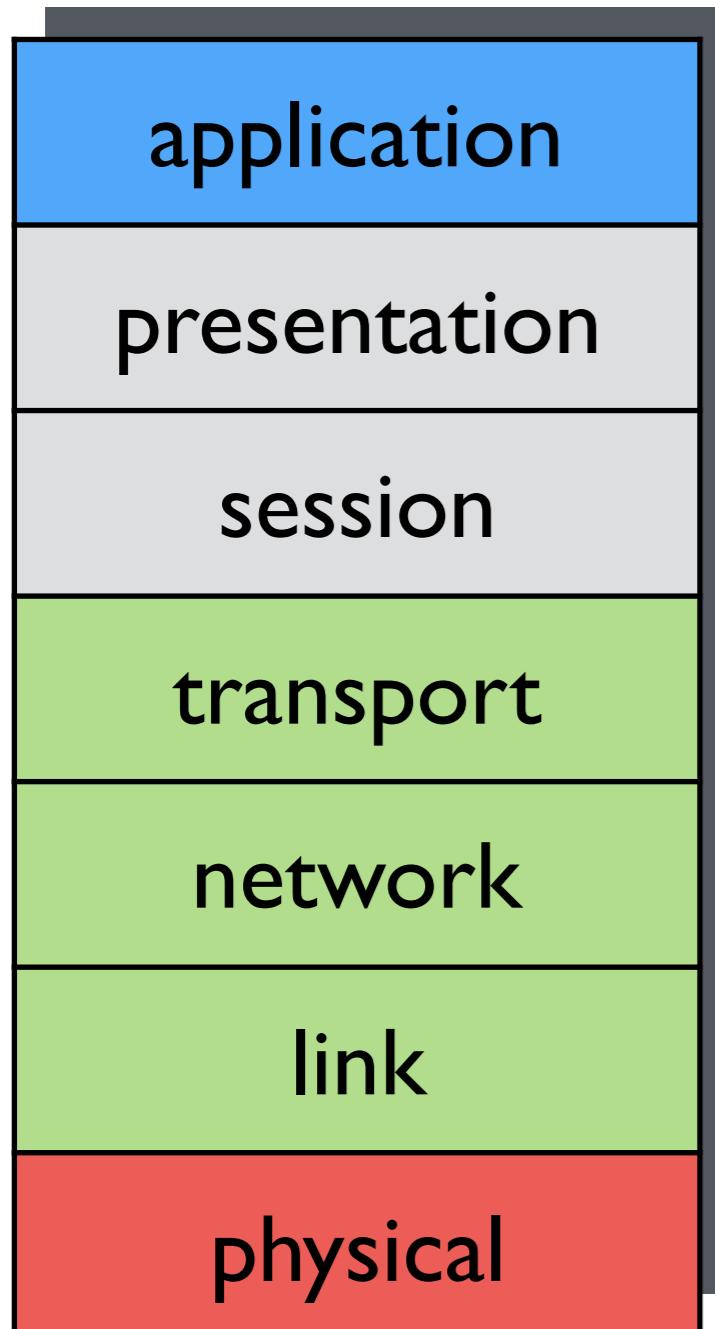
1. Convert hostname to an **IP** address with **DNS**
2. Establish a **socket** connection to the IP and port
 - Use a pre-defined standard to decide port (e.g., 80=web traffic)
3. Send a request for the video
 - Use a pre-defined **protocol** to format the request (e.g., HTTP)
4. Receive the video from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

