Lecture 1

* Routing
  + Paths
* Data Links – physical layer
  + Between 2 nodes (error detection, etc.)
* Narrow physical layer restriction:
  + Need to send data in small bits
* DNS – directory service -> name to ip address
* Network header
  + Routing info (all intermediary routers)
* Data link header
  + Only info about next router
* Physical layer
  + No header, splits DPDU (?) (frame) into bits
* At destination node, data link header is taken off, then the network header. Then, it sees final destination is this router and proceeds to take off other headers
* Data links – fiber, wireless, cable, Bluetooth, etc.
* TCP vs. UDP transport layer
  + UDP – no resending/retransmission
  + TCP – more reliable
* TCP abstraction
  + Sender and receiver have 2 connected queues and data reliably is sent from sender to receiver queue. No packets.
  + Uses data link to send packets between 2 routers, which sends bits at physical layer

Lecture 2

* TCP
  + Queue abstraction
* IP layer
  + IP header places source and destination
  + If IP addresses match, then we are at the right destination
  + Else, figure out the next router
* Data link
  + From mac address x to y
* Mac address
  + Unique 48 bit addresses uniquely identify everyone on a link
  + E.g., x->y z->l, y,z are mac addresses for the same router
* IP address
  + Temporarily assigned to device
* Physical
  + Bits
* RPC (remote procedure calls) vs TCP (Queue abstraction)
* Socket = queue
* Transmission
  + Bandwidth
    - Range of frequencies a particular signal is created from (infinite sum of which sine wave freqs)
  + Strict layering
    - Only read headers at current layer (e.g., IP only reads IP header)
    - Information can only be passed across interfaces
  + Physical layer
    - Frequency shifting vs amplitude shifting (baseband)
    - Bits to coded bits to add transitions
    - Send bits to modem to convert to energy
  + Channel
    - Physical medium to transmit from sender to receiver
    - A channel does not change frequency of sine waves (only maybe the amplitude and phase)
  + Any periodic function can be written as an infinite sum of sin waves with diff freqs
  + More harmonics/bandwidth -> better bit recovery
  + Harmonic = multiple of the fundamental frequency
  + Channels are sluggish (take time to respond to a change in the signal)
  + Channels have noise, resulting in a change in amplitude
  + Sample at middle points
  + Two limits
    - Bandwidth (Nyquist rate) – limit on symbols (signals)
      * Input: square wave, output: falling sine wave
        + When can be send the second signal such that it does not interfere with previous signal?

Bandwidth of input signal is 1/T

You can theoretically send signals every T/2 seconds (Send at frequency: 2 \* bandwidth = 2/T)

Max rate of sending symbols, not bits (baud rate)

* + - * Why not send multiple bits per symbol (6v – 11, 4v – 10, …)
        + Noise
    - Noise (Shannon) – limit on bits
      * Maximum number of noise to tolerate to distinguish 6V and 4V is 1V, which limits the number of bits per symbol
      * Shannon limit applies to any channel (wire, wireless, etc.)
      * S – max signal
      * N – noise
      * Signals must be spaced 2\*N
      * B – bandwidth
      * Shannon limit is B \* log (1 + S/2N) bits per second
* Channel needs to pass a range of frequencies in a range from highest to 0Hz
  + 010101010101 – highest
  + 000000000000 – 0 Hz
  + Non-periodic bit patterns

Lecture 4

Diagram

Description automatically generated

* Physical layer
  + Coding sublayer
    - Add transition bits
    - Initial training bits used to sync clock at the beginning
      * Physical clocks drift over time -> we need transition bits to keep them in sync
    - Example
      * Diagram

        Description automatically generated
      * Transition bits
        + Start detected by rising signal (edge detector)
        + Sample every 1 bit after that
      * Initial transition bit after each character being transmitted
    - Manchester
      * Sends double the number of bits (if Nyquist limit is B, we send B/2 amount of useful data)
      * 1->0 : 1
      * 0->1 : 0
      * Do not look at the signal for about 20-30%.
        + E.g., to send data 1111111
        + We need, 10-10-10-10-10-10-10
        + We must ignore the 01 transitions
        + No need to look at a specific point (middle of a peak), instead we watch for a transition from 0 to 1 or 1 to 0
      * The data 0101011 indicates start of the bit pattern (preamble phase)
      * Looking for rising and falling edges is dangerous due to noise
    - Phase Locked Loops (?)
    - Eye pattern (?)
      * Superimpose 2 signals together
    - Multiplexing (n times more bandwidth by sending more data at once)
      * Time
      * Frequency
      * Wavelength (visible light)
        + Send red, blue, etc. signals at the same time at the sender end
      * CDMA
        + Different frequencies at different times
    - Async vs sync
      * Async: Sender and receiver don’t have the same clock. Need start and stop bits and an accepted number of data bits (e.g., 1 byte)
      * Sync: Both the sender and receiver have synchronized clocks. Need transitions to ensure clocks do not drift.
  + Decoding sublayer
    - Clock recovery
  + Media transmission sublayer
    - Convert digital bits to energy (electricity or light)
  + Signal transmission sublayer
    - What the channel does to the signal (Nyquist and Shannon limits)

