CPSC 3600 Notes  
FALL SEMESTER 2021

INSTRUCTOR: DR. ANDREW ROBB

horizontal line

# 19 August, 2021

Network Programming

* The goal of networking is to get data to its destination as accurately, quickly, and securely as possible.
  + How does data know what path it needs to take?
  + What errors can happen?
  + What determines the speed of data in a network?
  + How can data be protected from manipulation by bad actors?

# 23 August, 2021

Chapter 1 Reading Notes (Homework)

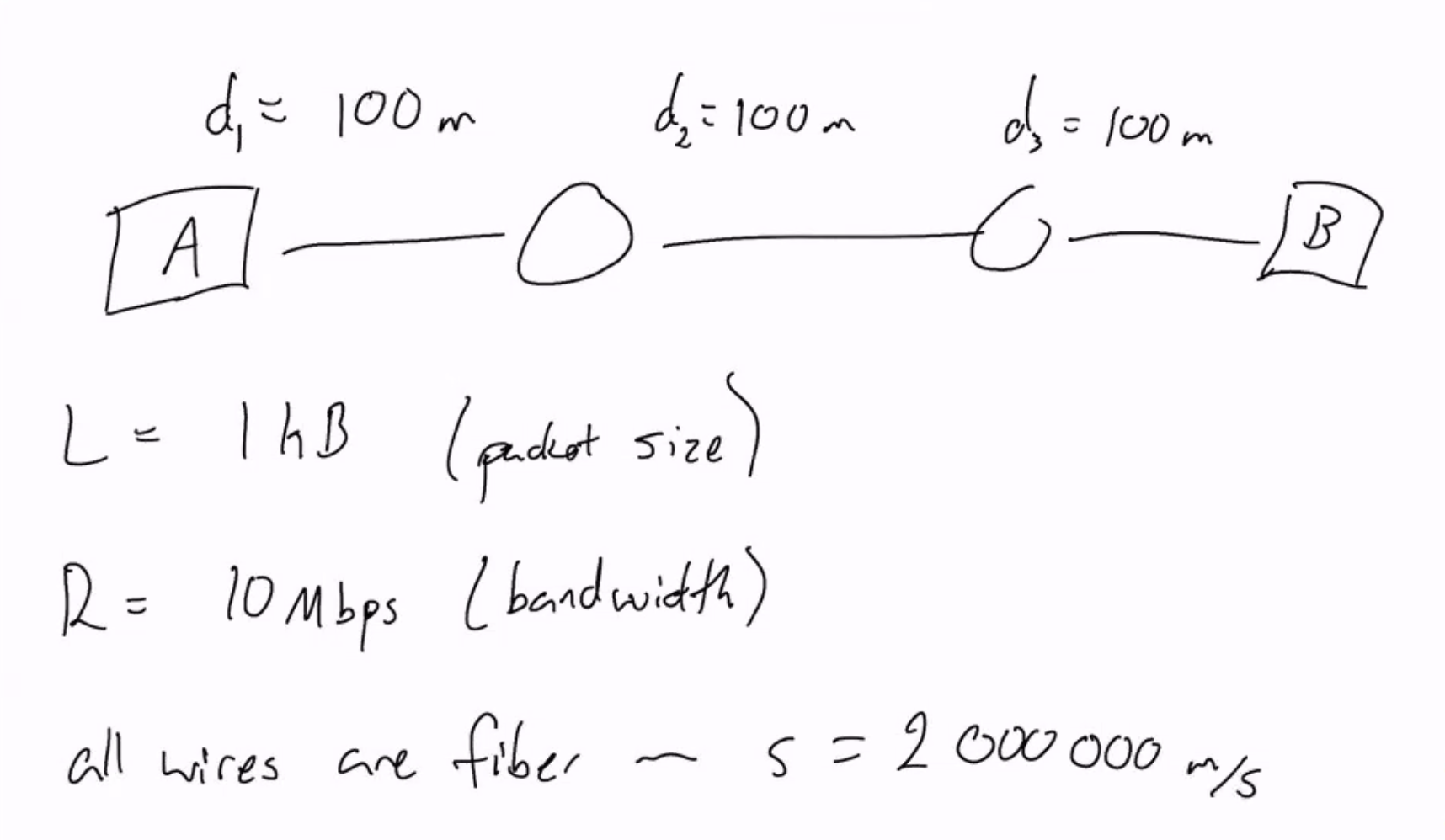
* LANs are local area networks. These are physical networks that provide the connection between machines within a place.
* IP is the internet protocol that provides an abstraction for connecting multiple LANS into the internet.
* TCP deals with the transport and connections and sending users the data.
* Four-layer model
  + A layer in this context, corresponds strongly to the idea of a programing interface or library with the understanding that a given layer communicates only with the two layers immediately above and below it.
  + An app hands data to the TCP, then the TCP makes calls to the IP library which calls the lan layer for delivery.
  + An app does not directly interact with the IP or LAN.
  + LAN is in charge of actual delivery of packets using lAN-layer-supplied addresses.
    - This is divided into the ‘physical layer’ which deals with analog electrical, optical, or radio signalling mechanisms.
    - And then a ‘logical layer’ which describes all the digital (non analog) operations.
  + The LAN diversion gives us the internet five-layer-model.
* Data rate: rate at which bits are transmitted.
* Throughput refers to the overall effective transmission rate taking transmission overhead, protocols, and traffic into account.
* Bandwidth can be used to refer to either of those above, usually used as a synonym for data rate. (term comes from radio transmissions)
* Goodput is sometimes used to refer to the ‘application-layer throughput’, the amount of usable data delivered to the receiving application.
  + Might be counted twice under some interpretations of ‘throughput’.
* Data rates are measured in kilobits per second (kbps) or megabits per second (mbps).
  + A kilobit is 10^3 bits
  + A megabit is 10^6 bits.
* Packets (frames at lan layer, segments at transport layer) are modest-sized buffers of data transmitted as a unit through some shared set of links.
  + Need to be prefixed with a header containing delivery information.
  + In the common case of datagram forwarding, the header contains a destination address.
  + Headers in networks using virtual-circuit forwarding contain instead an identifier for the connection.
  + Almost all networking today is packet-based and has been for the past 50 years.
* Ethernet allows a maximum of 1500 bytes of data.
  + Ethernet headers are typically 14 bytes, IP headers 20 bytes, and TCP headers 20 bytes.
* Internal nodes of the network called routers or switches will try to ensure that the packet is delivered to the requested destination.
* Paul Baran in 1962 introduced packets.
  + Donald Davies in 1964 coined the term ‘packets’
* Packets are buffers built of 8-bit bytes.