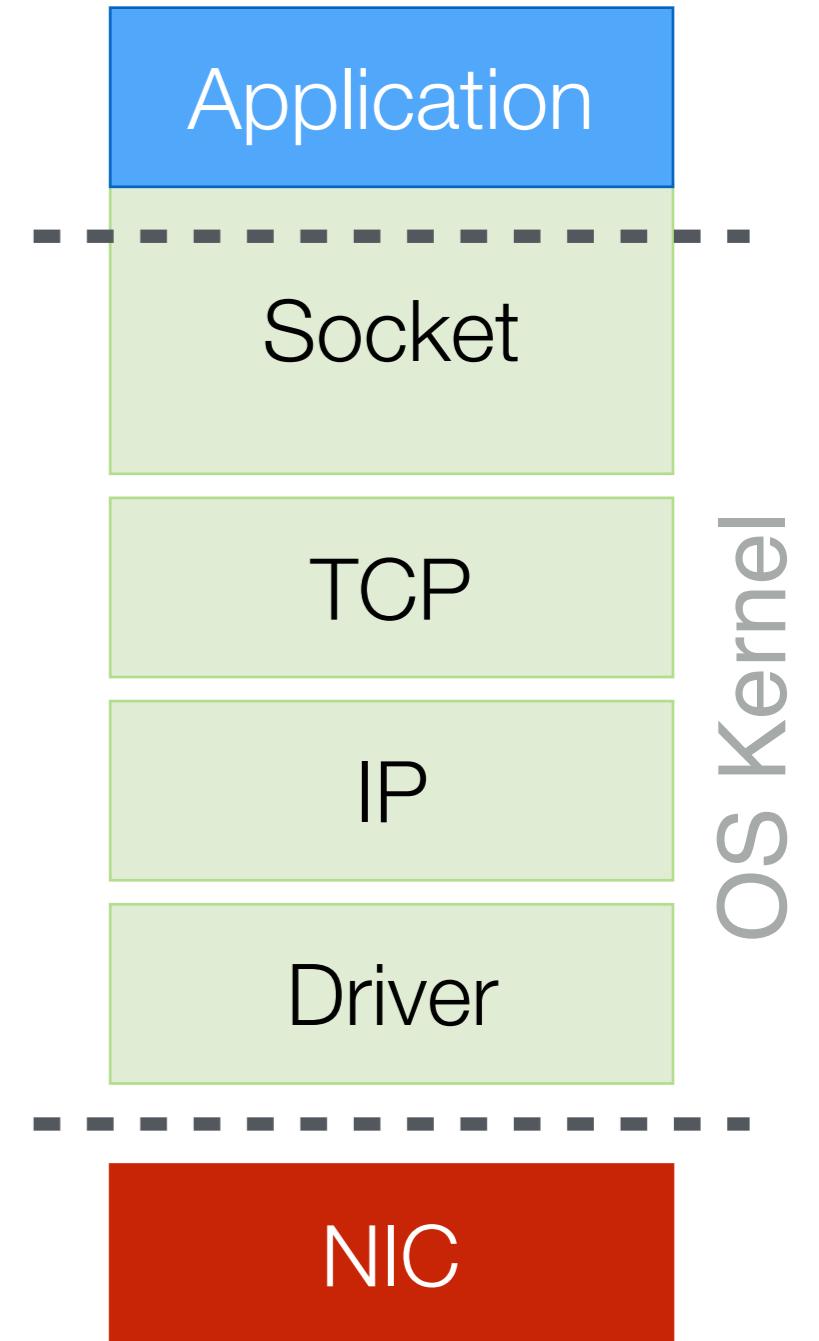
- Handles addressing and routing

Transmission Control Protocol (TCP)

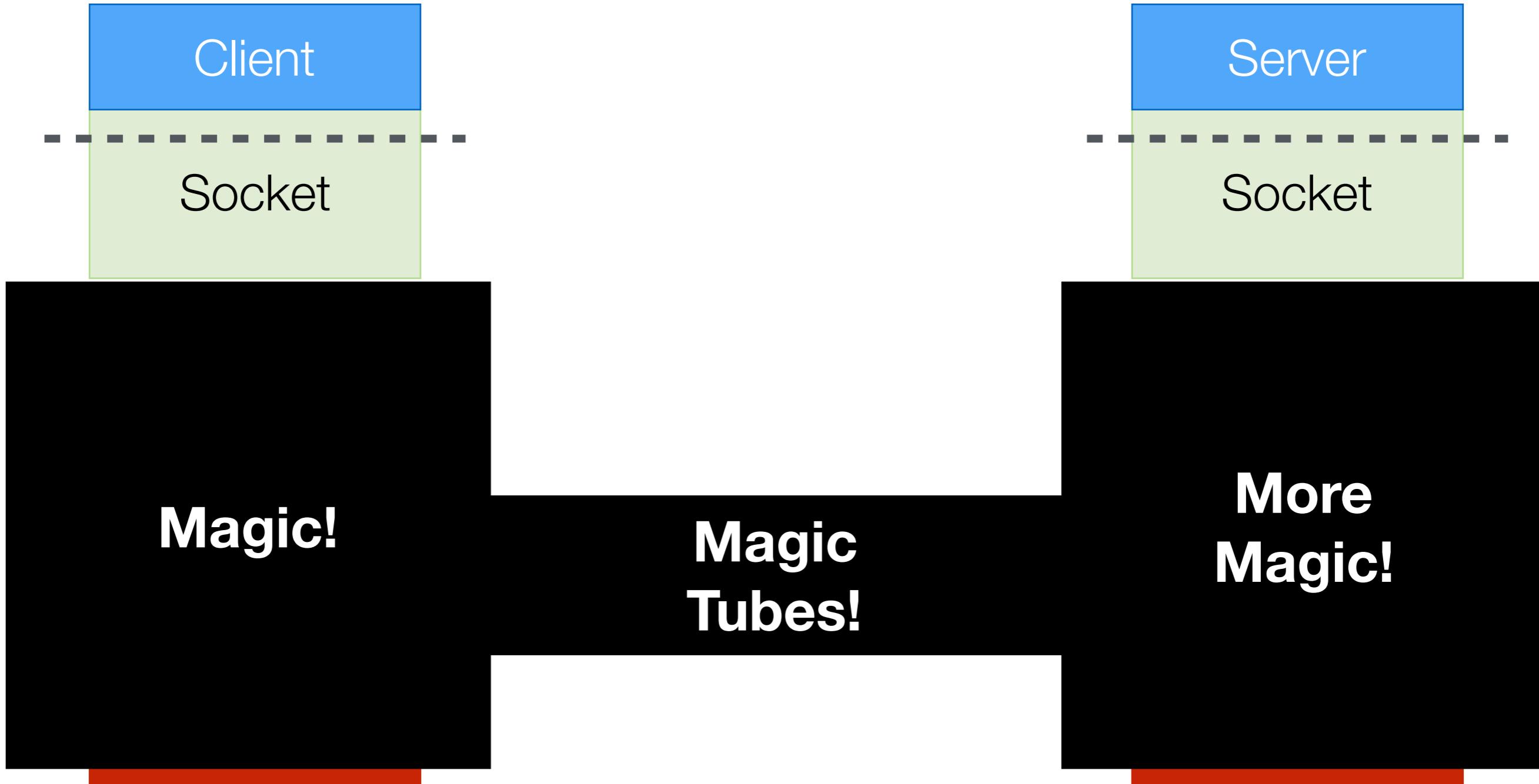
- Ensures reliable, ordered transmission of packets and manages congestion

