CS 118 Class Notes
W 1 M Lec 1-5-16

- Familiarize you with the fundamentals of computer networks and how they work.
- Careers will deal with networking situations
- It is important to understand how they work and how to make use of them
- Everyone that comes out with a CS degree should have a good understanding of how computer networking works
- Textbooks
- Shannon/Weaver, <u>The Mathematical Theory of Communication</u> (any edition)
- Corose/Ross(?) <u>Computer Networks: A Top Down Approach</u> (any edition)*
- * readings are cited from the Sixth Edition
- Programming projects
- Individual!
- Grading
- 30% projects
- 15% each for 2 assignments
- 30% midterm
- Feb. 4 in class
- 40% final exam
- March 14, 8-11 AM
- Projects are due as announced
- Due at the start of class on date indicated
- TA will set policy for late submissions
- Office Hours
- TTh 2-3 PM in 3532F Boelter Hall
- Other times possible by arrangement
- TA
- Seungbae Kim
- ksb2043@gmail.com
- Handles all project-based questions
- Have a good understanding of computer networks that you can apply for the rest of your career
- Not sure if tests are open-book yet
- Webpage for the class and the slides will be available on the webpage

Class Material

- Network Layers
- First principles that describe how communication can be done between anybody i.e. people and computers
- How people communicated through the course of time
- Computer networking
- Communication between them so there is information exchanged between computers

- Make use of that information exchanged
- We aren't too concerned about the presentation of data, except to the extent that they run into the same problems we see in lower-level communications
- Bandwidth
- Losing data
- We do not care about UI or how it looks on the screen
- Information
- Communication
- Methods of exchanging information between a fixed set (you know who's talking, listening) of directly-connected parties
- Can use any medium i.e. speech, telephone, email, Internet, etc.
- We have a set of pre-agreed rules
- We have an implicit rule of speaking in English, Reiher lectures, students have to raise their hands to query a question
- We understand how questions are dealt with and we are able to communicate in the matter that we wish to.
- What if we wanted the info to go to someone else in this room?
- We have to have some kind of <u>protocol</u>
- You raise your hand, you wait for the professor to call on you, etc.
- Single shared set of pre-agreed rules. There is an understanding of what should happen
- Somebody should decide to call someone else and this will indicate there is an incoming call
- Protocol Variations
- Alexander Graham Bell recognized you should say something to indicate there is a connection
- It is more common for the person being called to say "Hello"
- It can work the other way around, if there is an agreement
- Direct and doesn't necessarily go to one-to-one
- Can go from one-to-N or N-to-one (less common)
- This would require a lot of prearrangements
- The sender works directly with the party who is to receive the information
- Networking relaxes some of the assumptions
- We want some method where Reiher can talk to anybody.
- Should also be to people who you cannot directly communicate with
- If you cannot talk with your voice or sending bits out across a wire
- The person is not at the other end of the medium i.e. wire or phone
- You don't necessarily share the same set of rules
- Example: Chinese people saying "Wei?" because of crappy phone signal
- You want to be able to specify who gets information and parties don't need to be necessarily able to communicate with
- Don't necessarily need to follow the same set of rules -> go from little or no knowledge to how we can do this for parties we need to hear with.
- Communication
- Fixed set
- Directly connected

- Single, shared set of pre-agreed rules
- Networking
- varying sets
- indirectly connected parties
- Don’t share a single set of rules
- You don’t necessarily know how to get the information to him and you don’t know the rules to send information
- Summary
- Communication
- Methods fo exchanging info between a fixed set using single protocol
- Networking
- Methods of varying sets with multiple protocols
- Protocol
- Set of rules to enable exchange of info
- Layers
- International Standards Organization (ISO) model
- If we use layered approach, here are the seven layers we need
- Still being taught to people and there are never seven layers
- Internet
- Widely used and arguably has four layers (more or less)
- Not quite as well-defined from a typical layered system
- Names of layers really don’t tell you much
- You have many different types of layers
- If you have names on things, they are labels -> names do not inherently have any important meaning
- Saying something is a transport doesn’t necessarily mean much of anything
- What is that piece of software and what action is performed on it
- Many ways of mapping functionality into the hardware and software available to you
- Sometimes names are useful
- Names
- Important
- Allows you to specify this vs that
- We know which website we are talking about
- We know the correct actions happen to correct things
- We can use the names of the circuit to specify which one to deal with
- Names help us to get the messages to the right recipients
- Helps to keep track of which layer we are in
- Unimportant
- What they are called
- Summary
- Don’t care about names, just know the relationship
- Layers
- The ISO model was predicted about what they thought was possible

- Very much in its infancy and we discovered what helped and what didn't help
- We didn't actually bind ourselves to these models and bind them in this way
- Sometimes we skip over layers, or we duplicate layer functionality
- Too simplistic a way of looking at things
- Not really going to do what you need to do to understand computer networks
- Approach to Computer Networks
- Abstraction
- If we have something very complex and it is too difficult to understand
- We represent it with something simpler
- We hope this simpler thing can help us to predict the complex thing
- Just deal with the simple thing and don't worry about complexity
- Recursion
- Take a large problem and divide it into a combination of its components
- Down at the very bottom, we take the fundamental thing and we define how that bottom thing works and let's build up from that
- The goal
- Describe computer networks using abstraction and recursion
- Layers (Fixed?)
- Layer is the largest set of things that can communicate (have direct interaction)
- Shares a single common protocol
- A Roadmap of the Course
- Bits
- Very fundamental part of Computer Science
- Build up from the bits (start small then build up)
- Communication
- Share bits between two parties
- Channels
- Protocols
- Leads to Networking
- Multiparty complications
- Layers
- Naming
- Recursion/forwarding
- Leads to Examples & Mechanisms
- ~5000 years of networking to consider
- For as long as speech has been around, we have had communication via speech
- As civilization developed, we need to get information from one place to another
- Communication using voices, senses is insufficient
- Distance, language barriers, etc.
- How did they do it back then

- Couriers
- Human-based
- More reliable
- Problems?
- Slow
- Limited ranges
- Expensive
- Inefficient
- Vulnerable
- Pony Express
- Used in mid-19th century
- Couriers could get lost, killed, delayed, etc.
- Couriers could corrupt the message
- Hostile parties can prevent the couriers from getting where they needed to go
- Pony Express set up all these stations and they had to have a guy taking care of the horses
- He had to buy new horses and make sure there was food and things for the relay
- Carrier pigeons
- Unidirectional
- Knows where home is from a great distance
- It will go back home once it knows it’s directions
- You can take the carrier pigeon somewhere and it isn’t going back to where you are.
- Serious limit to the amount of messaging you can do
- Limited by the number of pigeons you got
- It can fly and carry a little bit of weight
- Tiny message on tiny paper
- To reset, you have to take the pigeon somewhere else to return home
- Tough to work with
- Hawks like pigeons, if it gets eaten, you are screwed
- Beacons
- Somewhere on a hilltop, you have a fire and you can light the fire and see it from a great distance
- Has the advantage of being able to work from a very large distance
- People can probably see it a hundred miles away
- If you set up relays, you will run into same problems that you run into at couriers
- Watch and wait for the other beacon to get lit
- Works over 100’s of miles
- Worked best at night
- Can’t see the fires at daytime
- Heliograph
- Optical communication (like beacons)
- Based on using mirrors and the sun

- The guy who I’m communicating with understands the protocol
- If he sees flashes of light, he will understand what I mean
- Obviously only works when the sun is out
- A little more flexible than beacons because it is moderately portable
- Hard to aim and you have to have some prearrangement to communicate
- Otherwise, we don’t know how to use this
- Flags
- Popular form of communications
- Most common for maritime communications at sea
- Used in Naval battles (indicates what to do and what the admiral wanted)
- Still used today to a certain extent
- Not used as much but used for a few other purposes
- If you see a flag up, this implies there are a few unsafe conditions and you shouldn’t go out to swim
- Origins
- Couriers: Spoken/written in prehistoric times
- Hooke
- Semaphores - analogous to flags + Telescopes
- Proposed increasing the distance a signal could be sent
- French Telegraph
- Semaphore telegraph
- The French created a semaphore telegraph that had towers and they had a telegraph that could go up or down and indicate a piece of information
- We understood what it meant and we had a time synchronization
- Problems
- Issues of contention within the same rang
- Flow control: People would be overwhelmed by too much data and lost data
- Error recovery - how to handle errors and treat them gracefully
- Emergence of electricity
- Electromagnets
- Led to electromagnetic relays
- Possibilities for using electricity to communicate via distance
- Cooke/Wheatsone
- Able to use multiple needles to send messages
- Based on where the arrows pointed, you can predict what the message said
- Morse
- Used electricity to send messages using a single relay
- Better in every way than the Pony Express
- Morse code sent symbols and letters
- Letters are encoded in a common standard
- Causes a particular electric signal to be sent
- Code consists of dots and dashes
- There had to be some space in between these and it was all standardized to be consistent.

- You also had to indicate you were done with a letter
- After a particular letter, you had a certain delay and the other guy could interpret it
- Letters were built up into words
- You would have an seven dot delay to indicate the end of a word
- This was a relay system but you didn't have to get on a horse
- Telephones
- Instead of having dots and dashes that only made sense to people who understood Morse code
- The telephone carried over actual voice
- You could only communicate via wires
- In order to have an effective telephone system, you need some form of networking
- Restaurants
- Wife
- Mom
- Airport
- Regardless of how you do it, you used wires in the olden days
- Radio
- People realized you could send signals though the air
- Used for the same basic purposes of telephony
- Sending the sound through the air in a broadcast fashion
- Thousands of people could hear the signal that was being broadcasted
- Marconi's device was the first practical device
- Could be applied to videos and other media
- Computer networking
- People realized that computers had to communicate with each other
- Originally, this was using very specialized networks
- The computers were sitting on the same campus and we had to run it from various points of the campus
- People began to think what was the right way to have computers talk to each other
- One of the important developments was packet switching developed in the 1960s
- This was one of the core developments that allowed computer communications to develop.
- Packet switching was developed and used the ARPANET (Kleinrock!)
- Used a machine at Boelter Hall sent to Stanford University
- The original development of the ARPANET was somewhat limiting
- We had to think of what to do in the long-run
- In the late 1970s and early 1980s, we developed new protocols called the Internet
- Funding came from DARPA
- Intended to be used for research labs and everyone wanted to use it
- Government funded network, there was a debate to use this for commercial purposes

- This is too good to just use for the government -> Make it available for everyone
- 1989: Use this for commercial purposes -> IBM wants to set up sites on the Internet and let them do it
- From that point on, Internet became the core of what it is today.
- It was still somewhat limited but of course, we could build anything we wanted to.
- Not just available to IT experts such as IBM, military, etc.
- How could we make effective use of this capability
- World Wide Web abstracted a lot of things
- Not a new hardware thing - built on top of software that was already there
- Characterizing Networks
- Digital communication -fairly gross generalization
- Numerical methods - distance the network covers and can transmit 10 MBits/s
- When we can apply a number to something, it is a more useful characterization
- Computer Science is essentially an Engineering profession - hence we should be able to apply numbers to things
- Typically, things related to time
- Things related to what you can push through the network and how many things can be moved
- Distance also matters - how far the wire reaches and signals go through
- Communications is all about time
- Highly reliant on time
- Information transfer from A to B
- It doesn't matter the medium - there is some amount of time to transmit information.
- Frequently what we do is transform information
- This requires some amount of time, will not happen instantaneously
- We need to talk to each other and this means we need to start it at A and get it to B
- B will send a response to me saying we have your information and there needs to be a confirmation that they have received information
- Communication/Network Measures
- Frequency
- Transformations and other kinds of processing
- This is how fast a processor runs and how fast it can process the transformations
- We generally don't have instantaneous communication
- Usually bound by a maximum of the speed of light and generally much slower than that
- Delay
- Propagation latency
- Access delay
- How long it takes for a person to deal with sending the information

- Loss rate
- Does everything that we send from point A to point B get to point B
- Rate vs Frequency
- Rate
- Number of events per unit time
- Frequency
- Time between events
- Not always directly related
- Depends on number of servers you have
- You can speed it up by having servers run in parallel
- If you have more servers, it can be achieved at a faster rate, but it's more costly
- The importance of speed
- Latency is a fundamental limit
- How long it takes to do something
- The more you can do, the more sophisticated result you can get
- Everything else is kind of a means to an end
- These things will tend to have an impact on latency
- Are we happy to overcome the things of the loss rate
- This is the choice we get to make
- Not always the case for all elements of computer science
- Things like screen size is not a latency metric (UI related things)
- What is latency?
- The time between certain things
- Measured as a delay between one thing and other things
- Event Y happens before Event X
- Time between when you start asking a question and when you answer back
- Give a webpage back with a certain transaction
- Latency is how long it takes to receive information from a destination
- The time between creating information at a source and receiving it at a destination
- Cumulative event: Effect based on several things
- Goes over a period of time
- Property of two effects
- Creating and receiving a message in question and the information
- Not a single value
- Many things going on that can affect what our latency is
- Several things will affect our latency
- Fixed cost - sometimes it doesn't matter the size of your message
- It will have some cost at the header
- Fixed length and a fixed amount of time that will create the header
- The OS will be involved in sending and receiving our message
- Put info in the memory card and it is not under the control of the OS
- Propagation delay
- In the medium I am using to go 500 feet, for example

- Proportional cost - more bits takes longer
- Typically related to the bandwidth I have available
- Transmission delay - getting all the bits out onto the wire
- Unpredictable aggregate effect
- Delays
- Loss in retransmission
- Relays in different ways depending on the circumstances
- It becomes a little bit complicated in certain circumstances to predict.
- Message size does matter
- It does have an effect on latency
- 5 root causes for latency
- 1. Generation
- Something happened to the guy causing the information delay
- We have a piece of information we want to send
- 2. Transmission
- The delay in transferring info from one place to another
- Takes a certain amount of time for the bits to propagate across the world
- Constrained by the speed of light - we generally achieve a fraction of the speed of light
- Constant in each medium
- Based on some proportional coefficient that limits the speed we can achieve from the maximum
- 3. Processing
- The delay due to the computational translation or frequency of information
- Guy had to get off his horse and ride off
- Delay with changing horses
- Have a router and send it through computer networks
- Has a delay associated with it
- There will be a processing cost and this means there will be costs
- 4. Multiplexing
- Adds to the latency in computer networking
- Based on the fact we are sharing things, not making use of most of the computer elements
- Multiple people all want to use those resources
- Someone else is using it at certain times
- Wait for other people to use a resource before we get to use it
- 5. Grouping
- Sometimes we see we can send out a bit and we have to slap a 100 bit header on top of the message
- Sometimes we put 100 bits on it and how do we send that?
- Send a second 101 bit message
- Instead of sending 1 bit, we wait!
- We will send 2 bits as one message, and slap one header on it.
- We gave it various ways of aggregating this information rather than sending it as separate messages

- Costs here that we sometimes have to pay based on grouping things together
- Summary
- Important definitions
- Understand theses!
- Names are just names
- Labels that allow you to distinguish one thing from another
- Just points to a particular thing but you don’t want to get too obsessed and make too many assumptions
- Networking has a long history
- Over the course of mankind, we have tackled the same recurring problems
- The way we describe networks
- Tend to be related to time, especially latency
- Very general term - can be used to describe many kinds of networking effects
- Q. Are certain latencies mapped to certain layers?
- A. Costs tend to be in latency, each layer is not done in exactly the way it could have! Because we go through recursive fashion, there will be some kinds of latency we pay
- The higher the layer of abstraction, the more latency probably.
- Formatting a message is far more different than raising the signal of a wire

Communication and Information Sharing

- Shared Information
- Technical level - what did you hear
- What dots and dashes did you hear in the Morse code
- Semantic level and you typed it out
- Effectiveness
- Is that what the telegram was supposed to tell you?
- Is your brother going to get married a month from now?
- Entirely different level of communication we are talking about.
- Syntax
- The level we start with
- Particular symbols have particular meaning
- Sequence and order matters
- Certain letters put together create certain words
- Semantics
- Meaning
- Assigning values to symbols
- What is the meaning of this set of letters?
- Assign values to symbols
- Whether it is a person or a program
- Analog vs Digital
- Analog symbols can be ambiguous

- The words are pronounced by Reiher in a certain way and some other people pronounce it a different way
- A word that Reiher says could be interpreted wrong depending on the person
- Digital signals tend NOT to be ambiguous
- They are, or they are not
- 0 or 1 (Definite)
- Computers don’t do ambiguity
- They believe in actual, particular values
- Hard to deal with any form of ambiguity in a computer program
- Computers are digital machines
- Signal that I send and this is converted down to a 32 or 64 bit value.
- Does not do analog.
- Discretization
- Basically says that there are ranges of analog values
- Choose one within a range and pick that value
- Treat other vales in those range and pretend they aren’t even there
- Convert it into a value we can represent
- Round any unambiguous values
- Effectiveness
- We do this because we are dealing with computers and we want what we said to be understood
- Syntax level
- Is this an ASCII ‘A’
- Semantic level
- Does this word have a particular level that people should understand
- There is something I want to come out of that conversation
- Agreement I want to reach and have a certain effectiveness.
- Care about precision and accuracy
- Precision: I might send it to person A and person B
- Will they agree they saw the same value
- Accuracy: Send message to person A and person B a message “YES”
- Did both see “NO”
- That is NOT accurate
- Which Shared Info can we Handle?
- Syntax
- If we set up our rules properly, we can usually handle syntax and understand some errors and correcting them
- Semantics
- Tough!
- If we always handle them in all circumstances, we reach a level of human intelligence
- Fruit flies <u>like</u> a banana, time flies <u>like</u> an arrow -> Natural Language Processing!
- In computer networking, we have trouble trying to deal with semantics
- Effectiveness is even harder

- Often related to intent
- Beyond technical means
- Sometimes a person will not understand what you said and bad things have happened because you didn’t get your intent through to them
- Don’t even try to deal with effectiveness
- Constrain the problem
- Deal with the computer networking problem
- Deal with any problem related to syntax
- Don’t forget about semantics
- Related to networking itself
- What we are trying to do with networking itself
- Not about what the message contains
- If you lost the message, what is the meaning of losing the message
- If you sent it once, do you need to send it again?
- At the receiving end, what happens?
- That level of control we worry about a little
- Ask your friend if the message was interpreted correctly
- You can worry about the effectiveness later
- What is Communications?
- Information heads to you guys and you have to first speak
- Convert ideas in my head into words -> Send words out across the air into your ears, and then move those into your mind
- Encoding words in my mind into words
- Decoding words into your ears into your head
- No method without encoding directly to convey information from one place to another
- Use the air to speak or a wire for Morse code
- Fiber optic cable to send IP packets
- Send information to something the medium can carry
- You have some raw information and a transmitter
- Takes raw information and converts it into encoded information
- Of no use to the destination
- Any type of communication or sending a large stream of data across the Internet
- Not necessarily the case the information you send is what you end up with.
- 32-bit machine and we might very well end up with converting 32-bit integers into 64-bit integers
- Meaningful to the destination
- Raw vs. encoded information
- At a general level here
- not just computer messages
- speech, audio, and everything else
- The receiver wants to get the raw information we can use
- The transmission medium is something we want to actually do
- Probably not the same as the raw information

- Not necessarily the same of render and receiver
- A German speaker sends a message to a Korean speaker
- Issue of translation
- Raw information is not necessarily the same
- Encoded information
- Characterized by communication medium
- Morse code - dots, dashes, spaces you don't send anything
- Electrical wires - encoded information might be voltage high, voltage low, etc.
- Messages and State
- Senders want to send a particular message
- syntax, semantics, effectiveness
- Receiver gets a message
- Particular syntax, semantics, effectiveness
- Entropy and Information
- Leads to a way of how much information can we send
- Shannon described this and worked out the limitations on the basis of entropy
- How unpredictable are things
- The more predictable, the higher the entropy
- What is the maximum amount of info we can put into a particular message
- Entropy
- The greater the entropy, the more you can send
- Example:
- Sender can send "YES" or "NO"
- Only sends one set of messages
- One bit
- Two messages
- two bits of information
- The more choices he has, the greater than uncertainty (higher entropy)
- Communications and Entropy
- For each source state i, there is an entropy H_i
- = -p_i log p_i
- The probability p_i(j) of producing symbol j form state i
- The total entropy of the source is a summation
- Max and min entropy
- Max entropy is when choice is 50/50
- Min entropy is when there is no other choice (extremes of the distribution graph) - means you have 100% certainty
- Returning to Our Example
- At Max when either message is likely
- At min when he always sends "YES" OR he always sends "NO"
- Generalizing the Example
- What if we send N symbols
- Importance of this

- If we have an understanding of how communications can work, we can say how much information we push through a channel.
- Channel has a entropy H and capacity C
- C/H - epsilon symbols is the maximum

W 1 R Lec	1-7-16

- Fundamental result from a perfect channel
- There are *i* different symbols you could send
- The weighted sum of all probabilities is 1
- The log of any will be the <= log(1)
- Unless the log is 1, it will be a negative number
- To figure out entropy is to multiply the probability of each with the log of each, summed together and multiply by -1
- Entropy is always a positive value
- Smaller probability means more choice
- The smaller the probability, the more negative number you get and you get a larger positive number for the entropy at the end
- You get more for the possible set of symbols
- What about zero possibilities?
- log(0) isn't defined, but the limit as something goes to 0 from the right is 0
- Predictability
- What if we are not sending random bits
- Unlikely -> there is usually some small amount of data
- You could generally only send syntax
- If we send the bits in true fidelity, the receiver can interpret it
- This means we generally get non-random data i.e. text, Morse Code, scientific data, images, etc
- The symbols are not uniformly distributed
- Given some symbols are already sent, it means some symbols are more likely to be sent
- The maximum possible value occurs if you are equally likely to send 0 or 1 (50/50 chance)
- Morse Code
- E and T are more common so the have short representation
- O is a less common letter, so it has a longer representation
- English language text
- YELLO_
- W is not a very common letter but we can predict it
- PIT_
- Many possible letters that can come after that
- There is a lot of context that shows there is redundancy when we send English language text.
- Written text -> redundancy is approximately 50%
- You only get half of the information when you send every English letter
- In the English language, if you are sending random letters, once you have sent off about 5 characters, you can probably predict what the 6th character is

- Implies redundancy in data and the probability that it is 1 in 26 is actually a lot less
- Less randomness in that 6th character
- Information vs Entropy
- Information is in terms of bits, entropy is measure of randomness
- Given that they are bits, the only values they can take are 0 or 1
- Assume each character is a symbol, there are 2^8 different ASCII characters, so different amount of randomness there
- The fact we have redundancy means we are actually sending less information than we think
- Sending bits
- We think are sending 1,000 bits, but we may only be sending 500 bits of information
- Instead of using 1,000 characters to send 1,000 bits, you can send some way of encoding those into 500 characters instead
- You don't get more information through, because you haven't changed the encoding
- If you encode it down to 500 chars, you have the same amount of information, but it is compressed
- You have gotten rid of a bunch of stuff you didn't need in the first place
- Push more characters effectively through and now you send 500x the length of a character instead because I have encoded it properly
- Morse code - dots and dashes
- People who devise the Morse code encoded it in such a way that he used relatively short encodings
- "e" is a common letter so it is a single dot
- less common letters like "z" has a much longer encoding in Morse code
- On average, "e" could be sent much more frequently than "z", and they can afford to have a longer encoding for z
- American English letter frequencies
- E
- T
- A
- O
- I
- N
- S
- How do we get more data through
- Recognize redundancy and encode the data we want to send to remove as much redundancy as possible
- Send short signals for common things
- Longer signals for less common things
- What do we mean by encoding?
- We have information source and an information destination
- Between, we have a transmitter and a receiver

- The transmitter is the one who does the encoding and he recognizes what we want to deal with
- The receiver at the other end decodes this
- The Perils of sharing
- Sharing a state - sender has some state and he wants to share the state with the receiver
- We have been primarily talking about perfect channels, and it gets received without loss and corruption
- If we are not careful about how we do things, the shared state won't be exactly the same
- If haven't achieved what we wanted to achieve
- Staleness
- You have an old state, but you have not changed your view of it
- The state changes for whatever reason and the sender still thinks it is in condition A and condition B.
- The speed of light is our upper bound for speed of communication
- The state will not be shared at any time between sender and receiver
- Unless you have a perfect understanding of latency, there is a period when the sender tries to send the state of the receiver to be the same as his.
- Capacity
- Fundamental problem
- Problem because we don't have infinite capacity
- Effect of receiving
- Entropy perspective
- Physics - entropy never decreases, it only increases
- What happens when you receive data, you know something more about the transmitter
- Because you know more info, this means entropy actually decreases
- Entropy is the randomness of the bits
- The receiver knows what the transmitter just sent
- We know the state is now B, and uncertainty at the receiver, so the entropy has decreased
- Over time, considering the universe as a whole
- Entropy never decreases, and usually increases
- We have increased entropy at the sender
- We sent something, but we don't know if the receiver got the information
- Character Transfer Protocol (CTP)
- Try to move characters from one sender to one receiver
- Communication topic
- We have a direct communication between sender and receiver
- Protocol is a set of rules that determine how we want to communicate
- Get a bunch of characters from the sender to the receiver
- Send them 1 at a time assuming a perfect channel
- Always receive what we sent
- CTP events
- We have a particular protocol to transfer these characters

- CTP rules
- Before we start sending characters, both sides say we are not connected
- We have another internal protocol that says let's get started
- Making a phone call
- Protocol for starting a phone call
- When the receiver gets it, we say "AHA" we are about to start the character transfer
- When both send it, we are in the connected state
- The sender will receive it as intended
- Once the transmitter is sent - he sends one character at a time
- Receiver will deal with it in any way he wants to until the protocol is finished
- Sender decides when to kill the conversation
- This end of file occurs when they determine a transmission
- Once you finish a protocol, you go back to the not connected state
- From not-connected to connected
- Transmitter initiates and then the responder responds
- They both have an agreement and a confirmation
- Simple transfer
- Receiver gets the characters in order because it is a <u>connected channel</u>
- From connected to closed
- Whatever you are supposed to do with these characters, you do it.
- CTP evolution
- As the protocol goes along, there is an increase in the shared state
- Gradually, those 1,000 characters go over there, and we go from non-connected to connected
- Limits of CTP
- Assumes a perfect channel
- We don't have perfect channels in the real world
- Someone can be very good, but no channel is perfect
- Possibility of loss, reordering and issues of flow control
- Inefficient way to move data
- Once we closed the channel, what do we know?
- The message!
- We have been building up shared state
- When we are done, we have the complete message
- Back to predictability
- We know a bit more than we think, if we have 1,000 characters
- We have another choice rather than sending 1 at a time
- Group them together
- Make a larger collection and send the collection
- Confirm what we got using "checksum"
- Mathematical derivation of the bits, used as confirmation
- Don't need checksums in a perfect channel
- But we don't have perfect channels in Real Life
- File Transfer Protocol (FTP)

- Move a whole lot of data in one chunk and do what the real FTP does
- Deals with channels that are not perfect
- Starting Point
- Using TCP port 22
- Be handled in a particular way
- From not-connected to connected
- TCP is an existing protocol
- It takes care of the initial connection step, using something kind of akin to two sides
- Transmitter sends SYN
- Receiver sends SYN-ACK
- Similar to the same protocol of a phone call - someone says “Hello” to initiate conversation
- Once confirmed, they can start doing stuff
- Perfect channel vs imperfect channel makes no difference in this scenario
- Imperfect Channel?
- Characters and information can get dropped
- “Lo and behold”
- “Some characters might be dropped
- “H e l l o” -> “H e l l”
- The issue we have is how do we know whether something went wrong or not
- Transfer a block of data at a time
- Get a collection of characters and put them in a block
- We want a whole set of bits that represent the whole state of characters
- The block of data represents all three characters
- Additionally, we will have a checksum
- This is a summed number of bits, send more bits than we thought before
- We send more characters
- Checksum is not fresh information
- If we send across this block, and there is no change
- We have only transferred four chars of information, but we used up five characters on the line
- Perfect channel - useless overhead
- Imperfect channel - use checksum to determine if our transfer was successful
- Effectively, the checksum has made the channel somewhat more perfect
- Write an encoding to send a different panel of data so it is closer to perfect
- If we don’t know if the data was possibly the same
- We probably wouldn’t have gotten the error at all
- Some forms of checksums are error collecting code
- Even with error collecting code -> there is a limit of how many errors you are dealing with
- You are either ignoring it, throwing it away
- OR, you request a retransmission

- Find the checksum to be the same, but the message is not
- Usually certain parts of the message will match the checksum
- You can increase the size of the checksum
- Better transfer
- Successively increasing shared state
- Agree on the checksum that is sent and agree to end the transfer
- Similar to what we did with earlier protocol when we during one character at a time
- FTP leap of faith
- Assume that if we divide up all our characters into blocks and we send a block
- If the checksum is correct on the other side
- We assume that not only is the checksum correct, which it might not be
- We also assume that by having gotten the block, we have gotten the overall message
- You could have an entire situation where the block disappears
- It dropped the entire block!
- You could have sequence numbers but this adds additional overhead
- This doesn't work with if the last block doesn't show up
- The next thing that comes in is if we have a closed connection
- If we know how many blocks we have sent, we can close the connection
- Deal with protocols as we handle these issues
- Many things that can go wrong with this network
- Sequence numbers can also solve issues of characters arriving out of order
- Reordering - direct physical links that reorder things
- Pony Express rider -> three letters
- Order really matters!
- Electrical links could mess up in complex systems of systems and routers
- Other states we can add
- How many outstanding messages
- Depends on characteristic of the link
- We can keep sending messages
- If we know the link can only hold a certain amount of data
- Pacing - presumably a receiver and we can deal with it at a certain rate
- Do something at a certain rate
- Do it at a certain amount of time, eventually the receiver gets overloaded
- Let's try to match the speed of the sender and receiver
- Issue of reordering
- We need something built into the state that handles the issue of something arriving out of order
- What to assume
- As little as possible!
- Postel Principle
- UCLA graduate who came up with a principle of how to design networking application

- Sender
- Be conservative
- Follow rules completely carefully and exactly right
- Only send necessary information
- If you don't think he will understand what you will send, don't send it
- Receiver
- Be liberal in what you tolerate
- If it is an error, take care of recovery mechanisms but don't take it personally
- Why this function?
- Continuous function
- Let's say we have *n* different possibilities
- If the probabilities are equal i.e. 1/n, entropy will increase with *n*
- More disorder in the system
- View this kind of thing to be a set of weighted sums
- Three things you might send -> 1/2 probability of the first, 1/3 probability of the second, 1/6 probability of the third
- Let's assume we are in the start state
- There is a probability we are sending something else
- 2/3 probability of sending something else
- Unbalanced tree of 32 choices
- 32 different symbols in which we want to communicate
- Not equally likely
- One of them could be sent half the time, and another could be sent a quarter of the time
- At any rate, what are the chance see will send that probable symbol
- Are we sending that symbol or any other symbol?
- If there are any other symbols, we have a 50% chance we are sending any of the other 31
- It turns out if we aren't doing that other one, we need more distinction for what is going to happen
- We need one bit that is going further to distinguish between bits for this action
- Clear up this example:
- Remember it is unbalanced
- Character that is 50% likely to occur
- There are also some characters that are fairly likely to occur
- They can occur < 50% of the time
- Can be a fairly unbalanced set
- There could be fairly popular characters and there is a chance it can be that
- If there are two of those
- They could have a 50% being popular characters
- There is only 1/4 chances that it can be a moderately popular characters
- If it is one of the moderately popular characters
- There are only a couple of possibilities that are fairly likely to occur

- We can say it is not one of the popular ones
- Gradually as we go down, we can use a few more bits
- We can have a bunch of bits for unpopular characters and less for more popular characters
- We want to send a short sequence of characters to see what the most popular things are
- These things will be sent very infrequently
- Given the average, how many bits do you need to actually encode the data I am sending
- Because it is unbalanced
- If each one is equally likely, how many bits will we need to see which of the 32 things we need
- How many bits do we need? We will need 5!
- If it unbalanced, we don't need 5, there is REDUNDANCY in this data, and if there are encodings, we can take advantage of this and use less bits
- Unbalanced tree and get a very short
- Good for computer communications
- We can reduce the number of bits and make more efficient use of our channels
- Depends on how your tree is unbalanced
- 50% repeatedly
- Unbalanced Tree - weighted sum
- Look at slide
- Balanced tree of 32 choices
- On average you need 5 bits because you have 2 possible options
- Balanced tree - weighted sum
- For 32 choices, it can come out to 5 and Shannon's equation has told us exactly how much entropy there is there
- This is the maximum entropy possible for 32 choices
- Whenever you have a number of choices *n,* it is equally likely
- If it is equally likely, it is 5 bits, since there are 32 possibilities, and there is 2^5
- The probability of each one is 1/32 and it turns out H = 5
- What if only 1 of 32 was possible?
- There is only 1 and you never send any other
- This means there is an entropy of 0 since there is no disorder
- No communication channel since the receiver already knows you will send a certain character
- The Perfect Channel
- Information Source
- Receiver will send encoded information and give it off to the destination
- The channel never alters the encoded information
- The transmitter and receiver always covert perfectly back and forth
- The Real World (The Noisy Channel)
- Here we start with an information source and destination as usual
- We have raw information in between

- We have encoded info in between, and we have some noise injected in channel
- Changes encoding into something different
- This is something you convert into RAW information
- Not the same as the raw information that got sent
- Among other things
- You have had an imperfect transfer of state from one side to another
- You have not decreased the entropy as much as you thought
- What is a noisy channel?
- A channel that introduces errors into transmission
- Not always what is received
- Could be anything whatsoever
- Could happen sometimes on a schedule and/or probabilistically
- We would like to understand how much information can be shelved through that noisy channel
- How much information can you get through then?
- May end up with same number of bits on the other side
- Corrupted bits will not be useful
- When you have a noisy channel, you have a condition called equivocation
- If we have equivocation, what we got is not necessarily what was sent, but it might be something different
- There is more than one possibility of what might be sent
- If we get a 0 and that's what is intended, no equivocation
- If we get a 0 and that means 1, no equivocation still
- No equivocation is just a consistent encoding scheme
- 50% change and inconsistency -> there IS equivocation
- From a theoretical perspective, we should know what we have based on what we receive
- Difficult to deal with -> on the y-axis we have what we sent and on the x-xis we have what we received
- Perfect Channel
- Looks like an XOR because there is NO equivocation
- Perfect channel has no noise
- Merge things
- Total noise - no communication can be possible -> no matter what is sent
- The receiver canNOT be absolutely sure if the sender intended to send a 0 or 1
- The fact that the sender is sending something is NOT necessarily communication
- If noise in channel means that you see a 1, we know that something was sent and there is no communication possible.
- This indicates we are using a different channel
- Split things
- More complex form of noise
- Capacity of this channel?
- Parts of Shannon's paper will cover the way to do these calculations