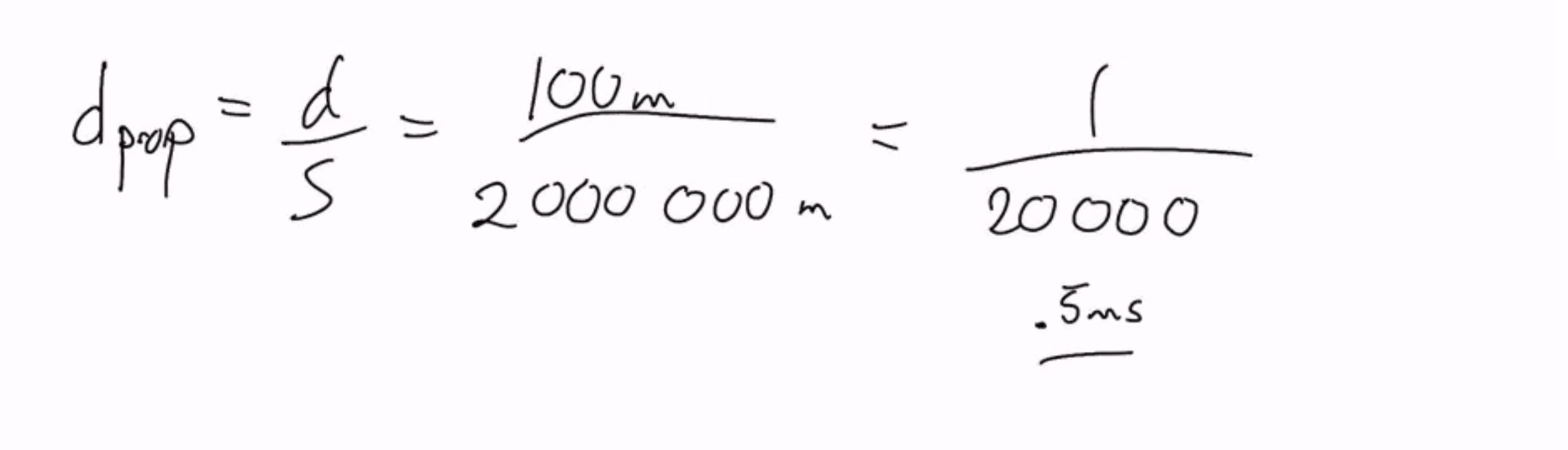
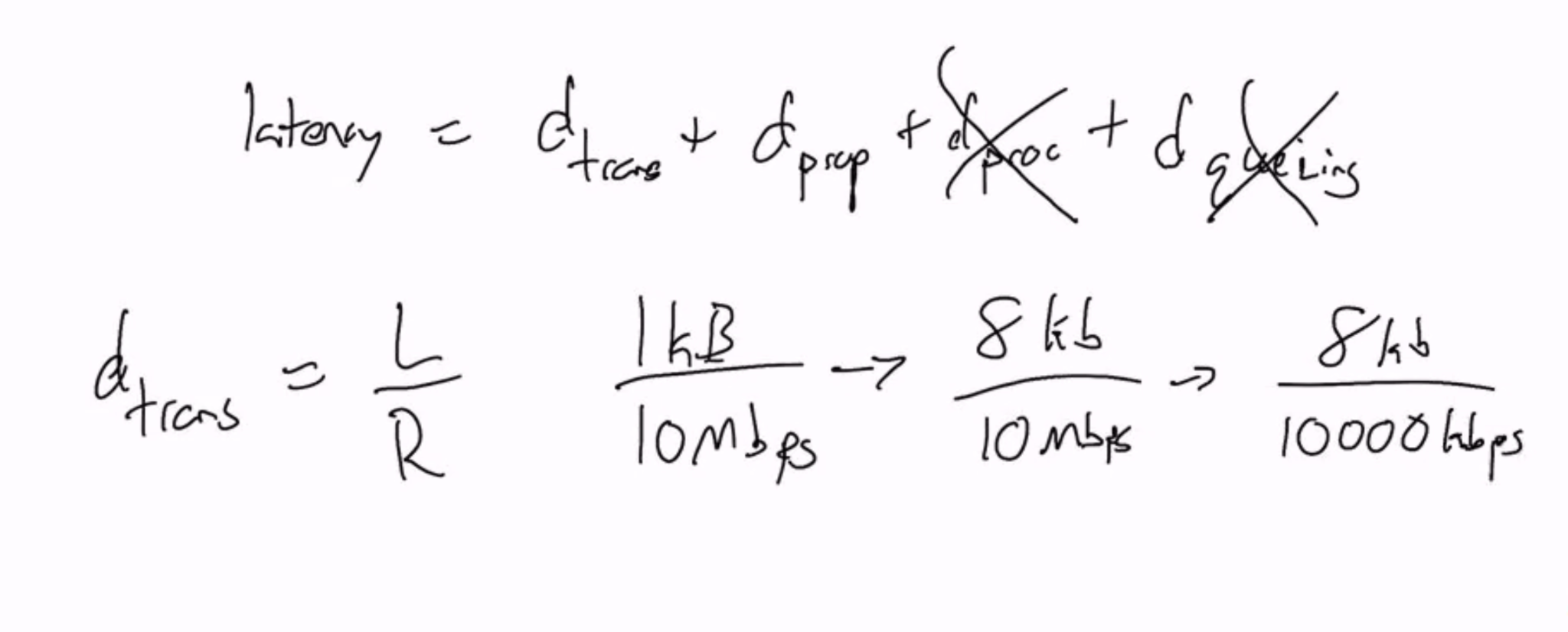
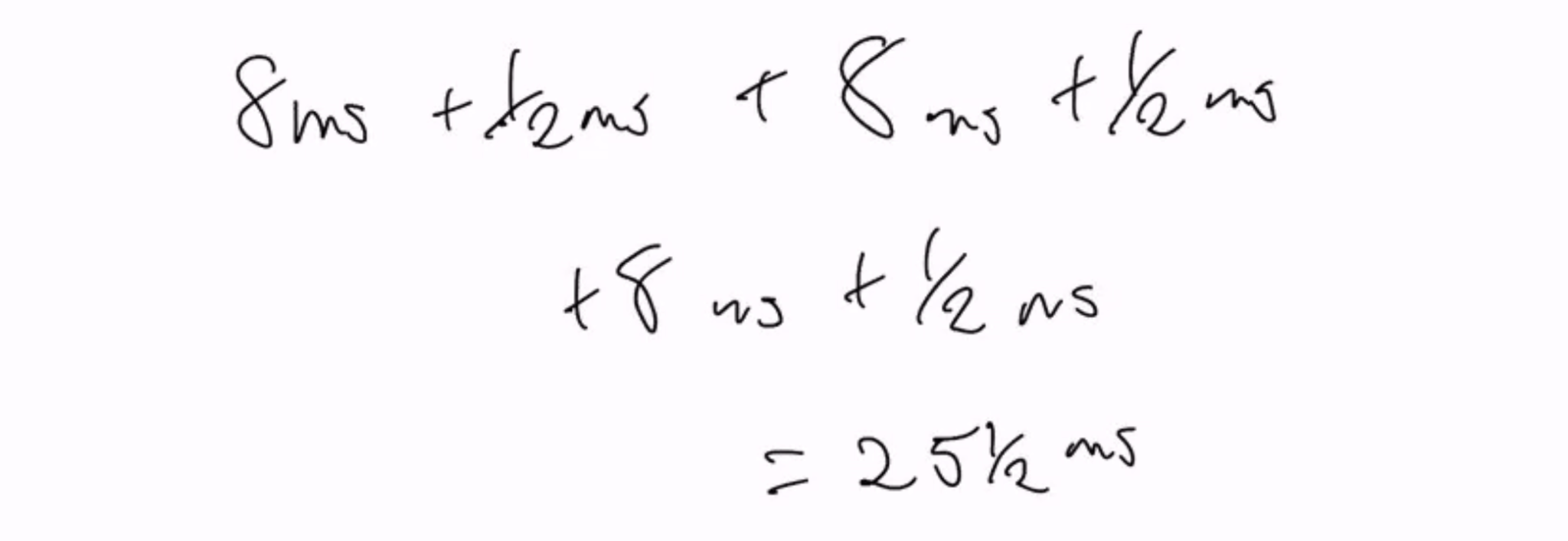
# 24 August, 2021