Lecture 5

* [www.google.com](http://www.google.com) -> DNS -> IP address
  + IP -> 32 bits
* HTTP atop TCP
  + HTTP header
* TCP adds sequence numbers to order data when received (after retransmission)
* Routing
  + Each router knows the next hop (forwarding table) and passes that to data link
* Data link
  + MAC in header
* RS-232
  + Interface between CPU and modem connected to twisted pair
* Cable
  + Twisted pair (last mile)
    - Signal is difference between both pairs -> signal gets cancelled out
    - Used to be low battery and low throughput and long distance
      * Now higher throughput less distance (cat 5/6 twisted pair)
    - Cable = coaxial
  + Cable is good at sending data to you
  + Coaxial
    - Used bw buildings
  + Twisted pair
    - Local
* Fibre
  + A lot of bandwidth
  + Each fibre cable is unidirectional
  + Total internal reflection – no refraction into the fibre cable
  + Multimode fibre
    - One part reflects (long path)
    - One part doesn’t (short path)
    - Limits bandwidth
  + Single mode -- Thin fibre
    - Wave will not be spread into parts... just one thin palse
    - Higher bandwidth because there is no dispersion (wave is not spread out)
  + Use monochromatic lasers so that the light waves do not interfere
* Wireless
  + Spectrum
    - Radio waves
      * Omnidirectional – goes in all direction
      * Goes through obstacles
    - Satiates, microwaves
      * Directional
      * Absorbed by obstacles
      * Signal to satellite from dish will be unidirectional
      * The signal back from satellite to its destination will have a larger direction range
      * Geosynchronous
        + Satellite should rotate around earth at same speed earth rotates itself

Antenna only needs to point in one directions

* + - * + Satellite needs to be far to achieve geosynchrony

Huge latency

Not as much bandwidth (cannot keep too many satellites in the sky)

* + - * Right of way
        + Satellites are not on the ground (bypass wires, …)
      * Low orbiting satellites
        + More latency
        + Cover earth with low orbiting satellites… handoff connection to closest satellite as one rotates away from dish
        + (satellites are not geosynchronous since they are too close)
        + Disadvantage: You need a lot of satellites that span around earth
* FMD = freq division multiplex
* TDM = time
* WDM = wavelength
* Interconnection
  + Router
    - More latency
    - Can redirect if a link goes down
    - Check for errors
  + Repeater (“hubs”)
    - Less latency
    - Cheaper
    - Physical layer
    - “Blindly” repeat bits
    - Fibre stays at the repeater level