- The rate of channel is the entropy of source (x) + entropy of destination (y) - entropy of source AND destination (x, y)
- Can calculate this entropy via information from the table
- Half the time he is sending a 0 and the other half, he is sending another symbol
- Do this based on the information from the Shannon paper and it comes out to approximately .81
- Read Shannon's paper and don't be a lazy bitch LOL
- After you do the reading, try to come out with the numbers

Working it out

- We are effectively communicating around 1/3 of a bit per second
- We aren't really sure if the symbol is correct, we don't have 100% confidence in this kind of situation
- Half the time you are going to send a 1
- If you do nothing further, nothing gets sent, and noise changes data
- No matter how clever you are, there is a limit to the amount of information you can push to the system
- On average, you will need ~3 1/3 encoded bits to transmit 1 actual bit more reliably
- .31 bits per second
- If you are really clever, you can get about 1/3 of a symbol per second
- Summary
- Communication is < most think
- Match the state at the sending end
- Information about the state can be based on entropy
- Character by character protocol
- Transfer information from sender to the receiver
- Use encoding to improve the ability to see information across a noisy channel
- Noise means many things
- Error
- Alters the relatively probability of what is sent vs what is received
- Instead of symbols, you would have an N x N matrix depending on the # of symbols you are capable of sending
- Two different symbols
- Destroy any symbols the sender tries to express
- Shannon limit
- Signal Power (S): How hard you are pushing
- SNR example - ADSL
- 40 decimals, which converts to S/N of 10,000
- Bandwidth is 1 MHz
- ADSL is capable of moving 13 Mbps
- They actually get 12 Mbps in real life
- There are upper limits
- Bump up S/N ratio in order to get more Mbps
- Error vs. delay

- Noisy channel coding theorem - Shannon's theorem
- Possibility of achieving the channel capacity
- Assuming we have a desired information rate *R*
- Send R < C, there will be a given code with probability of error epsilon where you can transmit information without error
- If R > C, such transmission is possible
- Shannon theorem again
- For a large enough message link in N, an entire probability of block error is <= epsilon
- If you encode it in a sufficiently long encoding, this would be < the capacity of the channel
- If the rate that I am trying to push through is > capacity, not gonna happen bruh!
- There is a possibility that this information won't be passed through without error
- If we take a long enough sequence of data and we encode something
- This will be received and understood with possible noise injected at the other side
- Consider a message of length N
- To get through this noisy channel without error, we have to encode it at a certain length
- Encoding takes time and it is related to the total # of symbols that needs to be sent
- Encoding time: you cannot send anything yet
- Compute: think to figure out what your proper encoding should be
- Emit: Information should go across to the other channel
- Why the delay?
- Even when computation is arbitrarily fast, the first bit of encoding needs to be customized to a set of symbols
- You cannot know what your encoding will be until you have all your symbols in the block
- Delay across the wire and how long it takes to encode
- If you have smaller blocks, you will encode it faster
- Shannon limit, and we don't need a smaller block to reach near the Shannon limit
- You don't have as much data going through the channel
- You will be able to get a lot of data out the door and get near maximal use of the channel
- Don't have to encode long blocks of data and you get to send the data out real quickly -> won't get as much capacity out of the channel
- What is "encoding a block"
- This means I believe I will get some noise, and we better compensate for that noise and see which symbols get alternated by the noise
- Add information into my encoding to allow you to detect incorrect errors
- Parity
- We are going to ensure there is an EVEN (or ODD) # of 1's

- This is used to help ensure a certain condition is always met
- When you add 1 bit to whatever your block length is, you want to make it an even # of 1’s
- The receiving end
- If the answer is YES, there is no signal bit that was flipped
- Receiver checks if the pattern has a correct # of bits
- Q. What if the parity bit added screwed shit up?
- A. It turns out if the parity bit wasn’t added at all, it would screw things up.
- Parity doesn’t tell you where an error occurred, it just said it has an error!
- You are always going to add a single bit for parity
- What can parity help with?
- Error detection
- If there are 5 bit blocks before, there could be 10,000 bit blocks and this means there will be an overhead of one extra bit if there is parity
- If on the other hand, we divide up into 100 bit blocks, and only one of those blocks had an error, then the only one we would have a problem with would be parity blocks
- If on the other hand, we use a 10,000 bit block, the whole 10,000 bit block is in question
- The whole 10,000 bits will be lost
- Parity cannot detect the 100 bit block and see if it is a problem
- Cutdown on the overhead and the cost of the problem becomes higher
- There is a limit here and you won’t be able to detect the number of errors
- If you have a kind of noise that might flip two bits, parity probably won’t do it for you
- Majority
- Send each bit multiple times
- Send 01011 -> send it 3x in a row
- One of the bits gets flipped in this case
- Majority of them are the same, but some of them are NOT the same
- If you have a fairly high probability of bit flips, you will use this because it will tell you what is wrong
- Some channels will have a kind of noise that affects individual symbols -> others will have noise that affects more than one symbol that is sent
- Overhead is immense in this case
- Good for Error Detection
- Hamming
- Let’s have multiple parity bits covering a particular piece of data to ensure it does not get altered
- Let’s be clever how we use it
- Don’t just have 3 parity bits
- Divide up data into subsets of bits and use parity bits for each subset of bits at a cost much less than the majority approach we just covered
- Receiver then checks to see if each of the sets of bits and if each is correct or not correct.

- Hamming algorithm
- All positions with one 1 in binary form are parity bits
- These are all powers of 2!
- What should these be set to?

W 1 Discussion 1-8-16

0. Web server (2 - 3 weeks)
- The project has not been scheduled yet
- Assigned in Week 3
- Should take 1 or 2 weeks
- Any C++ libraries we can use
- Pre-configured VM: http://metro.cs.ucla.edu/cs118/CS118.rar
- The spec will be discussed in two weeks

0. Reliable data transfer
- selective repeat on UDP
- congestion control (extra credit)

These two projects are individual projects but the 2nd one is a little bit harder than the first one

- Ask Reiher if the second project can become a group project
- We need to make groups of 2 -> 2 members can make this second project
- Testing -> submitting the web server code and the TA will test it himself.
- The 2nd project will probably have a demo and he will announce the demo schedule
- 15 minutes to show how it works

Three questions to answer

- What is the model for network programming?
- Where are we programming?
- Which APIs can we use? How can we use them?

- What is the model for network programming?

Client-Server Model

- Clients are users
- Server i.e. Google, CNN, UCLA server
- This model is asymmetric
- Client only requests the data
- Server responds with the requested data
- Clients initiate communication and wait for the server response
- When the server receives the request, it responds with data and it keeps well known IP addresses and port numbers
- In this model, client and server are not disjoint
- This means the client can be a server and vice versa
- You can run a personal server using your laptop
- Your laptop can also be a client if you request it from another server
- Server's service model

- Concurrent: server can process multiple clients' requests simultaneously
- Sequential: server simply processes one by one
- Hybrid: combines both features with the server maintaining multiple connections, but responses sequentially
- Project
- Request only one file and the server will request that one file
- Practice for socket programming
- Show it on your browser
- This is different from socket programming
- Underneath the application layer and this is the server's different types of service models
- Whenever you want to make a connection, you have to use socket programming

Where are we programming?

- Physical layer
- Data link layer
- Network layer
- Transport layer
- Application layer - client and server
- Typical transport protocols
- TCP
- UDP
- Socket API provides some functions that can generate sockets and put data into your socket
- From there, you can generate some packets and you can send it to the server
- Difference between TCP and UDP
- TCP is more reliable than UDP
- For the second project, you are implementing a reliable protocol on UDP
- Add features so the package won't be lost
- If we have a reliable transfer protocol, why do we also have an unreliable one
- If we have realtime video or audio applications, in that application, you don't need to use TCP because packet loss does NOT affect it too much.
- UDP is used for real-time applications and if a packet is lost, TCP tries to retransmit the package and it takes time.
- UDP doesn't want to take time if the application is real-time.
- When you see a video clip on Youtube, sometimes the quality of the video is not that good
- This is due to packet loss or packet delay
- The network doesn't want to wait to retransmit it.
- TCP: Transmission Control Protocol
- TCP transfers the data reliably and there are multiple TCP packets
- The application itself has to resemble the data, and this fragmented data is already ordered and there are no duplicates.

- TCP also controls flow control and congestion control
- If there is too much traffic on the network, packets can be lost.
- TCP can control the amount of traffic to control this kind of congestion.
- Retransmit the lost packages
- Why will packets be lost if there is too much traffic on the network
- TCP controls network traffic/congestion
- It doesn't care about the network condition
- TCP has a long header - and in that header - it has a receiver for buffer information
- Sender can always control its flow
- Full-duplex **byte stream**: there are two hosts - sender and receiver
- When sender is sending the data, receiver can also send its data
- Two way communication
- UDP: User Datagram Protocol
- If there is no flow control and no congestion control, UDP is still used in some real-time protocols
- DNS uses UDP
- There are duplicate servers and it cares more about the time between host and server
- If one server fails, it does not retransmit the weakest file again
- Instead, it will use another server

Which API's can we use? How to use them?

- What is socket programming?
- Each device has an IP address
- On top of IP, there are transport protocols such as TCP and UDP
- These sockets control TCP, UDP and allow IP level transfer
- TCP Socket
- Creates socket
- Establishes TCP connection and allows people to send and receive data
- Next week, he will talk about the socket programming examples in more detail
- Setup is included in the VM file
- You can run sample code and use this for the project
- It has already been implemented with some parts, so why reinvent the wheel?
- Are there other standard protocols besides TCP and UDP?

W 2 T Lec 1-12-16

- Review Comm. as shared state
- Information is based on states
- Describes the information that the sender would like to have
- Get the syntax of that state over to the receiver
- States change - that is the expectation
- States are going to evolve
- We are trying to build a model of what we got over there

- Sender should have understanding of how successful he was
- Has he gotten the information or not?
- Go through steps in which states change on both sides
- For practical communication, we have noise
- Something that gets in the way and causes information to be dropped
- This means that what we sent may not be precisely what we received
- There are ways we can overcome this by encoding information
- Let's encode the information we want to send
- We are more likely to be able to transfer the information successfully despite noise in the middle
- Communication: Roadmap
- The imperfect channel
- How to make channels from real mechanism
- Designing a computer system that automates the use of the channel
- What is "encoding a block"
- Noisy channels
- We want an encoding
- What we typically do is to take a block of data that we want to send i.e. bits of data and combine those into a single block of data
- We will then add some information and we will take what we want to send and put it into the channel
- The wire can raise and lower the voltage
- It is also possible to encode by adding bits that you will send over the channel
- We expect there is a possibility that it will affect some of the bits we are sending across the channel
- Methods
- Parity
- Majority
- Hamming
- Reed-Solomon
- Parity
- Count how many 1's you have and make sure you always send an EVEN # of 1's
- Always having an extra bit that is one size larger
- Let's put it at the end and then count up the # of bits
- If the # of 1's is even, put a 0 in the extra bit
- When you consider the parity bit, there should always be an EVEN # of 1's because you can CONTROL the parity bit
- If there is exactly 1 bit flipped, then you can detect this at the destination
- We are assuming that just 1 bit has changed
- We will change the # of 1's we have in the block of data
- All the bits for the blocks of data got through properly
- Regardless of whichever bit got flipped, there is still going to be the wrong # of 1's in the encoding version

- We check on the receiver end -> parity is not helpful unless the receiver knows you are doing parity
- If you get 010111, this is GOOD because you have an even # of 1's
- If you get 011111, this is BAD because something got flipped
- Parity is an error detection technique
- Detects any ODD # of bit errors on block
- Cost is cheap
- 1 extra bit per block
- Limits
- Doesn't know which bit got flipped
- Won't detect any EVEN # of errors
- If you use just parity, the best you get is detection
- Majority
- Send every bit several times
- i.e. you can send it three times
- 01011 -> 000111000111111
- If you know the ordering got changed, then you know something got flipped
- There are more of one than the other if it is an odd #
- Assume that the majority is right
- 010 -> assume this is 0
- 110 -> assume this is 1
- Error detection technique primarily but can also be error correction IF k < N/2
- Make an improper correction -> send every bit multiple times
- Gets expensive since it is N times longer. so it has a very high cost
- Typically not done
- Hamming
- Makes use of parity but also has multiple parity events and you are clever about how you use the parity bits
- Has some algorithm and take the bits of that data into sets
- Parity bits deal with only individual sets
- Each parity bit will compute things within the subset in question
- You can tell exactly what bit had the bit flip
- A Hamming Code Example
- Encode a 15 bit data item
- Noisy channel but not only detect, but also correct an erroneous bit
- Add 5 parity bits
- Do not use them as a single 5-bit #
- d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11 d12 d13 d14 d15
- Scatter 5 parity bits to the 15 data bits
- Which ones are going to be data bits and which will be parity bits?
- An example would be picking any bit whose binary value for position contains only one 1
- Any bit that has exactly one 1 in its position # will be a parity bit
- Everything else is a data bit

- Bit 1 (1), Bit 2 (10), Bit 4 (100), Bit 8 (1000), Bit 16 (10000)
- Powers of 2!
- Take data bits and put them in order of whatever positions are left over
- p1 p2 d1 p4 d2 d3 d4 p8 d5 d6 d7 d8 d9 d10 d11 p16 d12 d13 d14 d15
- Put binary identities of each position
- 5 parity bits
- How to compute values of these 5 parity bits
- For whatever parity bit, consider what position it is in
- Anybody who has a 1 in their position #
- p1 will be computed over itself
- p3 has 1 in its least significant bit, so it will be considered in the computation
- Parity bits cover whatever subset they have in common
- Count how many 1's that they have in common
- 1 maps to least significant digit
- 2 maps to 2nd least significant digit
- 4 maps to 3rd least significant digit
- and so on and so forth
- What does this buy us?
- It turns out that if we do it in this way, not only will parity bits be wrong, but also there is an encoding that tells us which bit gets flipped
- Hamming code example
- Compute parity bits using techniques we just discussed
- Red bits are the parity bit competed for this purpose
- If all parity bits match, you can get a block without noise
- What if there is an error?
- What if a single bit got flipped?
- Say, bit #3 changes from 0 to 1
- Now compute the parities on what you got
- Assume they receive a 2nd bit using the hamming code rules
- 0 1 1 1 0
- But in the message we have 1 0 1 1 0
- We get even more info from those 4 extra bits
- Out of the parity bits that I got, which parity bits did it match?
- Some matched, some did NOT
- The parity bits don't match 0 <-> 1 and 1 <-> 0
- The 0 should have been a 1
- That one that should have been a 0
- Bit flip finds exactly the position using the bit flip
- If you know which position got flipped and there are only 3 possibilities, you can correct that error
- Strip out the parity bits and tell the receiver your 15 bit block
- If parity bits are placed within the data bits, the parity bits have a chance of being wrong
- It will work out even if parity bits are affected by the #'s
- You know if a parity bit gets flipped using this technique

- If you work through the example, you will discover that you got an incorrect value!
- Hamming
- Error detection
- Does not correct 2 bit errors
- Error correction
- The limits are that it cannot correct more than one error
- We tend to NOT see single bit flips in a range communication but rather many bits
- Bits 4, 5, and 6 might all get changed
- Reed-Solomon
- A lot more complicated and won't really be discussed
- Adds t bits of overhead
- Works for burst (consecutive) errors too
- Noise in a wireless network will cover multiple consecutive bits
- Error vs. Loss
- Error
- Nothing is received
- How do you detect loss
- Time lets us know -> we wait and it doesn't come, after a particular period of time, we have lost whatever was supposed to be sent
- Some problem that will come eventually and if we are waiting for a check
- We can get tired and you lose it yourself
- Also possible that the envelope containing the check was lost forever
- Dealing with loss
- How to fix the problem
- Send it again and see the request going back to the sender
- Going across the network -> frequently the same network
- Go to some similar channel in the reverse direction to detect a particular problem of loss
- ARQ
- Sender
- Collects information into a block and sends that information as a whole block
- Sends in a block like fashion
- Slap an ID onto every block
- Necessary because the receiver has to be able to tell what they didn't get
- Have some ID associated with a block for an ARQ request to work
- Receiver
- Has to be able to recreate the block out of nothing
- In order to resend it, he has to have that data
- ID has more bits -> adds overhead
- Hold onto a copy of that block of data + ID in order to resend it
- How long do we keep it?
- Depends on exact technique we are using
- Sending blocks of data with IDs on hit

- Keep looking for missing blocks of data
- Most common way is to use consecutive IDs for a block
- Receiver gets blocks and can tell the sender which block is MISSING
- Sends general feedback based on what they receive and what they still need
- Receiver is able to ask sender for assistance
- Requires a *reverse channel*
- ARQ Variants
- Negative acknowledgements
- Let's you know what you are missing
- The message that is sent on the reverse channel -> this block has not arrived
- Lacking in confidence that it is coming any time soon
- The receiver reports what is missing
- Presumed to be gone and never coming
- Sender resends those explicit requests based on missing IDs
- Positive acknowledgements
- Sends a message saying they got the correct block
- Based on ID of the block they received
- At this point, the sender can take a look and say which ones they have received
- Sender look for gaps in the blocks
- This indicates loss
- Because the sender sees the ACKS coming back, they will assume ones NOT reported are MISSING
- Receiver can know they are not missing anything, it is just a waste of bandwidth on the forward channel
- Does anyone ever use both positive and negative acknowledgments?
- Generally one tries to build protocols as simple as positive, but it is possible. It is just unlikely because all this mechanism is embedded in code and this is run in a computer
- You occasionally have bugs in your code and you don't want to add too much complexity (overhead)
- As you get into more complex protocols, there can be more situations that can arise.
- You may have situations that are unlikely but could happen where you haven't really that thought through what could have occurred.
- What situations would you use positive feedback?
- Assuming you have a noise-free channel, negative feedback would be perfect
- Negative feedback is good potentially when you have a situation with a relatively low-probability of error
- 99.9% the block gets through, but every so often the block gets through
- If it is relatively common for the block to get lost, you generally want to use positive feedback
- Do you want to minimize the # of requests you send back?

- Acknowledgments are affecting the state of sender and receiver
- Share information about the data flow and how it is going?
- It really depends on your situation to generate an accurate picture to have some kind of back flow of information
- You want to know the information based on receiver's feedback!
- Variants of ARQ
- Stop-and-go
- Block has 1 bit # and if you want to send 5000 blocks, there will be an issue there
- This variant uses the positive feedback approach
- You send an ACK for every block that you get
- Constant feedback of acknowledgement
- You can only have 1 block of data outstanding at any given time
- Needs to hear about the first block of data
- Go-back-N
- Also positive feedback
- Variable length ID
- If it is the case that he gets an out of order block, we can't make any further progress
- I am going to send back information that he did not get that block
- Tell the sender we have to go back
- Start over and don't just resend all the other remaining, the block needs to be refilled
- It doesn't recheck what you have, it resets everything STARTED from the dropped block
- You know that the person at the receiving end -> has a lot of outstanding blocks at any given time
- As long as timeouts don't occur, it will be okay
- When there is a loss of block, there is overhead that goes along with it
- You get either an ACK for every block, some period of time after you send it, you will resend the block
- Selective repeat
- If you get an explicit NACK, they will resend the block
- Typically uses much larger ID space
- You don't want to deliver the blocks because you want to say you want to save those away and eventually the lost block will come in again OR it will time out.
- More complex state behavior for the receiver
- Buffer information because some of the information he needs has not come in the order he wants it to come in
- Reordering as error
- In addition to messages being corrupted or lost, it is possible for messages to come out of order
- This is relatively uncommon especially without timeouts
- If we want to handle that situation without retransmission, you need to buffer the data that comes in
- Basic components of a channel

- We need a way to affect properties of a channel and some ways of observing the channel to see what the channel's cost
- Signal
- Media
- Set of symbols
- A method of generating and receiving symbols
- Direction associated with channel
- Determines sender and receiver
- How to create signals?
- Learned in M117
- Move photons around
- Move electrons around
- Move atoms
- Water flow (not too practical)
- Types of media
- Characteristic types of media:
- Guided
- Transparent: fibers
- Some of it is electrically conductive
- Unguided
- Not much of a direction if any: free space
- Often done mechanically
- Freespace
- Unguided
- Mechanically conductive
- Transparent
- Use pattern of ripples
- Signal tends to degrade over distance
- Needs a clear path
- Some way to get from here to there without a big obstacle appearing in the middle
- Chunks of metal, mountains, etc.
- You can bounce signals in the atmosphere
- Needs to be some path you can follow
- Fibers
- Multi-mode
- Thick core
- Multiple channels on the same fiber
- Different wavelengths carried by the same fiber
- Single-mode
- Thinner core
- Fewer paths
- Potentially valuable because you can get greater distances from these
- Hollow core
- Characteristic that is like free space because it goes directly from source to destination

- Wires
- Guided
- Conductive
- Made out of superconductors or metal
- Number
- Single wire
- Multiple wire
- Communication symbols
- M117 gives a lot more information about the actual physical components
- How do we encode information on the channel
- How do we observe a channel to determine that a piece of information has been put on a channel
- Simplifies generation of information
- Some Example Encodings
- M117 material
- Different signal power/strength values indicate different symbols
- Symbol that indicates there is nothing there
- Null symbol which means we are not transmitting things
- See how the signal changes and be sure that the guy on the other hand will have different # of 0's
- Return to 0
- You have a high signal and you have a low signal
- If you go back to 0 volts, you have finished sending one signal and you are about to send another signal
- You could instead say you are going to put a value out for 1 microsecond
- The guy can understand the message encoding, and this implies a synchronization of clocks between sender and receiver
- May NOT be a feasible thing to do, it can be challenging to implement
- Generating and interpreting signals
- Generate a symbol pattern that corresponds to information patterns
- There is a way of putting out to the medium and it doesn't get translated on the other end
- Presumably at the other side, goes to the other side
- Interpretation is the same as the generation except in reverse
- Single channel (Direction)
- Receiver may not use that channel to send it
- Can the receiver support this feature or no?
- Simplex
- Transfers symbols in only one direction
- Data goes from sender to receiver, and receiver cannot send this information back the other way
- Duplex
- Simultaneously send signals in both directions
- You can use native bidirectional channels and people can simultaneously send information in both directions
- Different frequencies

- Half-Duplex Channel
- Share the channel
- The channel can have party A and party B
- Swap back and forth on who gets to receive on this channel
- Both have the capability of observing the channel
- Use a native bidirectional channel
- More frequently, you will have one channel that goes only in one direction and we wouldn't be able to read anybody's signals
- People will take turns and only A can send to B
- Rules in communication about who gets to speak next
- Military communications i.e. they say "over" to signal the end of a monologue
- One element of the protocol
- What Is a Protocol?
- A set of rules, agreed in advance, that enable communication
- Endpoints: the things we want to communicate
- Link: Enables action at a distance between the 2 endpoints
- Protocol: Specifies how to automate how these interact
- How to use these links
- Broadcast with many receivers
- How Do We Automate a Protocol?
- Computers have to be automated in order to make proper use of their communication channel
- Because a protocol is a fixed set of rules, we have the ability to automate this like a script
- You can specify the protocol as a finite state machine (or two in the case of sender and receiver)
- At any given moment, the machine is in one state (and exactly one state)
- Finite (and predefined) # of states
- You cannot generate new states all of a sudden
- Actions that are defined to have a transition to another state
- This is how we automate protocols
- Limits of FSMs
- Cannot count
- Finite state, so limits on count
- Why do we want a FSM?
- Keeps our state manageable
- Puts boundaries to prevent our state from becoming infinitely large
- Mealy machine
- These protocols get expressed as this form of FSM
- It has states and transitions
- In a Mealy machine, it has two things associated with it
- Input
- Output
- Networking situation: send a message to your partner on the other side
- Convenient way of expressing a network protocol

- Start state and you sit in state 1 until you get input b, send output y and then got to state 2
- Stay in state 2 until you get input a, and send out output x and go back to state 1
- Sharing simple state for networking
- The end is saying that A has some state and you want B to have the same state as A
- Communication is used to achieve the goal of sharing states between A and B
- Works fine if you have a perfect channel
- Simplest state
- Define two states: “round” and “funny”
- “A” decides to be in one of the two states
- “A” wants “B” to know which state “A” is in
- A gets to change state
- If we never change state, why bother setting up a protocol
- For some external reason, this might cause the sender’s state to change
- B has a similar state
- Has information that describes what is going on at model A
- Doesn’t matter if the names match or not
- B gets to change state too
- Changes based on input from A

How do we communicate?
- Each time A changes state, it lets B know
- Example:
- A and B machine on Page 51 in Lecture 3

When are we done?
- Depends on the situation, but if it’s a reliable channel, it WILL change state eventually
- Figure out a way to get this done eventually

Mutual state
- We could say that each of these two parties can have what the other guy’s state is
- Have some kind of idea that A knows B’s state and vice versa
- If not limited, can go into infinite loop
- Stop at one step
- Your state
- Your view of the other end’s state
- He has a view of a state and we can set up finite state machines

Simple Communication with confirmation
- New information gets generated at A

- Get to situation where A not only knows what his own state is, but also has an idea of B's state
- The vice versa is also applied
- We have not added more information A, it was just one UP/DOWN switch
- We have to have more states in this case to keep track of B
- Go to other state and go to intermediate state
- Send that message to B
- If you have a 0 message, you should go to state 0 and you also have an output
- Send a message called got0
- Now you know something about B's state
- This allows us to build the situation where every participant has an understanding of his or her own state
- Has a somewhat less perfect understanding of the other person's state
- He hopes he sends a Got1 message and this message could get lost if it was an imperfect channel
- A would have to say it got to that state
- B would also have to have more states in his state machine
- How far back should we go?
- How do we know if we should transition to another state?
- This is generally the point where we stop

Confirmation

- Channels are not perfect
- You may lose messages
- Errors can occur as well
- Detect when a message has come true and treat it as if you lost the message
- In certain circumstances, you can correct the message
- Treat it as if the message was NOT delivered at all
- Everything reduces to loss
- If it is a lost message, deal with it as a loss
- If you have a timer expire, you can give up but you don't have much in the way of choice

Time out on ACKs

- Timeouts add complexity
- Three-way handshake (TWHS)
- Send you a message, tells you I got the message
- Client-Server (Matt-Frank)
- Matt sends a signal to Frank
- Frank acknowledges to Matt that he got the signal
- Matt acknowledges to Frank that he got the confirmation (just in case scenario)
- We start with A at state 0 and B agrees

Specifying a protocol

- There are symbols
- Messages across a medium
- Something can happen and then a switch gets flipped
- A program can finish producing a result
- Events
- Incoming
- Outgoing
- Transition table
- Typically expressed as a state machine
- States at the endpoints
- There could be symbols incoming
- Change state -> communicate this change of state to your partner
- Little loops that tell you when the time expires
- Have a different output
- Output events
- What you send to your partner and send something to it
- Send message to partner and set a timer
- Transition table
- Maps events and states to other events and new states
- No magic jumping from one state to another

TCP state diagram

- State machines are common combinations of protocols
- If you understand the common things, it becomes easy to break things down in FSM
- There are a couple of handshakes here
- Sender and receiver agreeing on something
- Corner cases: At the start or at the end

Complication #2: sharing complex state

- What if we to share more than one bit?
- If you can share a SINGLE bit, you can figure out how to share more bits
- If you transfer one block of the file, you can ensure that each block that gets transferred will get transferred properly
- The guy on the other end if he gets all the smaller blocks will get the entire thing
- If all the individual chunks arrive, the guy on the other end can put them into the more complex piece of state.

What's the leap of faith?

- If we have a phone call, we agree on the blocks of bits set
- You can use a checksum, parity, etc.
- Identify each of the blocks with something like a sequence #
- Combine them into the complex state that we were trying to convey to a partner

- <u>Assume</u> the above sequence is true once the entire process has been executed

This 2-party stuff seems universal

- Applicable to a lot of circumstances
- All protocols should be described in terms of states, symbols, events, transition tables
- Under what circumstances do you transition states
- State diagrams can be looked at in full complexity
- Decompose into components and this can help you understand complex diagrams

Look at TCP

- States
- Connection status
- CLOSED, LISTEN, etc.
- Blocks transferred
- Fairly complex protocol
- Get an estimate of how long it takes to transfer data from sender to receiver
- Makes estimates and finds congestion between the wire between us
- Try to be a good guy, if we are congested, let's put less data and get less congestion
- Put information about how many blocks have been transferred
- Sender and receiver try to get into a mutual agreement
- Symbols
- Which block are you dealing with at this particular moment
- Events
- Input
- Messages arrive
- Timer expires
- Output
- Message departure
- Transition table

Why is this hard?

- Protocols can be large
- There can be many parts that leads to a lot of complexity
- There are familiar parts but there are lots of them
- There might be more than 2 parties that want to be communicating with each other
- Multicasts or broadcasts like news radios
- We may have multiple instantiations of the protocol
- Web browsing is an example that adds to the complications

Summary

- Noisy channels can be useful
- How to deal with loss (timeouts and retransmissions)
- How to detect loss (acknowledgments)
- Error detection and correction
- Computers can be automated
- High degree of precision of what needs to be done
- Protocols define rules that can be automated and implemented with FSM

W 2 R Lec 1-14-16
Multiparty Communications

- Outline
- Extending 2-party model to N-party
- I am want to send a lot of info to a whole lot of people -> broadcasting a lecture
- N-people who would all like to send info to one receiver
- Situations when you have multiples of information -> may require different treatments
- Shannon Channel
- Two preselected properties
- Don't change sender or receiver
- Unidirectional channel
- Shannon 2-party communication
- Start off knowing a great deal
- Participants, communication channel, actual ability to put info on the sending end and receive info
- May have to deal with noise and loss but we know that there is a sender and receiver guaranteed
- We have mechanisms to obtain additional information
- When did the endpoints share state?
- Communication between two people
- Dealing with timers
- Add a little bit of information on how to add the Shannon channel
- Decoupling party from channel
- There isn't usually one sender and one receiver
- Care about more general communication if there are other parties
- If you are a sender and there are many parties you want to communicate with
- Post on Twitter, sell on eBay, look up on Wikipedia….
- It doesn't make sense to have a permanent, always-on channel to each of them since you aren't in constant access with them ALL THE TIME
- We want to be able to say we can choose who we want at the other end of the channel
- Have some flexibility on who you are working with
- Channel vs. party
- Shannon channel
- Integrated with the endpoint (party)

- No choices - all information sent/received uses the only channel there is
- Cannot change who we are communicating with
- Separating the two
- Need to change who the endpoint is depending on what we want to do
- Abstract network components
- Endpoint
- Source or sink of information
- Links
- Like a channel
- Moves information from one place to another
- Components
- Shannon
- Party
- Channel
- Information
- 2-party interaction
- Multiparty, modern terms
- Endpoint, node, host
- Link, hop
- State, data
- N-party interaction
- We need to have networking that can support this generality
- Multiparty extensions
- Extend the system to specify whoever you are talking to
- Differentiate the receivers
- Make these names work not only the setting where people talk to each other
- Must work in setting of network communication
- Name could be an IP4 address
- Take this and allow this to specify who we are talking to.
- Getting through communication links for the party we are communicating to.
- We typically end up communicating to multiple parties at the same time
- Multiplex talking to different parties
- Web browsing
- Every webpage results in communicating to 20-40 different computers
- A lot of this is abstracted away from you under the covers
- The computer has to switch back and forth very quickly
- Keep everything properly connected
- Deal with things in the proper way and updates the state depending on the site
- You need some mechanism where there is some receiver on your system
- Network card sits on the BUS and sends messages to the CPU
- Go from that piece of information on the network card and send it to one of the 40 different parties we are communicating with.