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!

TCP Socket

Reliable Tube

TCP Socket

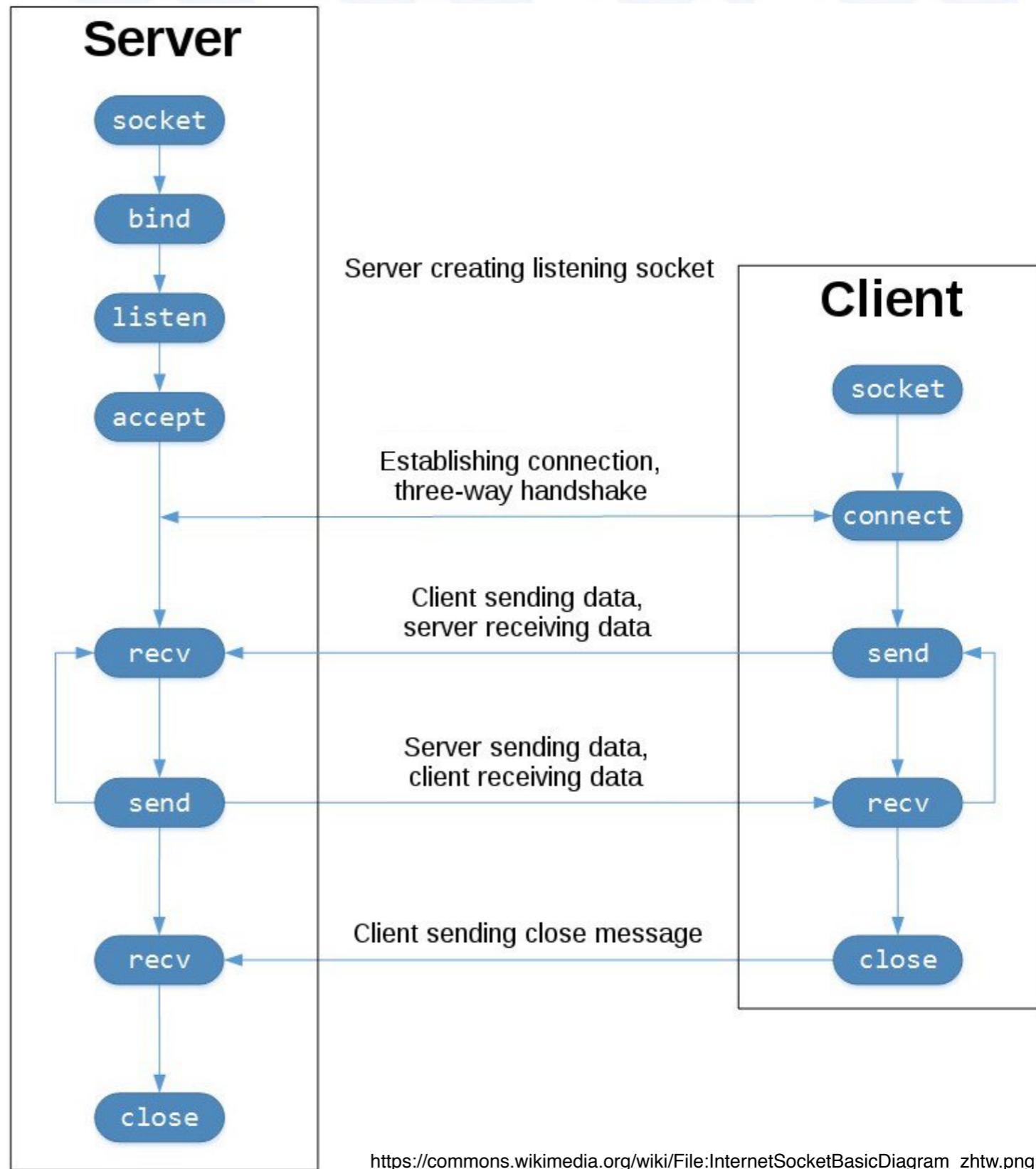
UDP Socket

Unreliable Tube

UDP Socket

Socket API

Socket
Connect
Bind, Listen, Accept
Send, Receive
Close



Cloud 9

(See instructions on website)

My Classrooms Workbench Create a new environment

https://www.awseducate.com/

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
	In this course, students will learn how to write object-oriented code using Java. Concepts will focus on object-oriented thinking, software composition, inheritance				Go to classroom ➔

The screenshot shows the AWS Management Console homepage. At the top, there are several tabs: "My Classrooms", "Workbench", "AWS Management Co...", "Workbench", and "Create a new environment". The main navigation bar includes