Data link

* Hop by hop should be an optimization (so that routers retransmit before TCP times out… not worth it if links are reliable)
* End-to-end ack is required to guarantee successful transmission
* Why framing
  + Headers
    - Individual bits cannot have headers (which would include destination, source, etc.)
  + Multiplexing
    - 2 senders and 2 receivers share a link
      * Each sender sends 1 frame at a time
    - Need frames with headers to multiplex (need discrete units to multiplex)
  + Error recovery
    - Easier to retransmit frames rether than the entire data stream
  + Frame boundaries
* Sol 1 : High Level Data-Link Control (HDLC)
  + Flag: 01111110
    - Start and stop delimeter
  + Bit stuffing
    - If data has pattern 11111, then add a 0
    - So, 111110 -> 11111
    - So, 1111100 -> 111110
    - Stuff an extra 0 after 5 ones
* Sol 2 : Ethernet
  + Inter-frame gap
    - Period of silence (0 V)
    - For 96 bit times
  + Period of preamble (56 bits)
    - 10101010…
    - For receiver to synchronize clock with sender
  + Start of frame delimeter
    - 10101011
  + Data
    - Includes header (dest, src, …)
  + Inter-frame gap
    - Period of silence (0V)
* Too small frames
  + Error rate
* Too large frames
  + Higher chance of error and need to recover a larger number of bits
* Usual amount: 64 – 1500 bytes
* If flag is 01111110, should add 0 after 5 1’s, not after 011111
  + First case: 0111111111110 -> 011111011111010 – transmit – decode -> 011111111110 (good)
  + Second case: 01111101111110 -> deflagger will see 01111110 as a EOF delimeter -> data after this pattern is lost
* If flag is 01010101, cannot stuff 1 after 010101
  + Data = 0101
  + Transmitted = 01010101 0101 01010101
    - * + -------------
        + Receiver sees this as a end flag -> data lost
* Sol 3 : Change the physical API
  + E.g., 4-5 encoding gives 0000 to 1111 data values, but there are many other bit patterns not used. Use those as control characters (SOF and EOF)
  + Add a third symbol at the physical layer (0 -> 0V, 1 -> 1V, frame boundary -> 5V?)
* Errors
  + Random error
    - 0 or 1 corrupted
    - Noise corrupts 1 bit
  + Burst error of length n
    - Bit 1 and n are corrupted. Bits in between may be corrupted
  + H(M) is 32 bits -> 2^32 possibilities -> 1/2^32 chance of getting the same hash for M and M’ such that M != M’ -> undetected error
    - Assuming errors and hash are random
  + Error correction is more expensive, not worth it when retransmission is few ms
  + xxxxxxx -> xxxxxxxppp -> ppp -> needed addition to make xxxxxxxppp divisible by 7 -> receiver will ensure xxxxxxxppp is divisible by 7 -> extract xxxxxxx as data bits (we add 3 bits because we add up to 6 to make num divisible by 7 -> 110 is 3 bits.
  + CRC uses mod 2 arithmetic
    - Addition = subtraction = xor
      * 10001 + 11100 = 01101
    - Multiply
      * Shift + XOR
      * 10111 \* 11 = 11001 (shifting just as normal mult -> use XOR when adding the terms)
    - Division
      * M = 1001, G = 101 => r = 3 (num bits of G) [If first bit of G is 1, then the remainder will be r-1 bits)
      * M’ = 100100 (add r-1 = 2 bits to M)
      * 1011
      * \_\_\_\_\_\_\_
      * 101 | 100100
      * 101
      * 0011
      * 000
      * 110
      * 101
      * 0110
      * 101
      * 011
      * Remainder t = 11 (CRC). [you no longer take G-t as CRC (only t)]
      * Final message = 100111 (which is mod 2 divisible by G=101

Each codeword has a set of valid codewords.

The idea is to maximize hamming distance between two adjacent valid codewords.

Parity has a hamming distance of 2

0 -> 000

1 -> 111

Hamming distance is 3

Hamming distance – 1 == resilience to error

Hamming distance of d:

C1 -------- d ---------- C2

|

|

|

d

|

|

|

C3

Any error of d – 1 bits on a valid codeword will result in an invalid codeword, so it will be dropped. => All d – 1 bit errors will be detected

Hamming distance of CRC is 4 (all 1-bit and 2-bit errors are detected… all odd bit errors are detected (3), but 4 bit errors are not detected)

Error correction: If hamming distance is 2d + 1, we can correct all d bit errors by going to the closest valid codeword. Error of > d bits will result in an incorrect correction

Media Access Control

* Reducing collisions
* Aloha
  + Fixed sized frames
  + Unslotted: within 2T will cause collision
    - Block size T. Any other frame starting before T seconds or during the T seconds (total 2T)
  + Slotted: within T will cause collision.
    - 0 … T … 2T … 3T
      * If frame is ready to be sent in (0, T] will be sent at T. Any frame ready to be sent (T, 2T] will be sent at 2T. => within the same T interval causes collision
* Ethernet
  + Variable sized frames
  + No acks in ethernet
  + Carrier sense
    - Don’t transmit/stop transmitting when you hear another signal
      * But if you already started sending data when a collision is detected, send more bits just to get enough collision signal so the other sender also detects the collision. (Jam)
  + No max packet size
  + There is a min packet size
    - If sender sends a long enough frame, then it will detect a collision
  + Sender needs to somehow know that a frame is dropped without acks
  + Ethernet has frames with a period of silence to determine end of frames
    - But we still need a length field to determine the padding bits that were used to reach 64 bytes (recall 64 bytes is required to ensure sender detects a collision of the frame it just sent)
* Statistical multiplexing
  + Bandwidth B, send B/x where x is number of busy users
  + Share

Questions

* Difference between Lan and Ethernet?
* Eye pattern? What’s the idea
* Binary exponential backoff??
* Doesn’t using hub completely chance way collisions happen we learned today?