- If you deliver the wrong information from the network, no one is going to know what the message is intended!
- Differentiate the parties remotely
- Sockets are a common way of doing this
- OS entity (111 type of thing)
- This is when the process is able to specify the party they would like to communicate with.
- They expect to communicate when there is a complete set of networking equipment
- Juggle between multiple senders
- Simultaneously deal with many different senders
- Talking to multiple parties
- Multicast and a broadcast
- Limited to the scope of the channel (in CS 118, it is the room we are in)
- Specify for a given network what we are saying
- Specify here are N-different parties -> send it to N parties and the other parties will not receive the message
- Multiparty
- Multiple endpoints
- Multiple senders, multiple receivers
- One option is complete connectivity
- For every party we would like to communicate to, we will have a Shannon channel to that receiver.
- This is the situation we have there, a lot of different separate channels
- This is a moderately busy network with 6 nodes
- Multiparty assumptions
- Every single channel is disjoint from all the others
- Other parties cannot hear what is going on in the other channels
- They aren't interfering with each other
- Disjoint in the state of the channel
- Every channel has its own input and its own output
- Why is this networking?
- No longer a single communication between one party and another single party
- You have N^2 channels, and you have to do complicated things to figure out what messages you want to do with it.
- Varying sets of parties and this is a small increment for the moment
- Everybody connects to the other.
- You don't have to ask someone to relay or store a message to another person.
- Still only one protocol and there are direct 2-party channels
- Everyone of them is independent
- Importance of multiparty
- Varying participants
- You get a lot more complexity
- Deciding where our message will go is more complex

- More complex state
- My single node could be communicating to several other nodes simultaneously
- Telling everybody about one large state
- If you communicate with N different parties, you have N different substates
- This is more commonly an approximation of what you can do
- Have a pretty good understanding of the communication with them
- Overlap
- Solve all the complex problems
- The power we get is vastly larger if we accept this vs a Shannon channel
- The need for names
- N participants (implies complete connectivity)
- Each node can interact with N - 1 nodes
- Once we put a symbol onto that channel
- At some point, which one are we going to use
- This symbol to this channel
- Some kind of identifier
- What can the name apply to?
- Identifiers can mean different things
- We can say everyone of our senders has N-1 channels and N-1 names
- We can also say endpoints have names
- In this case, the names that the red sender will have will apply to the green receiver and the blue receiver
- If we apply the name to the endpoint and we have another sender, won't they all have names that apply to the receivers or the channels?
- Potentially, we can use one name for a receiver that applies to all the senders, BUT we need to DISTINGUISH between different **CHANNELS**
- On an N x N graph, there is a channel:endpoint connectivity that is 1:1
- **We would have different names for channels, but endpoints can be shared**
- Names for receivers
- Standard is to name the nodes; we generally do NOT name the channels
- If you are talking about Shannon communications, forget about commonalities that they send it to the same guy
- We are talking about computer communication
- We have to have an internal name to describe the endpoint we are communicating with
- Could simply be an index
- One way of doing this
- Tends to say that we have a correspondence with the channel endpoint
- We could have a more general table-orientation
- Port
- Something we send to on another computer
- Helpful because we know that would happen on the other end
- There is not one piece of code doing everything

- Rather there are a bunch of different little processes
- Used for computer communication resources
- In terms of the code running on the machine, there is one processor over here that wants to talk to a different processor somewhere else
- For example, on your web browser, it is your browser that communicates to the server on the other side
- You don't expect an ad from another website to show up on your current process
- The web browser divides things up into separate processes so they do NOT interfere with one another
- Ports are commonly used for this purpose
- Channels
- Used more generically
- Wireless network - channel that is very local with implications you care about
- Many of our computer communications are NOT channels at all
- We are communicating across multiple channels that are connected with each other
- We want higher-level abstractions and we are talking to each other about something we are thinking about as a channel
- Typically, we also connect up to OS entities on each end
- Multiplex at the OS level and say where it is going to go.
- Socket
- Represent on-going communication between two parties
- Designed in 1974 for TCP
- One end of a 2-party communication
- Unix/BSD said that we need sockets for many different purposes
- From OS perspective, big data structure for the benefit of a local process
- Has a lot of state information about actions that are being performed
- Asks OS to set things up and sending messages onto the socket
- Change the communication properties of the socket, and so on.
- Nameable for the socket
- Port vs Socket
- Port is a place on the separate machine where you can communicate simultaneously
- Every single one has an abstraction
- Sockets are known about by the sending OS and the receiving OS by the sending process and receiving process, respectively
- Ports are known by multiple browsers/machines, but sockets are only known internally to the browser/machine that it is located on
- Receiver naming requirements
- Fully connected topology
- We decide we are naming receivers
- Every sender will get N-1 receivers
- There must be a unique name for every receiver
- Not necessary to have unique names for receiver between two senders

- Each sender could have a set of N-1 names
- One sender does NOT care about names of the other sender
- Guy at web browser 1 does NOT care about guy at web browser 2
- We need a bunch of names that are unique for each individual sender
- We do NOT care if he has a different name for the different senders
- Each sender has a different name for each guy
- Receiver name examples
- One sender can name the other ends to know where it needs to go
- Another sender can do the same thing but the names do NOT need to be **agreed upon** for each receiver
- Each receiver can have a different name **DEPENDING ON THE PERSPECTIVE OF THE SENDER**
- Unique method of associating a receiver
- Multiple senders
- We can talk to multiple websites with communications for different people
- How to deal with multiple communications?
- State machine that describes the state of the protocol for every channel.
- Develop a complicated state machine
- There will be a separate state machine for every channel
- We have to make sure that when a signal comes in and causes a state transition, it causes a state transition within its own machine
- Decouple the channel rom the party itself
- Socket
- Endpoint within the party and it is "disembodied" from everything else
- Associate a state machine and apply that message
- No need to worry about interference between other machines
- What's inside the party?
- Channel is NOT always active at any given time
- Implies that every state is associated with a channel
- We will have further state associated with it.
- Part of a finite state machine
- This is going to be something that has to be maintained by the OS
- Socket will have to deal with input and output
- Deals with state transition and sending things to other parties
- Associating with individual connections is used to determine how to deal with incoming messages
- How many machines are there?
- We can model this as one FSM
- This gets quite complicated, however.
- We usually think of them as independent to simplify the process
- What we really do in real systems is we generate something that creates a new FSM
- Put it into the start state and handle the associated task
- Use multiprogramming techniques to run it concurrently
- Multithreading/multitasking potentially going on in this scenario
- So what else do we have to name?

- Process/thread identifier
- Port is one way of doing that.
- Tie that port to a process and then we know what goes to another process
- Not the same as a socket, we don’t necessarily need to keep track of what the process connects to.
- Why?
- Party could be a process which could be communicating to many different remote parties
- Many different state machines
- For a given process, we have one memory space
- We need permission to address a location in memory
- This is done by a per-process basis.
- When a message comes in, this fiddles around with one of the state machines
- The process has to do this right and the state machine might interfere with other state machines
- Yes, we want state machines to be disjoint, but it doesn’t have to be disjoint within the process
- There are changes in the state of the process may have an impact on the state machines for other connections
- Talking a bit more of Operating Systems sort of things
- Not totally disjoint topics
- A lot of overlap from the other class
- Internal naming requirements
- How unique?
- We are talking about an N x N case and we need a “FSM” for each N-1 people
- Name must be unique within that set
- Name doesn’t need to be unique within that set
- Sockets are basically numbers that are unique within the machine
- Each could have the same socket #
- Can mean different things on different machines
- Think of localhost: it can be different depending on what application you are serving at the port!
- On a particular machine, a particular socket is tied to a particular process at all times
- The numbers are recycled. They can be reused as soon as you kill a process for a different process!
- The numbers are generic identifiers; there really is no special association between process and socket!
- You just have a whole bunch of sockets and you allocate this socket to a particular resource as a single time.
- Summary of multiparty naming
- We need a way to pick which channel/receiver we are sending to.
- There is a one to one mapping

- Out of N-1 partners, we decide which one we want to go to.
- Each channel has an incoming message and we have some index that has the right place in the state machine
- Other nodes don NOT care too much about these
- Multiples of communications
- Each party usually wants to communicate to multiple other parties
- Send information to multiple parties more or less at the same time.
- Shannon Channel
- Only people who have any change in their state are sender and receiver
- State is shared, but only between sender and receiver
- Multiple receivers
- Broadcast (1:N)
- Send same info on all channels
- Multicast (1:M)
- Choose a subset M from your N parties
- The remaining N - M parties will be unaware of this communication
- Broadcast
- Shares state everywhere
- No need to pick because it goes to all the places
- State now shared
- If everything has gone well, they now share that state
- When do we know when everyone shares the state.
- Sooner or later there is a need to coordinate on that.
- Handshakes with everybody
- There are other approaches that might be more effective
- Chang's "Echo alg"
- A way of confirming that everyone has heard the broadcast
- Complexities of communication copying
- Atomicity
- Losses do NOT correlate across channels
- We would like to have atomic operations
- The ability to say that before this moment, nobody heard about my state
- None of the N-1 participants have heard about my state
- After this moment, the remaining parties will have known what has happened
- State has changed as a result of this broadcast
- Distributed systems problem
- Everyone echoes back when they learn about the state
- All the participants need to be on the same page!
- If we want to work with distributed systems, we do care about this issue
- Reaching this point is an issue we can atomically say that we changed it.
- Efficiency
- Multicast
- Connected to N-1 systems
- We have to have a mechanism of picking
- Does the sender or the receiver pick?

- In this topology, once we know who our multicast partners are, it is quite simple.
- Certain need to coordinate when we know when all parties have received it.
- Things get worse…
- We have a bunch of people who want to hear about the change in state
- Conference call analogy
- We want what anybody says on the conference call to be heard by everyone else
- Potential issues
- Hangups - don’t send messages to the guy who hung up
- Another guy sends in - Add the party and include him in future messages
- Multicast complexities
- Indications are done with some kind of name
- Who does the picking in whatever kind of multicast group
- Is it done by a receiver who allows someone to do something on the multicast group?
- Put up a general # and provide code
- Full pairwise connectivity
- One topology
- Every sender connects to every receiver
- The sender knows exactly what he needs to do
- This is very simple to understand because it only gets more complicated
- Expensive to maintain and use
- Sharing Channels
- Needed because we have so many Internet users that adding nodes (O(N^2)) is very expensive!
- Endpoints
- We can share these (CS 111 concept)
- Sharing in different directions
- Full-duplex
- Outgoing destination
- Method of broadcasting/multicasting
- This is done for **SCALE**
- Scale
- A relationship between 2 variables and their ratio
- An independent variable that changes arbitrarily
- A dependent variable that is expressed in terms of an independent variable
- Scale magnitude
- Look at Page 44 of Lecture 4 slides
- Metcalfes’s law
- Not mathematically proven, but rather an observation of how things tend to work in the real world.
- This depends on how many different people there are in the network
- N nodes, value is N^2

- Bigger networks are obviously more valuable
- The ratio of value to cost is 1:1
- We pay a cost that is less than that
- 2-party sharing
- Make a 2-party channel into a 2-way channel
- Signals in different directions
- Bosons: These channels pass through each other like Pac Man ghosts
- Fermions: Named after Fermi the physicist
- For those that interfere,
- Separate them by space
- One sends the stuff in one direction, the other sends them in the other direction
- Separate them by time
- Time-division
- Time sharing control
- Prior agreement
- Sender vs receiver
- Change identities as part of the protocol
- Everything goes in the direction of left to right
- Go back and forth based on time
- Needs time synchronization and agreement on the clocks
- Time sharing control
- Prior agreement
- Central controller
- One side controls the flow of information
- The controller can decide when the other party sends and also how much information can be sent
- Sooner or later, the central controller takes command again.
- This party is the big boss and has complete say
- N-party sharing: 1 to N
- Share outgoing channel
- One channel to several destinations
- Send out to multiple people and eventually it goes down to other people
- Different possibilities
- 1:N - How?
- Receivers must read things in on a non-destructive fashion
- Someone else needs to read those signals
- Who is going to get those signals?
- We need to see a subset of those signals.
- How can somebody determine if they need to see that message
- Non-destructive reads
- A party does not destroy the signal so other parties can read it!
- Groups of identical symbols
- Perfect copies
- Allows sharing
- Destructive reads

- Dependent on type of channel you are using
- Everybody can see the signals in a non-destructive way
- Does NOT destroy the signal, everyone can hear the same signal
- The read can be destructive by nature
- If someone reads the particular signal, they can wipe it out
- The reading affects your value and something where you fiddle around with quantum states and you suck in the particles at a sufficient value
- Not a desirable effect because it is a pain in the butt for one-to-N communication
- Reintroduce the signal which gets quite unpleasant
- There are a few examples where this is NOT the case
- Quantum cryptography
- Quantum mechanic principle where only one party can hear information
- If anybody interferes with communication, we know somebody has interfered with communication
- Prefer communication media where reads are non-destructive
- Limiting 1:N transmissions
- How can a sender control which receiver gets a message?
- The right # of receivers should get it.
- Transmit at different times
- Expect different receivers to receive things at different times
- Different symbol sets
- Different encodings for different people
- N-1 people who will have different symbol sets
- Lecturing in Spanish - limits the audience to only a select group who understand Spanish
- Label the transmission destination
- Limiting based on an ID
- N-party sharing: N to 1
- Receiver will see what all transmitters set
- When someone sends a packet across the network, no one else can send a packet simultaneously
- If both are trying to send a message at the same time, neither would succeed because the medium will have interference at the particle level.
- We can transmit on different channels to avoid possible issues
- Three people communicating at the same time might work because we are using different channels
- Limiting N:1 transmissions
- Similar process as 1:N but it is more difficult!
- Any of these things above can work in that fashion
- Tell who is communicating
- Label communications (sender) will indicate through some ID who you are
- N:1 is harder than 1:N
- 1:N
- Easier because you have control and know everything

- You know who you want to send to, all the channels, you can coordinate everything
- You do need to coordinate naming with receiver, though
- ID that the receiver recognizes but this is rather trivial in comparison to other things
- Each set is unique in the context of that sender
- N:1
- There are multiple sources with different information
- Independent of others
- Coordinate things between each source
- Time -> we need to know the appropriate time at which to do it.
- Make an appointment and figure out when to communicate so it will be un-interefered with the other communicators
- Coordinating name
- Converse of 1:N naming
- N-party sharing: N to N
- What if we say instead of N^2 links, every body just has one link
- Linear O(N) amount of cost
- The ultimate shared channel
- One channel
- Add one node and make sure that they understand the channel!
- Single shared channel examples
- Freespace
- Communication is free and adding nodes is trivial because sender doesn't have to make adjustments
- Wired
- Create a wire-shared medium called bus
- Multiple people attach to a wire and listen and send via the wire.
- Ethernet
- People can connect to the some Ethernet and see what is going on
- Fiber
- N:N - combine rules
- 1:N - control receiver
- Each of the guys can communicate from 1:N
- Different channel for each of them
- Do it at different times with different symbols
- Communication medium is the error using the voice
- N:1 - avoid collision
- Different methods of communication
- Raising hands, emails, office hours, etc.
- N:1 - identify source
- Sending emails (associated with a different ID)
- Summary
- Channel sharing affects network size
- Distance, number of parties
- Limitations based on distance

- You generally cannot use a different network for different cities
- Limited by geography
- Sharing channel in general
- You need some kind of shared namespace -> put out signals on the same channel
- Each signal will be put out to some subset on the channel
- Allow those sharing the channel to see what message belongs to whom
- Figure out which N-1 channels to send it on.
- The guy on the receiving end knows who sent it because there is only one receiving end
- We need naming that is meaningful to one party
- We need a set of names with coordination among the names
- Share the channel among a whole lot to people
- Talking about independent machines
- The only way these machines can communicate is through the channel they are trying to share
- Use the channel to coordinate sharing the channel
- This requires a protocol!
- This protocol is quite sophisticated: all of us would like to use this network simultaneously. How do we use it without interfering with each other?
- Fair share and this results in more complicated protocols -> much more difficult than before.
- Use a protocol to coordinate the use of the network.
- Shannon's matrix of a noisy channel
- Sending a 0, there is a 50/50 chance of it being as a 0 or 1
- Recall this in a previous lecture
- Shannon paper allows you to do the calculation
- You will never be required to calculate the capacity of a noisy channel with Shannon's equation
- This is a good idea to have an understanding of the fundamental physical rules that describe what we all care about.
- If you are trying to build networks
- You are hitting the Shannon limits
- A basic understanding of the noise and the encodings you need to use will help you do well than if you don't have any notion whatsoever that you are limited.

W 2 Dis 1-15-16
Announcements

- Project 1 (Implementing a Web server)
- Individual
- 10% of total grade
- Graded by test script
- Project 2 (Reliable data transfer protocol)
- Group
- 20% of total grade
- Extra credit if additional features are implemented

- Congestion control
- Demo will be scheduled for grading
- 15 minutes and ask to run some conditions
- Put the parameters and it should work
- Update group information during Week 4

Project 1 Tips

- Pick a port number > 1024, and != 8080 or != 6769
- Part A
- In the "Web Server", the server prints out the widget message into the console
- Part B

- Parse HTTP Request
- Process
- Make HTTP response message
- HTTP 404 Error message: Server does NOT have requested file (CS 144!)
- If we don't have this file, output an error message
- Testing web server: Open browser
- Print out an HTTP request message in the console
- Parse that request message and find the file and send it back to the browser
- Test and run the browser and server on the same machine
- Use localhost or 127.0.0.1 as the name of the machine
- To test, use the format: http://<machine name>:<port number>/<html file name>
- http://localhost:8080/index.html
- Submission
- C or C++: choose one of them and write a report and it should not exceed 6 pages (except from source codes)
- High-level description of server design
- Difficulties
- Manual of how to compile and run the source code
- Include and briefly explain some sample outputs (test cases)
- Due on Friday, Feb 5th: 11:59 PM
- 10% deduction if submitted on Saturday
- 20% deduction if submitted on Sunday

Client and Server Example

- Client is connected to a server
- Terminal commands

```
./server 7890
Here is the message: Hi CS118
```

- These files will be uploaded onto CCLE

TCP Socket

- Create a service
- Establish a TCP connection
- Send and receive data
- Close a TCP connection
- After transmitting and receiving the file

TCP Socket: Service Setup

- TCP Server
- Creates socket in the server
- Calls function named bind()
- Binds the socket with its IP address and port number
- After binding the address it calls the listen() function
- Then, it calls the accept() function
- Keeps running in blocking mode
- Waits until it receives a connection request from any client
- Process the request (parse the request message)
- After finding the request file, write out the data into a buffer
- read(), then close()

- TCP Client
- Creates a socket in the client
- Calls connect() function
- Calls write() function
- Client can send data via write()
- This writes data out into a socket, which transmits the file to the server
- read() the data(reply) into the client, which is the web browser
- Client then needs to call close() to finish the function

Functions

- int socket(int domain, int type, int protocol)
- Returns socket descriptor or -1 (failure)
- Three types of parameters
- domain: protocol family
- Probably won't use the domain constants
- PF_INET: IPv4 is most likely to be used
- Difference between AF_INET vs PF_INET
- AF_INET refers to an address and PF_INET refers to a protocol
- type: communication style
- SOCK_STREAM for TCP (Project 1)
- SOCK_DGRAM for UDP (Project 2)
- protocol: protocol within family, usually set to 0
- int bind(int sockfd, struct sockaddr* myaddr, int addrlen);
- Binds a socket to a local IP address and port number
- returns 0 on success, -1 and sets errno on failure
- Three types of parameters

- sockfd: socket file descriptor returned by socket()
- myaddr: includes IP address and port number
- Note: socked and sockaddr_in are of same size, use socked_in and convert it to socketaddr
- When you bind the socket to the address, you have to use AF_INET
- When you create the socket itself, you put PF_INET so the socket uses IPv4
- addrlen
- int listen(int sockfd, int backlog);
- Put socket into passive state (wait for connections rather than initiating a connection)
- returns 0 on success, -1 and sets errno on failure
- Two types of parameters
- sockfd: socket file descriptor returned by socket()
- backlog: The # of servers that the program can serve simultaneously
- Pass in 1 for Project 1
- Numbers larger than 1 should work as well
- You can use multithreading to handle multiple connections
- Check a set of multiple sockets and whenever the socket has data to receive or send, it returns that specific socket
- int accept(int sockfd, struct sockaddr* client_addr, in* addrlen);
- Accepts a new connection
- Returns client socket's file descriptor or -1. Also sets errno on failure
- Three types of parameters
- sockfd: socket file descriptor for server, returned by socket()
- client_addr:
- addrlen:
- More Information about Accept()
- A new socket is cloned from the listening socket
- If there are no incoming connections to accept
- Non-Blocking mode: accept() returns -1
- int connect(int sockfd, struct sockaddr* server_addr, int addrlen);
- Connecter to another socket (server)
- int write(int sockfd, char* buf, size_t nbytes);
- Write data to a TCP stream
- Return the # of sent bytes or -1 on failures
- sockfd: socket file descriptor from socket()
- buf: data buffer
- nbytes: # of bytes the caller wants to send
- int read(int sockfd, char* buf, size_t bytes);
- Reads the file
- int close(int sockfd);
- close a socket
- Returns a 0 on success

About Port Numbers

- Some port numbers are already assigned to well-known protocols so you should try to avoid using those to prevent ambiguity
- It can show the port number allocation
- Port number 1-1023: Already used and it is better to choose a port number from 1024-49151

Take Care of Byte Ordering!

- Little endian: LSB is stored in the lowest address
- Big endian: MSB is stored in the lowest address
- Hosts may use different orderings, so we need **byte ordering conversion**
- **Network Byte Order = Big Endian**
- If our host uses little endian, we have to convert it to big endian because networks are in Big Endian.
- Byte ordering functions: used for converting byte ordering
- Examples

```
int m, n;
short int s, t;

m = ntohl (n)  net-to-host long (32-bit) translation
s = ntohs (t)  net-to-host short (16-bit) translation
n = htonl (m)  host-to-net long (32-bit) translation
t = htons ()   host-to-net short (16-bit) translation
```

Address Access/Conversion Functions
Two functions

- in_addr_t inet_addr (const char* strptr);
- Translate dotted-decimal notation of IP address (network byte order)
- int inet_aton

Serve Multiple TCP Connections Simultaneously

- Problem: accept() works under **blocking mode** by default
- **Non-block mode**: use select(): Not used for Project 1!
- Three approaches:
- fork(): each connection is served by a new process
- Easy to write, but expensive and hard to share data between processes
- You will be forking quite often, so the # of processes on a big system will be difficult to manage
- POSIX pthread: each connection is served by a new thread
- Hard to maintain

What is select()?

- A monitor for multiple sockets (or file descriptors)
- Given a set of sockets, if any of them were ready to receive/send, select() would exit

- int select (int numfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeeval *timeout);
- numfds: The highest file descriptor plus one
- readfds, writefds, exceptfds: set of sockets
- timeout: timer for select to exit w/o any changes
- select() waits to see if there is a socket to send or receive data
- If there are no sockets that are sending or receiving, it times out.
- Use a low value if you don't want to use the timer
- returns when some sockets are ready, or timeout
- return value: the # of sockets that are ready to receive the data
- Can handle multiple sockets
- Create sockets and put them into a read set or write set
- The select function will check whether some sockets are ready to send data
- How do we know if read socket or write socket is ready?
- Shown in an example
- Assume the server has created a socket **sock**, which is bound with server's IP address and port number
- Use FD_ZERO to signify an active socket
- Use FD_SET: Set of sockets that is just added
- Can have multiple sockets
- If you put something in a set, is it there until the connection happens? What happens between that set and how could it fail?
- Not sure (?)

What is fd_set?
- A set of sockets (or file descriptors) that will be monitored by select()
- Macros of set operation
- FD_SET(int fd, fd_set *set); //add fd to the set
- FD_CLR(int fd,fd_set *set); //remove fd from the set

How to use select()?
- First parameter of select is the highest # plus 1
- Second parameter is reading the fd_set
- Put this set into the second parameter and don't write an exception there
- If it returns a # < 0, this means there is an error
- exit with a bad error code
- Since there is only one socket: There should be some data to read
- FD_ISSET(sock, &active_fd_set): creates a new connection request
- Returns 1 means that the function was successful in creating a new connection request

HTTP
- HTTP: a stateless protocol on top of TCP
- HTTP is based on pull model
- Communication between web server and web browser

HTTP Example

- Figure that can be an example of the first project
- Client Request
- Sends an HTTP request message to the server
- This is the HTTP request message and this is in human-readable form
- First client should show a GET request
- When it parses the request message, it will give the HTML index file with the header
- Server Response
- Contains a code like 200 OK
- It has an index.html file and gives information like date, context type, etc.
- index.html file itself
- Data should be located there
- Header information
- In the Project 1, we need to print out request message in console, parse the request message, and make this response message
- We don't have to write the client code

W 3 T Lec 1-19-16

More Details on the Noisy Channel Rate Example CS 118

- Page 2 on slides
- Matrix that describes the input you will receive in the channel
- What is the capacity of this last channel?
- Rate of channel = H(x) + H(y) - H(x, y)
- Entropy of input + Entropy of output - Joint entropy of input and output
- Sender is equally likely to send a 0 or 1 in this situation

H(X)

- X is the original source of information
- Two possible signals
- 0
- 1
- Each is equally probably
- The inputs are going to be equally likely
- Here, they are because we said so

Calculating H(X)

- Page 6 on slides
- Two outcomes -> just labels when we come down to it.
- Each of the two outputs are 50% likely

H(Y)

- Y is the received signal, which was affected by noise
- H(Y) is the entropy of that signal
- The equation here is a bit more complicated because what you receive depends on what you sent

- The probability of receiving that symbol will be very low
- The probability that any particular symbol got sent

Calculating H(Y)

- First summation over i and j, second over i
- What does p(i, j) mean?
- Probability that signal i was sent and signal j was received
- No noise would be the probability that signal i was sent
- There will be calculations based on the probability that a given signal was sent and the possibility that signals were received

The p(i, j)s

- Obtained through the matrix
- .5 chance of sending a 0, .5 sending a 1 (based on your number of choices)
- To obtain the probability of the event, it is multiplying (chance of sending a value) * (chance of receiving a value)
- You find (chance of receiving a value) by looking at the value on the matrix
- By summing up all the probabilities, they should add up to 1 because this means ANY of the four events occurred!

Back to H(Y)

- Assume log_2
- Fill in #'s obtained through your matrix and plug them into the equation

Working H(Y) out

- You will get a negative value from the log, so entropy takes the negative of that to get a positive value

Calculating H(X, Y)

- H(x, y) = -(p(0, 0) log(p(0, 0)) + (p(0, 1) log(p(0, 1)) + (p(1, 0) log (1, 0)) + (p(1, 1) log (1, 1))
- R = .31 (check calculation)
- See whether he sends a 0 or a 1
- This really means that we will take some set of input bits and encode it to some integral number of output bits
- If we can't get a full third of a bit sent, we cannot do any bit than encoding it into 4 bits
- If we gathered up 100 bits, we can send 310 bits to get that fitting output

Modern R values

- Noise is a lot less than other wireless networks

- When ISB advertises a smaller value, there may be noise and they may share a channel

- They will gather in 100 houses and they will send traffic down one table
- There will be one bigger cable to take those cables in
- It will be bigger than the rate of any one of those tables
- When you send non-random information, there will be redundancy that can be compressed
- Try to have thinner cables and put compression into the concentrators and hope it all gets compressed
- This is great as long as people at the end are NOT encrypting their data
- Discussed in CS 136
- The data in question after you have encrypted it looks random
- NO redundancy
- Compressor gets nowhere because they see 0 redundancy
- As a result, encrypted data from end to end will be unable to be compressed
- This means people have to buy bigger cables
- Wouldn't it be great to decrypt data
- We have to be able to decrypt data using the secret key
- Man in the middle
- We have to decrypt and re-encrypted + compressed
- They have decrypted your data and they can see what you thought was secret
- NSA can see all your stuff, don't send bad stuff!
- CS 136 is offered in Spring -> take this course!
- Connections to allow everyone's house to connect to the Internet
- Box contains a concentrator and will encode them into a single signal to wherever their offices are.
- How do we share channels?
- They cannot afford to run a wire from your house to their central office
- This connects up to the Internet
- One cable going out to ISP office to somewhere else on the Internet
- If you are using the Internet, you are sharing channels
- Channel is wireless network i.e. UCLA Web or UCLA Wi-Fi
- You are sharing that channel and it looks like some people might be sharing things currently

Outline

- Ways to share the channel
- Divide up signals to determine which signal belongs to who
- Label (name) implications
- Emulated sharing
- Explicit coordination
- Method of working together on the same channel with a laid-out plan of what to do
- We can get good sharing of the channel without necessarily having prearranged agreements

Ways to share a channel

- Difference channels
- Different times
- Whoever is using the channel at a different times (clocked schedule)
- Different symbol sets ("languages")
- It can carry different frequencies and overlay the signals
- Label the transmissions
- Tells you who is doing what

Different channels

- Spatial Division Multiplexing (SDM)
- Specially distinct
- Run a bunch of different wires and this is very costly
- Expensive: Anything that involves physically laying wires costs hella
- We don't like this very much but if we have to do it, we do it
- People have to lay wires and this is expensive because it is a physical service
- Dig a trench, lay out wires, cover up the trench, test it
- Not very scalable
- Each time you want to add a wire, you have to dig the trench again and it's a pain in the ass
- Nobody likes pulling wires
- This is an easy way to get the job done but it's going to cost an arm and a leg
- There are better ways to do it with great scaling

Different times (TDMA)

- Take turns using the same channel
- Color bands (Lecture 5 Page 5) represents who is using the channel at different time intervals
- You either have the entire channel devoted to you or you cannot do anything
- Atomicity in the TDMA system
- TDMA concerns
- Time interval size
- Has to be **AT LEAST** long enough for **AT LEAST** one symbol
- Otherwise, it is a waste of time
- Time interval allocation
- Fairness
- Everyone gets the same amount of time
- Starvation
- CS 111
- A bunch of people want to use the resource, and potentially, certain groups do not get the resource based on the scheduling algorithm
- Efficiency

- If the guy has nothing to send, you have wasted some of your channel, so you have to keep track of unused time slots and keep note of users who are inefficient
- Gap between intervals
- “Guard time”: Use the channel and he is supposed to stop at a certain time
- If you start and the very moment his clock hits a certain time, what happens if your clock is off?
- You start clobbering the other guy’s signal and vice versa
- There has to be a syncing in the transition
- Otherwise you corrupt each others’ data
- You need to add a padding called “guard time” as a safety precaution
- If there is a little error, then it is fine because you ignore what happens in the “guard time”
- Impact of guard time
- Guard time avoids sender overlap
- All receivers should see non-overlapping slots
- The clocks keep getting further and further apart
- You start to run into problems of overlapping signals
- You may get more or less data through
- Consequence
- TDMA needs clock coordination
- TDMA has distance limit
- The Aerospace Corporation built GPS
- Works because of clock synchronization on the order of several miles
- Extremely difficult technical problem
- Average technician is NOT going to worry about that
- Why don’t we have a wireless network over the whole city?
- We are going to have to worry about distance and we run into other problems with sharing schemes that are limited by distance
- Different symbol sets
- Different representations using different physical properties
- Different logical representations we can lay on each other

Frequency (FDMA)

- Split channel capacity into non-overlapping ranges
- Split the channel into non-overlapping ranges
- Works with electromagnetic radiations of various kinds
- The colored bands indicate a particular party performing communication
- Frequency vs time
- No one else can use that range of frequencies ever
- We could do it so the blue guy can get multiple rates of frequencies

FDMA concerns

- Width of the band (bandwidth)
- Band allocation
- Fairness

- If we are not careful, we could starve people
- Efficiency (unused frequencies)
- Wasted resources if bands are not allocated well
- People are not careful about the frequencies they send out.
- Drifting is BAD, this means that there is NO room between non-overlapping frequencies
- If somebody has a bit of drift, they suddenly start making use of part of the band and we have inefficiencies in the channel
- Putting gaps between bands
- Can be applied to frequency as well as clocks
- "Guard bands"

Impact of guard bands

- Guard bands avoid sender overlapping
- Receivers should see non-overlapping bands
- Doppler effect since motion affects frequency (Physics 1B)
- If they don't agree, they speak over each other and corrupt each other's information
- Limit to how little you can send something if you want to translate to a particular rate

Different alphabets

- Code (CDMA)
- Division between both frequency and time
- You get this piece of frequency for this particular time
- Color coding implies you have partial overlap that can enable communication in the presence of increased noise
- Code Division Multiplexing

xDM vs DMA

- xDM: {Space, Time, Code} Division Multiplexing
- Sharing (dividing a resource) by multiplexing (merging) or demultiplexing (splitting) based on spatial, temporal, or coding context
- DMA: {S/T/C} Division Multiple-Access
- Using xDM to coordinate shared access of a channel by multiple sources or receivers

Often used somewhat interchangeably

Sharing compared

- TDMA
- If we look at frequency vs time, all the parties are color coded
- They get the entire frequency here
- FDMA
- Instead, we always get the same band of frequencies that we can use
- As we can see in the middle, we will divide the frequency up into pieces and we will send it to different senders and receivers

- CDMA
- White slots indicate we don't have anything to send at the time
- For TDMA and FDMA, will there be times when we have no information being sent?
- Yes, that is possible
- For CDMA, is there a way to for sender and receiver to coordinate who gets what piece
- Some other method where we can get this effect
- Get through the coordinated stuff today and how do we go about doing this
- Divide CDMA into so many pieces
- Assuming we don't have parties that go forever, we will still have that same issue
- Who gets that period of time in TDMA vs FDMA

Label (name) implications
- Names can be very arbitrary but they need to be able to indicate who we are talking about
- If we aren't sharing channels and we only care about multiparty, we still need names
- We need to figure out on the sending end which channel we are sending on
- This channel could be devoted to this sender
- In order to do that, we need information inside our nodes
- This can be purely internal between each node
- If we are also sharing channels from one party to N-other parties, we have an issue
- We need some way of naming things i.e. in a particular naming approach
- We might need to attach names to what we are doing
- Figure out when a particular band or frequency belongs to another one
- Other people will have to know that as well and both ends have to agree on any information put out on the channel

Destination names
- Context (1:N)
- Know the channel
- MAY mean the receiver knows the source
- Uniqueness
- Names have to be unique as far as the sender is concerned
- Shared channel
- More difficult because you don't have N channels, you only have 1 channel
- Name must be agreed upon between the sender and the receiver
- Must be known in this context to be shared across the channel

Source names
- Context (N: 1)

- Know the channel
- Unique per-sender on this channel

Combined names

- We are obviously going to be doing something where we send both kinds of names
- Full connectivity (N:N)
- We don't really want to know very much about names except for local names
- On the receiver side, we need to know very local things
- Not very useful for senders/receivers
- We don't need to share a lot of stuff
- It may be useful to coordinate sender and receiver names
- We need the ability for everyone to know the names
- Anybody can send a message to anybody else, so they need to know the receiving names
- Try to have the same name for sending and receiving
- When messages go out from him, the guy sending it puts it into the same name for messages
- We can have a different sending and receiving name provided that everyone knows
- We are going to have to put the sender's name and the receiver's name
- The N receivers have to see the messages and say this one is for me because my name is in it.
- Be able to determine who sent me this message
- This is precisely what we do on the Internet
- Every packet has a sender name and receiver name for precisely these reasons
- The Internet is one big shared channel
- Dig down deeper and it isn't one small channel

Name assignment

- A-priori here (for now)
- Part of the protocol configuration
- Name is sent by some process we are considering
- An agreement on the protocol
- Organization out there such as IANA that says this is your range of unique Mac addresses
- If you must go to one place to get your name, it has to ensure it hasn't given this name to anyone else
- Issue of cost here: someone has to be that entity and be able to figure out which name you are giving out
- System of some kind to ensure that names are unique

Emulated Sharing

- Devices can emulate sharing

- There are signals that go in to one end and things that come out on the other end
- Look at cabling on the back of a router and we are using a device to emulate sharing
- Multiple connections coming into that device and we have multiple inputs that come out to one shared signal
- We are dividing that into multiple signals coming out

Demultiplexer

- 1:N
- One source, multiple receivers
- Isolates receiver from sharing
- Source still thinks the channel is shared
- Needs to indicate the destination
- Receiver thinks it has direct channels
- Doesn't need to know whether to listen
- Each channel potentially connects to one receiver -> one fully connected network
- Anything coming in from this channel must be for me
- What's in the box?
- Computational device that does stuff
- It splits symbols up into sets
- Has information about destination names to demultiplex (pick output port)
- Names that are associated with information
- Differences between what people see on the other end
- Can remove the differences (translation)
- i.e. using a FSM
- Translate a language form the top guy and send it to the bottom guy

Multiplexer

- N:1
- Multiple sources, one receivers
- Get a single signal in the output channel
- Other side of the demultiplexer
- Sender looks like he has a dedicated line from the receiver
- We think we have a devoted channel to the Internet, but it is actually a devoted channel to the multiplexer that concentrates this information
- What's in the box?
- Merges/interleaves symbols
- Combinations of 100's of symbols
- The multiplexer knows what is on the other side since there is only one thing
- All of the hundred lines connects to one thing
- Implicitly, it knows how to slap the necessary name on it
- Receiver thinks we have a shared channel and we better have some way of knowing the names

- Add source names to the output
- Do the translation business
- Combine them together and do code division on the outgoing part

Switch
- N:N
- Multiple sources, multiple recvs
- Have them do multiple parts i.e. multiplexing and demultiplexing
- Includes equipment such as Internet routers
- Pros
- Coordination is internal
- Easier at the low level to manage channel wiring
- Makes it easier to deal with wiring issues
- Explicit coordination
- Efficient
- Shared channel
- You know all the details (omniscient)
- Global balancing
- Simple to implement
- Cons
- Maximum load and switching it can do
- Trust issue: everyone has to trust that the switch behaves as expected
- Sender and receiver had to trust each other
- I assume that the sender is sending good stuff and receiver is getting information properly
- Guys in the middle i.e. certain Internet companies and ISP have made deals with the NSA to give them traffic that wasn't meant to the NSA
- You have to be careful about using switching technology in the middle of the network
- Source/Destination need to participate in naming
- Names need to be unique (pain in the ass!)