links for "Services", "Resource Groups", a bell icon for notifications, and account information ("vocstartsoft/user131101=timw... N. Virginia"). The title "AWS Management Console" is prominently displayed.

AWS services

Find Services
You can enter names, keywords or acronyms.

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 across your AWS accounts.

 Create a new environment 

 Services  N. Virginia Support

Environment settings

Environment type  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.

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.



 [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++

Javascript
Python
Swift
Kotlin
Scala

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

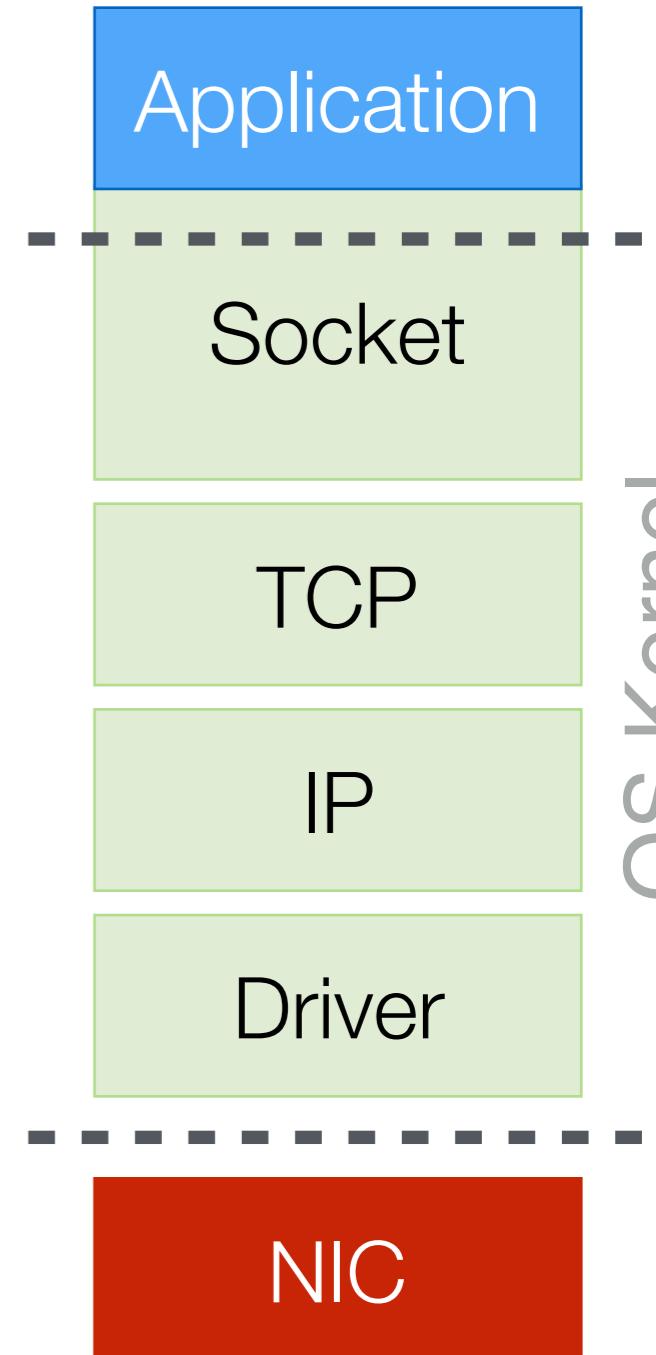
- Is all data guaranteed to be sent to kernel?
Probably not!