Media Access Control  
 Ethernet

* One big wire between a number of nodes
* Carrier sense possible: any node can detect collisions on the common wire

802.11

Hidden terminal problem

* A -> B -> C
* C cannot detect collision since it would happen on common terminal B

Solution: RTS/CTS

* Request to send / clear to send
* ( A **(** ) B ( **)** C )
  + When B sends, it goes through everyone in his range (both A and C). But A and C cannot hear each other
* A sends RTS to B
* B sends CTS
* When C overhears a CTS sent to A, it keeps quiet for duration of the transfer

Ethernet for short distance

Interconnecting ethernets (data links) without routers

MAC address unique (48 bits)

IP address temp (32 bits)

Multicast

* Ethernet is broadcast medium -> can send to a set of people
* MAC msb:
  + 0 -> multicast broadcast
  + 1 -> 1 sending to 1 receiver
* MAC address on all devices are unique:
  + First three bytes are manufacturer (intel, dell, etc.)
* Hardware looks for a set of addresses: its own MAC address and the multicast addresses
  + If an address matches anyone in a set
* Hardware passes up to software via interrupt only when address matches the few addresses we are listening to
  + Software makes a call to hardware to specify MAC addresses in the set
* Hubs
  + N wires going into a hub and all connected together in series.

Bridges

* Ethernet 1 (I1) ----- bridge ------ Ethernet 2 (I2)
  + Bridge stores a mapping of MAC to I1 | I2
  + Algorithm:
    - Receiver F(D, S) on Interface X:
      * I = Loopup(D)
      * If X == I:
        + Drop
      * Else:
        + Send F to interface I
  + Adjust algo to learn the Loopup table based on source addresses
  + Flooding: Haven’t learned destination address yet, so we send it to all other interfaces
  + Filtering: We learned the interface of the destination, but source and destination are on the same interface, so we drop the packet
* All flooding stops soon due to acks
* How to handle when device changes interface (moves to another location)
  + Add timers that will reset each entry in the Lookup table
* Transparency
  + A bridge is transparent if you can remove the bridge and the nodes will not know the difference.. Router is not transparent because nodes need to directly communicate with the router
* Promiscuous Receive
  + Bridge must receive everything (read everything)

Realization

* Problem: ethernet runs at 10Mbs and routers run on Kbs, so we couldn’t efficiently use routers to connect ethernets. Also, there was no standard routing protocol
* Bridge needs to be fast
  + Bridge picks everything on the ethernet. Router only picks packets addressed to it
  + Solution:
    - Runs 10Mb/s
    - Memory shared by two ethernet chips connected to the ethernet interfaces
    - Binary search lookup engine
    - CPU talks to binary search lookup engine to perform the algorithm (flooding, filtering, etc.)
  + Bridges cannot have cylcles (topology must be a tree)
  + Every connection to the ethernet has a mac address, so bridges have at least 2 addresses
* Spanning tree
  + Pick root (bridge with minimum MAC), then build a minimum spanning tree
  + The distributed algo
    - All bridges send message (Root, Distance, Own ID)
    - Initially, all bridges think they are the root (at distance 0)
    - They broadcast their state and update their state accordingly on receive
    - Each ethernet picks designated bridge (db) that is closest to root
* Bridges vs. Routers
  + Both can connect ethernets
  + Bridges connect Ethernets within data link
  + Routers need a new layer (IP)
  + IP: 32 bits (written in 4 decimals separated by dots)
  + MAC: 48 bits, in pairs of hex
    - Everything connected to the ethernet has a MAC address
  + From bridge POV, routers are just devices with MAC connected to ethernet
  + From router POV, ethernets connected via bridges are just one big ethernet
* Routers
  + When you need a mac address on a line, routers broadcast ARP message on a line and ask everyone to respond with their MAC addresses. Then, that info is cached in an ARP table
  + Translate from IP to MAC using ARP
* Bridges are bad
  + Address Incompatibility and Max Packet Size incompability
    - Router has ability to chop router into pieces and give sequence number... Bridge has no choice but to drop packet if an intermediary ethernet does not support address or packet size
  + Routers are fine with handling prefixes of IP… Bridge addresses are flat (48 bit) without any hierarchy, so need to store entire addresse
  + Spanning tree is bad in wide area
    - Fine in local area network, but in wide area, you might need to travel a long distance even if destination is close if the bridge happened to be off
* Levels of addressing
  + Human friendly: Bruinlearn.ucla.edu (e.g., “file.txt”)
  + Locator: IP address (e.g., file\_descriptor)
  + HW level: MAC address (e.g., plate, sector)
* Translation from Human friendly to locator: DNS server
* Translation from IP to MAC: ARP