What about circuits vs. packets?
- It will matter a lot later
- Circuit switching vs packet switching

Explicit resource coordination
- Why coordinate?
- N:1 sharing needs to avoid collisions
- N different people can overwrite each other's stuff
- Collision is bad in this case
- There can also be more noise and less efficiency
- Where is 1:N sharing coordinated?
- Somewhat easier because you can do it in your box and he can make his own decisions
- Why explicit?

- It is simpler, and the focus of this class

A-priori coordination

- Part of the pre-shared rules
- Contains everything you need to do to do coordination
- No other computer ever uses that spectrum
- Very inefficient
- Stick with those time-slots forever
- Fixed allocation
- Various kinds of inefficiency
- Build it into the protocol and have some kind of starting point

Limits of a-priori coordination

- Fixed allocations do not work well if you have dynamic usage
- If people have a highly variable schedule, you will have inefficient use because you cannot predict dynamic usage
- Expensive because adding/deleting nodes is difficult

Centralized coordination

- He makes all the decision and it does NOT have to be at fixed intervals
- This means he can ask the source how much information to store in the next time interval
- Get this information from all the sources and have the following set of allocations assigned

Requirements for central coordination

- Symmetric channel
- All possible sources needs to be able to respond to the manager
- You could have only nodes and you can hear from the manager about what you want

Central coordination protocol

- Polling
- master and slave concept
- Channel
- Bus
- Often very short -> occurs within a chip
- We like to move a whole lot of information within the bus
- Changing who is on the bus at human time scales
- Relatively static, relatively short
- Master make all the decisions
- Master will go to each slave and ask if they have anything to send
- If they do, master will keep track of that and it can decide to assign a slot or wait to poll the remaining slaves
- The scheduling process tells everyone what the master’s plan is

- This could be a truly shared Bus; it does NOT necessarily have to be on a switch
- When he is done with that, there will be cycles of what he had gotten

Limits of central coordination

- Load
- Pushes all the work to one manager
- Limit of how much that master can do in a certain amount of time
- Fault tolerance
- If the manager fails and cannot poll anybody, nobody else can talk
- The same effect occurs if the channel fails
- Trust
- You have to trust the Bus master, but if he/she/it is an asshole, you're kind of screwed

Hierarchical/delegated coordination

- Extend central coordination
- Get information from a bunch of submitters
- Tends to have higher overhead

Benefits of explicit coordination

- People who are sending always require the same capacity on a channel over a given period of time
- Manager knows everything so he can work it out so he gets the most efficient use of the channel
- Trivial to avoid starvation and ensure fairness (under the control of the manager)
- Manager has complete power over fairness

Limits of explicit coordination

- Possibly very inefficient
- A lot of polling messages can be meaningless
- The signal is sent and the receiver interprets the message
- You use up some capacity of the channel purely for this purpose
- Lots of overheating for polling which can require a lot of messages to parties NOT involved in the communication
- Vulnerable
- One of the N parties is trying to fuck you over and this can cause many problems
- Failure to use the channel at all in this case

Use cases for explicit coordination

- One party that has a lot of capacity and communication capability
- He is already going to be gathering up everything
- Satellites

- They cannot communicate with each other unless they have the satellite sending from 1:N
- Airplane/blimp
- Going through ViaSat's network and you need that to get access to Wi-Fi
- Single access point
- One piece of computing equipment
- All the messages going to and from you are going to that equipment
- Decentralized Sharing
- We can have decentralized sharing and we could extend 2-party and N-party masters

Overall sharing goals

- Fairness
- We want allocation to be fair to those who need it
- Starvation-Free
- Someone gets to send at least something
- Efficient
- Try NOT to throw away capacity
- Talked about in 111, ways to allocate the processor to determine who got how much memory

2-party master

- One side controls the system
- Other side obeys the master in whatever fashion the master says -> we still need coordination
- One of the two guys will be the master
- Issues
- Who gets to use the channel when to send data?
- The slave has to NOT be sending
- Tell the master when he would like to send
- Use the channel and have issues here
- Who is the master vs who is the slave?

2-party controller selection

- Who gets to be the master?
- O.K. Corral analogy: whoever uses the channel first wins
- Backgammon or coin flip: whoever gets the highest role/heads goes first
- Tie-breaking
- O.K. Corral: not needed (both dead!)
- Backgammon: try again!

Tie-breaking 101

- Computers are deterministic
- When we say we are generating a random number, we really have a pseudorandom number that has a deterministic output

- If we run similar code on both sides and both sides generated the same pseudorandom number, running it again will probably have the same pseudorandom sequence
- Solution
- Each piece of communication has a serial number that can never tie
- To achieve this, we will generate a unique ID using a central authority such as from IEEE
- Used for wireless networking equipment

2-party fault tolerance

- Who cares about the coordination?
- They cannot communicate if one of the party fails
- “Fate sharing”
- Does not work well when you have more than 2 parties
- For exactly 2, this is perfectly reasonable

2-party bias

- Someone got to be the master, and the master always gets to send
- The slave can send and this means that the master has the ability to send more often
- If he makes the request, he can overwrite the master’s information
- Impact
- Biased controller can undermine the concept of fairness
- If a master is trying to be fair, there can still be issues

Why are there problems?

- Client request might occur just after every poll
- Master can have x milliseconds without further communicating to the master
- Every so often, the client may NOT have something to send
- The master sends what he wants to send, but this happens at certain intervals
- Polling intervals can be a problem if the client’s message is generated after initially saying that he didn’t have anything
- There is a period of time where the master cannot use the channel because the master is waiting for the client’s poll response
- He is only going to poll every so often, so there are situations when the client has to wait
- He will get the channel until the next poll comes in.

Solving 2-party control

- Transfer control of a master
- You get to be a master for 5 minutes and then he goes back to you
- This should even out the bias
- Transfer the master’s status back and forth
- Needs a protocol to do this

- Need something to solve the bias problem
- If the master is sending a whole lot more than the client does, this could be perfectly fine.
- Ping pong
- Status changes and you could have some protocol saying that the status has changed.

W 3 R Lec 1-21-16

- We no longer have N^2 channels for N participants with our new algorithm
- We want to be able to share our channels
- We want a full duplex channel with two parties
- How will we coordinate the use of that channel
- Two party master
- Decides when each participant gets to use the channel
- Master tells the other guy you may communicate now for this period of time or the following amount of data
- We also talked a little bit about channels where we have a whole lot to parties sharing the same channel
- Only one of them will be able to make effective use of that medium at the same time
- Introduces noise into the channel
- Significantly lower rate of what we can send
- More than one party could put data into a shared channel at a single time
- Extend the idea of a 2-party master to an N party master
- One of those N parties will make all the decisions about who gets to use the channel when
- This person then polls to figure out if they have data to send
- Get particular period of time and one guy makes all these decisions
- Most general way for this to work is for slots set aside for nothing to send.
- Usually what we will do is that the master will go around and ask for anything to send
- Collect info about that and make decisions about when to send
- Obviously there will be some issues about selecting who the controller is
- Fault tolerance can be an issue here since if the master fails, we have N-1 where N>2 and we don’t want failure of master to prevent the use of the channel
- Bias problem with the 2-party master
- How to select our master
- Go first (time)
- Whoever said he wanted to be the master first would be the master
- Highest-roll (value)
- Pick a random value and whoever gets the highest value becomes the master
- Same tie-breaking

- If two people talk at the same time or put out the same number, this approach is NOT going to scale well if N gets to be very large
- Many selections result in ties if there are a large # of participants
- Birthday problem
- Sharing Ben Franklin's birthday (Jan 17)
- About 10%
- Probability that two of you share one birthday?
- Roughly 90% for 40 people
- Each of these N parties is choosing a number out of the days of the year
- If there are at least 40 parties on the wire, we can get a tie on one of the #'s
- It could also be the winning # if you have a tie at all.
- How do you decide who gets the number on the wire?
- Choose the #, put it out on the wire with your identity, then you wait.
- Wait for your submission, otherwise, do it again.
- You can also say there is a temporary master
- If you use deterministic methods, you may never break the tie.
- If N gets large enough, you tend to get ties
- You need a much larger random space than # of candidates to avoid the high likelihood of ties
- Your algorithm has to overcome the fact that you have a tie and is non-deterministic
- We should be able to communicate nearly as well with the master
- N-party fault tolerance
- Sharing of the fate between ability to communicate and the need to communicate
- N-party bias
- As N gets bigger and bigger, controller has much more power.
- The only time that the non-master would be able to communicate was when he got a poll from the master
- Master could decide when to send the polls and there would be a significant delay in having the other party communicate
- The master got to communicate a lot more often.
- Treat clients preferentially
- Through malice or a bad algorithm, the master can give N-1 preference to a group more than others
- If the client needs something, give him the use of the medium for some period of time.
- N-1 client has worst preference and cannot send anything until the other group has finished.
- Only one party can effectively use the medium at any period of time.
- There are issues of NOT having a controller as well, but this is often what we do in real situations
- Not everyone might be treated fairly by the master and the master can receive a lot more benefit than anyone else
- Solving N-party control

- Talking stick derived from aboriginal tribes
- The chief would hold a specialized stick and he could say whatever we wanted
- Whoever had the stick would be handed off power
- Handing off the master role and after a period of time i.e. # of sends
- We hand it off and this is beneficial because the master gets better use of the medium than anyone else
- Always has preferential use of the medium depending on how he treats the other members of the group
- In the long haul everyone will get fair treatment
- Token bus derived from IEEE
- Problems with token bus
- What if you lose the stick?
- The stick probably won't get lost easily in a circle, but we are talking about a situation where we are doing computer communications
- Who gets to decide who is using the communication medium?
- What if the person forgets who is using the communication medium?
- We need a master selection stage and a regeneration of the token stage
- We also need to prevent false alarms of regeneration of the token
- We need to make sure people do NOT cheat
- People will try to "slip in" and use it a bit more
- This leads to collisions between data and this reduces the efficiency of the channel
- Cheaters will create a problem
- Might not be cheating intentionally but there might be a flaw in the hardware and some other problem
- People build cheating versions of protocol i.e. bitTorrent
- People only build versions that help themselves i.e. downloading illegal content
- Membership Change
- Change the circle of people sitting there and figure out the state of other members of the group
- We need a way of repairing a broken ring of people
- Sharing without a master
- Outdated method: we don't use token rings for computer communication anymore
- Build a need to speak to your friends and you probably don't have anybody pointing to you and telling you when to speak.
- Telephone companies said we would have a shared line ("party line") in the 1950's in rural towns
- Who gets to use the party line at any given moment?
- Aloha!
- Computers aren't so good at social types of things so we need something more well defined
- Aloha! was built in 1971 at the University of Hawaii

- The University of Hawaii has a # of different campuses across the different islands
- Use wires and have some way of sharing that wireless channel
- Have one shared channel and they will be able to communicate with each other I
- If you have a message to send via radio, send it!
- If you heard your message perfectly, life is good.
- If you heard your message get corrupted, send it again.
- The reason you didn't hear your message was because someone was talking at the same time.
- If you send a message and it gets clobber, this has a domino effect where there gets repeated noise
- This could possibly lead to an infinite loop of clobbered messages
- Using random delays
- If message is stepped on, don't send immediately
- By the time your waiting period is over, you won't step on each other's messages
- The time you send your message is wasted space in the channel
- Treated as total noise
- Probably both of you are going to wait a little bit of time
- During this time, there is wasted time because no data goes through the channel during this period
- Wasting certain amounts of your channel during these backlogs
- One solution
- Slotted Aloha
- You can only send in messages at a particular slot of time
- Wait for the next slot and decide when to send
- Only send at the start of a slot
- On collision, you decide whether or not you want to send at a particular slot
- Role a pair of dice and do pseudorandom # generation
- You might have a moderate probability of sending during the slots
- Pure vs Slotted
- Pure
- Send at whenever time
- Has a ton of collisions!
- Slotted
- Common slot time
- Each user backs off to a different extent
- Still has some problems with collisions but this ensures that collisions occur at particular ties.
- Assuming fixed-size messages
- Assuming Poisson arrivals
- Red is unslotted, blue is slotted
- In the y-axis, this is the throughput
- We get a whole lot more through slotted than unsolved

- As the load increase, they approach zero meaning that past a certain point, there is diminishing return for slotted Alohas
- This means the Aloha solution is far from perfect!
- Alternative to Aloha!
- Listen on the channel first, and if you have something to send, then you send if channel is NOT in use.
- If the channel is in use, don't clobber packets by sending, be patient!
- Discussion group rules
- Talking with friends in a group
- Message to send
- If we do this approach, we need to put something into the shared channel and there seems to be nothing in the channel
- Did we hear our sent message?
- We want to avoid collisions, so you can confirm if it is sent if there is no disruption in your channel
- Even though you listen for quiet, you might still get a collision
- You will know because you listened and we better resend
- We go through the same thing with random back offs

Summary

- Many ways to share channels
- Time division multiplexing (TDMA)
- Frequency division multiplexing (FDMA)
- Sharing suggests coordination
- We can do it using the talking stick approach by changing the master all the time
- Problems with failure, cheating, poor utilization, etc.
- Sometimes, we can do better without having a master.

Outline

- Carrier sense channel sharing
- Relaying
- Allows more people to get into the network without sharing channels

Sending without a master

- Listen to see if we hear it
- If we did, we're DONE
- Otherwise, resend it.
- CSMA (IEEE 802.3)
- The carrier here checks if the the channel is idle
- "Listen before you talk"
- We expect to see 7 parties on the shared channel
- Sharing the channel where at most one party can send data on that channel
- One says that we would like to share a message to 2
- I don't hear anything
- Find a message to 2 and it goes across the medium to 2

- 1 and 2 were closer together on this channel

CSMA and persistence

- No point in sending when the channel is busy
- You will clobber your own message and everyone else’s message
- Everyone gets rekt
- Do you keep listening?
- Listen until the channel is free!
- Persistent CSMA
- Be persistent and keep listening until it is your chance
- Non-persistent CSMA
- Stops listening and checks again later at a random interval
- If the message is missed by the person, they will get it later when they return to the channel
- The interface means we can do it simultaneously.
- We are listening and if the slot is busy, why wait for the rest of the slot?
- Generally speaking, something that is non-persistent means there is random back off.

CSMA and behavior

- Pay attention to Lecture 6, Page 8 slides

CSMA and collisions

- CSMA involves sharing a channel
- Multiple senders can all put messages on the same channel
- We listen and do NOT hear anything
- More than one sender can try to use the channel at once
- Listeners will NOT send messages when others are talking
- Inevitably, there will be collisions somehow
- This corrupts both messages and makes BOTH useless

Collisions

- If two groups hear nothing, assume nothing, and try to send information simultaneously
- This happens quite frequently!
- Both messages get destroyed here

CSMA variants

- Listening business
- Once you send packet to shared medium, you listen to see if the message got all the way through
- If so, your message got through without a collision
- Otherwise, there is a collision somewhere along the line
- Collision Avoidance
- Spend a little more time upfront listening
- You don’t send things if you have a collision
- CSMA is for collision avoidance
- Listens first before sending, but does NOT necessarily listen afterwards

Why not listen?

- Not practical for some wireless channels

- Harder to build a wireless network card that both listens and sends simultaneously
- Send it hat a relatively high power and you want it to dissipate as soon as it goes through the air
- If you send, you are probably having stronger signal at the sending point
- Relatively hard to send received signal over the signal you are sending.
- More sensitive equipment is more expensive
- If both tools are rated the same, obviously you get the cheaper one
- They won't detect collisions if they actually occur

Acknowledgments

- Listen to determine if there is a collision
- Solve the problem at a higher level
- If you get any message across on this channel, get it back to the guy and send it back saying I got your message
- Visit a channel and if you hear an acknowledgment of that packet, all is well.
- We can assume the message got clobbered and they can be very short if there is NOT a whole lot of data
- You should all instantly see that we are going to waste a certain amount of the capacity of the channel

A note about CSMA

- Benefits
- Collision avoidance and this is heavily studied because Wi-Fi is really important
- People put a lot of effort to squeeze out utilization and chances of using some kind of collision

Ensuring channel capture

- Start sending data
- Collision can occur in unpredictable ways
- Preamble
- If you want to send a message, listen to the channel.
- If the passage is free, send an arbitrary message as a test value
- If it works, you have captured the channel at a free moment
- The preamble determines that no one was trying to send at the same time as you.
- You are unlikely to have a collision in this scenario

Flooding the channel

- You just put a bit pattern out and see if it gets corrupted
- Put an unimportant signal into the (apparently idle) channel
- Keep it out on the channel for a while to make sure people did NOT do it simultaneously
- Keep the preamble out there and wait long enough to ensure that your preamble has NOT been corrupted.

More on flooding

- Time going down and we listen
- It has to be of a particular length and we know the distance it goes

- Wait for a full round trip
- The total amount of time for it to be sent and come back
- If it takes a millisecond, then we know this information and can gauge when we CAN send things
- You have to know how long the medium is and this will be prearranged
- The true broadcast form of Ethernet had a known maximum distance
- Since there were limits to Ethernet length, we knew we could limit it.
- If we are NOT looking for perfection, we choose some distance which is how much insurance I want to pay for, and throw junk on the channel without corruption.
- This is overhead: reduction of amount of information that can be sent across the channel
- We need to know how long that blue band will be to send information in collisions
- Higher speeds, a given preamble is going to be “shorter” and take less time to get there
- If Wi-Fi does flooding, can anyone mess up the flooding by making the preamble very long
- A preamble with a particular # of bits cannot go a bigger distance
- There is a limit on the channel length

Limitations of no-master sharing

- Channel length
- Protocol overhead

Preamble vs. channel length

- Preamble
- 7 bytes = 56 bits
- There is overhead except it cannot send the information for a particular length
- It has to go to the other end and the first bit has to propagate out there
- This puts a limit and it has to have particular channel capacities
- This has to have a link longer than 1866 m, otherwise, you cannot detect all the collisions
- How does this cause a faster link to have a shorter distance?
- This length is 56 bits and this affects your preamble
- How long does it take to send 56 bits to send something across the network?
- This means if all you send is 56 bits, it will take less time for you to send things on a very fast channel
- You might hear things only 5 or 6 meters away
- Ethernet at a particular speed might be rated for something

Protocol overhead

- Converse of channel length limit
- Faster symbol rate = longer preamble
- Larger overhead in this scenario

Backoff

- Two people can listen and they might have to back off if necessary.

- Used in pretty much any scheme

Capture effect

- Collision backoff is not fair (known in 1994)
- Two parties who collided each generate a random #
- One of them gets a shorter backoff than the other
- One guy will be able to send first and the other guy will be able to send longer
- One guy got a shorter one and he is going to have a 2nd collision
- The more collisions you get, the more you have to back off.
- If you have NOT collided, then you may have to listen for a little while and you have to wait for some random period.
- If you have a collision from A and B and A is the winner, this means A is the preferred candidate for a while
- Any collisions on B won't be transmitted
- He has backed off for a longer period of time
- He didn't have to wait very long because he didn't have to wait for a collision
- A's 2nd packet keeps it busy during his 2nd backoff for B
- The other guy can get the short end of the stick and probably cannot send as much
- This means when you are sitting there, you may have a long wait

Single, shared channel limit

- Signal power
- Power gets absorbed as we send things through a signal
- As long as we have some constraints, it spreads and dissipates the power
- Number of receivers
- You need to send with enough power to get enough signal to hear it at a sufficient power
- We need to understand what the signal means
- Puts an effective distance limit
- Protocol effects
- Fairly generic figure and what typically happens
- Contention based CSMA protocol
- Token passing protocol won't have collisions (it has other problems but collisions ain't one of them)
- Protocol variations
- Persistent to a different degree
- It turns out that if you look at Pure ALOHA, you will get poor efficiency under almost all circumstances
- As we go up, we see that we do better and better and better.
- There are other protocols that work better at low loads that work better at high loads

Protocol effect implications

- Channel negotiation takes time
- Uber won't be using a single shared channel

- Relies on signal and relay, this connects people up through different paths so that their message does NOT collide at all.
- Strike a balance between connecting nodes and sharing the channels that are used in that network.
- Things aren't shared between two parties
- For example, I'm going to use a satellite link to go across the U.S. and Germany
- This won't be shared!
- If you don't use a preamble, what you send could potentially get dropped

Topology

- It is hard to deploy a single shared channel
- The wire goes where the wire goes
- The wire doesn't go to certain areas

The hidden terminal problem

- Situation where we cannot reach all nodes using the medium
- Wireless networks i.e. 802.11
- The others stomped on each other too much and they were using the same frequency range
- Generally, they were a distance apart and if they are far enough apart, they won't hear each others signals.
- What if there is a guy with a laptop as the middleman?
- There will be an incomplete sharing situation
- Not all modes reach each other!
- A and C won't know their transmissions collide!
- If A and C send at the same time at the same frequency, A and C won't be able to hear respective frequencies, but B can hear!
- We cannot solve this problem through preambles and listening
- We have to use acknowledgments in this situation!
- Otherwise, we assume there is a collision of some sorts
- Acknowledgements have some overhead themselves
- They have a round trip time and we can have essentially a half-efficient use of the channel
- We wait a round trip time to send and receive
- Have more inefficiencies built into your protocols.

Naming implications for shared medium

- Sharing is sharing
- We can have some way of determining information on the shared medium
- Avoid getting the stuff we shouldn't be getting and we should use names to prevent this problem
- In 1:N, only destination name must be unique
- Like a mailing address to identify who gets the packet
- In N:1, only the source name must be unique
- Each packet needs a source name to identify who sent it
- In N:N (with no other info), source/destination pair must be unique
- He won't be able to tell which webpage updates things
- We want unique names!

The cost of naming

- Worst case: N^2 names
- Costly to add more parties and does NOT scale
- Simpler cases: N names
- Adding 1 party adds at most 1 name
- Scales well

Shared media naming techniques

- Central authority
- Someone is responsible for slapping a name on a device
- We delegate high-level names and we need part of the name to be a high-level name to choose yourself
- Guy in China is given part of a high-level name and give it whatever you want
- There are also 64 bit addresses used for other protocols
- ATM NSAP
- Had a central authority where people give out part of the names
- IPv4 addresses
- Had a central authority (currently sitting in Marina Del Rey)
- Delegating it to a more international body
- You could also do self-assignment
- Choose a random # and if you choose a random # generator
- The chances of this would be pretty small
- If they are already using it, do it again!
- One thing we can do, especially on the shared media is to be able to broadcast the whole network.
- Why waste space given that everyone is going to hear the message?
- If the particular address is C, we know everyone in the address and they get it.
- There is a particular address that is reserved for broadcasting.
- Everybody who is connected to that network is supposed to get that message.
- Predefine who they are and assign a special name to that set of N
- When you see this special name, you are supposed to listen
- If a guy does NOT have an address and wants to connect, he needs to reserve a special address as a temporary thing
- Sort of like an interim driving license until you get a real one

More switching

- Switching emulates sharing
- Multiple independent non-shared channels in a switch
- Make it look like a shared channel
- Each of these independent things do NOT worry about complex issues because they are NOT shared.
- Switch will deal with issues of non-shared channels dealing with shared channels
- What makes it useful?
- Simple wiring (direct to closet)

- Relatively easy to do
- They don’t interfere with each other very much and it is easier to coordinate
- Circuitry in a box does all the coordination
- All of this can be decided in a central place
- They can be more efficient and fair.

Simpler wiring

- First step: everyone on a bus
- Anything you put on the channel is public
- Anyone can see it!

Simpler wiring 2

- Second step: delegate to a shorter bus
- We don’t have to worry about this bus issue and avoid a dedicated channel for these pieces of equipment
- Less possibility of collision and less overhead

Simpler wiring 3

- Third step: box it up!
- Simpler to manage and operate
- One centralized point for many decisions

A note about MAC protocols

- Media access control
- Just a name
- Protocol that controls shared access
- If you don’t have shared access, you don’t need MAC
- It is implicit what you are doing
- On a one-way channel, there is no issue of who is the master vs who is the slave

Simpler wiring 4

- Why do you need a box in the box?
- You care about putting your packets in and sending anything through the box
- Convert it to a set of non-shared channels
- A lot more simple for everyone except the switch
- This makes life a whole lot simpler
- We don’t actually have 20 machines connected up to the same shared medium
- This may use the Ethernet protocol to connect from the Internet to a switch
- This does NOT take into account problems of collisions

Benefits of independence

- Tell the switch we aren’t worried about other people’s cables.

Enhanced coordination

- The switch can see everything!
- This can take care of everything and normally we will have shorter cables
- If you have done a good job, it will be x/2 meters form the switch
- This leads to all sorts of benefits from the physics of computer networks

- You can run master based protocols because they are now all fair
- We can make sure there is no starvation

So what’s a switch really?

- How do switches work?
- Shared media
- Piece of hardware sitting in the middle
- Star coupler, bus, memory, etc.
- A control algorithm that says I have a whole bunch of links coming in and out
- Makes decisions on what to do next
- A bunch of internal relays

Goal: scalable communication

- Direct, dedicated channel to everybody else
- The problem was that it scaled very poorly
- Only 2-party channels
- Have N^2 of those and this was scaled very poorly
- Have problems regarding maximum distance since it was limited to a direct signal
- Shared media
- Limited by signal sharing based on distance the signal goes, frequency, etc.
- Also limited by MAC protocol
- Listen before communicating and dealing with collisions
- Great for a small group of people
- Does NOT work so well for larger groups

Sharing and relaying

- Emulate full connectivity
- We have one switch that come into the switch and go out to the switch
- It is in the building or a small physical environment.
- We probably have one cable that goes from the switch to the receiver.
- The switch still has problems but it manages itself since it has a central communication point.
- We won’t run into another collision and we can avoid having to send to the same receiver.
- Instead of both needing to back of, they can either resend and avoid business of saying each person must back off.
- There is one box with one piece of hardware.
- It cannot be longer than that because there are physical limitations on how long it can be.
- Limitations based on how far a signal propagates through space.
- Benefits by sharing more effectively in a small group
- Solves problem of large scale computer networking.
- Let’s pretend we have full connectivity but we don’t
- N^2 case
- Go from communication for a single link to networking
- This gives an illusion of full connectivity

- Any party will be able to communicate to any other party
- We will have the effect of N^2 cables without N^2 cables
- Of course we have complexities and tradeoffs

Sharing vs. relaying

- Sharing
- Works well at small scale
- Does NOT scale well in terms of # of parties involved
- Scaling size limitation
- Radio can only reach so far in physical space
- Relaying
- Spreads out the work
- Distributed protocols where people have to agree on complex ways
- Allows complex topologies
- Arbitrarily complex topologies in this situation
- Build huge networks like the Internet using Internet topologies
- Good size scaling
- Internet works on a global scale

Relaying to the Rescue

- Communicating through a third party
- Transitive closure of this
- Eventually, A can get a message to X

How does relaying help?

- Allows us to remove some of the channels
- N^2 lengths for N participants
- We can start cutting links and end up cutting things

How can we relay? #1

- Most links in the Internet are two party links
- Essentially they have some media that connects them i.e. a wire or fiber of some kind
- We could use switches here and place it in the middle
- They do NOT share a communication path with a switch in the middle
- They can set themselves up like old telephone networks.
- It does circuit switching: a continuous path that is always in place between two endpoints
- Guy on the left can communicate with the guy on the right using a switch
- Conversation can go over that circuit as long as it lasts

The good, the bad, and …

- Good news
- Relaying reduces the # of links
- Bad news
- Limits how many pairs can communicate at a single time
- Communication made it impossible to communicate to someone else.
- Perhaps one person can use the link at any given moment.
- I do NOT want to communicate to the other billion users at this moment.
- Tradeoff is that we cannot based on the topology used by switches and relays determine the amount of communication that takes place

- Inevitably, we will run into that limit and we won’t have an unshared channel between two parties.
- The links we need will be used by other people.

Circuits - a sure thing

- Trains on a track
- NY to Philadelphia or NY to Boston
- Same starting point
- Set switches to control the direction you are going.
- Atomicity in the sense that the entire train goes one direction and the switch can allow you to change the direction of current flow
- Please set up all the circuits so you can set the train to go to Boston
- Rather inflexible after you have decided on a choice
- Not necessarily using all the resources, either way, they aren’t used by anyone else.
- It may be that the train has multiple ways that are more circuitous or not.
- No competing traffic
- Fixed delay, fixed jitter, fixed capacity
- Sending things from Point A to Point B, does it take exactly the same tim?
- There is no jitter because it takes exactly the same time.
- You cannot share resources concurrently
- No one else can use your resources while you have it.

W 3 Dis 1-22-16

- The server can run in a loop and there is no requirement
- Test with just one file and see if the request file has been shown in the client web browser
- We will test three times and make sure that all the test files pass
- Select() function
- int select
- (int numfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
- Returns the number of sockets that is ready to read or write
- If you give it just 1 ID, it returns 1
- Example code uses multiple sockets and we can understand how the select function works with multiple connections
- Parameters
- numfds: the highest file descriptor + 1
- readfds, writefds, exceptfd: set of sockets
- timeout: use the timer to set this parameter
- We don’t need to use this select function for Project 1, but it is HIGHLY RECOMMENDED for Project 2
- There will be only one client and one server
- Implement the timer so we don’t have to use the select function to implement the timer
- This is the easiest way to make a timer on your Project 2

- fd_set struct
- Bit mask that represents the state of each file descriptor
- fd_set Example
- fd_set is a set of sockets to “monitor” some activity.

Code high-level description

- master_socket is used to make a new connection with any client
- Each connection has its own socket and the buffer has an fd_set
- Initialize the client socket as 0, so there is no connections
- Clear the socket set
- If there is a connection, you can put the client socket in the FD_SET
- Set the maximum file descriptor number as sd
- Client socket can handle 13 clients
- Make the new socket and put it into the client socket
- When you use the select function, you have to check all the values to make sure the socket is available

HTTP vs HTML

- HTML: hypertext markup language
- The language that is used for the HTTP
- Defines multiple tags to be used for web documents
- HTTP: hypertext transfer protocol
- This is the protocol that is used to define how the client and server communicate
- Rule used to establish communication between a client and server

What is a protocol?

- Protocol is a set of rules governing a conversation between a client and a server
- There are many protocols and HTTP is just one type of protocol
- FTP. IP, TCP are other types of protocol,
- HTTP is the highest layer in the network (application layer)

An HTTP conversation

- Set of rules governing the conversation between a web client and a web server
- GET is used to as a request to obtain a Web page
- GET <file path>
- The server has to send this to the HTTP client if the <file path> Is valid
- It is already defined by HTTP -> rule between the client and the server

Network protocols

- This kind of protocols are defined in RFC (request for comments): each document has its own #

Q. Difference between HTM and HTML?

A. They are interpreted the same on any device

HTTP is an **application layer** protocol

- This figure shows the connection between client and server via the cloud (network)
- Web client and Web server are application programs
- Application layer programs do useful work like retrieving Web pages

The application layer is the boss - the top layer

- Your boss says: Send a package to Miami
- Transportation does NOT matter. It just has to get there!
- The application program says: Send the request to the server. It doesn't care about how it is sent, just send the damn message!
- There are 5 TCP/IP layers, the application layer and 4 lower layers

Application layer protocol examples

- HTTP (Hypertext Transfer): Retrieve and view Web pages
- FTP (File Transfer): Copy files from client to server or vice versa
- SMTP (Simple Mail Transport): Sends email
- POP (Post Office): Read email

When sending emails between your gmail account and the UCLA mail server, you are using SMTP to send the email and the retrieval of the emil uses POP

The TCP/IP protocol layer

- Transport layer: Makes logical connection between client-server. It also controls transmission speed and can retransmit a packet if needed
- Internet (network) layer: It cares how to use which route to reach the destination
- Data link: Routes data packets within the local area network
- Physical layer: Specifies the medium between the two nodes i.e. binary ones and zeros are differentiated

Real World Examples

- IP uses IP addressing and routing, so it is located at the Internet level
- TCP is at the Transport layer
- 802.11 is the data link layer

27.13 Request and response messages

- There must be a blank line between Headers and Body
- Body can be present only in some messages
- Request a file without the body

27.14 Request and status lines

- Print out request header from the client and parse that request header and make the response message

- When you read the request line, you will have request type and space

27.1 Methods
- GET: requests a document from a server
- HEAD: Requests information about a document but not the document itself
- POST: Sends some information from the client to the server
- PUT: Sends a document from the server to the client

27.2 Status codes
- 404; Not found; The document is not found

27.3 General headers
- Cache-control: Specifies information about caching
- Look at slides

27.4 Request headers
- Look at slides

W 4 T Lec 1-26-16
- There are limitations based on the characteristics of the shared links
- For some limited set of parties N
- You can have one link that is shared by all of them.
- This is how we have everyone connected on a network so they can communicate with everyone.