What is the internet?

* The internet is a specific network composed of hardware, software, and protocols.
  + The world wide web is a piece of software that runs on the internet.
* Other large-scale networks also exist (Global lambda integrated facility, internet2). These are typically dedicated to research or government facilities.
* The internet’s physical infrastructure comprises:
  + End systems (user computers, servers, cell phones, etc)
  + Communication infrastructure (cables, wifi, radio links, etc)
  + Switching and routing hardware.
* Some hardware can be devoted explicitly to the internet (backbone routers), others may be general purpose (like a laptop)
* Some hardware is nested. The internet is a network of networks. Parts of the internet are discrete units capable of internal communication that is distinct from communication with the wider internet.
* Software provides services to applications
  + Low-level software: transmits data along communication links.
  + High-level software: answer queries for information required to determine the route a chunk of data should take/
  + Applications: used by people to transmit/consume information.
* Examples of software includes:
  + Implementations of various protocols(TCP, SSH, BitTorrent, etc)
  + Network drivers that control impulses along communication links.
* Internet protocols are agreed upon standards that define how messages should be formatted, processed, and sent.
  + The format of bytes/information in the message
  + The order messages should be sent and received in
  + The actions taken upon transmitting or receiving a message
* Protocols on the internet are defined by the IETF (Internet Engineering Task Force)
* The internet is a particular network composed of specific:
  + Hardware
  + Software
  + Protocols
* The world wide web is an application that runs on the internet.
  + Analogous to microsoft word running on a pc
* We can divide the major internet hardware into three groups
  + Hosts/end systems
  + Communication links
  + Routers and switches
* Hosts and end systems are the places people care about going to:
  + Houses...malls..parks...schools….etc
* Communication links are the roads people drive on:
  + Highway, two-lane roads, etc
* Routers and switches are the infrastructure facilitating the flow of traffic:
  + Intersections, traffic lights, bypasses, crosswalks
* Data Moving through the internet:
  + Cars, bikes, semi-trucks, etc
* A second grouping is also helpful: the edge and the core
  + The edge of the internet is where hosts and end systems live
  + The core is where communication infrastructure and routers live
* Any system that a typical user will directly interact with lives on the edge
  + This includes client machines, web servers, loT devices
  + This is where applications and data live
* The core transports data between edge nodes. The hardware and software in the core is largely invisible to typical users
* The internet’s core consists largely of routers and communication infrastructure.
* Routers live at the intersection of multiple different lines of communication. Their job is to determine which connected line of communication data should be sent on to reach its destination.
* Edge devices connect into the core by communicating with an edge router.
* Edge routers aren’t particularly special, except that they are the first router in the core that data passes through.
* Edge devices connect to a core router via the ISP (internet service provider)
* Many different media are used as communication infrastructure in the internet. Common types include:
  + Twisted-pair copper wire
    - Least expensive, most commonly used medium
    - Two insulated copper wires twisted together.
      * Twisting reduces electrical interference from other nearby pairs
    - Many pairs can be twisted into a single cable
    - Data rates range from 10 mbps to 10 gbps
      * Length of cable negatively affects data rate
      * Cables can be 100+ feet long
    - Ethernet cables are usually twisted pair copper wire
  + Coaxial cables
    - Two copper wires, but not twisted, instead one is inside of the other, separated by insulation
    - Commonly used for cable connections
    - Carries multiple signals simultaneously on different frequencies.
  + Fiber optic cables
    - Glass fiber carries light pulses, each one is a bit of data
      * Light kept inside of the cable using total internal reflection
    - Supports high-speed transmission (10 gbps to 10+ tbps)
      * Information travels at speed of light in glass
    - Low error rates
      * Immune to electromagnetic noise
      * Very low signal attenuation (up to 100km)
    - High cost (including infrastructure)
      * Transmitters, switches, receivers..
      * Have to convert between light and electricity
  + Radio links
    - Unguided medium, signal carried in electromagnetic spectrum
    - Potential issues include reflection, interference, obstruction
    - Several different options
      * Terrestrial microwave (45 mbps channel)
      * Wifi (11 mbps to 1.3+ gbps)
      * Wide-area cellular(10+ gbps)
      * Satellite (kbps to 45 mbps channel, high latency)
        + Geostationary vs low orbit
    - Transmission speed affected by the radio frequency used
      * Higher frequency = faster transmission rate
* The internet is a network of networks ……
  + End systems connect to internet via an access ISP
  + Access ISPs have to be interconnected to enable communication
  + These clusters of networks have to talk to each other
* Current global structure of the internet evolved organically over time, driven largely by economics and national policies.
* The internet is a very complex system that is run by many different people and operates on many different devices
* Just as the hardware of the internet is organized in a hierarchical fashion, which makes it easier to manage, the core software of the internet also uses a layered architecture
* Layers are black boxes with exposed interfaces
  + Each layer hands off data to the next layer and doesn't care how the next layer is implemented.
  + Layers only communicate with layers directly above and below them
  + The Internet Five Layer Model
    - Application Layer
      * Where data is generated and pushed into the network
      * Almost any networked application we use resides here
      * When an application wants to send data into the network, it passes data to the transport layer, which begins the process of moving it through the internet.
    - Transport Layer
      * Implements TCP (Transmission Control Protocol) and UDP (User Datagram Protocol)
      * These are responsible for getting the data from source to destination.
        + Allow data to be delivered to a specific application, not just a machine
      * TCP also provides many guarantees. UDP provides almost none.
      * TCP has more overhead than UDP
      * Addresses receiving applications using a port number.
    - Network Layer
      * Internet implements two major network layer protocols
        + Internet protocol v4 (IPv4)
        + Internet Protocol v6 (IPv6)
      * These protocols are responsible for routing data through the various routers and machines residing between the source and destination.
      * Addresses receiving device using an IP address
    - Link/Logical LAN Layer
      * The internet implements one major protocol at the link layer, ethernet
      * Ethernet is responsible for moving data between adjacent routers or machines
      * The Link and network layers are separates to support scalability
      * Addresses receiving device using a MAC address.
    - Physical/Physical LAN Layer
      * Many different protocols are supported at the physical LAN layer
      * The physical LAN is responsible for converting digital data into a form that can be transmitted across communication infrastructure.
  + Each layer is responsible for a different part of the data transmission process.
* When the transport layer receives data from an application, it divides the data into a series of packets
* Each packet contains headers and a payload
  + Headers contain information about getting packet to destination
  + Payload contains a chunk of data
* The packets are injected into the network, and forwarded to their destination by the core routers.
* There are two main ways to support ‘simultaneous’ users on a shared network
  + Frequency domain multiplexing
  + Time domain multiplexing