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

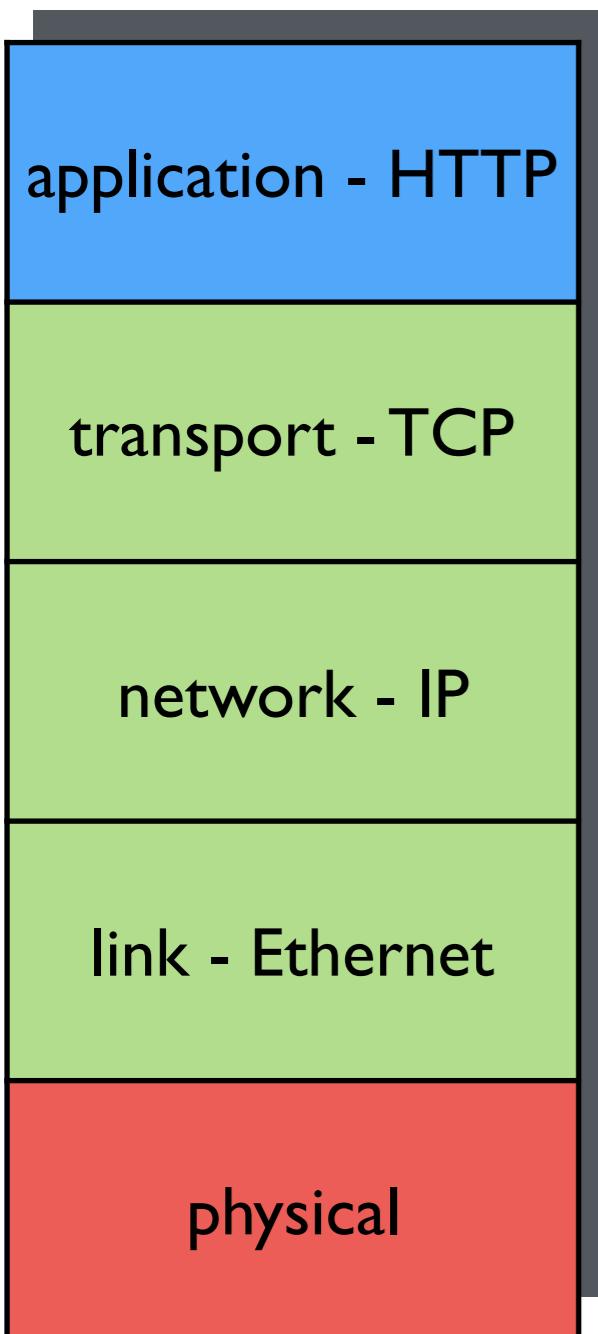
link - Ethernet: Send chunks of data

physical

What is a packet?

It's really just a blob of data!

- But its structure is well defined by protocols



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							
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)													