Sharing and Relaying
- Let’s say we have full connectivity
- Use relaying where we have information go from point to point to get from the source to the destination
- Makes it possible to support large scale network
- Sharing
- Only scales up to a certain number of nodes
- Has size limits (hard to share across large geographical distances)
- Relaying
- Allows for complex topologies
- Efficient at large scales
- Good number scaling
- Good size scaling
- Can cover the entire world with the Internet
- Still need to work with other planets but it works pretty well on Earth!

Relaying to the Rescue!
- Communication through a “transitive closure”
- If you relay across a whole lot of different hops, you can get to a faraway place with the relaying principle.

- Before, we need N^2 links, which was bad in terms of cost
- Relaying allows us to remove some channels
- You could have as few as N links for N nodes

How can we relay? #1

- 2-party channels
- Use a medium that lets us do an action across a distance
- Use switches to relay between points
- If our message does not get where we want it to go in the first hop, we could do things using circuit switching.
- The green dotted line is the circuit so that the information goes to the top guy.

The good, the bad, and ….

- Good news
- Relaying reduces the # of links you need

Circuits - a sure thing

- Trains on a track
- Scheduled ahead of time
- Tracks are arranged in that particular fashion
- The resources are locked for the entire path
- Guarantees that there is no competition
- Guarantees no competing traffic
- No one else can use a link, only the guy who has it.

Circuit movie transfer

- One long, continuous path
- Certain amount of data
- A need for the data to arrive in a fixed amount of time
- reliably with no interference
- Assuming a perfect channel, there is in order delivery and nothing else gets in the way.
- Reasonable thing to do for data transmission.

Circuits - pros

- You can do all kinds of planning based on understanding how the circuit is supposed to behave.
- Get rid of a lot of overhead that is implicit in the circuit.
- You know exactly where the data is going out to because that is where the circuit has been set to.
- It is implicit in the fact that the circuit has been set up.

Circuits - cons

- Fairness on a per-circuit basis

- Try to be fair, but once the circuits are set up, you no longer have any option but to seize previously allocated circuits
- If you told the guy to get a link with a capacity, he will
- But if conditions change, it will block paths that you can no longer use.
- We will have fewer than N^2 links, so with circuit switching, each link is used for one particular path.
- This implies that devoting a link will remove the possibility that other paths cannot use these paths through the network.
- There will be a limited capacity
- You look at the capacity of the smallest link; it cannot be any bigger than that!
- If you were more flexible about information, you are not going to get this capacity

How can we relay? #2
- Packets to the rescue
- Packet switching is the key that allows us to be able to use our current Internet
- Divide up the use of links into time settings
- Each party gets the link based on time division multiplexing
- Circuit will no longer block traffic
- There is the possibility that multiple parties can get connectivity between the sources

Baran's study
- Having a single link between nodes will give us connectivity
- We don't necessarily have different paths through the network
- With one link per node, we won't have different paths!
- His question was "N^2 is too much, N is too little. What is a pretty good medium?"

Baran's insight
- N links is minimum
- N^2 is the largest # that makes sense
- The happy medium is about 4N, which is nearly as good as N^2 according to experiments that Baran ran
- 4N reasonably approaches the N^2 line and this insight was useful in getting pretty good assurance that we will still have a connected network.

Packet example
- Cars on a highway analogy
- Each car is sort of its own packet
- Getting from a particular point A to point B
- Based on traffic conditions, we don't need to schedule ahead of time
- Multiple vehicles can use the road simultaneously
- This allows us to share the resources, rather than a train.

- If we are driving, we can take many different roads and we aren't bound by path
- Problem of variable delay
- We may have chosen a bad path, and there could be jitter if we don't know the same amount of time
- Sometimes it is quick, sometimes it is slow.
- If we use packet switching, we are going to get variable jitter
- Sometimes, it is quick when you drive, sometimes it is slow depending on the traffic situation
- Variable capacity
- Driving on the freeway vs driving on local streets

Packet movie transfer

- Circuit switching was a good example for movie transfer because we had requirements on how fast things should go.
- Each packet would be sent independently and a decision would be made where to send it next.
- Eventually, all the packets would have to get to the destination eventually.
- This implies that packet 6 may arrive before packet 5 if there is path independence
- If we are moving from Point A to Point B and someone wants to send an email, they can potentially share the same path.
- This is NOT as perfect a mechanism for that particular application
- Sometimes, pieces of the movie won't arrive on time, so the guy at the destination has to be careful about building up the overall thing.
- He may be getting bytes out of order and this is tougher with packet switching

Packets - pros

- Sharing is really good for packets
- We get much more fair usage of links since people are using the links
- In the 1970s and before, they had inefficient use of the circuits
- For packet switching, we can move a whole lot more data
- We essentially don't have to use that link in that period and it is fair over time periods
- Primarily telephone networks use circuit switching, but not too many nowadays.
- Generally speaking, we aren't going to be using circuit switching.
- Packet switching is also more fair over certain time intervals
- Later on, when you get more information, we can over a period of time be fair.
- It will all be much fairer and we can be dynamic and agile
- Anybody who needs one link cannot get there because link was already reserved from the A to B circuit.
- We are no longer limited by the capacity of the smallest link chosen by the circuit

- The limit is our total bandwidth from source to destination
- We can have dynamic path variation
- We can set up a circuit and we would be out of luck
- If we cannot take a path, we try another one and assuming the packets are set up properly, we will eventually reach our destination

Packets - cons

- More work
- Every single transfer has to be divided into packets and at the end, we put the packets back together
- Use checksums because we don't know what happened in the middle
- Capacity overhead
- Every packet is going to have to carry extra information
- Data came in on one link and this implied that it would go out on the other link
- With packet switching, we don't know that and it can send stuff in on multiple links
- There has to be information on different packets
- The guy at the destination end doesn't really need information that allows you to route the packet
- It could be going to multiple different output links
- Fields that allow us to demultiplex and allow us to share multiple links
- We have to use some buffering to handle reordering and loss
- If packets are out of order, we can throw them out and request a retransmission
- Otherwise, we can store the out of order packets and hope that the remaining packets come in later
- This is NOT an issue at just the destination end, you also need to do this with switches at the beginning of the network.

Circuits vs. Packets

- Circuits win when
- You can predict data patterns
- You don't need to share
- Train analogy: Setting up a circuit for a train makes sense
- Service guarantees are vital
- When we use information on the BUS for DMA Transfer in a computer, we need to use this link for the period of time to get efficient and guaranteed service
- Data length is long
- Overhead you pay is going to be too high and you want to make sense of the circuit ahead of tim.
- Packets win when
- Data patterns are unpredictable
- Sharing is important
- Relatively small amount of links, greater requirement to share our links and use packet switching

- We like to have good performance from our networking
- Works very much faster than human perception, when we talk about things done at levels of human perception, we have more flexibility of how things should be from Point A to Point B.
- If you get something within a second, you are probably pretty happy
- Computer has a lot of time and it is much more flexible to decide how to handle and manipulate data.
- Data length is short (relative to path setup cost)

Goal: scalable communication

- AT&T ended up NOT becoming the dominant Internet company in the world
- They focused on circuit switching rather than packet switching
- This is costly and they are NOT dominant.
- We wanted scalable communication (large groups of people could talk to one another)
- Impossibly expensive for what we wanted to achieve.
- We could only go so far -> Ethernet was great and it was NOT a feasible solution for things like setting up the Internet
- Relaying let us set up links and this scaled beautifully!
- The billionth + 1 person costed us 1 extra link
- This sounds too good to be true!
- There are costs, however.
- Sharing channels relates to what we did on the previous channel
- Don't need a master in order to share a channel

Summary

- We don't need a shared channel
- Relaying can be done via packet switching or circuit switching

Network Topologies

- We use some kind of relaying so we don't have N^2 channels and we will have channels that go across long distances.
- We will have N total channels, which nodes do we want to connect
- What form of connectivity do we use?

Parties and nodes

- Shannon talked about parties
- Moving into the world of modern networking
- Uses nodes to move physical information
- Switches, routers, etc.

Channels and links

- Shannon talked about logical connections between parties
- Draw a physical line between two nodes
- Copper wire, coaxial cable, etc.

- Links are physical connections between two nodes.
- Network topologies where links connect nodes.

Ring

- N parties and you can get by with the smallest # of connections possible
- We will connect links in a circuit and there will be unidirectional links
- If 6 wants to talk to 7, he just sends it to 7
- If he wants to talk to 5, he has to go all the way around.
- Usually we have a bidirectional ring where things can go in both direction
- In that case, we can still go in the opposite direction and allows for more efficient communication.

Hub and spoke

- One of the nodes acts as a switch
- There is a single level hub and spoke
- We can get by with N-1 links, and everyone who wants to communicate goes through the switch in the middle
- A multilevel hub and spoke has one green hub and he connects up a bunch of blue guys, it can get to be a very large, complex network.
- If the hub fails, we are screwed.
- Not a great solution unless we have a highly reliable hub node.
- If the hub node fails and we don't need to communicate, that could be okay.
- We don't use hubs much because of this weakness.

Regular mesh

- We would have 6 nodes on each hexagon.
- Depending on the lines on the hexagon, it would connect up to 6 other nodes because of the hexagonal nature.
- This is just how you set up your links and this allows you to tolerate multiple failures.
- Some of these lines get to be long and complicated.

Manhattan network

- Named after the Manhattan grid
- As we can see, most of the roads go north, south, east, or west, and they intersect in that fashion.
- The ends of the island go to the other end of the island.
- This is a 2D torus
- You have limited yourself in a simple way and we can do an analysis that we assume our network is a Manhattan network

Torus

- Instead of a 2D torus, we can have a many dimensional torus
- Each dimension is set of rings
- 1D = ring
- 2D = Manhattan network

- 3D = see slides

Hypercube
- log_2 N links per node
- Not going to be too subject to failures
- The paths tend to be relatively short
- As you double the # of nodes, the number of links per node increases by 1
- Inevitably, if you have a billion nodes, you will have a lot of links
- This isn't used for computers too often, but can be used for parallel processors
- The biggest hypercube that Reiher used was 64 nodes

Q. If you added 8 more nodes, isn't it only the outside nodes that increase their links
A. You will say you have gone from a 24 node hypercube to a 32 node hypercube. There are phantom nodes, which have phantom edges (in principle, those are connections)

Multistage Interconnection Net (MIN)
- You need to go through stages of nodes
- Not used for general relay purposes
- Inside a supercomputer, you may very well use this kind of switching

Examples
- Crossbar
- Clos
- Based on the address, we can figure out which of the outputs we can go to.
- There won't be switching from a guaranteed # of hops.
- Most useful when you want a guarantee on the length of time to travel from one place to another
- Each node should be no longer than a certain distance apart.
- This technology will be used in a box
- There will be a parallel processor that uses a butterfly switch

Irregular mesh
- A choice and usually the choice we end up making.
- Irregular pattern and we have to decide which mesh we are using.
- Lacks a pattern
- We need a whole lot more networking that we thought we would need.
- This is more than what we need.
- Could be a Euclidean mesh
- Could be quite irregular

Impact of topology
- Naming
- The node can be the place in the grid where things are located

- In Manhattan's grid, In one direction, they are numbered, in another direction, they are lettered
- Based on the knowledge of topology, we can map our problems onto nodes
- Divide things up into pieces and put communication patterns to determine which piece goes on which nodes.
- Align these communication patterns of topology -> each bank needs to handle ATM exchanges
- We don't want ATM's to be working on their own, so this adheres well to the hub-and-spoke model.

Scale and Size

- Limits of a single link
- With a single hub, there won't be anything that is further away than one link distance
- Repeaters can spread out the power of links, but they tend to be expensive
- Real links are based on electromagnetic hub spectrum
- There are limits to the amount of power based on physical constraints
- Limits of a shared link
- Protocol limits the distance
- If you have a lot of people connected up to it, there will be power issues
- Talking about real signals and power is a consideration
- There isn't a distance limit anymore.
- No node count limit.
- Complexity limit
- More nodes == more complexity
- Internet has a complex topology and it is difficult to predict how things will behave
- The more relaying we do, the longer it takes to do things
- Takes processing time
- The more often you relay, the more processing time

WAN

- Wide-area network
- Usually inter-city to global

MAN

- Metropolitan network
- Fairly big chunk of a network

LAN

- Local area network
- Usually within a building or a floor of a building
- Typically has a small # of people
- Usually no more than 100 m between participants

- First type of *AN network

Other *ANs
- DAN
- Desktop
- PAN/BAN
- Personal/body
- SAN
- Storage
- CAN
- Car or campus
- HAN
- Home

Networks without a "AN"
- Intranet
- Network internal to a corporation
- Internet
- Network of heterogeneous networks ("inter")
- Access/Aggregation
- Connects multiples LANS and WAN
- Sharing a medium, so we can figure out how to deal with collisions in the wireless network.
- It's primarily a power issue; the federal government only allows a power up to a certain range.
- You don't want your wireless router to stomp on another group's router nearby.
- We are using the air typically to get from the portable computer to a wireless access point.
- At that point, you are using a wireless link.
- In Beijing, they will be sitting there on their laptop. Between the global wireless links, it will be by satellite and they won't be used much anymore
- We generally use cables that go under the ocean.
- The Wilshire building connects Internet to Japan, Australia, and China
- These cables are very expensive to lay and they are lying on the floor essentially.
- Interplanetary
- Very large delays
- Challenging to use network technology to communicate to a rover on Mars
- Limited communication possibilities
- Challenging type of network to perform in.
- Limited basis of things to talk to.

Relaying and choices
- We are making a bunch of choices and sending a packet out.
- Every time it hits a node, it may be destined for a particular node.

- Usually, every node that gets a packet coming in, it has multiple ways of sending a packet out.
- For a ring, it is very simple
- It is either for you or it isn't.
- Sooner or later, it will get to the right place.
- Your problem is sending out the link and slightly more complicated topologies.
- We have three directions we can send it out to the right
- Some issues with relaying

Relaying rules

- What do you do when you get a message?
- Deliver if it if it is for you
- Otherwise, relay the message!

The when question

- A message starts to come in
- *When* do you start sending it out?
- Significant amount of time when there are a lot of bytes that arrive
- Fundamental decision to wait for everything to arrive, or decide something quicker
- We want our networks to run fast!
- We would prefer to make a quick decision rather than a delayed decision.
- Perhaps, you may want for the entire message to arrive.
- Avoid sending corrupted packets
- You don't know if all the bits got flipped until you safely check at the end result.

Cut-through

- We probably need more information to see what links to go on.
- Especially if we decide that our packet is what we do.
- Time is going down, sitting here at the node, and we see the first few bytes of the packet
- If we are doing cut-through, we start sending a few bytes on the packet to know which link we are going on.
- Eventually, the entire message goes out and this will be relatively quick.
- Needs a slight delay but then we are set.

Store and forward

- If we have the entire packet, we know how to relay it and we can then send it.
- Fundamental pair of choices

Collisions

- Different type of collisions than what we saw when we had a shared medium

- Multiple parties use the same input and output link.
- The node will do relaying and if we look at the addresses, we see if both go to the same link more or less simultaneously.
- In come the 2 messages, and we don't want to have a collision!
- We can delete one
- Not an ideal scenario
- We can save one
- One of them will start sending output whenever we decide to.
- The other one will be put aside and once we finish using it, it might not be true and there can be a very popular output link.
- They are a machine that has certain capabilities that are designed or built into it.
- Under some circumstances, depending on how we decide thing, we may use up all the memory available.
- Requires storage
- A buffer requires storage
- Save it in a queue somewhere

Buffered vs. bufferless
- If we had perfect throughput, we could start to approach the perfect line and make full uses of our throughput, unlike unbuffered inputs.
- Buffering is heavily used in computer networks
- Throughput is a question of how much capacity it could use for a given time.
- Generally speaking, we prefer saying we can deliver packets.
- Anything else, we can compromise on that saying we can move data.
- We need to trade some delay and transmit over what it could be in principle.

Queuing
- Treat them all together uniformly in some fashion
- There may be other principles and sometimes, we will drop messages

A short foray into queueing theory
- Mathematical theory of handling lines
- When we are building up a buffer, queuing theory is highly applicable to this.
- Handling lines of waiting customers in efficient ways
- Very applicable to networking
- A fundamental question
- If the outgoing link is busy, we need to form a queue of people waiting to make that link.

The basics of queueing theory
- We have a server
- Server can do things

- We have an input queue
- The server takes some amount of time to send work to an output
- That time generally is not constant
- Work is also arriving at some rate that is NOT constant

What's trouble?

- Trouble can be dropping any message
- Usually considered when your system is NOT stable.
- Maybe it is okay to drop a message as long as you can keep up with the amount of work you want to do.

Little's result

- Found out a principle of when your system will be stable.
- If you can handle L = lambda * W messages, you are good

Drop behaviors

- Tail drop
- N messages that we got are better and deserve better treatment, so drop the one that just arrived (stack-like)
- Head drop
- There have been messages that have been there too long, so we have to push the older members out (queue-like)
- Random drop
- We have N + 1 messages, we can only save N, randomly drop something
- Random early drop (RED)
- Let's not wait before we are full, under certain circumstances, let's potentially drop a message
- Instead of queuing, there is some chance we will drop someone at random.
- Increase probability of drops as space decreases
- Various drop behaviors that are built into the equipment.

Priority

- Some messages are more important than others (George Orwell's Animal Farm)
- Low priority people are more likely to get dropped than others.

QoS

- Quality of service
- We would like to make some guarantees to a node about their messages
- Part of the problem is that we no longer have guarantees about how long something would take.
- We no longer have that guarantee, and it is great for certain purposes.
- We have to deliver frames at a certain rate, or else it looks crummy.
- We want to guarantee a service despite the fact there is a sharing of links.

- We don't have one queue per outgoing link, but rather we have multiple queues
- No guaranteed qualities go into a 3rd queue
- Deal with things in a priority basis
- This ensures we are getting as close to a guarantee as possible.

Returning to the question of choice

- The post office does something similar
- Look at the address of where the mail is supposed to go.
- Figure out where the messages go and take a look at the address.
- Take it to wherever it needs to be sent.
- Naming becomes very important in order to tell what to do with it.
- Specifies what we do with it.

Naming and relaying

- How can we decide if a particular name implies a particular choice
- Assume that relaying node makes a particular choice.

Terminology

- Namespace
- Set of possible names
- Within a namespace, you have multiple names
- Can also be called **identifier**

Naming vs. identification

- Naming:
- The act of assigning an identifier to an item
- Identification:
- The act of selecting a subset of items based on a name
- You already have names, how do we match that item to the name associated with it?

What can be "named"?

- Place (source or sink of data)
- Endpoint, node, process (inside an OS)
- Location
- Communications medium (path of data)
- We can name an entire path which goes from here to there, etc.
- Information (the data itself)
- Having bits and pieces of a movie file
- A behavior
- A sequence of events, an algorithm, etc.

Finding names

- You have to know things like A-priori knowledge
- Not associated with a message that has gotten somewhere else

- Information is lying around to figure out what it should do with that name.

Purpose of Names

- Support shared channels
- Support relaying
- Support layering
- Provide other information
- Support naming itself
- How do we get information from naming to the relaying node.

Support shared channels

- Source identifier (N:1)
- We typically have a source and a destination ID
- We do things with these two IDs.
- Source who sends it and destination that receives it.
- If we were doing circuit switching, we could have a common source and destination in this case.
- Requires coordination if there is only one group.

Types of group identifiers

- All
- Broadcast
- Some
- Multicast
- Any
- Anycast
- Someone just has to get it and the network can decide who will get that particular message

Broadcast

- All everywhere
- Rarely true
- Idealized case but unrealistic
- All within a range of names
- Usually reserved for a LAN
- Received by everyone who is on the LAN.
- All within some metric

Multicast

- We can do the same things as a lower level network
- Networking equipment can identify the correct audience in this case

Anycast

- We want any member who can take on the job
- One of the server guys needs to take care of the job, and they could do this by broadcasting.

- Send in any cast address and it to at least one of them.

Support relaying

- Structure in the name can help
- Name space is a hierarchy and that is frequently done in terms of how we set up our namespace.
- You can typically dial any country in the world using some kind of country code
- You have a DNS country code suffix that can handle a lot of issues
- Every country has a 2-letter code and that is supposed to be a message that is associated with that country
- Zip codes are another example

Types of Name

- Flat
- We don't have any particular semantic meaning
- There is nothing about the way you put the symbols together that matters
- Structured
- Has syntax and semantics
- There is a particular way of setting up things
- Single-level
- Every possible name is sitting in a big pool called the name space
- Mult-level
- Divides it up into multiple levels and this has a namespace that specifies the city and the state.
- We can have many different things i.e. different zip codes, cities, or addresses

People

- Size
- Can be long or short depending on culture
- Organization
- 2-3 level hierarchy of flat
- Family name, personal name
- Almost any name is possible within the family name.
- Content
- Typically there is a surname (family)
- Usually flat
- Can change (marriage)
- Given name
- Flat
- Nickname
- Flat
- Broadcast/multicast support?
- Geographically limited, geo-limited multicast (all brown-eyed people)
- Example

- Indiana Jones (Henry Jones, Jr.)

W 4 R Lec 1-28-16
- Midterm: closed-book, closed-notes
- Names
- We are going to have to associate names with the packets so we can figure out where to send them.
- Many different types of names used for many different types of purposes.
- Houses and Buildings
- Typically a string of characters
- Organized name i.e. The White House
- General address, the house or apartment will have a hierarchy
- If you try to be fully general, you typically have between 4-6 level of hierarchy
- Country, region, city, street, unit, subunit
- There are several levels built into this kind of address
- All of them tend to be flat space (unique names)
- For every street, one side has an even # and the other side has an odd #
- Put them in order on the street so you know where to look!
- Analogous to sorting
- This makes it great for figuring out which is the oldest house on the street
- No broadcast or multicast on for this.
- Geographic
- You can specify any location on Earth with a geographic name
- Size
- 3 groups
- Latitude, longitude, altitude
- 2 level organization
- Latitude + longitude go together
- Specified in degrees, minutes, seconds
- Or degrees and fractions
- It has to be a # and it has to be some degree of level
- Altitude is typically done in meters or feet
- Set a box around a portion of the globe and would specify how you would receive information from that basis.
- Geographic names
- Geopolitical names
- DNS sets up a portion of the namespace for each country code
- .ca
- .us
- .cn
- .jp
- .au
- .in
- Topological names
- Name that I am specifying is in relation to something else

- Each is further subdivided
- Look at written notes
- Each level deeper in the tree appends a character to the end of the string
- Telephone
- Size: 6-N digits
- Organization
- 3-4 levels of hierarchy of flat, variable
- Ordering of #'s doesn't matter and it is always a set of #'s in variable order and variable size
- Some country codes are short and some are long.
- The U.S. got a very short # and some countries have very long #'s
- Region codes
- U.S. has 3-digit area code i.e. 626
- 825 is the # on the UCLA campus
- Telephone companies could use this to set up a telephone connection between a caller and a callee
- Between 310 area code (West LA), let's go to UCLA and find the right circuit
- Flat # (unorganized set of digits)
- No broadcast/multicast support
- Ethernet
- Size
- 6 bytes
- Has an address that indicates which network you are on.
- 2 level, flat hierarchy
- No sub-hierarchies in it
- Set group of binary bits
- Fixed
- 3 byte OUI (Organizationally Unique Identifier)
- Says who built your Ethernet card
- Flat 3-byte NIC (Network Interface Controller) address
- Network Interface Controller
- No other Ethernet will have your Ethernet card
- Obtain your very own OUI (2^24 possibilities)
- For every Ethernet, it will broadcast to everybody.
- IPv4
- Name we put on every packet to say who sent this and who received this
- Has various different versions
- Used primarily today
- IPv6 is the successor version of IPv4 and should take over in the near future
- Sender and receiver
- Each name is 4 bytes long
- Organization
- 2-level hierarchy of flat, variable sizes
- You could view any IPv4 quantity as composed of two parts

- You get to assign any addresses that start with this high order prefixes in any way you want.
- We can get a prefix that is a single byte and the total size is 4 bytes, so there is a limit on how big this prefix can be.
- You either get an 8-bit, 16-bit, or 24-bit prefix
- IANA does this and its purpose is to assign Internet #'s or names
- Many addresses get handed out and they save it for future use
- In particular, the People's Republic in China did not get many addresses
- In contrast, the U.S. got many addresses thanks largely to Leon Kleinrock (badass)
- Gave him a whole bunch of addresses because of ARPANET
- UCLA came to the CS department and asked for their IP address
- UCLA has many addresses
- You can assign any address you want for the low-order part.
- Some of the prefixes are reserved for special purposes
- If you put your prefix on that packet, it shouldn't go out to the Internet
- Only known locally for your own purposes
- We will talk in future classes about why you may want to do that.
- Every /24 address, all 0's would be a broadcast to the local area network (LAN)
- You are supposed to have broadcast capabilities within that network
- Example
- 128.9.160.161
- These are just put in when we write those names out so we can make it easier to figure out what is going on.
- Composed of 4 bytes
- 128
- 9
- 160
- 161
- You can have 2^32 different names, but this turned out to be not nearly enough
- IPv6
- **8 bytes (1 byte = 8 bits)**
- Needed a bigger address
- Wanted to make sure they never had a problem again
- 2^64 possible addresses
- Name of sender and receiver
- In addition to having a much longer name, there is a different organization to how the bytes are set up.
- We are going to have a multi-level 6-byte site prefix
- Set network prefixes:
- Instead of saying some people get a lot, some people get a little, let's say people get a particular 6-byte size prefix
- Total of 8 bytes, so this means you have a 2-byte subnet.
- 8-byte interface ID

- Similar to Ethernet we talked about
- Organizations get assigned one or more site prefixes
- These are handed out in a particular 6-byte prefix
- Two bytes are then used by you to set up whatever addresses you want.
- 2^16 bits which is usually plenty for most machines
- One-hop broadcast
- One of the 2^16 bits address is reserved
- Every single node with 6-byte frequency should get a copy of that message
- Example:
- 2001:0:9d38:6ab8:3c1f:108d:b898:6b35
- Delimiters that are not represented in the real name (in this case, colons)
- The only purpose of the colons is to provide more readability
- If you want more addresses, use bigger names
- Less payload in every packet
- More overhead
- Host names (hosts.txt)
- You have the ability to have a translation between names that are more meaningful
- Set up a file in your computer with a bunch of translations
- Here's a # it matches to, and host names can be 1 to N characters
- Variable sequence of chunks with individual components of the name
- Periods have to be put in for this purpose
- Meaningful
- Labels are separated by "."
- Server machine -> essentially one you are getting the slides from
- Association between host names and servers
- Example
- lever.cs.ucla.edu
- Each level can have diff # of chars
- The dots (.) allow you to separate specific sequences of chars
- No limitations on subfields and you can do pretty much whatever you like
- **DNS (Domain Name System)**
- Human readable names that let you navigate your way around the Internet
- Look a lot like the host names
- Most of what you see in host.txt files is much of what you see in the DNS name
- Can be divided into different #'s of fields
- Separated by periods and the periods are meaningful
- Has to come out from the right DNS name and they can be of variable length
- It can be one character long or 50 characters long
- All labels are separated by dots, and they are delegated right to left
- Some organizations get to have a name starting from a rightmost part
- Entities from that point on receive the suffix

- You will have sub organizations
- .edu example
- You can sub delegate and give authority to another group to prefix any names
- Given .edu, you can go to the UCLA to have ucla.edu
- Given ucla.edu, you can go to CS to have cs.ucla.edu
- And so on!
- Computer Science department looked up lever.cs.ucla.edu, did NOT see it available, and granted Reiher permission to use
- Could be different #'s of characters at each level
- Could have as many levels as you want
- TCP, UDP, SCTP, etc.
- Built on top of IP and uses IP version names
- There are going to be other things and there are particular systems that you look at.
- For port #'s, they are already preassigned and people will use this set of port #'s
- Port 80 is for HTTP
- Port 53 is for DNS
- Strict adherence to these rules, cannot change!
- There are ones that users can register and keep the ports set up
- Others are dynamic and can be switched out of the ports at any time
- After you finish running, you can give up access to the port and free it up for that purpose
- Does NOT have broadcast or multicast support
- Specify one destination port
- ATM
- Low-level form of network communication

Names also provide more information

- Names do more than identify
- Other information can be seen from this
- Cell area code
- The area where you first bought the cell phone
- Doesn't change when you move
- Social security number
- Given to you when you are born
- Helps you for your retirement
- Encodes information such as birthplace and birthdate
- People (lineage)
- Parental
- Ethernet
- It has an address and you can look at that address and figure out who made the card that this particular packet is going to.
- Based on manufacturer
- IPv6

Summary

- Full connectivity won't work, shared channels won't work
- We need to relay!
- More scalable network
- Arrange network with some topology
- Usually we have some degree of organization
- Topologies have a very big impact on reliability
- Share a bunch of stuff
- Needs more shared rules to avoid congestion and collisions
- Usually we handle this by queuing up the packets and we have to have some rules for what do we do when buffer space is full
- Cannot do relaying without naming scheme
- Primarily IPv4 and IPv6
- We stop using IPv4 and IPv6 names at the edge and we usually use another address like MAC address for Ethernet, Wi-Fi, etc.

Layering

- The Internet made it clear you have to layer to have global level communications

What is a Layer?

- The largest set of parties that can communicate
- Using a single protocol
- Single name space
- You can communicate to anyone in your layer
- Agree on same protocol, same namespace, etc.
- Very consistent conventions for people on the same layer
- Ability to agree is very important
- They all behave in the same way
- This could be a layer as well
- Reach wireless network and have a wireless card
- There could be a shared namespace which will be the address and they will use a protocol where they will share a medium without a master
- Transitive closure of a set of nodes via communication

Limits of a single layer

- Homogeneity
- Homogeneity 1
- No important optimizations based on context or environment
- Cannot say you will do this differently if you are connected from point to point vs wireless
- You should see that the things you want to do in the wireless network environment are very different from the tasks you want to do between two computers connected via copper wire
- Make some choices and see if you want to avoid being in bad shape
- Makes it difficult to pick which layer you work with if there is only one.
- Hard to customize and optimize for particular scenarios

- Homogeneity 2
- Incidentally what phone companies do (one and only one layer)
- Circuit switching
- Don't build anything on top of it and do it our way or go away.
- No incremental evolution
- Needed an answering machine that was a big technological innovation
- Call waiting and simple innovations
- Hard to do any innovation whatsoever
- Run into a problem where everyone makes a mistake
- We need to do something else
- Create a flag day!
- Everyone using a flag turns on a switch and transitions to a new protocol
- Examples:
- Multics transition
- NCP to TCP
- Very incompatible
- Everyone got together and flipped a switch to turn off NCP and turn on TCP
- This is a **TERRIBLE** way to handle change
- It would be disastrous with this large scale of Internet users
- Very unrealistic
- Q. What if someone forgets to flip a switch at that time?
- A. Unless a fairly high # people switch, you probably will have a disconnected network
- Complexity
-
- They would essentially be cut off from the network unless they change.
- A lot of complexity to deal with as many cases as possible
- Difficult to manage, difficult to route
- Hard to coordinate many things such as changing the paths
- One, huge name space
- Essentially groups of layers can make it much easier (modularize the layers)
- We had to have layered protocols to do different things in different protocols
- Q. What if you have too many layers?
- A. It has overhead because there are costs involved in running some of the layers. Some of the layers don't bring much value and it is more costly to include these layers.