# 26 August, 2021

* Latency, Throughput, and Errors
* Latency
  + Four sources of latency in the network
    - Transmission/Bandwidth delay
    - Propagation Delay
    - Processing/Store-and-forward delay
    - Queuing delay
  + Total latency equals the sum of all of these components
    - Total = trans+prop+process+queue
* Sources of Delay
  + Transmission delay
    - The time required to transmit a packet onto the ‘wire’
      * Packet size measured in bytes
      * Transmission rate measured in bits/second
      * Trans = packet length/transmission rate of link = L/R
  + Propagation delay
    - The time required for the data to travel along to its destination.
      * Data moves at different speeds in different mediums.
      * Prop = distance/data speed on ‘wire’ = d/s(prop)
    - Can be much larger in other supported medium
      * Request for comment (RFC) 1149 defines the IPoAC protocol
    - IP over Avian Carriers
      * In 2001 the Bergen Linux group transmitted 9 packets over a distance of 5km.
  + Processing delay
    - The time for the receiving device to read and process the packet.
      * Delay is usually negligible, typically microseconds
      * Comes from checking for bbit level errors, routing the packet and storing the packet for forwarding.
      * Process can’t be solved using a simple equation.
    - A router has to wait until it has received the whole packet before it can begin forwarding it.
      * Routers need a dedicated buffer to store packets in before forwarding them.
      * Introduces transmission latency because each router has to retransmit the packet
  + Queuing delay
    - The time the packet spends waiting to be transmitted because the outgoing link is busy.
      * Queue is variable and depends on how much other traffic is present on the network.
      * Queuing delay = R/L
        + R = link bandwidth (bits/sec)
        + L = packet length (bits)
      * a = average packet arrival rate
      * La/R ~0, avg queuing delay is small
      * La/R~ 1, avg queuing delay is large
      * La/R > 1 avg queuing delay is infinite
    - Infinite Delay
      * It approaches infinity over time, not immediately upon entering this state.
      * Delay approaches infinity because the receiving buffer grows infinitely.
        + Because of store-and-forward, routers have to buffer incoming packets
        + When La/R > 1, the buffer grows faster than it can be emptied.
        + This results in dropped packets. If a packet is received when the buffer is full, the router has to throw it away.
* Terminology
  + Mbps vs MBps
    - Lowercase b =bits
    - Uppercase B = bytes
    - 8Mbps = 1MBps
    - 1Kb = 1000 bits
    - 1 Mb = 1000 Kb
    - 1 Gb = 1000 Mb
* Example (Fiber optics) (Transmission + Process)
  + L = 1 kB (packet size)
  + R = 10 Mbps (bandwidth
  + S Prop = 2,000,000 m/s
  + Latency = d trans + d prop (for now)
  + d trans = L/D = 1KB/ 10Mbps = 8 kb/ 10 Mbps = 8kb/ 10000 kbps
    - d trans = 0.0008 sec
  + d prop = d/s = 100m/ 2000000m = 1/20000 = 0.5ms
  + 8ms + 1/2ms + 8ms + 1/2ms + 8ms + ½ ms = 25 ½ ms