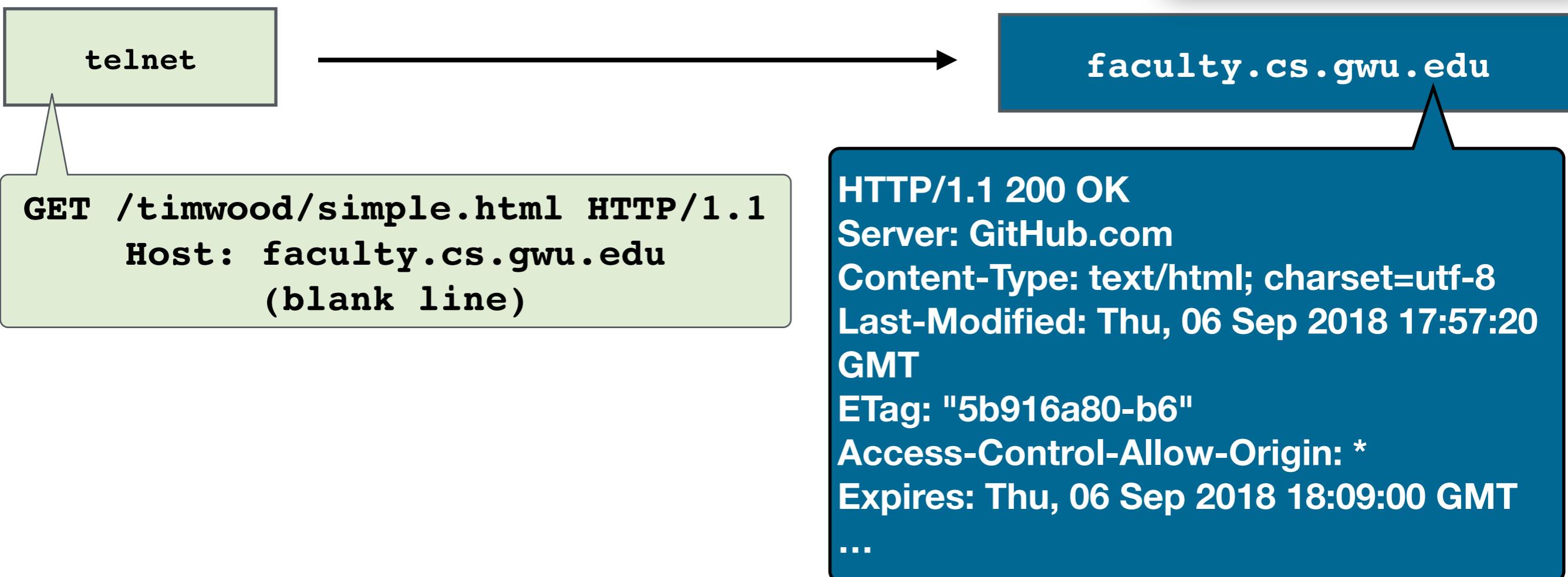
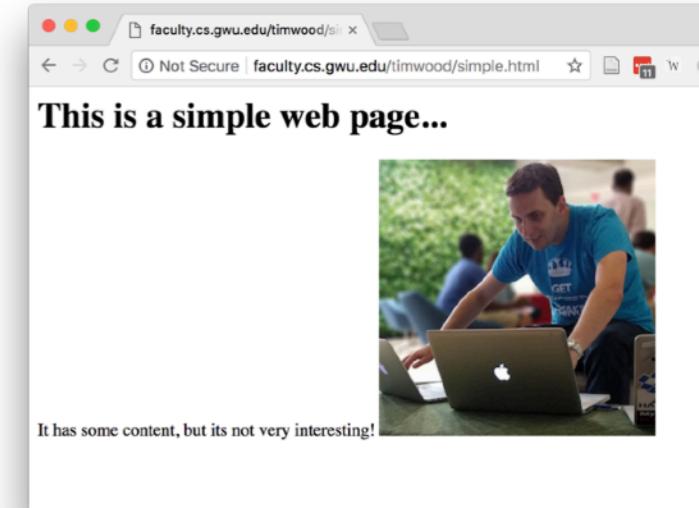
TCP Segment Header Format										
Bit #	0	7	8	15	16	23	31			
0	Source Port				Destination Port					
32	Sequence Number									
64	Acknowledgment Number									
96	Data Offset	Res	Flags		Window Size					
128	Header and Data Checksum				Urgent Pointer					
160...	Options									

GET /index.html HTTP/1.1 ...

Let's try HTTP

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

Usage: **telnet host port**



Let's look at packets!

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

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								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								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