Reasons to have multiple layers

- Abstraction
- Model
- Allows you to abstract away many features that aren't necessary in understanding and using things

- Picasso’s bull that go from very detailed to general but you can still recognize that it is a bull
- Abstraction and Layering
- We can communicate without going into too many of the low-level details
- The layering has allowed us to abstract away many details
- Do not worry about all the little things
- Protocols don’t have to be complex enough to think about shared mediums and other channels
- There will be behaviors that you simply will not see.
- Fewer nodes that you will need to worry about.
- Don’t worry about the other billion nodes that will be on the Internet
- IEEE 802
- Inherited a lot of computer stuff (cosponsored by ACM)
- Organization that set standards
- Set up 802 (networking standards)
- Addressing
- Formatting a frame
- Data detection
- Error detection
- Lower layer
- Bridging (switch control) and configuration
- Emulation
- Pretend we have something we don’t have
- Valuable characteristic to have in computers and networks
- Replicate the capabilities of something else
- Try to put things together so that several individual things provide capabilities you will have in another system
- Emulation and Layering
- PPP (point to point protocol)
- Use a modem and get to a phone line
- Don’t worry about going across the modem, ran another layer called PPP
- Hid a whole bunch of details from the dialup connection
- Internet generally had more bandwidth than your modem had
- Containment
- I want to have a network that behaves in a LAN (physically or conceptually (machines belonging to a company))
- Treat these guys as one unit
- One namespace or one protocol
- Ensures that everything that happens is well understood
- Specify sets of names and enforce limited functionality
- People on the Internet can run a set of protocols that they won’t be able to run.
- Broadcast in a way that people won’t be able to do.
- By having a layer of my own, I have a boundary where I can enforce other boundaries.
- Containment and Layering

- Put up a firewall
- Hides a network from the world
- Hide/protect the world from an experiment
- Run their own protocols and set up whatever is in their own environment.
- Point at which they contain the badness in their own experiment.
- Limit any possible damage of their own system.
- Scale
- Limit complexity
- The moment you inject a network in the packet, we can solve it using more layers
- We will route in the following way and then we will hand it off to someone else
- Do whatever he is going to do
- Hand it off to another layer and get the packet to its ultimate destination
- Overcome physical limits
- Where your shared features can reach
- If you want to share at a wider level, you need to switch to another layer in order to achieve the scale that lets us get globally or even city-wide
- Scale and Layering
- Tree routing
- Use layers and we know who our descendant layers are
- Particular place is supposed to go right there and we have to route that.
- If we don’t know layered tree type routing, we will push it up to the next layer.
- If it knows, it routes down.
- Make sure packets get to the place they are supposed to go
- Take care of any addresses in that route
- Don’t worry too much about details that are hidden due to the layering
- Cloud routing
- I am told you can do something for me and we can have a lot of complexity
- Autonomous systems on the Internet
- Verizon is an example
- You route via autonomous systems
- Handle addresses of the following nature and you may have route things to 8 different nodes.
- This allows me to treat your layer as a cloud and you can do it.
- Don’t worry any further

Connections and layers

- Dovetailed
- Linkage of layers going from one to another
- Stacked
- Your layer is on top of my layer, and I presume there are other layers below your layers
- Talk about the differences about using these two styles of layers

Translation gateways

- Dovetailed layers tend to be separated and this causes problems because the layers don't speak the same language
- Convert what is understood over there to another
- Two-way translation between the layers since they won't necessarily understand
- Translate based on two different pieces of knowledge
- Presents two views
- Each side sees something different

Benefits of adapters

- Does what is best based on the medium
- Can make local optimization and make choices on its own
- If you want to run a newer version of 802.11, you don't have to change anything in his subnetwork, so don't worry about it.
- Backwards compatibility

Lost in translation

- Based on the fact that protocols have idioms and are meaningful in that protocol
- True with people as well!
- Telephone game
- If you come up with a phrase and you keep doing that through 20 people and ask what is the phrase, there is a high probability you will lose things
- Same thing can happen when you lose translations in networks.
- There is a high possibility that translation is imperfect

Semantic gaps

- Turing machines can have "completeness"
- Completeness: Measure of how applicable something is to ALL cases (compatibility)
- Certain things under certain circumstances cannot properly translate
- What works out here may not come out quite right
- Some things just don't translate
- Idioms
- "Ballpark"
- "drop a dime"

State problems

- In order to translate properly you have to maintain state of what happened in the past
- Translation gateways require the same thing
- We think that our status is the following, and they agree on this.
- The state you have on your side is NOT necessarily the same state on the other side
- Translations and keeping your state updated.

- If you are not careful, your states will explode.
- What if the guy has an error and loses part of his state?
- This means the protocol will not work perfectly in its translation.
- We need to make sure information goes through all the layers
- If you have an alternate path but goes through different layers
- We may NOT have a truly consistent method and neither path is completely right.
- I will have two different gateways communicating with each other
- If you go through a bunch of gateways, it gets to be quite complex

Mutual modeling

- My world
- Proxies that know information about the other side
- Your information is only as good as the proxy
- What happens on your layer depends on the state of your proxy, it all depends on the information you receive (good or bad)

Scale of translation

- Each layer has more languages
- Between every pair of layers has to be a translation gateway
- The more languages = more translators
- Explosion in the # of translators you need
- You are going to have a requirement for more and more translators
- The more languages you add, the more translators you will need.
- Get someone involved in each layer to work with someone else who is involved in other layers.
- Full understanding requires someone from both teams working.

Stacked layers

- The other option is to stack them
- Put one on top of another
- Every layer in the stack has its own responsibilities
- Things that it is supposed to take care of.
- Don't worry about the problem
- Just say that he will take care of the error so don't worry about it.
- Allows us to have customized layers and we don't have to worry about lower layers
- Each network is doing more or less the same thing.
- Different protocols that helps us get the packets to me.
- Handle packets for a particular character and they can do whatever they want with their own layer.
- Try to depend on the layer directly below you.
- Limit your knowledge to the interface you are given
- In a transitive closure sense, you will depend on things below you, but you make the assumption that your descendants will do what they are supposed to do.
- Icing connects the layers via translators

- A network isn't exactly like a cake but it is more like it than you think

Internet Model
- original model all did things totally differently
- It was supposed to be a layer that connected these subnetworks that couldn't communicate with each other.
- This was the model for the Internet
- How do we make these different nets work together?
- One layer to rule them all! (Lord of the Rings analogy)

IP Layer
- The common Internet layer
- Essentially the key to the Internet's success
- Says "I am a packet switching network"
- Let's have a variable size instead.
- Include source and destination packet instead.
- Within the IP layer, we will have a one address space and each is supposed to have a common namespace
- High bits are assigned by a central authority
- Low bits are locally assigned
- Best effort communication
- Try to deliver it to the address you specify
- You won't guarantee that packets you send will be in order.
- It is NOT your problem to ensure that delivery happens in order
- Layered protocols can be kicked up to a higher layer.
- Some people care about loss and reordering
- Others care about simple forwarding
- Assume the network is not very smart
- People can NOT deal with the problems, so we deal with this by handing the problem of to someone else.

Metcalfe's Law
- If you have N elements in a network, how powerful is your network?
- He said N^2 because you have possible pairs communicating
- Other metrics
- This suggests that you have 2^N possible subsets
- The best possible network is one where you maximize N
- This is why we built the Internet!
- Let's build something that takes little tiny N's and form one big N

Internet principle
- Talking to everyone poorly is more important than talking to anyone well
- Get a bit from here to there than worry about moving data very efficiently
- It is more important to send a message through the network than for the message to be encrypted

- Long story short, **Capability** is the most important characteristic! Screw the other metrics
- Move the bits, that is what the Internet is about.
- Being able to communicate is everything according to the Internet principle

Benefits of a common layer

- Common expectations
- All the things know if they buy into IP, they can be assured of certain things when using the Internet
- This encourages them to say that whatever they do locally should be forgotten when you get to the translation gateway.
- Don't worry if he is a wired network, just do what the common layer wants me to do and just do it well.
- Be able to communicate as you go through the class!
- Stacking supports the recursion

The Hourglass Principle

- The little tiny waist in the middle limits the rate at which sending go shortage
- Let's say we have a bunch of high-level protocols
- Web browsing, network file system, etc.
- Down at the bottom, we have a whole lot of ways we can use the communication channel
- For every high-level requirements, we will figure how to build PPM, CDMA, NRZ, etc.
- We want to run HTTP over that.
- Do the same thing over DNS, FTP, etc.
- Wouldn't it be nice to map these things?
- Each time, we find new ways to map things and we can do that.
- All these lines are translators that we are talking about.

The Narrow Waist

- Everything translates to IP
- Move files around, access files remotely, play a game, etc.
- Move things down via IP and eventually it will get where it is supposed to go.
- Once we have where it is supposed to go, it should go back down to IP and everything gets translated down in a fair manner.
- That is how we are going to do transmission!
- All high-level protocols should understand IP
- Every one of the methods of moving bits around has to understand how to translate that movement into frequencies moving through the air.
- Everything translated into IP in the middle

How many waists?

- IP is one waist to our hourglass.
- 802.*
- All of the hardware you buy will match one of those 802 protocols
- Everything pretty much matches that at the low-level
- Throw away a lot of other locations and many games are HTTP
- Youtube videos use HTTP
- Gmail uses HTTP

Issues with a common layer

- Design a waist that is successful and this means we have thrown out a lot of possibilities
- Throw about possibility for reliable delivery
- Lowered expectations of what we can expect
- Do things in less efficient fashion
- Moving data to a wireless network at the other end, move it to a different source point.
- Doesn't expose you or optimize anything.
- If the least common denominator isn't sufficiently strong, it won't work out too well.
- What if one purpose wants to do a file transfer to the rest of you?
- Using the same wireless access point.
- Take file transfer and transfer it into 802.11 packets.
- Set the speed of transmission to match that congestion level
- You take the packets on your machine and wrap them up into an IP packet
- Goes up to the wireless access and drops it down to the other machine
- IP will get stripped off and you will do whatever you are supposed to do with the file transfer.
- You were talking to him, so you want it to go across the same network through the same wireless medium.
- Paid a cost of overhead and potentially TCP overhead in order to have generality.
- Customize my transfer regardless of where your location or my location is.
- The easiest thing to use is IP in general, so we use it even if it is NOT the most efficient.

Why has stacked layering won?

- Benefits outweigh the costs
- Critics as recently as 20 years ago said to get rid of a common layer
- Otherwise, it won't be sufficiently efficient
- IP isn't going to cut it here
- Parallels "home field advantage"
- European Latin -> universal language of Europe in the olden days
- The lingua franca (native language) of networking developed as networking itself developed

- More likely to do things that are translated correctly into IP and it's a common idea for everyone to work with.

How layers are used

- Dovetailed (peer) layers
- Translation
- Guy on Net A has it in a format that is understandable and usable on Network A
- There may be a translation of how to divide things into different packets as well as bit representation of data
- Peer - what is translated?
- Guy on the other end may not be speaking the same language at all.
- Don't think of layers as being distinct based on physical machines
- Be a little careful about not being fooled that layering depends on physical machines
- Stacked Layers
- Where does communication go?
- Net A wants to talk to Net C
- They will communicate through the Internet
- We are still going to model this as if they were communicating with each other
- He translates it and hands it to another guy on the Internet and we think of this as a subnet to subnet interchange
- Relay a common language and translate it back
- Hit Ethernet on the other side and completely change from IP to something else.
- Deliver it in that fashion
- It doesn't actually work this way using subnet to subnet transfer
- Staying in the waist
- Common as primary
- Send IPv4 packet so send it across the Internet and he understands the guy on the other end knows this information.

W 4 Dis 1-29-16

- No specific requirement, about header files for the project
- Test web server on the grunt web browser
- Ask the grader to use multiple browsers: there shouldn't be problems because of the types of browsers
- Chrome checks some header fields that is NOT checked by other browsers

Transport layer vs. Network layer

- Network layer
- Layer 3
- IP address is used for identifying a host

- This allows your computer to communicate with other servers or other hosts
- Even those outside UCLA
- Deals with logical communication between two hosts
- Transport layer
- Layer 4
- You are using the port # to identify a process
- What is the difference between a host and a process?
- A process runs on a host.
- Each process has its own port #
- Let's say a web browser has port # 80 and P2P uses port # 5678
- Each process has its own port # and each host has its own IP address
- If you are using Internet and also you are using Wi-Fi, these two connections have different IP addresses
- One host can have multiple IP addresses
- You can find out a specific application that is running on the host.

Network Layer

0. Addressing
0. Routing

TCP vs UDP

- UDP: contains port # information, checksum
- No-frills extension of "best-effort" IP
- Does NOT guarantee connection between two hosts
- TCP: reliable and guarantees in-order delivery
- Sets up a connection via three way handshake
- If there is a corrupted packet, it throws it away and requests retransmission of lost packets
- Three different ways to ask for a packet i.e. positive acknowledgment. How does adding a packet work?
- Just keep sending and acknowledging them to start over.
- Cover all different types of reliable protocols
- Controls flow between two hosts and handles network congestion

Multiplexing and Demultiplexing

- One host can run a lot of applications and the way to send packets is by connecting to the Internet cable and you are running multiple applications.
- This is why we need to multiplex these packets from these applications.
- The PC or laptop can multiplex the packets from the applications in order to send the packets to its own destination.
- A host can receive IP datagrams
- Each IP datagram has source and destination IP addresses
- Each IP datagram carries one transport-layer segment
- Each segment has source and destination port #

UDP (User Datagram Protocol)

- The UDP header has four spots of information
  0. source port
  0. destination port
  0. length
  0. checksum
- Whenever it receives applications data from the process, it forwards the UDP packet.
- Doesn't establish a connection and thus, no connection state
- Header is 8 bytes (small packet overhead)
- Length represents the length of UDP header + UDP data
- One thing you should know is that UDP and TCP use checksums.
- It NOT only checks the data part, but also the header part.
- This figure shows the range of the checksum
- UDP header with application data
- This information is carried by the IP, which carries protocol information so you know what is the next transportation layer protocol
- This protocol specifies that this data is UDP packet
- Why do we need to check this part as well?
- Check the port # as well to confirm that it came from the correct source.
- Listen to the data that has been sent
- The source and the destination are aware of the logical connection
- TCP uses the handshake method before it translates the data
- UDP doesn't care about the type and the destination can receive some packets.
- No way to find out how many lost packets you have.

Popular Applications That Use UDP

- Multimedia streaming
- Telephone calls, video conferencing, online gaming
- Real type applications so you don't want a delayed packet sent across a network.
- When you talk across the phone, you don't want any delay because it causes confusion
- Retransmission is NOT working here because it is real time transmission
- Simple query protocols like Domain Name System (DNS)
- Uses UDP because it doesn't want to put more overhead
- It carries the packet size
- We still use UDP because it is fast and is only 8 bytes
- TCP is at minimum 20 bytes so it is a lot bigger
- TCP has greater packet overhead

We are using the main server to ask the IP address of each domain name

- This guy asks for the IP address for CNN.com and the main server gives back an answer, which is the IP address
- Very simple example of DNS

- Structure of DNS name servers are NOT that simple
- Domain name is hierarchical

Principles of Reliable Data Transfer

- How to deal with bit errors?
- Error detection (e.g. checksum)
- Receiver feedback
- Retransmission
- Sender could think the packet was lost so they might send a new packet.
- How can the destination host figure out if two packets are the same?
- How to deal with duplicate packets due to retransmission?
- Sequence number
- How can the sender detect that ACK or data is lost?
- Timer

Reliable Protocols

- Stop and Wait
- Go-back-N
- Selective repeat
- Requirement of 2nd project
- TCP
- Don't go in depth for an undergraduate class

Stop and Wait Protocol

- The fundamental technique to provide reliable transfer under unreliable packet delivery system
- Waiting for feedback from the receiver (**acknowledgment)**, before transmitting the next one
- This is called stop and wait. Send one packet, stop and wait (**ACK**) from the receiver
- Uses **sequence number** to avoid confusion caused by delayed or duplicated ACKS
- Look at diagram on slides
- Left side is sender and right side is receiver
- This diagram represents the communication between two hosts
- Sender sends pkt0 and rcv receives pkt0 and gives feedback.
- After receiving ACK0, the receiver sends the next packet, which is pkt1
- After receiving ACK1, we are sending pkt0 again
- Same sequence number since it only sends one packet.
- Doesn't need a large range of sequence numbers with a series of zeros and ones*
- This label shows delayed packet transmission
- Receiver receives packet1, congestion is delayed and ACK1 packet arrives on the sender side after timeout.
- Before that, the sender thinks the packet is lost and you have to rcvACK1 again because it is delayed.

- Sender doesn't care if it is the previous one or not.
- Send pkt0 again and we don't want to assume packet loss.
- The retransmitted packet is received by the receiver and sends ACK1 again.
- Sender needs to ignore this ACK1 because it is already received

Project 2
- Reliable data transfer protocol
- Go-back-N / Selective repeat
- TCP
- Congestion control (extra credit)

W 5 T Lec 2-2-16
- Testing familiar, basic concepts
- Short answer questions
- Understand what the readings are on.
- Not expected to do proofs or serious calculations
- From the Shannon paper, there is a lot of formulas and math. Do we need to know the math?
- If you have the general idea, you don't need to know everything. You won't have to have proofs and math, but you need to know pros and cons.
- Why is Entropy a good way of understanding information content of the source.
- More of it will be from the lectures than the readings.
- Study readings as well!
- Includes the content covered today!

A layer vs. layering
- A layer (one layer)
- Homogeneous network with shared attributes
- Namespace
- Protocol to communicate
- Largest group that can communicate via transitive closure
- Layering can achieve effects and we can connect layers edge to edge, or in a stackable format.
- Create something in a bigger network usually starting from other, smaller networks that are quite different.

Stacking via transit
- Transit purposes can unify several different layers
- Intention of the Internet
- Take a whole lot of little tiny networks that couldn't communicate with everyone else, and create a big framework that could communicate with everyone else.
- Translate capabilities
- Moving between one network and another and achieve commonality of abilities.

- Adjust some things for different capabilities of different networks.
- Protocol will require you to keep a state machine at each of the communicating partners.
- What other services could you provide?
- You could provide almost anything!
- The features will be implemented using either hardware or software
- We can do that with a great deal of computation but that is something a layer can do for us.
- We want to add capabilities and add new things we can do with that network.
- Implement the new capabilities that go on top of everything else.

Useful functions

- Emulation
- Don't want people to worry that they have a more complex network.
- Can we emulate a single link across many links.
- Splitting
- Used for big message networks who want to communicate with networks who prefer small messages.
- Deal with problems of noise and errors.
- Try to correct those errors in a layer.
- We want a few messages arriving in a different order.

What defines a layer?

- Use a single set of rules.
- This means they can talk among each other.
- How they do it is an element of the definition of a layer.
- Protocol
- Implies what services we can provide in the layer.
- Does it provide reliability?
- Is it more like a wire or is it more like packages?
- Heterogeneous layers
- This is a sense of what can you combine together into a group?
- 802.11 wireless is only good for 802.11 wires
- You cannot combine an Ethernet with a wireless connection
- OTOH, the Internet lets you do this and allows everything to operate together.

Layers as building blocks

- What are we expecting from below and what do we expect from people above us?
- Each layer expects that the layer below it accepts certain inputs and gives out certain outputs.
- The transport layer will provide capabilities of moving large quantities of data from one place to another.

- **If the transport layer is TCP, it will give ordered messages or messages in order with no loss.**
- **If you say it is a UDP, it will give the messages on a best effort basis.**
- Different layer Notes (**ATNLP**)
- Physical allows you to move information from one side to another with just a channel in between.
- Single hop, not multiple hops.
- What is usually done in the definition of the physical layer and Internet stack is that it sends a bit from one side to another.
- Physical layer defines parameters for high and low voltage
- Linked layer is about moving a message of some kind, NOT a bit!
- It encompasses a larger thing. The Ethernet is an example.
- We sent frames that are this long with headers and the following set of bits.
- We look at this particular set of bits and it tells me who receives a particular messages.
- Network layer: We have an entire network of multiple links
- Let's get a message from one of those places to another place.
- Worries only about individual messages in the Internet packet.
- For IP, it is about the best effort
- If you wanted to say you have a movie here and you want to see the movie there, it is NOT the network layers problem.
- If you want to get the movie from here to there, there is a transport layer that does this.
- Transport layer: Big transfer of data, a phone call you want to maintain, online video game i.e. Xbox Live or LoL
- At that layer, we are now talking about a longer-term thing.
- No longer a single packet, maintain a long term relationship between two parties.
- We now worry about things like sending a bunch of messages.
- Did all the packets get there?
- What if we have congestion in the network?
- We worry about all this stuff in the transport layer.
- Application layer: Chooses a transport i.e. TCP
- **TCP is for reliable delivery but it has huge overhead, so it is slower.**
- **UDP is another example with less overhead but it is faster. Info is less reliable.**
- Given what the transport layer and what the application needs, the application layer figures out which transport system to use.
- There are real-time constraints.
- Streaming a movie needs to drop frames.
- If you save a movie to show later, you would like to show the whole movie.
- What do I do at the application level?
- What are the needs and requirements?

- Transport layer cannot operate unless there is a layer that provides information (network layer)
- Assumption that we get packets from the underlying layer
- If you would like to move a packet from one side to another, the layer above is expected to oblige that request

Layer Expectations

- Talk a common language for each layer.
- Each layer will share a namespace.
- This implies we take whatever name and translate it to a meaningful name at a higher layer
- Your layer is supposed to do this. Is it exactly what I want?

Things you can do with layers

- Each network can communicate with its own members, but you can't talk from Ethernet to the STAR network.
- InterCommunication problems between various networks.
- **All of them should talk to the Internet, we need a common language between networks!**
- They cannot talk directly to each other, but they all can talk to the Internet
- Messages go out on a token ring and this bridges communications between layers.
- Create new services
- You can deliver packets and IP will oblige that request for you, but if you want to move files that contain medical database of important information, you want to know that they are in a proper order.
- Makes sure that these problems are resolved.
- Resend that packet and make sure there is no congestion.
- We keep adding new traffic and new requirements to that link.
- It doesn't go up as you add more traffic, but rather it goes down!
- If you hit congestion in the middle, let's back off a little.
- Maybe everything will get better.
- I would like to create a new service, something that works in the face of congestion
- Build a TCP layer on top instead!
- Glue Layers
- Put together different requirements and expectations
- Lossy -> reliable
- Likely to have losses (used a fair amount in networks)
- Noise is essentially loss, and we can create encodings.
- Reordered -> ordered
- Jittery -> periodic
- Organize the time interval when you receive a message
- Stream -> message, Message -> stream
- Change a stream of data into a bunch of separate messages and vice versa.

- Multiple stream -> 1, 1 stream -> multiple
- Combine multiple streams into one or break up 1 into multiple.

How layers interact

- Translate from the level down to the lower level
- Creates common services for the lower level.
- The IP protocol will help you build TCP to guarantee in-order, reliable delivery.
- Create extra things using IP layer.
- Use lower layers and have names that you use yourself.
- Expect when you go to the upper layer that it understands the name you are using.
- The TCP name is essentially a combination of IP names and a port #
- Use this as part of your name and the TCP layers.
- If the IP layer gives it to the TCP layer, it will know what to do with the packet.
- You can handle moving stuff around as necessary.

Things that make you go hmm…

- Upper layer
- Expects its name
- Provides its services
- Lower layer
- Translates its names
- Creates its services
- Build an interface that lets you use a layer without completely changing everything you are using.
- Have at a very generic level the ability to say that all of your many layers do things on the way up vs doing it on the way down.

So what really defines a layer?

- Largest set that can communicate with a single name space and single protocol.
- Network has a high probability of success
- Every layer is its own one unifying thing.

Examples of layers

- Ethernet and ATM (physical layer)
- Implemented by hardware.
- Above Ethernet
- IP: general thing that is not tied to hardware or network architecture
- Works on any kind of network.
- Above IP
- TCP, UDP
- Above TCP

- HTTP expects to get the whole webpage and doesn't want to be missing pages.
- Worry about asking for webpages and getting webpages.
- How do we get reliable delivery of those webpages?

Different expectations and capabilities

- HTTP
- Provides request/response messaging
- Wants a webpage and implicitly a request-response kind of thing.
- TCP
- Two way communication, make sure it gets there effectively
- It cannot do something all on its own but it needs to get messages elsewhere on the network.
- You want reasonably timely delivery. You can get messages that take more than two minutes.
- Loss at the level that TCP provides above it due to congestion.
- We really don't want errors or flipping bits.
- We do NOT want disorder.
- What TCP does is deal with the ordering issues itself.
- Does NOT expect alterations in its packets.
- IP
- Relies on possibly several different networks
- Ethernet: Come through without error i.e. using checksums.
- Messages could be lost but we expect a shared message on the Internet.
- Limitations and expectations
- Wire that everyone shares.

Messages inside messages

- Grab a message off the wire and look at what you got.
- Messages inside messages
- The layer above it wraps stuff around it.
- Frame captured off the Internet
- These are bits pulled off the wire.
- Moves all the protocols as one unified thing.
- Very structured, hierarchical organization.

Messages spread across messages

- This can happen in IP in particular.
- Say we have all these networks sitting at a ring.
- We have a part of the protocol that is x bytes long.
- The problem is that not all the bytes will be the same length.
- If there is a difference between any of the packet lengths that each node sends, there needs translation to be involved to be compatible.
- The long messages need to split up into several short messages.
- Can he accept packets of my length or will IP worry about that for you.
- If it runs into part of the information it goes through, it splits it up.

- Represents the same message and combined together, they are a single message.

Summary

- Layering allows us to create powerful networks from components.
- Start small, create big things with many modules/components
- Ultimately, every computer can add, move, shift, and perform logical operations.
- Up at the top, we want to run algorithms to perform certain actions.
- At the bottom, we use AND, OR, +, move, etc.
- We have to abstract things away at the high-level, but the low-level implementation is very complicated to work with.
- Web designers use middleware that sits on top of a programming language that sits on top of hardware.
- Stack layers for transit
- Big networks that have many members have more power.
- Having the power to use a computing or network device is achieved by stacking layers of protocols
- Layer requirements
- Expects things from levels below it.
- If you don’t get what you expect, you cannot do your job!
- Layer capabilities
- Provide many features and implementations.

Network Layering, Naming, and Name Resolution

- The layers have to share a namespace.
- Different layers can have different namespaces.
- Layers are supposed to work with each other.
- This implies taking something and converting it into a different layer.

What do names mean?

- Names are just labels?
- There aren’t any necessary meaning besides pointing to that thing.
- Unless you know you have something built into what you are dealing with, you shouldn’t take things at face value.
- The thing you can be sure of is that it is unique within a layer.
- Sometimes, if you look at a name, you can understand the physical location.
- You can understand some things about the network location sometimes.
- Depends on the name and depends on the circumstance.

Impact of multiple layers

- Take something at one layer and move it up.
- The thing moving up through multiple layers will have different things at different layers.
- We will have names for things at the middle.

- The ability to have different layers is by relaying.
- The things in the middle are NOT destinations themselves.
- The relays themselves will have names.
- This all implies we will have to have name translation.

Multiple name spaces

- The mere fact you have the same name in two different layers doesn't mean it points to the same thing.
- As you go from layer to layer, the names won't necessarily be the same as before.
- As we go up to the IP level, we go up to the IP address.
- It probably has a DNS type name referring to our machine.
- Very different types of names dealing with characteristics of those names.
- It could be that at each of these levels, we are running a protocol.
- The state machine could have some general idea of the other state we are communicating with.
- Have a name kept at each layer that we are communicating with.
- If we don't have a name, what is going to happen when you work with this message?
- There are limits because there will be ambiguities.

Translation

- Mapping between names labels)
- DNS: translate the DNS name to the IP address
- Do the translation in multiple directions in both cases
- Google

Layering requires translation

- Change of layer = change of name
- There is a name coming in from the layer below and above.
- For your layer to do its work, it has got to know the names of the things it is operating on.
- Find a translation -> find a local equivalent about the thing it is talking about.

Example: my PC to your MAC

- Labels in the common layer:
- My PC = banana
- Your MAC = apple
- Label when we're the same lower layer:
- My PC = silver
- Your MAC = copper
- They can no longer use the labels in the common layer. They have to use the labels on the CURRENT layer.
- Even though they are referring to the same thing, we have to know which namespace we are currently in.

Getting there…

- It doesn't know about silver for copper, it only knows about apples and bananas.
- Apple and banana
- It might be you are talking to silver and copper, but it gets translated when you get on the Internet
- Create the IP names at the edges because the Internet has become such a popular protocol.

Layering

- Nodes in the network have multiple names at multiple layers.
- At the higher layer network, there is a node called A.
- Two names at one machine.
- Bridge between two networks to communicate at a higher level.

Result

- Each layer has its own name, so layering requires resolution.
- Requires certain other things that will be helpful in networking.

Name semantics

- It is merely an indication, we can have names that do deal with semantics though.
- Route
- Name could specify a route
- Ownership
- Describe who owns the information now and who suspects the name to change.

Naming human factors

- Can we guess the name off the top of our heads?
- It depends on the character set we use.

Expressing names

- IPv4 - Decimal
- IPv6, Ethernet - Hexadecimal
- Old DNS: a-z, 0-9, "_" but cannot start/end with "-"
- New DNS
- There needs to be backwards compatibility
- This is how you group components of the name
- 4 groups of 8b
- IPv6 has groups of 16 bits, we will have characters that don't occur on the basis of human context.
- 0's in the long run won't be all that important.
- The implication is that everything in between is 0's.

Remembering names

- DNS's original purpose
- Easier to remember text than it is to remember #'s

Guessing names

- It turned out to be useful for DNS's evolved purpose.
- Apple.com
- It might take you to Apple records instead (Paul Mccartney)
- Apple computers (Steve Jobs)
- US copyright has multiple categories
- One set of names -> either Apple Computer gets it or Paul Mccartney gets it.
- Guessing won't always work.

Naming management

- We won't guess DNS names as much as we used to.
- There are issues of the management we have to worry about.
- Names change and how do we deal with these changes
- Jump to an indirect place and how do we get this information.
- How do we build a system that supports naming and name transmission.
- Have an authoritative method of assignment
- There is some guy and when we want a name, he will give you a name.
- If we work in a local network, we don't have to go to the authority.
- Let's automatically look at the names we got locally.
- Worry that we might have collisions and how we avoid those collisions.

Name Dynamics

- Add names
- Happens all the time
- Remove names
- Entity is going to disappear from the network
- Change names
- Used to have name X and we want to change it to have name Y

Adding names

- Not just "collision avoidance"
- In the U.S., we can't just say we figured out some houses on the street.
- Choose the appropriate number to correspond to your system.
- If there are no integer sized numbers, this can cause issues.

Removing names

- Easy to stop using a name, but how do we send this information out to everyone else in the system?
- How can we say that "foo" broke and inform everyone else?

Changing names

- The object isn't going away but we want to rename it.
- You have to avoid the collision as you would for any addition.
- You have to inform people about the change.
- Send things and it will get dropped because it doesn't mean anything anymore.
- Tell them explicitly.
- The packets eventually get dropped and they get bounced back to the sender.
- Let's forward it to the new name and move it on.
- Here we are working within a single layer.

Benefits of Indirection

- Allows us to do self management
- We can choose some IP address and assign it.
- Otherwise, we can have someone else figure it out for me.
- Lever vs IP address
- Express lever and use indirection to get it to the right place.
- This machine may get different IP addresses when you are at work.
- You would have different IP addresses depending on if you at work vs if you were at home.
- You could work with a permanent name that is not tied with a different location.
- Express the location of the machine without worrying about physical location.

Name overloading

- Aliases
- Multiple names for one party
- Several different names and you can access the application with one of those names
- Proxies
- One name for multiple parties
- There are certain cases where proxies makes sense.

Use of Aliases

- Upper common layer that bridges the two networks.
- Egress layer (side you're leaving)
- Ingress layer (side you're entering)
- Emulation
- Have one machine and have 12 virtual machines on that one machine.
- Within that machine, you can decide based on the name for what you want to do.
- They are all going to the same place and this is useful for other purposes.

Use of proxies

- Load distribution

- Multiple destinations that map to the same name
- Does NOT have one immense supercomputer that has everything that gets sent to www.google.com
- IP addresses are distributed to handle the load
- Localization
- Send informations from “nearby” IP addresses
- Get quality of service by moving it across shorter distances.
- How do I want to translate a DNS to an IP address based on location.
- Fault tolerance
- If the first machine goes down, you can translate it automatically as long as everything else has been done correctly.

Implementation issues
- Name management
- How if we care about consistency will we deal with locking?
- A lot of naming systems are distributed.
- Information in distributed databases and solutions have tended to arrive from that field.
- Use of hierarchical things and typically, we use structure in names
- Makes it a lot easier to build a distributed name transition.

Terminology
- Resolution
- Translates a name in one context into a name in a different context

Resolution mapping
- Name(type1) -> name(type2)
- Aliases
- Types can be the same
- Address resolution
- Address will specify in some fashion on how we get from point A to point B

Resolution mechanisms
- Fixed
- Very permanent
- Local
- Purely local and no one else uses it
- Central
- One authority
- Distributed
- Clearly more complicated but scales a whole lot better.

Fixed
- Express IPv4
- A lot of broadcast over the current link
- This is at the IP layer

- To do an Ethernet broadcast, you have to use the Ethernet fixed address (all F’s)
- Everyone who uses that protocol uses that address
- No further resolution is required as it is built into the protocol.
- Send it to yourself if there is a particular address.
- Send it right back to me and send it in to incoming messages.
- Similar but different for IP version 6

Fixed pros and cons

- Pros
- Always works
- Learn it once, it works everywhere
- Cons
- Limited to a very small set of names
- Inflexible and not scalable
- Hard to add new operations (add/delete/change)

Local

- Names you can translate into the context of your own machine
- Go to the Hosts.txt
- DNS names that match with IP addresses
- Express DNS names with the Hosts.txt file
- Translation I am taking that tells you what you want.
- Services.txt
- Part of the TCP name
- HTTP -> 80
- SMTP -> 25
- Purely local ports
- Your machine can be changed and have different results from other people

Local pros and cons

- Pros
- Easy to implement
- Easy to make changes
- Very cheap to resolve since it is file-based
- Cons
- Hard to manage because people don’t know how to do this properly.
- Hard to scale since you have to maintain this.
- You cannot do aliasing or proxy if you focus that much on using local.
- Inefficient to do certain operations (add/delete/rename)

Central

- A single server at a known address
- Works somewhat better than locally
- Whenever you want to translate, tell the Central system.

- The original Napster which copied music did this with a single Central server.
- A little like broadcasting queries

Central pros and cons

- Pros
- Easy to maintain consistency and deploy
- Can be pretty efficient using a real database
- Performs reasonably well even for a fairly powerful server
- Napster supported a large # of users using this technology.
- Cons
- Doesn't scale eventually.

Distributed

- Spread out ability to change from form one to another.
- In-band
- You have an ability to go to a location and say "Translate this name for me and send it off to a service"
- RPC, TCPMUX use
- Name I am requesting, forward it to the right place and use the file.
- In-band pros and cons
- Pros
- Easy to deploy independently
- Can be very lightweight
- Cons
- A lot of replication of effort for scalability
- Out-of-band
- Step back and talk to the web server now that you know the location.
- Structure our namespace so it is in a hierarchy
- Divide up tasks more effectively.
- Gives hints on how to find a person who can do the proper translation for you.
- Search
- I don't know who has my translation but I can find out based on clues.
- Ask people to get a more clear picture of who to ask.
- Broadcast
- * Flooding information, send a request to everybody in the hopes one person knows how to translate it.

Hierarchical

- DNS
- Used to translate text to their making IP address
- Has some order in the name with name components divided by dots
- Right to left to organization
- reiher.cs.ucla.edu
- Starts at .edu -> .ucla. -> .cs -> reiher

DNS Hierarchy

- implicit ROOT way at the top
- Has suffixes below
- arpa
- com
- edu
- etc.
- There can be a plethora of many other names
- Things under net are people who support networking ,etc.