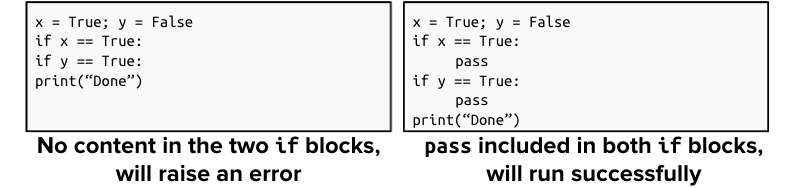
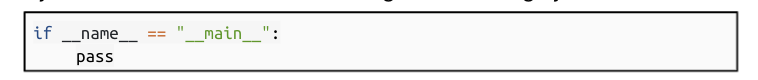
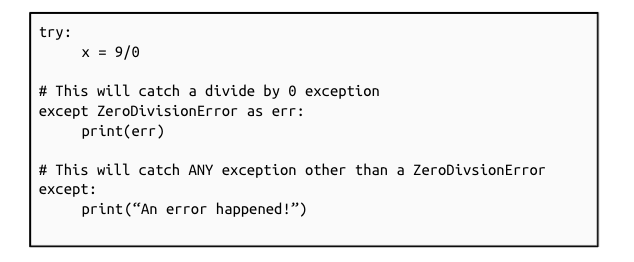
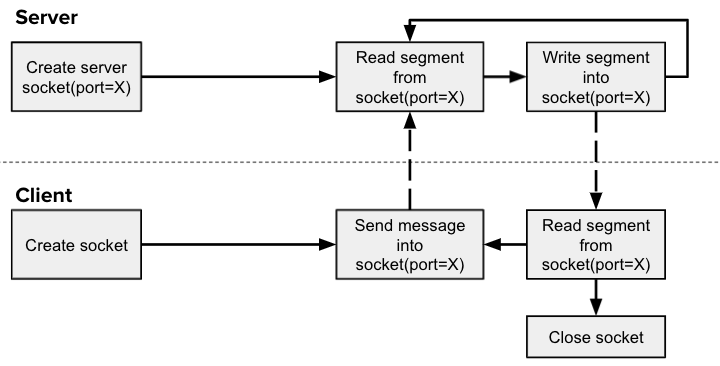
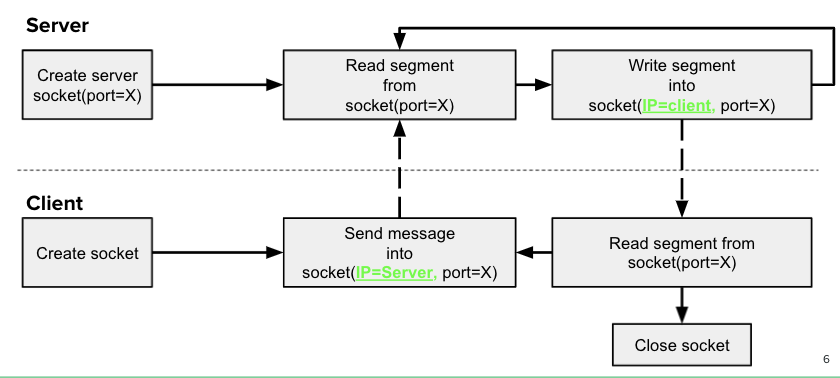
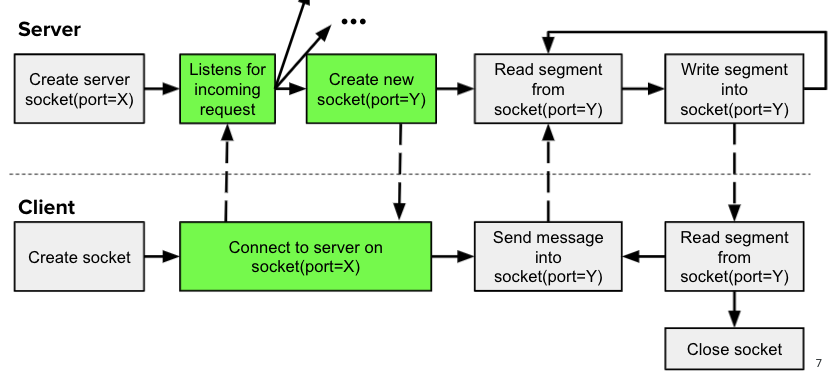


* Throughput
  + Bandwidth is the amount of data that can be transmitted per second
  + Throughput is the amount of data that is transmitted per second.
    - Can be measured instantaneously, or on average.
  + Goodput is the amount of USEFUL data transmitted per second.
  + Think about it like water flowing through pipes
  + Bottlenecks
    - The bottleneck link is the link on the end-to-end path that constrains throughput.
* Sources of Errors
  + Corrupted packets
    - Packets may be corrupted in flight
    - Usually not possible to tell which bits were flipped, we have to throw away the packet
  + Dropped packets
    - Packets queues have finite capacity
    - Possible to have a packet arrive but not have space for it (then packet is dropped)
    - Lost packets may be retransmitted from the previous node, by the origin node, or not at all.
  + Misordered packets
    - Applications want data sent, not packets
    - Network system breaks data into packets for efficiency
    - Packets need to be received in the same order they are sent, or data cannot be reconstructed correctly
    - If packet is received out of order, the system might not know what to do with it and it will be dropped
  + Collisions
    - A collision occurs when two or more hosts attempt to transmit on the same media at the same time.
    - When a collision happens, both packets are dropped and a mechanism is triggered to retransmit the packets, hopefully without a collision
    - Impossible on non-shared media

# 31 August, 2021(Python Slides)

* If you are using every bit, you will be looking at very large queuing delays.

Python!