Peering

* ISPs decide to carry each other’s messages for free

IP’s original goal was internetworking: 2 independent networks communicate together

Each packet carries a header with network number (IP address). Gateways (routers) was to get the packet to the right network given the network number (IP address)

Incompatible max sizes of networks -> IP routers had to fragment packets when necessary

UCLA Network ------- Router ------- Stanford Network

|

|

|

Berkeley Network

Basics of IP

* Names vs Addresses
  + Use DNS to get from name to IP address
  + Use ARP to get from IP to MAC address
  + Use bridging table to get from MAC address to interface
* Class A
  + Starts with 0, First byte was the net prefix, the rest were the host
* Class B
  + Starts with 10, First 2 bytes was the net prefix, the rest were host
* …
* Old IP forwarding
  + A table for each class, class A, class B, …
  + Check what class this IP is, and look up correct table
  + Send to appropriate router, default router if no entry found
* New IP forwarding
  + Look for the longest matching prefix in a table inside the router
    - Look for longest matching prefix because it is closer to the destination. E.g., why go to ucla.edu when you can go to cs.ucla.edu instead, which is closer
  + Crossbar switch
    - Make parallel connections between input and output interfaces
    - Not on exam
  + Routers need to buffer packets and send ARP for them to get the destination Mac address
  + If two endnodes have the same subnet (compare IP addresses using masks), then we can ARP to get MAC address of other endnode and communicate directly without a router because we are on the same ethernet.

R1 E3

| |

DHCP -------- E1 ----------- E2 -------- R2

ARP: IP to MAC

1. Message comes from network to R1 for E1

E1 asks DHCP for an IP address and asks for the IP address of R1

Consider a packet comes from network to R1 with IP address of E1. R1 doesn’t know who the IP address is. Then R1 buffers the packet and broadcasts an ARP request (with the IP in the request). E1 sees its own IP address in the ARP and responds with its MAC address. Then, R1 sends the message to that MAC address.

1. E1 and E2 want to talk

Everyone on the same ethernet must be given prefix 128.3.\*. Since E1 knows E2 has the same prefix IP address, so it does not need to send its message to a router. E1 ARPs to get the MAC of E2. Then, E1 can send a message directly to E2 through the ethernet without a router.

1. E1 sends to E3

DHCP gives E1 IP address of one router. Let’s say E1 has IP of R1. Then, E1 sends message to R1 with destination IP address of E3. R1 sees that it has to send the packet back on the same interface to get to R2. R1 sends a REDIRECT to E1 that says: “in the future, if you want to talk to E3, send your messages to R2 (and gives the IP address of R2 to E1)”.

If no IP address matches within a router, then we send the packet to the border router (which is the default router). (table in router == forwarding table)

IP Forwarding with ACLs

* ACL:
  + When you get input with (src, dest, protocol, port1, port2) in router, you check to ensure this tuple matches a list of rules you allow. Otherwise, drop it. (Layering violation).

Trie

Multibit trie

Ternary Content Addressable Memory (CAM)

Route computation

* Trying to build the routing table
  + Prefix -> interface
* Distance vector
  + Nodes send hello messages to neighbors 🡪 if no hello messages are received in a while, then a node is dead
  + Each router keeps track of:
    - Current shortest distance vector and neighbor nodes to shortest distances to all nodes
    - The distance vector received from neighboring nodes
  + They announce their current shortest distance to neighboring nodes
  + Count to infinity problem
* Link State
  + Send the estimates of neighbors (instead of estimate of yourself) to your neighbors
  + Link state packet
    - Current node, neighboring nodes and distances to them

Policy routing

Who? ISPs

Why? Security, economics

How? Small modification of distance vector (called path vector)

Add parameters: sequence of ISPs (you don’t like the ISPs, so you may discard it)