Roots

- Need to start somewhere to get to any other node in the tree
- You need some knowledge to even begin starting translation.
- We need to find the Root to access where everything is.
- Translate any DNS name in the world, use one root to translate anything else.
- This means that the root is important, like one of those central authorities.
- Most clients don't start at the root.
- We have many different copies because each of those copies can have multiple copies of itself.
- Each one has multiple machines that provide the translation.
- This is plenty and we have never had problems getting things from the DNS system.
- Handle trillions of translations per day.
- Usually, we never go to the root and everything we have is pretty static.
- If we know a translation of something, we can usually start with that translation and you will usually be right.
- Don't worry about the root at all, there will be a server that can do DNS lookup for you.
- The DNS resolver in the ucla.cs environment can handle this for you.
- This connects up to the hierarchy and goes into this and figures it out for you.
- This means it may start at the root

Iterative vs. recursive

- Iterative
- Whoever is doing the lookup can translate one component and translates it for you.
- Gets the next DNS server and feeds back an IP address for you.
- How do I translate this one? It tells me
- Keep doing that until the whole name is translated
- Recursive
- Send everything down to a different server and eventually it gets down to the base case

- Eventually, someone has translated the whole name and it returns it back to you.

Iterative DNS

- Get www.cs.ucla.edu
- I don't know how to get anywhere except the root
- Go to the root and look for .edu
- The root looks at one component of that name and shows you how to get to that.
- Send back to the client and look for ucla.edu
- Continue the process, look for cs.ucla.edu and keep going until you reach the value you want.
- This is the worst case of the iterative approach

Assume you know where edu is

- You can skip steps based on your knowledge base.
- If you know where everything is, you can jump to the cached translation

If nothing is cached, it takes N round trips to resolve an N component DNS name

Recursive DNS

- Assume we know nothing except where the ROOT is
- Strip off the .edu and keep breaking that down.
- Send the rest of it to ucla.edu
- Keep sending the information down via a recursive call and it will eventually go off to the web server

If nothing is cached, one round trip to resolve a four component DNS name + more "behind the scenes"

Iterative vs. recursive

- Iterative
- Pros
- Client retains control
- We could cache things locally
- Cons
- Sends more packets
- Longer delays
- Recursive
- Pros
- Faster
- More efficient
- Uses aggregate caching
- If we need to know an address, we can take the entire translation and give it back to the client as well as the layers above the path
- If it is cached once, you essentially saved the whole address so you can grab from your cache
- Cons
- Client hands over control and it will be more generic.

Other DNS uses

- Load distribution
- One name can be linked to many addresses
- You can have 500 different machines with 500 different memory addresses
- Translate this to the many different addresses
- Cycle through and go through all 500
- Alternatively, you could have gotten the entire list
- Not load BALANCING; it just gives different translations but does NOT ensure BALANCE
- Localization
- Hulu does this by guessing where you are locally so it is more convenient to access the IP address. If you are in So Cal, you are going to use the Santa Monica address rather than the New York address

DNS pros and cons
Pros

- Similar names end up in one place (localization)

Cons

- Vulnerability if the root is NOT there
- The root has always been there but then it gets scattered across the world.
- If there is no root, it may not be able to translate many addresses.
- Caching and replication consistency issues will come up and this could end up in one place.
- Google does NOT necessarily want everything localized

Search Approaches

- Unstructured
- Ask a neighbor who knows
- Eventually someone can find it
- Structured
- Use information to make an intuitive search approach.

ARP

- Ethernet responds using the ARP protocol
- Works only on Ethernet or shared channels
- The problem with this is that people can lie.
- This is called ARP poisoning!

Broadcast pros and cons

- Pros
- Easy to configure
- Cons
- Bad for scaling

Distributed Hash Table (DHT)

- Structured search
- Using a hierarchy (like the DNS)
- The node is either going to have the translation you want or it will have the translation.
- One or two step process. The person who knows this is another 3rd party node.

DHT variants

- Connect up translation nodes in various geometrical structures
- Ring, hypercube, etc.
- Widely used
- Bittorent tracker component
- Freenet anonymous network

DHT pros and cons

- Pros
- Fault tolerant
- Scalable
- Load distributing
- Spread out
- Cons
- Way more complex
- Hash avoid locality of similar names (advantage or disadvantage)

Caching

- Relies heavily on name translation
- Expensive and we don’t want to do it more than once.
- Keep a copy at the endpoint or somewhere in between

Cache timeouts and performance

- Timeout issues
- Don’t keep cache entries forever
- What do we do if the copy expires?
- Wait for someone to request that translation.

Cache invalidation

- Timeout since it was first retrieved
- Timeout since last request

Summary

- Different names for different layers
- Name resolution
- We need to start somewhere and go in both directions
- The only reason we can do this is through the high use of caching

W 6 T Lec 2-9-16
Layers, Naming, and Sockets
Outline
- What's a party?

Recall: definitions
- Communication
- Exchanging information between a fixed set of directly connected parties
- Networking
- Parties are not directly connected and don't share a single protocol
- There will be complications in this case
- Protocol
- Set of rules that everyone knows in advance
- Entities that must know these rules are the ones using the protocol to communicate.
- The communicating ones are the only ones who need to know the protocol.

Names, layers, and translations
- Representation of a name for each of the layers in which he is making use of that layer.
- Look at Lecture 10, Page 4
- Jim is going to have the same thing: a computer that is represented at different layers
- Bill has a name at the blue layer, purple layer, and yellow layer
- Jim also has a set of different names for each layer
- Both of them have different names at the blue layer and the orange layer
- Go back through that stack, changing our names as we go

Parties
- Parties wish to exchange information with each other.
- Where is that party?
- Somewhere in the world, there is a computer that represents that party and this computer has a physical location
- Logically, it is located in a layer somewhere on a stack

A network layer
- Nodes
- Create information or absorb information
- Links
- Can have multiple parties set up in them

A closer look
- What's inside a node?
- What communicates to the outside?

One View
- Computer communicating to multiple networks
- One node to multiple channels

Another View
- Proc A is communicating to Google
- Proc B is communicating to Amazon
- Proc C is communicating to Facebook
- Three processes have channels to one party each
- This has naming implications because they are communicating to the same computer but at different layers
- We have to deliver it to the computer first, but then we have to know which Process to deliver it to.
- A process will have to know how to render itself in the context it asked for i.e. if Process B asks for Amazon, it has to render content from Amazon
- Processes have to figure out this message goes to particular portions of what we are doing

A closer look v2:
- What's inside a node when there are multiple channels on a single network

What Does That Mean?
- Names within a single layer representing a channel

A closer look v3:
- What if a single node has several channels on different networks?
- Several types of external names.
- We really have a wireless network that has some kind of channel to a wireless router.
- One is on the Internet, the other is on an 802.11 wireless network
- One with an Internet name and the other has a wireless MAC address
- Use a wireless channel to be on top of the other (they are related to each other)
- We could have a wireless link as well as an Ethernet connection that are probably not related to each other.

Inside vs. outside names
- These names all belong to the same node
- Three different names we will use there
- Two way communication so he has a name for channels that go out to those sites

"Outside" names
- Name that is a source or destination of something on the network layer

- N:1 if you are one of the N, you want the 1 party to be able to determine that you sent it.
- Likewise, for 1:N, the 1 needs to know which member of N he is sending to.
- Outside names like you identify these things!

Nodes vs interface

- Nodes
- Talking primarily about the endpoints of communication
- The guy who is creating the messages and who is going to consume finally those messages
- Source/sink of all network channels at a single place
- Interface
- The way you attach one of those nodes to the network.
- Not purely the hardware interface
- As some layer in the network stack, this is the endpoint that leads to your network channel

Node names

- They have to be unique across a particular layer
- We are talking about a layered architecture
- Each has a different name and a different layer of the cloud
- There must be no other node that can have that name, but **nodes can have multiple unique names**!
- Within that layer, using any of those names will get you to that node.
- Node names are equivalent, you can use any of the aliases

Interface names

- Way of distinguishing message coming from Amazon vs message coming from Facebook, for example
- As long as you make sure that the interface name is unique, you don't care about collisions between interface names
- A node can have multiple interfaces and multiple levels
- You could also have one piece of hardware doing the networking and you would have multiple interfaces across each network.
- Endpoint names are equivalent within an interface

Node name uniqueness

- Lecture 10, Page 21
- A, B at the bottom left -> A == B
- A, C at the top -> A == C
- C, D, E on the bottom right -> C == D== E
- Multiple interfaces here
- 2 == 8
- 2 != 4
- 4 != 4 because they are two different types of interface

- Value doesn't matter, they are not in the same context!

Strong vs. weak endpoint models
- Strong
- Names are very specific
- Name a particular interface
- We have three interfaces (C, D, E)
- If a message is for C, it will go through the C interface
- The thing directed to C will make sure that things happen in the right way at this node.
- Like having a house with multiple doors and you can direct which door people enter
- Downtown Abbey
- Guests to the front door
- Tradesmen around the back
- Preferential treatment in this case!
- Weak
- Name belongs to the node, NOT the interface
- We don't care a whole lot of the interface it comes in.
- If C, D, and E are node names and NOT interface names, it is okay for it to come to either interface.
- They all arrive at the same place.
- It doesn't matter which door you knock on and you just need to get to your house.
- All roads lead to Rome
- Interconnected
- Bad if there is too much bad traffic (barbarian invasions)
- It is a matter of what you hope to achieve
- You can have multiple interfaces
- If you are talking to Google, Amazon, Facebook, you need a different virtual interface for each
- You need to slap a name on it that is associated with that interface

Kinds of outside names
- Ethernet
- We are sitting on the Ethernet and use that name to get to me
- Probably an interface name so fairly strong
- If you want Ethernet address, it will go to a particular card
- IP
- Typically a weak name
- Somehow, this connects to several networks in several ways
- You don't care how it gets there (path independent)
- You can have multiple IP addresses for the same node
- Everything coming to IP address A has one way of receiving it, IP address B has a different way
- TCP, UDP

- Tends to be something that is a little closer to a strong node
- Includes both IP address and port #
- Specifies things like a message that is supposed to go to an email program or secure shell program
- You need to specify a port that is related to a particular connection that comes out like an interface

A look inside the endpoint….

- You have a bunch of data with different finite state machines and Turing machines
- All of these are connected out to this endpoint.

“Inside” names

- Names within a party
- They are going to be either source or sink names depending on your view port
- What do we need to refer to?
- The message is going to be built out of data
- Who uses that data?
- Which process is associated with that data

Object related names

- File names (static data)
- This particular virtual device
- Socket descriptors

Process related names

- Process
- Arbitrary integer definition
- Thread
- Operating System threads vs user threads
- OS threads will have thread names
- user threads will NOT have thread names, but the process gives it a name to identify it
- Other related names
- You will probably see an integer to represent the name of an integer
- These numbers are essentially a name related to a particular process

OS Review

- Process
- Smallest independent running program with its own memory space
- Protected from the other processes
- Represents possible set of addresses that the hardware can express
- 2^64 possible addresses you can express
- Resources are allocated to processes

- This process gets to use this resource and this includes things like code, pieces of RAM, threads, etc.
- OS will know of several OS-level threads
- Threads can be supported by the OS or individual user applications can support multiple threads within the processor
- Threads
- Smallest independently-scheduable running progress
- Associated with a particular process
- Conceptually, every process has AT LEAST one thread
- Multiple cores can be executed instructions simultaneously
- Single core can only run instruction at a time
- Can have a pseudo-simultaneous effect however, and it is usually indistinguishable from simultaneous execution in most cases.

Why we prefer process names to…
- Thread names
- Not quite as unique
- Every process can choose its own thread names
- There could be multiple machines with the same name.
- This implies the OS cannot figure out those thread names under those circumstances
- We generally don't know if we are talking about OS level threads, user level threads
- We tend to address with the expectation of the process
- File, I/O, etc. names
- Typically do not use network names as files

Properties of inside names
- Syntax
- What is the namespace format
- If you have an integer name, that is one form of syntax
- If you have process names that are character strings (usually not used), this is another form of syntax
- Unique to the node
- Inside names that only go to that node
- Doesn't matter that we don't understand the syntax
- Value
- Unique within the node
- Your process 73 could be sending messages to my process 73
- Clearly this is NOT a very useful name for network identifiers
- We need something more unique!

Job of an OS
- What does it do at a high-level?
- Coordinate resource sharing
- Make sure they don't step on each other's feet

- People do NOT interfere with each other
- If we have one 802.11 card on our device and we communicate to 50 different devices, then we have to share
- This is the OS's job to mediate the sharing of that a device
- Provides abstractions
- People don't worry about which register to load, all the gory inner details
- Provide a higher-level understandings for processes to use to make use of the system
- Gives an illusion of using the compute all by itself
- Someone has to build these abstractions

How do OSes abstract layer endpoints?

- If we have to use sockets in this case
- You can do communicating across networks without sockets
- If we use ordinary user level communications i.e. servers, tablets, etc.
- For most programming, we will use sockets to communicate to the network.
- Sockets
- 1970
- Same research that created the ARPANET which led to the Internet
- Networking infrastructure lead back to the research
- Communication endpoint from the point of view of the "user"
- Programmatic view of an endpoint
- Generally a two-way channel
- You can do a lot as a bidirectional channel
- It is an inside name that is going to connect up the process that leads to the outside world

What does this mean?

- Network interface card and putting information out on the proper way
- Eventually leads out to Google
- Create messages that go out to Google
- Consume messages that go out to Google
- Connect the process up to the channel so that they can name the channel and send information out to Google
- Process A is going to connect up to Channel X here
- Socket is the name that Process A which indicates that it connects out to Google

Room for confusion…

- Unix-style systems also use sockets for all kinds of things
- Includes IPC within a machine
- Different address spaces
- These two processes are purely within a single computer and one mechanism that they use is sockets!
- Same kind of sockets we will talking about

- Used for non-network communications
- Sockets are used for network communication

Inside and outside

- How do we link: inside names and outside names
- Throw stuff outside our machine and it could be routers or switches

Linking the two

- Bind
- Another unfortunate overuse of a term
- Used at a much higher level to deal with the DNS system
- Translates names from DNS to IP address
- We are referring to binding an inside name to an outside name
- We will do the right thing to associate data from inside name to outside name
- Widely used for Operating Systems for Windows machines and many other types of commodity machines
- Links an internal name to an external communication layer to get it out of your machine and hopefully to the right place

Two sides to a socket (Client-server type operation)

- Server side
- Client side
- Channel in between
- Lecture 10, Page 42
- Operating Systems have to perform some operation on each side

A socket (either side)

- One socket on the client side machine, the other on the server side machine
- On the client side machine, the socket we create will be bound to the networking channel sent to the server side
- This gets down to channel coming down to the client.
- Two-way communication in both directions
- We can have one of these internal names that we can hook up to the external name.
- We need a socket name and runs a system call known as socket
- Connects one of the processes to an outside name
- Example call is the code segment on Lecture 10, Page 43
- We have parameters specifying what we want to do with this particular socket
- The system call will return either -1 for failure or a pointer
- If we don’t do anything else, we can’t do much with the socket

Common kinds of sockets

- Quite general

- Many different types of sockets
- Datagram
- Not associated with any piece of information you send before or after
- Stands alone
- Does not care about what comes before or after
- Not associated with anything you have done before.
- You have a whole lot of information and what you say will depend on what each person says.
- We don't want any unassociated message and we can instead set of streams and sockets.
- Streams allow us to have ongoing two-way communication
- Start using the shared context to exchange machines
- We have some kind of virtual machine (state machine on each side)
- Get them into the proper state and have them exchange information on different sides
- We typically use sockets in theses two ways
- Datagram
- Stream

Miscellaneous
- Google
- Process name and send it over to a unique one.
- Needs a unique identifier

Bind
- Link to a socket on an address on the server end
- We have a process here and there
- The hooking up of the processes is internal to the channel and that endpoint.
- We have to connect to the channel on the sending side
- Send it to the web server on lever
- It is a process that connects up to the channel and connect it to that channel
- Socket on each side
- Socket on the browser side is NOT the same as the socket on the server side
- They do independent work but they have to agree that the channel is being used
- It is usually done by convention, where the web server we want to talk to is part of the naming we need
- Lever is a Linux machine that does many different things
- How do we make sure the message that comes to lever arrives at a different web server?
- It needs to go to lever's web server rather than a secure shell login
- We need conventions and agreements ahead of time where we have a name that we attach to the IP address.

- We make use of those choices as appropriate
- Easy way to do it is to have an agreement of how it should be.
- For other purposes, we need some other way of doing it and we need a brand new application that everyone loves.
- How are we going to have people understand how to find it on your machine.
- We need to connect our process, which has a socket connected to nothing to our particular name.
- We are in bind that has a socket that is a generic thing and we haven't specified where it connects to.
- We need to take that socket and connect it to some network channel.
- Then when we put information from the channel, it goes where we connected it to.
- The channel is defined at the bind statement (Lecture 10, Page 45)
- This is where we define the channel in the server parameter
- Let's say the address we want is the web server on lever. We have to somehow or another contain the address for the website on lever.
- What port # is it?
- All web server port #'s are at 80
- Specify an outside address that makes sense and it will be done with sockfd
- We need a pointer to a struct sockaddr* to refer to the server we are using
- We can have all kinds of sock addresses, so we need sockaddr_len
- This should be able to connect our inside name to the address of the channel we would like to make use of.
- If we want a web request to lever, having performed the bind, it has to go to lever and if everything goes well, then our message gets to lever's web server.
- This is what we hope to achieve here.
- We can send other types of stuff i.e. UDP
- We need a somewhat different type of thing here
- UDP is a transport protocol that lives above IP but does NOT try to maintain a connection
- It does NOT have any memory but still, we want to send a whole lot of messages to one place.
- They all have to go to one place and we don't want to specify the address every time.
- Send them all to the same address
- UDP wants to limit messages you receive
- Maybe you only want messages to come from one particular party

Server first steps

- Socket
- What happens when the server wants to create a socket?
- On the server-side, we want to provide service to people.
- We need a channel on which to have it sent to me.

- The socket needs to be attached to the channel
- Socket will have a channel placeholder which is local to the process
- Use it for whatever purpose I want
- Connect it to a channel i.e. Channel X
- Bind
- When you start on Process A talking to that socket
- On this side, Proc A is aware that this means you want to use Channel X
- Look at info coming in from Channel X, and send info outgoing on Channel X

Stateless: receiving messages

- Sockets are stateless
- Don't try to maintain state for each message
- Deal with them separately
- If you Recvfrom, you can find out who sent it to you
- Lots of people could be sending from the socket and you have to figure out who sent it to you
- Lecture 10, Page 47
- Here is a structure that describes an address and send it to whoever sent it to you on this socket
- Here is a server and you have a socket connected up to a process
- Accept the message and tell the OS who sent the message
- You can have a whole lot of people sending you messages from the socket
- Sendto
- Send a message
- They can go to a bunch of different places
- If you send a message, you have to specify who it goes to
- Otherwise, the OS wouldn't know
- You have to specify the OS to whom it goes to
- Lecture 10, Page 48
- Specify the address of a client in the case that we haven't found a bind
- We need to know the outside address
- Use the same socket to talk to multiple endpoints as long as it isn't bound to a single endpoint.
- Most commonly, we want to bind our socket to a single endpoint
- Here is one end of a channel and if it came in from one channel, we know where it is going to.
- With TCP, you can be reasonably sure that it gets received
- With UDP, you cannot confirm this

Server side - connections

- Web server process
- A lot of people want to talk to our web server and we want to provide services to all of them
- Give people access to the webpages that we provided them.

- We don’t want to specify individually that all these channels belong to someone ahead of time
- We cannot determine who will use a channel beforehand!
- For the first time, you go to lever and you want to download a set of slides
- How does the server of lever know it is you who is requesting the download?
- It could accept a request from anyone (it is an open web server)

Listen

- Server will set up a socket and we need to hear from people who want to communicate to me.
- Bind process to socket but it won’t necessarily be bound to a particular address
- When a message comes in, I am not sure who it is from
- Say that you want to listen in on this connection and we can see what comes in.
- Mark the socket in question and something the server does
- Listen to anyone and if I like what they say, I will set up a separate connection for them (promiscuous server)
- How many can you support simultaneously?
- You have to specify the second parameter MAX_CLIENTS
- You are expecting a certain amount of packets to come in.

Accept

- Set up a socket and you are listening on the socket waiting for someone to give you web content.
- This is in a promiscuous mode so you have to set up something in this regard.
- This is something the server does, NOT the client!
- If you have a socket already, you want a brand new socket to the client.
- If anyone wants something, you then accept the request if you like it.
- Sets up a brand new channel and we usually will have a problem with TCP that will allow us to communicate.
- If you have the old one, you can set up a brand new one because you want a different socket associated with a different channel
- Assuming we are talking about a different level, we want an IP address + port
- Type socked will contain that information
- Go into his message, put out a bunch of stuff, and put it into that instruction
- External outside address
- Is there a way to exclude certain clients?
- They will listen to all clients, but they don’t have to specify all the client.
- In order to filter, you will examine what you heard, and pick what you like.

- MAX_CLIENTS is how many calls you want to have active at a certain time, so you can throw it out.
- Denial of Service attack if people keep spamming the crap out of you and you keep saying NO. You cannot do anything else in the meantime.
- If you take whatever comes in there, we have not specified what the address is, and they could be from anywhere.
- We would then have to multiplex from within our own program because we would have to set up our own data structure in this case.
- Accepting is like saying we have a brand new socket call and we are doing a bind on that socket.
- This is for a special case that is very common and we are going through an example to do an ordinary operation.

Client side - connections

- Socket
- Something to connect to and we want to connect up to a channel
- He makes the socket call and creates a brand new socket using bind
- This is how we set up communication channels for ongoing communications

Connect

- Do what we expect to want happen, and he is listening on this channel and he is listening to everyone.
- Cannot communicate on a long-term basis and it is how he wants to talk to me and this is what connect does on the client side.
- When you talk to this address over here, we expect to listen on this connection and he is going to set up a brand new socket associated with a brand new channel.
- We can just simply connect and we will have this socket to eventually do web browsing on lever.
- Bind that socket to lever's IP address on port 80
- Not good for long term
- We are expecting to set up a brand new socket and we now want to bind it to that one
- We can now use this socket to do all these communications
- Why would we not create a socket and bind it to port 80?
- We aren't going to use port 80 because he is using it to listen on. When he hears that, he will perform accepts on a different communication channel.

When the server listens and sees that someone wants to connect, they create a separate one and bind a new socket to that address.

- You are behind the scenes doing this every time you connect to a new server

sockaddr and names

- sockaddr is a data structure that has been filled ahead of time

- If we have received a message, we are going to pull that information out of what we received
- Usually an IP address and a port
- Sockets are really general and could be many other things

Send

- Write data on the connected socket
- Send information down that channel and we provide the length
- Operating system says you would like to use that socket, so we will see what it is connected to and we will put information out to the channel
- Comes in on the receiving end and the receiver can grab the data

Recv

- Reads data from a connected socket
- Put that information from there

Putting it all together

- Build an entire set of calls that we can use to create a client-server interaction
- **Server will create a local socket and bind to it**
- **Client will create a local socket and connect it to the server's external name**

For example, (Lecture 10, Page 59)

- The server creates a socket and binds to that socket
- It then listens and eventually, it will send a request
- Client then has to create his own socket and it will get to the server
- Client connects and his expectation is that the server is listening on the channel
- If we are talking about doing the web server, it should connect out.
- While we are doing connect, we will make that accept call that we mentioned before.
- We can then see that the middle arrow between client and server and this establishes the TCP three-way handshake
- Passes information about what port you will use and you can properly connect to the server
- We have a socket on the client side and it could be hooked up to the server
- If we put something out to the server, we then know what to do with it.
- Client does NOT have to do a bind; the connect will implicitly do that for you.
- We are just using port 80 to talk to you and this is how we will talk to all to you.
- You are going to specify the particular address you will talk to
- You can still get the message to the right place but there has to be a corresponding address

- We can bind to one address permanently and from then on, things have to branch out from there.
- If connect works properly, you will end up with a brand new socket that connects up to a channel that is attached to the web server and this is your channel
- accept() is a blocking call, so you don't know when the client will make the connect call
- I make the accept call, and if there is a connect waiting, return immediately
- Otherwise, the process blocks until somebody does the connect
- Set up the socket in such a way to set up their own separate sub sockets
- This is the way it has been designed and we can throw away all this stuff
- There is one piece of code that will receive messages from those 5000 clients
- Looks at the message and tries to find which one it belongs to.
- Setup a separate thread for every client.
- Whenever data comes in from that client
- We know it goes to the right thread and it performs work for that client
- The OS will do multiplexing in terms of the sockets
- What the process does is a matter of programming
- You can have a multi-threaded program

No bind, no problem

- If we omit it, we can figure out where the message should go and there should be sufficient information
- The operating system can figure out what to put into that particular position

Automatic source addressing

- Destination address has to be in the system call you make
- It is already in this socket hooked up to a particular channel
- In the send to call, you will have the destination address
- Full address family and we have to see what layer we are working in
- This is going to be implicit in the type of name you use.
- Give it a name that is a network layer name, you will deal with that accordingly.
- Put it into a particular message that goes out
- Source address is an outgoing interface

What about ports?

- Port distinguishes TCP or UDP layer
- Look it up from a published list to understand the one you send it to.

What's your port number?

- Messages
- You just sort of have to know who you want to send to.

- You know that based on previous history of the port or you look at connection history
- Connections

Port numbers
- Ports are local to a communicating pair and won't mean anything except to the endpoints
- On those endpoints, it won't know anything else about that
- <u>Sometimes</u> ports can have common meaning
- Sometimes, they work with registering ports which have specific meaning

Two meanings of ports
- During first contact - expected process
- Web server (80)
- Secure login (443)
- email server (110)
- After that, you will use a unique name and you no longer care about the port #

Port meaning
- By common convention (assumption)
- Groups:
- System ports: by original convention
- User ports are assigned
- Dynamic ports are unassigned
- By other coordination
- Parties will be very limited and they can go ahead of time and everyone is communicating to everyone else
- You know these are at the ports we should be using and there is no negotiation.

W 6 R Lec 2-11-16
Issues
- Create a placeholder and connect it to a channel using bind
- What if you decide not to use bind?
- Things don't necessarily have names
- What are the boundaries in sockets and the relations we get there

No bind, no problem
- You can still use socket to send stateless (connectionless) messages
- Somebody will have to figure out what do I do with the message
- Something the OS provides for you
- Something dealing with the message you send to the socket

Automatic source addressing
- There are certain things you have to do

- Call to send to via the destination address
- Unnecessary when you do a bind
- If you don't do a bind, the socket could be used to be send a message to any arbitrary address
- One of the things to consider is the "address family"
- What protocol should you use to transmit the message
- If you had done the bind, you need to bind it to a communication channel
- A type of address that you use will implicitly label what part of the stack you will be dealing with.
- If you use that kind of address, that is what the OS will use
- Pick an outgoing interface to use
- If you have multiple choices, you will have some procedure to select which one you will use at any given moment.
- If you don't tell me, you can send over the Ethernet.
- Whatever, it is there is an address that gets put to a message that gets sent this way.
- Some of it will be deduced or chosen by the Operating System.

What about ports?
- Put the port into a destination address in this way.
- Sending a UDP message this way, include the port you want to go to.
- You need to know which port to send to!
- Worry about how the IP address gets sent later
- You have to know which port you want to send to.
- It could be that it is from a published list and this port on all machines will be used for this particular purpose.
- Another reason you could provide a port more arbitrarily is if someone sends you a message from that particular port.
- He will send it back on a different port.

What's your port #?
- When you are using the socket, you need to somehow or another specify which port you would like to have for outgoing messages.
- Depends on if you are doing in individual messages whether you are binding or NOT binding.
- If on the other hand, you are sending individual messages, or you are replying to the message, use the one you wrote.

Port #'s
- Indicates the socket to use on each machine
- Some ports have common meanings!
- Used to set up client-server communication
- Already registered through an international organization

Two meanings of ports

- During the 1st contact - if you don't know which one you want to use, who should I talk to in order to set up the socket?
- Once you have decided the port you are going to use, keep using that as an endpoint identifier.
- No longer has any special meaning, merely an identifier for socket destination's end.
- User ports are specifically for users to do particular things
- Dynamic ports that are able to be used by anybody.
- When it wants to create a new port from a new socket, it can choose from one of those sets and we can have a negotiation between client and server.
- Well-defined ports with what the # is, go from a web server today and a web server tomorrow
- Don't mean much of anything except the ones that are preassigned for any purpose

Having no name

- Bind with no name?
- You can't really do that because bind is a system call that has to have parameters
- In place for some of these parameters, you can put a "0" there
- This says that you don't care to the OS, any address will suffice!
- OS cannot deal with sending addresses without names out, so it will pick out an arbitrary name.
- We don't have to figure it out ahead of time.

Q. If you don't specify a name when you bind, do you have to specify a destination?
A. It arbitrarily chooses a message for you as well. You only do this if you don't give a damn about where you are going. Try to use a protocol where you aren't concerned who provides its answers.

Q. Is it like broadcasting or do you have choice who you send to?
A. OS will choose an arbitrary address rather than pure broadcasting, but when you connect up to the socket, it will tell you which address you have for the socket. Assuming you are working with a channel that has broadcast capabilities. You cannot broadcast to the Internet.

Socket types and boundaries

- In many cases, when you are sending out network communications at a high level, we aren't thinking about packets and we are saying we have a bunch of information here and we want to send it back to there.
- The pieces need to be a size your network can actually carry!
- With TCP, for reasons of my own, since we know characteristics of the path, we have decided that we need to split up our 1000 byte message into two 500 byte messages and put it back together on the other end
- There will be some degree of effort to get information from the other side
- UDP won't be guaranteed in order
- TCP will be reliable and in order

Summary

- Names are used for many purposes
- Networking
- Internal processes
- We need to bind the names we need to understand
- Take inside names and purely local to our own system and connect them up to outside names in order to deliver our messages coming back
- Sockets are the general paradigm we will use for this purpose
- Work for a different OS and if we get a smaller OS, it may be much more raw and we would have to do much more with those purposes.
- Expectations associated with certain names and this is the broadcast address
- This is the address that relates to the application we are using.

Automatic Naming

- There is no pre-assigned naming generally
- We will usually use automatic naming
- If we don't have a name, we cannot have our messages sent to us properly
- Use automatic mechanisms in many cases

What is automatic naming?

- We don't want some human system network administrator to fiddle around with the machine and we don't want some human being to send information to us.
- We want to automatically send an appropriate name to some network entity
- If we don't have a name, we need to get a name
- We need to do it relatively quickly!
- Frequently, we expect these things to change over the course of time.
- It probably isn't the same name as the first one.

Why automatic?

- We don't have anything to begin with so we need a starting name
- We could have done human prearrangements but those are harder to configure
- If things change, we have to automatically switch to a new name

Because it must be!

- If you have N:1 communication, then you can send something to that one party
- He just won't know which one since he doesn't have a name to distinguish you
- 1:N send it to everybody and some of those people won't yet have a name

- A few things we can do if we don’t have a name
- You can actually assign a name and use these operations without knowing a name to create a name and assign it to a party that needs one

Ease of configuration

- We want to be able to turn on a device and use it.
- Plug-and-play
- Have a much more automatic way of doing naming.
- You may have many different devices and you want to configure them to the default
- Maybe with a few little changes, you will customize
- Other than that, it is much like the original configuration
- If you had to do all the configuration yourself, it would be painful

Adapting to changes

- Mobile computing
- Change the network link as you move from time to time
- Switch from one wireless point to another wireless point
- You get a different address space and whether you send it to another computer lab.

Mobility

- Requires changes because of a change of a physical location
- In particular, UCLA has a block of addresses you can use for your IP
- UCLA is not going to be all that excited about when use one of their addresses if you work at a company like Google (far away from Westwood!)

- Changing networks is essentially changing a name space
- Sometimes your smartphone can use 4G, other times it can use WiFi
- You don’t want too many user visible changes

Renaming

- Every so often in the telephone network, eventually it gets too big
- LA used to be 210 but now we also have 310
- Working on my lap top but we need something only available from a UCLA IP
- Connect to a VPN (Virtual Private Network), put more layers into your network stack with things you haven’t seen before
- You could use this to access things like private videos at UCLA

How can you get a name?

- You don’t want humans fiddling with names themselves
- Few users have any understanding of network names, and they are for the network stack code.
- Not meant to be handled by human beings in most cases
- What are your options for getting a name automatically?

Alternatives

- Design-in (preconfigure)
- Randomly pick a 32 bit address and hope for the best
- Get an appropriate name from someone else

The $1 solution

- Effectively what you do in a # of cases
- The $1 bill has a unique serial # and we will use either that serial # or a function of that serial #
- Put the $1 bill away so no one ever uses it again to get the #
- We would never do this in Real Life, but the concept is similar

Ethernet

- Two part solution
- IEEE assigns an identifier to anyone who is willing to pay for it
- Pay about $2,575 and use it to get your address
- Figure out the cost to send one of those addresses (about a millionth of a penny)
- Only buy these things if you are going to manufacture equipment
- Lecture 11, Page 14
- Start assigning numbers out of Organisationally Unique Identifier (OUI)
- Use another hundred thousand cards coming out of your block
- Make sure the address you slap out of the Ethernet card is unique
- When it connects to a foreign country like Zimbabwe, you can be confident no one else will have that Ethernet address
- This assumes you don’t have rogue manufacturers who are careless

Ethernet addresses

- Fixed devices are write-only by the manufacturer
- You will have the Unique Burned-In (BIA)
- Can also be writeable
- For multicast addresses
- In order to set multicast, this multicast address will be associated with this set of MAC addresses

POTS, non-SIM cellphones

- Essentially, you have a telephone # attached fairly permanently to the wire attached into your house.
- It is your phone #, and that is how it will get to the right place

Dude, where’s my card?

- Have a smart card of some kind associated with your communication device
- You will have information on the smart card that will have a portion of your address

- Comes down to the nano size

SIM-based cellphones
- Have two different names
- One is associated with the phone itself called IMEI
- 14 digits
- 6.228 bytes
- SIM card (Subscriber Identity Module)
- You can take a SIM card out and put a different SIM card in
- How they get that different #
- You can change the name on your phone by pulling out the SIM card
- Telephone numbers link ICCID to your phone #
- You can blacklist that and your SIM card will no longer be there

Getting the boot
- Power-on configuration
- Boot time will want the address
- Has a lot of variance on what it can be at runtime
- You could ask the user what is your address?
- Generally speaking, not a good idea
- Another choice is figuring it out for yourself

Rolling the dice…
- Randomly generate a #
- Big address space with 16-byte addresses
- Pick a random 16-byte address
- You have to think about the fact that it doesn't always work out fine.
- You don't get the same uniqueness out of the Ethernet hierarchical approach
- You can potentially have many collisions with the most popular first names
- There can be possibly be a few collisions if you pick in a sort of random fashion.