* Python is portable!
* Python is simple!
* Python provides straightforward access to low level socket API.
  + Networking concepts in Python easily transfer to almost all other languages.
* Two major versions of python are still used, v2 and v3
* Python3 is what we are using.
* No compilation is required (an interpreted language)
* NO SEMICOLONS
* Python scripts are executed directly via the command line.
  + This assumes your path variable includes the python install directory.
  + C:\> python my\_script.py
* Variables do not need to be declared in advance. Just assign it a value
* Variables are not statically typed
  + You can assign a value of any data type to any variable
* You don’t need to create a “new” variable that is the same name as an old one, you will just overwrite the old one.
* Strings
  + Can use either ‘ ’ or “ “ to define a string
  + Can be concatenated as string1 + string2
  + Non strings must be converted to strings before concatenation using str() function
* Lists and Arrays
  + Similar data types
  + Both store data, are mutable, can be iterated through
  + Lists are built in data type, arrays must be imported
  + Lists can store objects of different types, arrays can’t
  + Arrays are optimized for arithmetic operations, lists don’t support them
* Dictionaries
  + Store key-value pairs
    - Sometimes called hashtables or hashmaps
  + Key must be unique, value can be anything
  + A dictionary can store keys of different data types
* Operations
  + Standard arithmetic operations
    - / float division
    - //integer division
    - \*\* exponentiation
  + Logical Operators
    - True, False
    - Notl, or, and
  + Whitespace
    - Each block nested inside of another block must be indented once more than its parent
    - Be consistent!
    - Python doesn’t use {}, only indents
  + Pass
    - Any statement that creates a new block MUST contain something
      * If, for, while, functions…
      * Sometimes you want to stub out a loop or a function, but aren’t ready to create content yet
      * Put the pass statement in block
  + Loops
    - For loops in python function like foreach loops in C++ or Java
    - Rather than iteratively changing an index value, you iteratively examine new elements of a list
    - If you need to loop through a list of numbers, use a range to create it
      * Creates a list of all number within a given range
    - If you need both the index and value of each element in a list, you can do that using the enumerate function
  + Functions
    - Specified using the def keyword
  + Main Method
    - Python script files can be run immediately in the console. They do not necessarily need a ‘main method’
    - Python defines main methods using the following syntax:
    - When you run a python script, it will look to see if a main method is present, if so it will begin execution there. Otherwise it will run the file from top to bottom.
* Object Oriented Programming
  + Classes are defined like class FootballPlayer: pass
  + Code defined in a class can reference the current instance of the class using the self keyword. Self is similar to this from c++ and Java
  + The self argument must be the first argument in all class functions
    - You do NOT pass a value for self in when calling the function, Python handles this automatically!
  + To define an instance-level variable within a function, preface it with self
    - If you don’t call self, the variable will be defined at the function-level
    - If you try and call an instance-level variable without self, it won’t work.
  + Create a constructor using the \_init\_ function
    - By convention, if you know you need to create a variable using another function, it is good to use \_init\_()
* Exceptions
  + Raised by programs when something goes wrong.
    - Dividing by 0 will raise the ZeroDivisionError
  + You will encounter exceptions as you code
    - Sometimes this is a sign something is wrong
    - Sometimes you just want to execute different code after the exception.
  + If not caught, a raised exception will terminate the program.
  + Catch exceptions using try...except blocks
  + You can raise your own exceptions using the raise command.
* Modules
  + Python automatically packages code into modules
    - All code in a single file is automatically a module
    - Sets of files can also be packages into a module
  + Code in a different module must import other modules to use them
* UDP and TCP Sockets
  + Two major types of sockets
  + Two places sockets are used
    - Client and Server
  + Each of these have different usage requirements.
  + General Process:
  + UDP Sockets:
  + TCP Sockets:
  + Setting up Sockets
    - You must import the sockets module to be able to use either UDP or TCP sockets
      * from socket import \*
  + Setting up UDP sockets
    - UDP Client
      * udpClient = socket(AF\_INET, SOCK\_DGRAM)
    - UDP Server
      * udpServer = socket(AF\_INET, SOCK\_DGRAM)
      * udpSever.bind((‘ ‘ , serverPort))
  + Setting up TCP Sockets
    - TCP Client
      * tcpClient = socket(AF\_INET, SOCK\_STREAM)
      * tcpClient.connect((serverIP, serverPort))
    - TCP Server
      * tcp\_Server = socket(AF\_INET, SOCK\_STREAM)
      * tcpServer.bind((‘ ‘, serverPort))
  + Sending and Receiving with UDP Sockets
    - UDP Client
      * udpClient.sendto(message.encode(), (serverName, serverPort))
      * newMessage, serverAddr = udpClient.recvfrom(2048)
    - UDP Server
      * message, clientAddress = udpServer.recvfrom(2048)
      * udpServer.sendto(newMessage, client address)
  + Sending and receiving with TCP Sockets
    - TCP Client
      * tcpClient.send(message.encode())
      * newMessage = tcpClient.recv(2048)
    - TCP Server
      * tcpServer.listen(1)
      * connectionSocket, addr = tcpServer.accept()
      * message = connectionSocket.recv(2048)
      * connectionSocket.send(newMessage)
  + Cleaning up Sockets
    - Once you are done with a socket call socket.close() to free resources
      * tcpClient.close()

# 2 September, 2021

* Encoding/Decoding Data
  + Socket.send and .rec() expect to receive a byte array
    - You need to convert your data into an acceptable format.
  + You have two options, appropriate for different messages
  + If all you need to do it send a single string, use encode() to convert the string to a byte array
    - decode() will convert the byte array back to a string
  + Both encode and decode will accept an encoding type and turn it into a uft-8
* Pack/Unpack multiple data values
  + Use pack() to convert a tuple of variables into a single byte array
  + You can convert it back into the same tuple using unpack()
  + These are Python features, not socket features
  + Pack() accepts a packing format and a set of objects as its arguments, and returns a byte array
  + Unpack() accepts a packing format and byte array as its arguments, and returns a tuple of objects (a tuple is similar to a list)
  + FOR ANY FORMAT STRING, IT SHOULD START WITH AN EXCLAMATION MARK! (python convention)
  + You must import pack and unpack from the struct package
* Packing Strings
  + If you want to include a string in your packed object, use the format string ‘s’, prefixed with a # indicating the length of the string in chars.
* Streaming and buffering data
  + All data comes in as bytes
  + Pack oconverts collections as variables into a byte array
  + We need to concatenate the received data to a buffer containing previously received portions of the message.
    - Bytes objects can be concatenated using the + operator
  + Once the whole message has been received, we can decode it or unpack it
  + Simple option: use a fixed message size. This is appropriate when working with messages that don't vary in length much.
  + Second option: include message length in header
  + Full protocols usually include many values in a header.
* LIVE DEMO/EXAMPLE:

//client

from socket import \*

SERVER\_NAME = “cirrus4.computing.clemson.edu

SERVER\_PORT = 3604 //encourages to pick bigger than 1024

my\_sock = socket(AF\_INET, SOCK\_STREAM)

my\_sock.connect((SERVER\_NAME, SERVER\_PORT))

message = input(“>>>”) //prompt to get user to enter something

my\_sock.send(message.encode())

byte\_array = new\_sock.recv(1024)

response = byte\_array.decode()

print(response)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//server

from socket import \*

INCOMING\_HOSTS = ‘ ‘

INCOMING\_PORT = 3604

server\_sock = socket(AF\_INET, SOCK\_STREAM)

server\_sock.bind((INCOMING\_HOSTS, INCOMING\_PORT))

server\_sock.listen(5)

new\_sock, client\_addr = server\_sock.accept()

byte\_array = new\_sock.recv(1024) //makes it so that no other code after this can be computed until this step it done

message = byte\_array.decode()

print(message)

message = message.upper()

new\_sock.send(message.encode())

# 7 September, 2021

* Internet Applications and Hypertext Transfer Protocol
* Network Application Architectures
  + Two main system architectures
    - Client server model
      * Clients request information from the server, server provides information to the clients
      * Clients don't talk to each other
      * Very distinct roles
        + Clients don't communicate to servers
        + Servers don't initiate with clients.
      * Servers must always be on or the application won't work
      * Need a fixed IP address so clients can contact them
      * Bigger applications need better servers
        + Often need giant cluster of server to handle all of the clients
        + Also needs high throughput links to handle larger amount of traffic
      * Clients only on when a user needs them
      * Ip address doesn;t matter because server doesn’t initiate contact with them
      * Rarely needs to be as powerful as a server.
      * Follows the request response messaging pattern.
        + Client submits request in agreed upon format
    - Peer-to-Peer (P2P) Model
      * No central server
      * Focuses on communication between peers
      * Bittorrent, bitcoin, skype, etc
      * Benefits
        + Self scalable, as more users come online, computing power automatically increases
        + Cost effective
      * Potential problems
        + Security, privacy becomes complicated
        + Often requires much more complicated models
        + Potential issues with performance and reliability.
* The Web and HTTP
  + Web is an application
  + Web’s application protocol is HTTP
  + Browsers (clients) exchange HTTP messages with servers to receive web data
  + Web pages consist of many objects/files
  + HTTP Protocol
    - Defines how clients communicate with servers in order to request control
    - Communicates over TCP sockets
    - General Process
      * Client initiates TCP connect to server, using port 80
      * Server accepts TCP connection from client
      * Exchange HTTP messages between client and server
      * Close tcp connection
    - HTTP is stateless
    - HTTP Request message
      * Client requests some kind of data or protocol
* Method Types (request messages)
  + GET: most common, requests content, no entity body
  + POST: requests content and sends content in body, used online forums
  + HEAD: like GET, but only wants the header, for debugging
  + PUT: upload a file to a specific location on server
  + DELETE: delete files on the server specified by url
* HTTP Response Message
  + Like HTTP requests, has message types, responses have status codes
  + Examples include: 404 not found, 400 bad request, 301 moved permanently, 200 ok.
* Cookies
  + Four components to cookies
    - Cookie header line in an HTTP response message
      * Set cookie
    - Cookie header line in an HTTP request message
    - Cookie file stored on the user’s host, managed by browser.
    - Database on the web server linked to cookie information.
* Web Caching
  + Store previously requested data close to end users’ machines
  + 1.Browser sends request to cache
  + 2.cache checks if it has requested file
  + 3.Returns it if it does, otherwise sends request to origin server
  + Both client and server
  + Caches usually installed by ISP
  + Why do we cache?
    - Reduces response time for client’s requests
    - Reduces traffic on institutions access link
    - Reduce overall traffic across the internet as a whole
* What is LAN utilization?
  + We know how many requests, how big, the total bandwidth

# 9 September, 2021(DNS)

* If changing to a 1 gigabit internet, it will be very expensive.
* Cache hit rate is if there are so many requests made, how many of those are fulfilled!
* DNS: Domain Name System
  + Websites have two types of identifiers]
    - Hostnames/URLS
    - IP Address
  + Equivalent to people
    - Names, SSN, passport #, etc
  + Internet needs a system that maps URLs to IP addresses
    - Routers need to use IP addresses
    - Easy to remember google.com, hard to remember IP address
  + DNS Process
    - 1. User inputs hostname.url
    - 2.Client host queries a dns client about hostname
    - 3.DNS application sends a query to a DNS server
    - 4.DNS client eventually receives a reply including IP address
    - 5.Returns this to the host application that originated request.
  + DNS requests can add considerable delay, dns server ,ay have to query other servers to find the correct ip address
  + DNS servers will cache new IP addresses to shorten future requests and reduce internet load
  + Host Aliasing
    - Host names can have aliases
  + Mail server aliasing
    - Email servers can also have aliases
  + Load distribution
    - Large sites are hosted on multiple servers, each with unique IPs
    - DNS servers rotate through the IP address given back to the clients to help balance load across the servers
  + Distributed Database
    - Many different DNS servers scattered across the world, each hosting portions of the distributed database
      * Removes single point of failure
      * Handles large amounts of traffic volume ”Close” to all clients
      * Easier to maintain individual databases.
  + Three classes of DNS servers, ordered in a hierarchy
    - Root DNS servers
      * If a local DNS server doesn’t know a mapping, it contacts a root DNS server
        + 13 DIfferent root servers, each replicated many times
      * Root name server
        + Returns name mapping if known
        + If not, returns address of an appropriate TLD server to request
    - Top-level domain DNS servers
      * Manager “top-level” domains
      * Different companies manager different TLDs
      * Ideally TLS server sends client address of authoritative DNS server than knows the actual IP address
      * May be other levels in the hierarchy though.
    - Authoritative servers
      * Where the mapping between hostnames and the IP addresses are stored
      * Organizations run their own authoritative DNS servers
      * Alternatively, you can register your own hostname with an authoritative DNS server
      * Once the request gets to this level, the client will get the IP address it is looking for.
* DNS query count
  + DNS queries sent over UDP
  + MUltiple queries required, can be:
    - Iterative, Recursive
  + Same time requirement
* DNS caching
  + Can cache hostname-IP mappings to speed up the request process
  + TYpically caches for 2 days, then released
  + Can also cache connections to TLD servers, bypassing root DNS servers
  + Caches can be out of date, domain names remapped to different IP, won’t be known internet wide till TTLs expire
* DNS Records
  + Each entry in a DNS database follows the same pattern
  + TTL is the time to live in the cache
  + Type determines what name and value mean
    - A, authoritative
    - NS,domain Name
    - CNAME, alias name
    - MX, email hostname
* DNS messages not human readable
  + DNS messages have 12 bytes for header section
    - ID number, allows clients to match response to requests
    - Multiple one-bit flags
    - Length of different payloads also listed there.
* Nslookup
  + Lets you send a DNS query message