Replace distance with a path

DHCP

* Gives you
  + Your IP address
  + DNS server IP address
  + Router IP address

DNS

* Given name address, gives IP address

ARP

* Router sends ARP request to get MAC address of a node given the IP address

Autonomous System

* A signle org/domain that uses link state or distance vector
* It is connected to other ASes
* E.g., UCLA, CMU, etc
* Every AS (when registered) is given an AS number
* AS number vs prefixes
  + Each AS already has a prefix, why have an AS number then?
* The border routers (which are inside an AS and are connected to other border routers in other ASes) send their own prefix and the AS path (a sequence of AS numbers to get to a destination)
  + Each ISP can then decide their own policy (e.g., if they don’t want to go through a particular AS number, then it could not pick that route)

Border Gateway Protocol (BGP)

* Canonical path vector protocol (how it is done now)
* BGP params
  + Prefix
  + AS path to a node
  + Next hop (IP address of interface of next router to go to)
  + Community – “No transmit policy”
    - Equivalence tags for multiple routes
    - Packets with community D can internally travel but cannot be sent externally
    - E.g., ISP packets coming in to UCLA (which pays the ISPs) are labeled with community D and we ensure those messages are not sent out externally (because otherwise we are providing service to the ISPs… but they are the service providers)
  + Origin
    - Route came from IGP or EGP
  + Local pref
    - Statically configured ranking of routes within AS
  + Multi Exit Discriminator
    - If an AS has multiple exit links, then it can load balance between the links and use the MED number to prioritize which one gets more traffic
    - Since BGP packets sent out are for data messages coming in, then MED going out controls data packet priority coming in
* eBGP routes (external) are directly connected to each other. Does not go through AS
* iBGP routers are connected together via an AS (form a TCP connection and looks to be directly connected, even though not directly connected.)
* Data packets flow in opposite direction of BGP update increments
* Local pref -> priority
* BGP runs over TCP
  + So routing is abstracted away at the BGP layer… border routers appear to be connected directly
  + sets up a TCP connection and exchanges increments reliably
  + Unlike distance vector, it doesn’t need to send its entire state, just the incremental changes… Which are guaranteed to arrive due to TCP
* If a route is no longer available, send a withdrawal increment message

DHCP

* Your IP address
* 1 Router’s IP address
* DNS server IP address

DNS

* Name -> IP address

NAT

* How to use 1 IP address for N > 1 people

iBGP

* Internally, border routers of an AS form a TCP connection (virtual connection)

Alternative to iBGP: route reflector

* All borders within an AS send their data to route reflector and the route reflector send it back to other border routers
* This is only for BGP packets… data packets flow normally
* This method has more latency

eBGP

* Boarder routers are directly connected

DNS root servers

* Roots know where every other DNS server is
* But not necessarily know geo locations

We go directly to the root DNS, then the root finds the local DNS of the destination that knows the IP.

NAT disadvantage:

* You can’t have a server behind a NAT

TCP

* Why transport layer?
  + Reliable, in-order, byte stream
* Segments with sequence number sent to IP
* What is a connection?
  + State kept at the sender with a connection ID
  + Connection table has connection ID key
  + Connection id = (src\_ip, dest\_ip, src\_port, dest\_port, protocol\_id)
* Why not have just one connection for all data?
  + If connection between 2 processes is slow, then it should not affect other connections
* Why not keep connections up always?
  + Expensive
  + Every pair of hosts on the internet
* How do we address the receiving process in the receiver machine?
  + Not OS process names, need OS independent name
  + We use ports instead.
* Flow control
  + Speed matching between sender and receiver
* Congestion control
  + Speed matching between sender and network

UDP delivers messages … UDP delivers to a process (IP delivers to a host)

Read/write messages (similar to a segment/packet)

TCP interface is a byte stream

Puts bytes into a buffer and packages them into segments and gives them to IP

Other application reads segments in buffer. Application interface reads bytes

Why not use TCP for video, games, etc,

* Too much overhead (extra acks)
* Latency… sometimes when packet is dropped, we wait so reliability and in-order is correct

Why use UDP for DNS, NTP (Network Time Protocol)?

* DNS does not need to be in order
* DNS does reliability (retransmission) itself, so we shouldn’t use TCP

TCP vs go back n

* Same as go-back-n except
* When TCP sees 3 duplicate ACKs, then it knows something went wrong
* Good for one packet loss
* This optimization breaks if packets arrive out of order