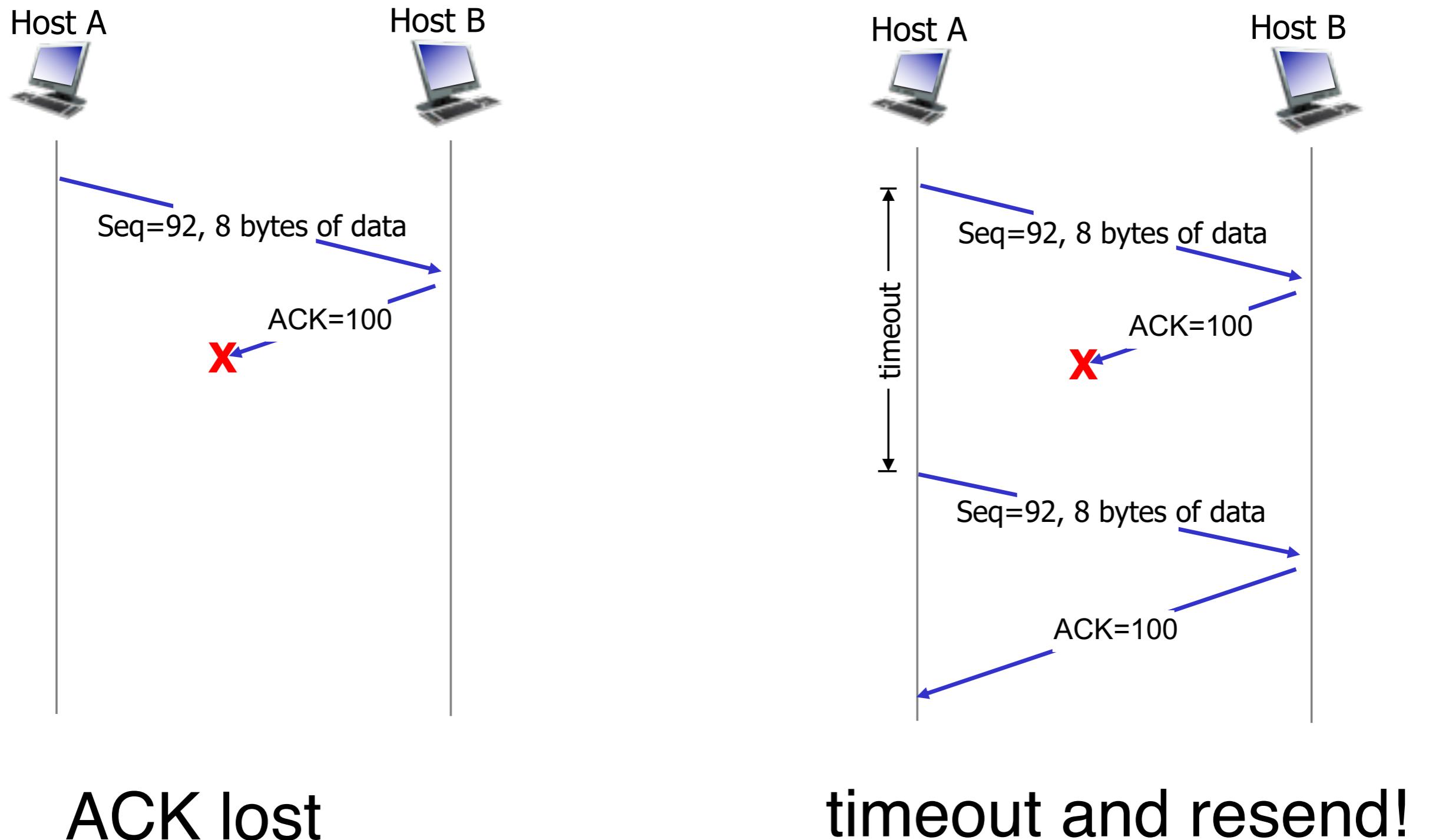
- Client/server start connection with a random sequence number
- On every send, add the total amount of data transmitted
- On receive, reply with ACK specifying last seq number received

What to do...