# 14 September, 2021(RDT)

* Reliable Data Transfer Principles
  + The internets underlying infrastructure is unreliable
  + We need to impose reliability on this channel through protocols.
* Acknowledgement Messages
  + When a recipient receives a packet, it sends an ACK to the sender
* Retransmission
  + Sometimes Packets are retransmitted
* Timers
  + Ongoing timers track how long it has been since packet was sent
* Sequence Numbers
  + Each message (including ACKs) is given a unique, ordered ID number
* Dropped Packets
  + Two possibilities
    - Original packet was dropped
      * So, retransmit the packet
    - The ACK was dropped
      * We will need to retransmit the packet and will do this when the timer expires.
* Corrupted Packets
  + We will retransmit the packet!
  + If the ACK was corrupted
    - Wait until the timer runs out them we retransmit the packet.
* Delayed Packets
  + Can cause problems
  + Handled using previous mechanisms
    - If there is duplicate data, receiver ignores it
    - If there is duplicate ACK, sender ignores it.
  + There are two methods of recovery
    - Retransmit on timeout (sender)
    - Retransmit on duplicate (receiver)
    - Bad things happen if both sides implement retransmit on duplicate
      * Following the first retransmission, all other packets are also duplicated.
* The sender always waits for the timeout to occur and then resend, then they restart the timer.
* Example:
  + Sender sending 2 pieces of data.
  + The receiver only has to send the ACK#
* The big thing the sender has to do if something was submitted wrong is to catch that something is missing or something went wrong.
* On an exam:
  + Interpret a finite state machine
  + Identify a problem with the finite state machine (what went wrong in given protocol)
  + Adding a piece to a finite state machine.
  + PRACTICE WITH FINITE STATE MACHINES!!!
* LOOK AT STUDY GUIDE

# 16 September, 2021(GBN)

* Pipelining:
  + Allows multiple packets to be sent before receiving an ACK for them
  + We’ll limit the number of unACKed packets that can be sent.
    - If too many are sent then they will get stuck in the queue.
* Go-Back-N (GBN
  + If the GBN protocol, the sender is allowed to send N packets without receiving an ACK
    - N is our window size; this is a ‘sliding window’
  + Once the sender has sent Pktn it must wait until it receives an ACK
    - Once it receives ACK, it can send Pktn+!
* Selective Repeat
  + Receiver individually ACK’s all correctly received packets
    - Buffers pckts for eventual in-order delivery to upper layer
  + Sender only resents pkts for which ACK not received
    - Requires a sender timer for each un’Ack’d pkt
  + No longer consecutive un’ACK’s window, now just keeps an un-ACK’s count
  + Individually ACKNOWLEDGED!!!!!!!!
* When to use selectors is when you have MORE THAN ONE SOCKET!!!!!!!!
* DO NOT USE SELECTORS IN THE PROJECT!!!!!

# 21 September, 2021(More GBN)

* Project 3: Go-Back-N
  + Implement GBNHost.py
  + Run RDTTester.py
  + In receive\_from\_application\_layer
    - self.simulator.start\_timer(self.entity, self.timer\_interval)
* The rest is questions about the study guide
* Transmission: filesize\*8/bandwidth
* Propagation: distance/speed
* Processing: .1ms
* Queuing = incoming date range / bandwidth
* The internet is more heterogeneous because it has grown up more organically with many different mediums, radio links, protocols, etc

# 30 September, 2021 (TCP)

* TCP: Transmission Control Protocol
  + Reliable
  + Connection-oriented
  + Stream-oriented
  + It is responsible for delivering data to specific applications on specific machines
  + Multiplexing: takes data from a bunch of locations and sends them out through the same network link
  + Demultiplexing: receives a bunch of data from a network link and delivers it to each of the appropriate applications.
* TCP functions as a byte-stream
  + Applications give data to TCP, splits it into packets and then sends it out.
  + TCP receives packets, combines the data into a stream of bytes, and then delivers this to the appropriate application.

# 5 October, 2021 (More TCP)

* MORE TCP
* FB was taken down via a dns error
* TCP samples the rtt by computing the time between when a segment is sent and ACK’d.
  + Doesn’t calculate RTT for all segments, only ones running one at a time.
  + Don’t calculate sampleRTT for retransmitted segments
* COMPUTE THE AVERAGE OF THESE SAMPLE READINGS! Individual readings may not accurately represent RTT
* EstimatedRTT = (1-a) \* EstimatedRTT + a\*SampleRTT
  + a is recommended to be a = 1/8

Calculate TCP timeout with: estimatedRTT + (4 \* DevRTT)

Calculate estimated RTT with: (1-alpha)\*estimatedRTT + alpha\*sampleRTT

Question 1.2)

a = ⅛, B =¼

So, (1-(⅛) \* 10) + ((⅛) \* 8) = 9.75 or 39/4

(1-(⅛) \* 9.75) + ((⅛) \* 7) = 9.40625

(1-(⅛) \* 9.40625) + ((⅛) \* 9) = 9.35546875 so this would be rounded to 9.36

Question 1.3)

DevRTT = β \* | SampleRTT- EstimatedRTT|+(1- β)\* DevRTT

(¼) \* |8 - 10| + (1 - ¼) \* 1 = 5/4

(¼) \* |7 - 9.75| + (1 - ¼) \* 5/4 = 1.625 or 13/4

(¼) \* | 9 - 9.40625| + (1 - ¼) \* 13/4 = 2.5390625 which is 2.54