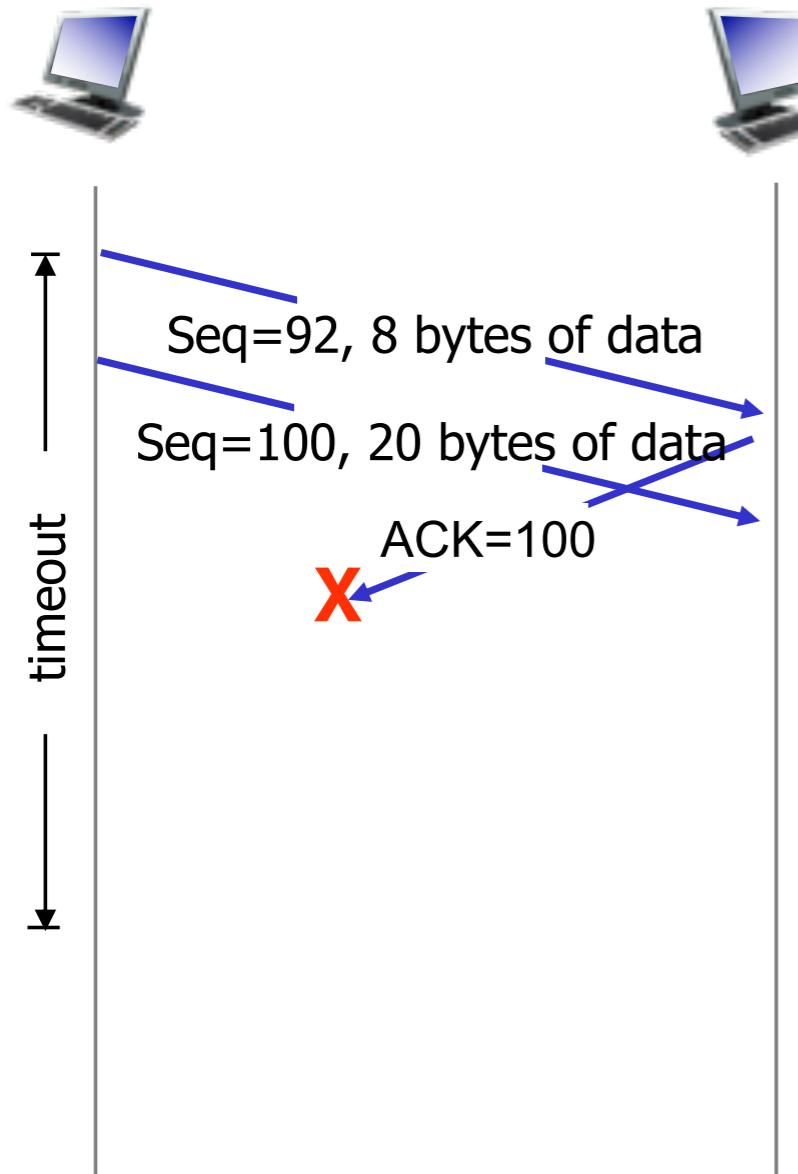
- If no ACK received?
- If receiver

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									

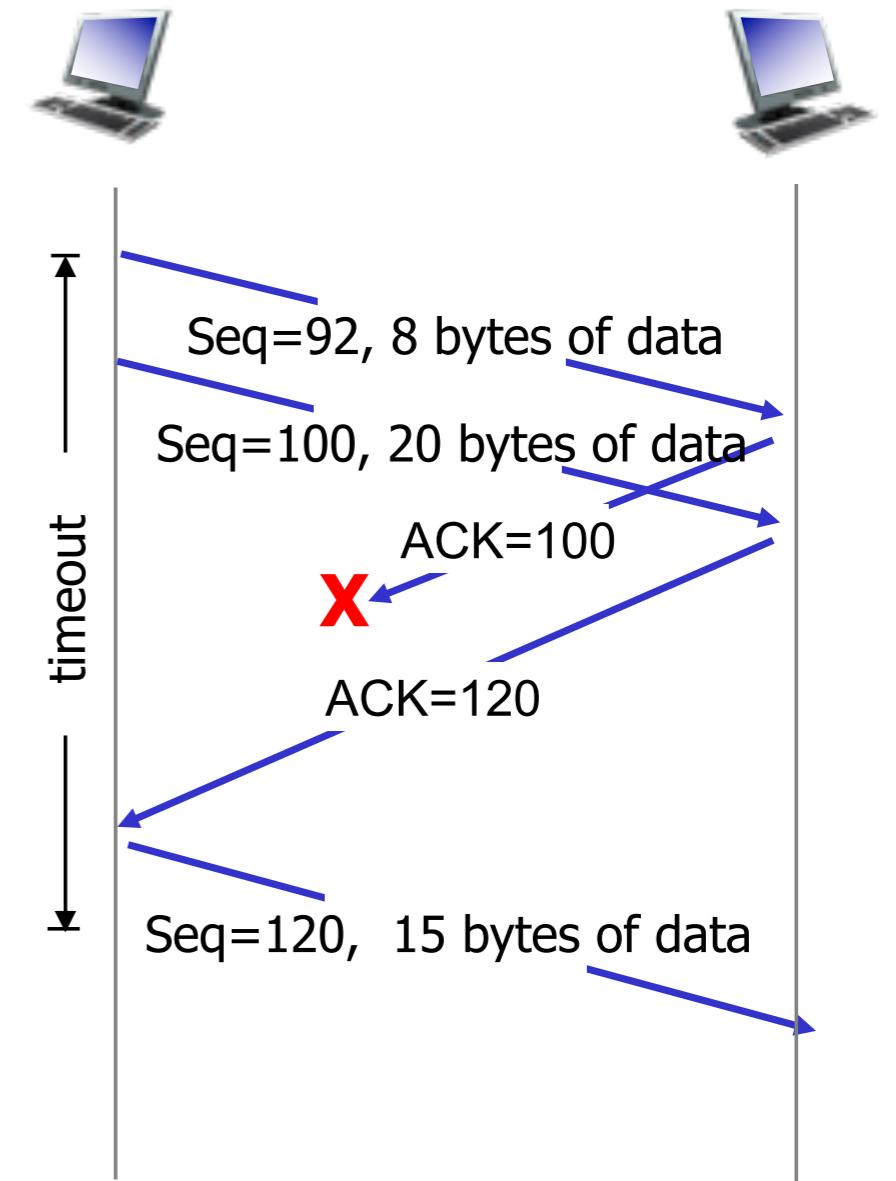
What happens?



What happens?



1st ACK lost



2nd ACK is
cumulative!

TCP Reliability

<i>event at receiver</i>	<i>TCP receiver action</i>
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 <i>duplicate ACK</i> , 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

Project 1

Reliable UDP File Sender