IPv4 link local
- We have link local addresses
- A portion of the IPv4 is set aside for this purpose
- Any address that begins with 169.254.x.x
- First two octets means you can use that local address on your link
- There you can be talked to on that Ethernet for your own address
- Someone across that company can use that address
- That set of addresses can be used for your Ethernet
- The address over here that serves as 169.254.x.x and the one elsewhere can mean two different things
- This isn't going to be good for sending things across the Internet

- We have to be sure that the address we are sending from this space belongs to someone else
- Broadcast and be guaranteed that it is a local address
- If we only need a link local name, pick one at random
- This may NOT guarantee you that you will get a unique name
- You do a broadcast type test in order to get the following local link name
- Keep performing the process until you get a unique name
- Under some circumstances, you may get one of these types of addresses
- Weird error messages like a link local address that cannot be routed
- Probably means you don't have Internet access to do this
- If you only care about your LAN, it is a perfectly good address and it works just fine

Pseudo-what?
- What we mean by random is having NO predictability
- You cannot be sure what you want to get
- You can have sequences of #'s that are NOT random.
- What this means is that if you turn to that notion of entropy, you will have sequences with maximum disorder

Random # generation
- Presumably you can pick one # out of it
- Cannot use a Turing Machine to generate a # in finite time

True random
- If you have access to truly random sequence #'s, use that
- Radioactive decay - shows true randomness, Brownian motion
- Choose fairly high assurance that it is random in nature

Pseudorandom
- Generally we cannot get truly random #'s
- We have deterministic algorithms that simulate this
- Not a random sequence of #'s
- Call a pseudorandom # generator using the rand() call
- How to avoid them generating the same sequence is to use a different "seed" (input) each time
- If everyone picks a different seed, you will get a different sequence of #'s
- You can get a reasonable approximation of a series of #'s
- If you have an Ethernet card lying around, the Ethernet card will have a series of addresses that are supposed to be unique.
- We want different sequences to generate an address and we want to ensure that each address won't be the same.
- Different seeds will likely end up with different addresses
- Observe processes in my computer that are very highly unpredictable
- Mouse movements are less reliably sources of unique seeds

- Sometimes a good idea NOT to have a genuine choice out of a random sequence
- We would want something more repeatable
- Simulating conditions of parameters setters
- We would like to run some simulation that has randomize characteristics but can still be effectively controlled
- We want the differences in parameters to determine any differences in results

“Spot” the difference

- Lecture 11, Page 28
- Pseudorandom does NOT have so even a spread

IPv6 link local

- Often it is a good idea that you try to be a little bit careful about your choices
- Verify that you don’t collide with random sources
- I am going to generate an address that is good for my link
- I have a MAC address and I will use this to generate my address
- Only works on local Ethernet
- This is where the test is going to work and this ensures you are NOT unique
- The protocol doesn’t support that test
- It is only on your own local link
- Some kind of local broadcast medium
- Some other kind of broadcast medium on the channel

iOS Ethernet anonymity

- Generate a random MAC each time you wake up from “sleep”
- Hope it doesn’t collide when you pick a random link
- There might be a collision, but fortunately, MAC addresses are quite large
- SSID requests
- Something that comes from your network behavior
- Persistent name that moves from place to place.
- Everything associated with this name will keep a record for that user.
- Keep changing your name as you get different names
- Hopefully the merchants will have a harder time tracking you.

Q. Why is it called iOS Ethernet anonymity?
A. iOS concept created by Apple. This is more of a mobile Operating System concept.

Asking DAD for help

- Generate a # that you hope is coming from a random sequence
- Hope that it is another approach that means ask something else
- Duplicate address detection
- Similar mechanism for other protocols

IPv4 duplicate detection

- Use ARP
- Probe and see what IP address it is
- The destination IP address can be broadcasted and this will be useful
- Only works if you have a MAC
- Try to get a name at the next layer up
- NOT going to work for everything in the world
- If it is somebody else's address, it will mess things up for the other guy

Crossing the streams?

- They don't really translate the addresses here

Implications for IPv4

- Exchange the ARP for the purpose of outnumbering the IP
- Cannot do this for devoted #'s and it only works for link local addresses
- It only works on local Ethernet
- It might perhaps be used on other links and we won't hear responses after we get it

IPv6 Neighbor Solicitation

- We don't have broadcast, so we use multicast instead
- Who do we send to?
- Who might know if that IP address is already in use?
- The guy using it
- Create a multicast group to include the guy using it.
- If he isn't using it, we can use it, and no one else in the network will be bothered

More parental support - IPv6

- I am available here and we can be available to get this address
- This helps create an address that you knollw is unique
- Another portion will be attached to the router
- It knows that it has a global unique address
- You are the only machine that has that MAC address
- By combining two things together, you get a unique address
- Address is good globally and won't have a link local address
- Throw on a portion of the address that is guaranteed to be unique
- This is another problem since it was configured by a human being
- This ensures an IP version 6 address where it is unique to itself.
- It can use that already existing piece of information to generate more unique addresses

IPv6 example

- Listen for router advertisements

- We will get one of these and take the router prefix and combine them together
- In IPv4, the address is good locally, but you cannot use it globally.
- In IPv6, we can use this mechanism to get a globally unique address

Asking someone else

- Generally speaking, we can ask other entities for help in getting a brand new name that relies less on getting anything unique on our own.
- Ask someone a question
- Ask for an address if we don't have a name of our own
- If we don't know who is out there, we can broadcast assuming we have broadcast capabilities
- If OTOH, you know through other mechanisms that someone is out there, you can ask directly
- You can put in an address 0 to say you have no address

What layer do you ask?

- IPv4
- Another layer (generally)
- IPv6
- Things not put into IPv4 have made their way into IPv6
- We always need to get an IPv4 address and we need to get one.
- There is no way built into the protocol and we need to use the solution discussed moments ago

IPv4

- We still primarily use IPv4
- How do we get an IPv4 address?
- Go to a different layer of software
- We could use the ARP solution
- Generally gives us only a link local address
- Don't go down the stack, go up the stack to a higher layer using UDP and the application layer on top of that
- We want to go up the stack built on IPv4
- We use DHCP generally (which is an application protocol that sits on top of UDP)
- We have no address,and we try to use IP
- Hence, we use address 0 (this is our source address)
- The goal is to get a better source address than 0.
- We can get one of N people and we can have 1:N communication
- Everyone who is address 0 will listen to this
- Get to one point where we have our very own address
- This means the entire Internet
- We want an address good for very little to an address that is good for anything

What can someone else tell you?

- We will ultimately use some broadcast capability
- What can we do and what answers can we expect?
- We could get some address with some other information for various purposes
- We could get something back where we can use an address for a while

Reverse ARP

- Let's have a broadcast request and provide an IP address
- We want to know what the ARP address is
- RARP
- I am at this MAC, I want an IP address

RARP limitations

- Can only provide an IP address
- To make effective use of standard Internet routing, we need more of that
- Our local link will connect up to the Internet
- We need to know how to get a DNS network and deal with translation.
- It is also the case we need a preconfigured server that runs on its own protocol
- Gets IP address but does NOT use IP address to do it.

BOOTP

- Bootstrap Protocol
- Needs a static, preconfigured table
- Provides a bunch of information like IP address, DNS list, program, etc.

DHCP (Dynamic Host Configuration Protocol)

- Replaced BOOTP
- We don't have an IP address, we are going to use IP to get an IP address
- Funky
- We can do limited things
- Send anonymously to somebody, typically with broadcast
- Can receive from somebody where we don't know who we are sending it to.
- We need to give it an IP address to the guy who has an IP address
- Improvement on some of the other protocols
- The reason I am doing this is because nodes are coming and going
- We don't want this IP address to be assigned to somebody who is going away and coming back
- After some period of time, we have to get another IP address and we notice our lease is about to expire.
- I am going to take away that IP address and later use it to somebody else

Steps in DHCP

- Used in manner like ARP

- I am working at the IP layer and what goes underneath will get translated down to the Ethernet
- Defined at the IP layer, NOT at the lower layer
- You can use any of the things located below that IP can be translated into.
- Use it for anything that is willing to speak IP and translate to IP above
- Client wants an address, and we have a server that is willing to play the game.
- We need at least one server with a b lunch of servers that you can talk to.
- Client wants an IPv4 address, and we can also get an IPv6 address this way too.
- Get a UDP discovery request
- If on Ethernet, you can use an Ethernet broadcast.
- There can be several servers who hears this broadcast and we can get a block of IP addresses we can give to you.
- Offering you this IP address and send this message from address 0.
- Broadcast this entire link
- One signal sent across the broadcast system
- NOT going to be any collision at the sending side
- All of them will get that message and will presumably send a response
- If we don't have collision avoidance, one will work faster than the other
- One of them will seize the broadcast medium and get the broadcast out first.
- If there are collisions, back off and we will have retransmissions
- Because it is broadcast, the guy over there hears the offers and we wait a certain amount of time to hear these offers.
- Embedded into these offers is an IP address that is supposed to be for him.
- We know who gave him the offer because the source address is in it.
- He is still sending with address - because he doesn't know who got the address
- Both heard a bunch of broadcasts offering you an address
- I am taking your offer for the same offer
- We cannot send back the same IP address of the offer you are sending
- Send it unicast because you know which server made the offer
- Server will get that and the client will say that the address is already in use and send it back to the client
- If OTOH, there are multiple people who asked for it, the guys who came in 2nd loses, and the offer is rescinded for them.

Q. How does the server send a unicast if the client is still called 0?
A. Server is broadcasting. The client assumes he can get it and he expects to hear back from them and he won't have it. If one of them gets address X, he can lay claim to it. There is going to be that message as well.

- Bind it using the underlying protocol

- Bind address X to that MAC address Y
- X translates to that MAC address Y
- The other guy wants that address X and will get denied by that server

Q. How do we forward back that the binding happens? How do we know to send it to you if you send an IP address 0?
A. They do a similar thing on Wi-Fi cards where there is a Wi-Fi address. To send an IP address, you have to be able to cross a broadcast (Ethernet) link. What the DHCP server does is once that particular client has won, he is going to take information and send it to the lower layer. Ethernet address X will belong to MAC address Z. Anything onward will translate when they go down through the stack of layers into address Z. Ethernet itself does NOT use IP address to route. They look at the Ethernet address.

- You do have an Ethernet address Z that went out to that discovery message
- Supposed to be a general protocol that works higher than the IP level
- Used to work on a broadcast medium
- Kind of a degenerative case
- The only guy at the other end of the link is the client

Q. When you send on the Ethernet layer and it switches, it will switch that name as it goes through. How do you communicate to something else?
A. This is typically NOT done across networks. Typically, you have DHCP host on the broadcast network and you won't have to route it. It will have toe emulate the broadcast and it will have two networks outward. This means you have to broadcast it in multiple directions, and there will be translations that happen at each point. Usually, we just set up broadcast DHCOP routers on the medium itself. We can also have at the end of the link information where it came from.

- There is generality where there is a high capacity, perhaps the load on one DHCP would be too high.
- We would like to have protocols that work generally to let multiple parties work together.

Why two phases?

- Offer can time out so client tries to respond to an offer before it times out.

B/Mcast vs unicast

- Unicast where possible
- Leased info and we want to renew the lease

DHCP relay

- A little like the proxy ARP but in both directions
- You can deal with response properly

What can DHCP configure?

- IP addresses are the primary purpose of DHCP
- If we need to get from my machine, what is the next thing that I am going through.
- Which machine should you be at the lower layer to get information for this layer of the Internet.
- Leasing addresses here for a fixed period of time.
- This information is for the response in DHCP
- We could always go to the DNS route if we know where it is ahead of time.
- We don't want to start at the DNS root if we have a lot of messages.
- Don't overload the root.
- Try to avoid going through the DNS hierarchy
- At the very least, for all the clients he is going through, he will cache the result.
- We can have other things that DHCP tells you about.
- It is useful to know what the timeout is and we are going to lose it at noon tomorrow.
- Make sure it is NOT a slow clock and we want to avoid losing the lease in 24 hours.
- We could in principle put a bunch of other stuff in the DHCP server and ask for local type information as well.

DHCP events

- Request
- I would like an address
- Offer
- Here is an address you can have
- Request
- Client picking an offer
- ACK
- Server confirms the offer

USB

- Network type device that has the same kind of issues
- Take a flash drive and plug it into the USB bus
- Needs a name, but what name will we give it?
- Not useful for connecting generally anything in the world
- That hardware will assign addresses

Name service for self-namers

- Tell you what your address is and anything sent out will be listened to.
- We need a simple way of doing addressing since it is a simple type of network.
- Recall: bind
- Maps part of what a process does to a particular TCP/UDP port

- Goes out through a particular TCP address or comes out through a similar type of thing
- How does a guy know what is going through you.
- How do I know to connect up to port 80 if you want to get to that machine's web server.
- We know the names of machines we want to connect to but we don't know the port #
- We can go through the register and this particular service goes through this port.
- Other parties that you like to go to service.
- Big hierarchical DNS system
- Human-level names to IP address systems
- You will register my service to this port to a different type of translation

Telling everyone else…
- How do others know your new name?
- You know your new name
- The guy you got it from knows your new name
- What about the rest?
- You have to tell them what your name is!
- DNS
- I am going to change my name
- Because I was given a new IP address, you used to say my DNS name matches to Y, now it matches to X.

Using the net to find names
- DCHP provides a certain amount of "glue" for the client
- Go from local Wi-Fi network and the router is the hop in the middle
- Subnet that says which portions of the IP address space are local
- Local portions can be broadcast and non-local portions have to be routed through a router.
- We can say how we deal with those IP addresses.
- DNS servers will get names for you
- Gives a translation
- Needs a higher-level name to understand how to do the translation
- Know what the DNS server's name is.
- Glues you into the overall Internet an includes what the response is to you.
- One or more DNS servers we can use.

Configuring DHCP server
- Hands out IP addresses
- We are giving out the IP address for Internet communications
- It has to get to you
- It has to be a name that no one else has.
- Get hold of exclusive rights to use that name.

- Nobody else is supposed to assign those IP addresses to a machine.
- If you assign to the block that is yours, everything else will work just fine.
- People connecting up to me can get one of those other 256 names.
- All messages on the Internet will go to you, who I just assigned it to.
- Assumes good behavior from everybody else on the Internet.
- Pakistan decided to take the addresses of Youtube, and Pakistan didn't want some videos out related to religious issues
- They didn't want Pakistan to send irreligious images.
- Therefore, many messages intended to be sent to Youtube were sent to Pakistan.
- Individuals, use IP addresses that aren't theirs
- If I spoof your address, the responses don't go to me, they go to you.
- Distributed denial service attack
- Hope people play by rules and get the addressees they are supposed to get
- Tell the DHCP server here you go.
- Big organizations can say we have one really big block and we don't want to block out the smaller servers
- Big block can be used to dynamically send to smaller blocks

Pros and cons

- Don't screw up on reconfiguration
- It will work
- Changes are manual, which is a pain
- Pick at random
- Can work
- Might not work
- Ask someone else
- Easy for client, allows coordination
- Everyone at the top still has problems

Impact on in-progress comm.

- Address X that belongs to him, now belongs to someone else
- You can continue to use that name or it can shift on a new name.
- It can shift and depend on what you want to communicate with.
- Who told you it was time to shift.

W 6 Dis 2-12-16

- Be careful about packet loss
- It is NOT the same size as window size
- We can make a delay so it doesn't hit the sender at the same time it is sending
- TCP: Overview
- Has some other features in addition to reliable, data transfer protocols
- You have the option of getting extra points for Project 2
- 20% of Project 2

- 20% of the total grade so 40% of possible grade
- Full congestion control implemented: 4% boost on your total grade
- There must be one sender and one receiver
- Reliable and in-order byte stream
- Pipelined protocol
- TCP can send multiple packets to the other side
- Same as callback-N and select-and-repeat
- Receiver window size that is used for flow control and the congestion window size is used to control the congestion in the network.
- The TCP can fragment the whole message into multiple segments
- Sender & receiver have buffers
- Sender's application writes data in its buffer and sends its packets to the receiver side
- Receiver buffers the data in its buffer and it reads its data to its application
- Full duplex data: you can communicate simultaneously. If you are receiving some data, you can also send some data
- You can communicate in two ways once connection is established
- Connection-Oriented
- Establish a connection before you send in some data
- Three-way handshake to maintain a connection
- Has a connection teardown process to finish a connection
- Controls flow
- Prevents overflow of receivers buffer

TCP segment structure

- UDP header has information that is the same as this
- checksum
- src port #, dst port #
- Why doesn't it have an IP address here?
- It is handled in another layer!
- It is handled in the Network Layer, not the Transport Layer
- Length of the packet
- Each unique condition between two processes can be found out using src IP address, dst IP address, src port #, and dst port #
- By using the sequence #, you can make the packets in order
- To give feedback, you can use acknowledgment #
- Acknowledgment # is feedback from the receiver side saying that you received this packet and you are expecting this next packet
- You can handle packet loss and determine which packets have NOT been received
- This header length information indicates the length of header and this length means that the length of UDP header + application data
- The size of UDP header is fixed and it is always 8 bytes.
- Here, TCP has an Options field and the header length can be minimum of 20B, but it can be up to 60B

- This is why TCP carries header length information
- There are 6 flags
- Not used often:
- Urgent
- Push
- Used frequently
- Reset
- Synchronize
- Finish
- You can know how many packets you can send to the receiver
- Urgent data pointer: Not frequently used
- Options are not used often either

TCP seq. #'s and ACKs

- How 2 hosts communicate using TCP
- Host A
- Tries to send data C
- Seq # is 42, ACK # is 79
- Host A expects Seq # of 79, and the host data (Host B) sends Seq # of 79 back
- Seq # becomes the previous ACK
- When Host B sends ACK, this is the next expected sequence #
- ACK # is the next expected sequence # (43 in this case)
- It has to keep the sequence # here!
- Both sequence #'s and ACK #'s are represented by bytes
- This is why both sequence #'s and ACK #'s increase by 1

TCP Round Trip Time and Timeout

- How can I set the timer value?
- TCP uses roundtrip time to figure out this timer value
- Default value is 200 ms
- Changes during the communication
- Delay between two hosts are too long, there should be a delay in packet transmission
- Timer value will increase
- RTT == round trip time

TCP Round Trip Time and Timeout

- EstimatedRTT = (1 - alpha) * EstimatedRTT + alpha * SampleRTT
- Value of RTT fluctuates, but if you use the previous equation, you can get the estimated RTT
- Use this EstimatedRTT in this case
- Setting the timeout
- Calculate by using the equation
- TimeoutInterval = EstimatedRTT + 4 * DevRTT
- This is the TCP timer's timer value

- Don’t worry about this in the Project because the timer value is fixed

TCP reliable data transfer

- Sends multiple segments with cumulative backs
- Uses retransmission timer
- Difference from previous data transfer protocol is that retransmissions use duplicate ACKS
- This means that in Project 2, we need a way to figure out packet loss/ packet corruption
- We can just send a packet in the header that says retransmit
- NACK can be used in general, but this is NOT going to be used in the Project 2
- If TCP receives same ACK # again and again, this means there is packet loss because the sender triggers retransmission
- Dealing with corrupted data
- If you receive a packet, you will just send an ACK packet, or just ignore it.
- Q. If we didn’t acknowledge how #’s progress, is there a way the receiver can acknowledge the receiver # that they saw

Receiving three more same ACK packets

- If you have a duplicate ACK, don’t send the segment again

TCP: retransmission scenarios

- Timer timeouts and you have to send the data again
- ACK # is the sequence # + # of bytes of data
- You have sent 2 ACK packets here and if the timer package is too short, it times out
- When it times out, the sender sends the packet again and it should be corrected

TCP Flow Control

- In flow control, you don’t care about the network, and sometimes it is congested by many other hosts
- These hosts are using the same network and they generate many packets

TCP Connection Management

- Utilizes a three way handshake
- To exchange data segments
- If there is no more data to send, server sends FIN back
- Why do we need a timer if no one else is going to use the channel?
- Port # is staying active, and when you request a packet, if you try to run the client with the same port #, it will NOT work
- The timer is the reason for this.
- TCP is way more complicated
- To get extra points, you should start the project early!
- Not that easy compared to Project 1

W 7 T Lec 2-16-16

Recursion and Networking

- Normally, people talk about the Internet stack of layers (ATNLP)
- If you think about the whole idea about layers and functionality, it becomes more natural to build multiple stacks with #'s on it.
- In order to achieve the effect I want to achieve, you need to build a customized stack

Outline

- Recursion
- Stacks, hourglasses, and directed acyclic graph (DAGs)

Preview and motivation

- Putting the pieces together, find what is necessary to actually be able to perform networking

What do we have so far?

- Communication
- Describes how they can transport info back and forth to each other
- These parties have to have a protocol that describes how they share information.
- What if the aren't directly connected?
- You can use layering for that purpose
- Higher layers let you have homogeneous indirect communication
- homogeneous: Share a common namespace
- It has to have the capabilities of relaying as well
- Go from direct link to direct link
- Not sufficient to have a single layer, there have to be different things you have to do
- Different needs for the application level as well as different types of networks we are using.
- Have a bunch of different things to put together in different ways
- Heterogeneous layers need to be combined together
- Accomplish this by stacking
- Allows indirect communication that is heterogeneous
- Assuming we can put things together, we can still communicate even though we don't share the network.
- Go from layer to layer and work with a different type of namespace
- Change names as you go through the layers

Putting them together

- Layers can describe other types of interactions between shared information
- Single protocol and single namespace per layer
- Put the pieces together in many ways

How do we know:

- Figure out whether to use TCP vs UDP, HTTP vs secure shell, etc.
- How do I put together the different pieces?
- Figure out what to stack on top of the other
- Which layer in the stack can you figure out at the resolution point.
- Figure out how to take the name you got and which layer above us can we translate to.
- We usually don't have a translation mechanism i.e. DNS level name.
- This is pretty rare
- We usually won't be able to make this because we cannot resolve those names.
- Resolution mechanisms help us know which layers we **could** stack
- The question is SHOULD we stack them?
- Which layer should we use?
- We want to be able to reach our end goal more efficiently.
- Make it closer to getting the message to where it should go.

What's missing?

- We need a map!
- How does each layer relate to each other?
- What should we do to figure out which layers we should be using?
- Not so much about per-node traversal
- Rather, the layer protocols we should use (one level higher than nodes)
- Go through a set of layers and try to find which one you should pick
- Each layer requires us to convert to a new namespace and a new protocol.
- You want to get it to the appropriate layer at that node
- For example, in an HTTP message, get to the HTTP layer
- We need breadcrumbs to find out where we came from in the layering sense.
- Two kinds of movements going on.
- We are changing directions many times (the path matters)

Maps and map use

- Using the map
- Recursion comes into play
- Hourglass Principle

Using recursion to describe network layering

- Great concept for understanding how networking works, but NOT necessarily how the code, hardware, and most architectures work.
- If you look at the Linux kernel, you probably won't find ANY recursive code.
- This is more for a high-level understanding of what's going on.
- Easy to get confused about this point.

What is recursion?

- Induction
- Prove something with a base case and an inductive step.
- Base case:
- We have an assertion that we claim to be true as a starting condition
- Inductive step:
- Prove a composite case <u>assuming</u> already proven cases
- Example:
- 0 is a natural number
- X + 1 is a natural number if X is too
- We can use the fact that X + 1 is always natural and extend this outward to infinity

Inductive proof

- Lecture 12, Page 12
- Add them together first because it is a sum, then plug in and verify it works

Recursion: backwards induction

- Start at something fairly complex and make it simpler to assert that it is definitely true
- Do this in computational terms
- Reductive step:
- Reduce a complex case into components
- Rinse and repeat until you get down to the base case
- Base case:
- You don't need to do anymore recursive steps and you just have a definition for this
- Everything else will flow upwards as we unroll the recursion
- Eventually we can get the value for the complex case

Recursion: example

- Lecture 12, Page 14 and 15
- Check if negative, otherwise check if n == 0, finally run the recursive call if n > 0
- Each call at the top will call a simpler version
- Eventually we reach our base case, and we return each value up the recursive tree

Fibonacci series

- Base:
- Fib(0) = 0
- Fib(1) = 1
- Reduction:
- F(n) = F(n - 1) + F(n - 2)

Properties of recursion

- Base case:
- Similar to induction
- Self-referential reduction case:
- Supposed to be simpler than the original case that we started with
- Induction in the reverse order

Differences

- Induction
- Starts with the base case
- Uses finite steps to go to infinity
- Recursion
- Start at a finite place and work our way down to a base case
- Get down to a base case rather than building up from a base case

Properties of recursion

- Recursive step is self-referential
- Whatever parameters are to the recursive function are what the recursive function returns
- If we have a recursive function called recfunc, it should return type

Variants of recursion

- Regular recursion
- Reductive step is an arbitrary function
- Must include self-reference
- Self-reference has to be **simpler**
- We can figure out Fibonacci numbers using this function
- Needs a base case to work properly
- Those things are required in our recursive function

Why simpler?

- Reductive step must simplify
- Otherwise, you get into an infinite loop and you're screwed.
- Make sure you simplify, otherwise you never will get out of the recursive call.

Tail recursion

- Same rules as regular recursion with one additional thing
- Self-reference ONLY as the sole last step

Why tail recursion?

- You start building up stack frames and there are particular costs
- You can use a "goto" or use a loop in this case
- Take the code and replace it with a while loop
- Lecture 12, Page 24

- Make sure you have the same effects as tail recursion
- Allows us to avoid actually implementing recursion
- The stack frame has its very own storage associated with it, but we might need extra variables to hold things temporarily

How is recursion related to networking?

- Treat layered networking as a recursive operation
- Base case: communication
- Two parties share a direct link
- 802.11 will go from my machine to your machine directly
- If both machines understand the same namespace, the reduction steps is the networking!
- Reduction steps: networking
- Stacked layering is the regular recursion
- Relaying is the tail recursion

Stacked layering as recursion

- Start at HTTP layer go down to TCP layer down to any subsequent layers
- If P is the guy who wants to perform HTTP requests, P and Q will NOT be on the same network necessarily and won't necessarily use HTTP to get from P to Q
- You have to translate P to X and translate Q to Y down at the physical layer
- If X and Y have direct communication, you share the physical network
- This turns the P-Q layer into the X-Y layer
- Go down gradually into X-Y.
- Perform physical communication on X-Y
- There has to be some layer in the stack that allows the source to reach the destination
- For scaling reasons, we will have to do some relaying (analogous to tail recursion)

Relaying as tail recursion

- A can reach C
- If A shares a physical link with B, and B shares a physical link with C
- Transitive property
- A -> B, B -> C steps are like tail recursion

Recall how stacked layering works

- Get to the layer you share with destination
- Go down and up to reach your end goal.
- Lecture 12, Page 28

Where's the elevator?

- In particular, when do we want to go down to the layer
- **When we don't share a layer with our current destination**
- What if we can't go down?

- Go up instead

The basic block

- The elevator will be a recursive function
- The block
- The elevator:
- Lecture 12, Page 31
- Call some layer in which you inject to the system
- Provide information and your goal destination
- Go step by step
- Initial creation of the message (gets called multiple times)
- Process the data
- Make it sensible within your own layer
- Check if you are at your destination
- If the message is for you, return and go back up the stack
- Often, you won't be at the destination though
- Call layer on S' and D' (these are the lower layers' namespaces)
- Take whatever we need

Why prepare then send?

- You can't change the order
- If it is for you, put it into the format and give it to you!

Why does this work?

- Effective recursion works so this should work.
- We don't necessarily change this to a name that is an alternate alias for our destination
- We may get to a point that changes D' to lever
- We will be doing some interesting things with the translation of our names.
- Get from here to there with a direct communication link

An example: DNS request

- User requests *gethostbyname*() to the OS
- gethostbyname should suffice
- Start with the DNS root or start at a local node
- Ask yourself recursively if it's for you
- Call a layer that can help you move things
- Need a way to get to the server
- Destination is going to be the name of the server

Recursion steps

- Look at the slides (this one is hard to trace through)
- Use a transport protocol and it will specify a UDP name
- Translate to bob.com and have a particular port you want to hear on.

- This will be important since we may want to go back to our network and hear a UDP port.
- Translate UDP because we decided that is the next layer we want to use.
- Same recursive block that we used in principle
- Format a proper UDP message but we cannot send it physically since it doesn't have physical connections
- Let's use IP to get it to some other place
- Provide an IP name!
- One of the four octets (52.3.5.3)
- ns.com will get translated to 2.43.13.123 -> there has to be a table for those connections
- Call IP
- Keep going down to the atomic layer (bottom-most layer)
- Send it once you get the BASE CASE

What about at the receiver?

- Message comes in at some base protocol
- Somehow to another, we have to get from here to there
- How do we do this?
- Fiddle around with port #'s mechanically
- Conceptually, we need to use recursion and recurse in the opposite direction
- What happens at the receiving end and what happens in between?
- The very first thing we will hear about is on the physical interface.
- We will essentially have to move it up the stack to the appropriate layer
- It doesn't understand the DNS protocol and it won't go down to the layer that can handle this message

Recursive block at receiver

- Pop back up the stack
- Recursive calls in the opposite direction
- At each recursive call, we went from data source destination to message S', D'
- Eventually, we will have something like S', D", D"', etc. (some crazy recursive result)

Interfaces

- What does the block input?
- Source name
- Destination name
- Message
- All of these things will be in understandable formats that the block knows
- If this is a DNS block, names coming in at the top will be DNS messages
- Output is the same thing as input
- Same effect as if we were going down through another recursive layer
- Same kind of block at the other end

Process the message

- Protocols can be modeled as finite state machines (FSM)
- Every layer is going to have its own finite state machine
- DNS has its own, UDP has its own, IP has its own, Ethernet has its own, etc.
- Physical layer does NOT have a FSM because it moves electrons around
- Either in a default state, or they have been put in a state due to leftover information
- Tape-in is the “input” message to be shared
- Tape-out is the “output” message

The role of naming and routing

- Resolution tables
- Use this name to get there or you can’t get there!
- You need to have the ability to translate the name into something that will let you make progress
- This implies if a high layer says it can get to a name, there has to be a low level name that can fulfill that progress
- Does the translation as well (DNS server name to the IP address)
- Translate to a name that will help you route the message

Stacks, hourglasses, and DAGs

- Recursion allows you to go up the stack and down the stack
- If you have choices to make, which will you choose?
- Do the proper thing and process things at the destination.
- We need something like map that gets us through the set of layers for the networking operation we want to perform

Stacks

- A linear chain of layers
- Rare in practical networking to go multiple directions

The Hourglass

- The stack is what you use for any individual message
- You often have choices, before you build that particular stack, you can do many things
- Use the hourglass approach to try to maintain control of the set of paths that we have available to us.
- The top half describes reuse
- The bottom half implies that you have choices

Top half

- We have protocols at the top
- Narrows and narrows until we hit the IP
- It has a narrow waist and everything gets translated to IP

- If you can satisfy your request by another process on the same machine, you will go through the waist of the hourglass.
- Everything above us shares the IP layer
- Go through the same point in the hourglass stack.
- Everything coming in will come out of the IP layer at the bottom

Bottom half

- We can have very low-level physical approaches on how we translate signals
- Below IP, we have choice here
- Different physical media have different properties
- Some are good at having short-range high bandwidth
- Take IP and translate it to the appropriate low-level physical mechanism
- We don't need to know the details of each level of the hourglass, but there's a lot of other options that get introduced
- A lot of things can be done here
- CDMA is one way we can do this by sharing a channel like the Ethernet.
- Take a common thing (IP) and translate it to something good in our specific case

Who talks to whom?

- What we do coming down may be the same as going back up
- You start and finish at the same layer
- Lecture 12, Page 58 (Diagram)
- If sender chooses to use TCP, the receiver will also use TCP
- IP has a relay node in the middle
- Goes across the SONET and then ends up at the Ether layer
- We don't share the SONET level at there receiving end, and we don't share the Ether at the sending end
- We keep translating names but we have to remember something
- Breadcrumb analogy
- We need to be able to send things and see what we are trying to do.
- Use the layer that he gave me.
- Don't worry about the SONET stuff because he will never see that.

The DAG

- Representation of how you move through the layers
- What choices you have
- There are going to be at various points and choices that we have to consider
- We needed a map that says we got from here, to here via the graph
- This graph lays out the alternatives and the arrows essentially allow us to send messages to different destinations and follow the links
- Acyclic graph because you don't go in circles
- There aren't going to be cycles in it

DAG Components

- Recursive block (RB)
- Translation table (TT)
- State instance (SI)
- Representation of one of those protocol finite state machines
- Structure
- Directed acyclic graph
- TT as primary nodes
- SI are intermediary nodes
- Network protocol layers will have their own namespace and you have to have a translation to the namespace that is understood by that layer
- Make sure you are working on names that you understand
- Finite state machines will decide if a message is for you or not
- To go to another layer, you have to go to the FSM for the next layer
- Translate names you understand to names that the receiver understands

What does the DAG indicate?

- Indicates recursive steps
- Fan-out
- We have alternatives
- TCP layer
- We can use IPv4 or IPv6
- Fan out graph
- We can have a finite state machine and a # of choices of what we can do next
- Fan-in
- Protocol reuse/sharing
- **If we have only an 802.11 network, ultimately the only way to get messages off this machine is to run it down the Ethernet**
- You will have to reuse the underlying network we have available on the machine

FSM rules and state

- The protocol runs and does something and it shouldn't reset to the beginning
- The FSM describes what you do.
- If you never do anything, you can have somewhat more simple approaches than FSM
- We aren't going to try to remember anything between messages
- No state maintained between messages
- Perhaps we can get an acknowledgment or a timer
- There will be some kind of input that will let us know
- State will be the "breadcrumbs" that indicate how we move through the map

Follow the yellow brick road

- As we go down, search all the options for each step
- Decide which of the alternatives we should use
- Use the translation table to associate each choice
- Follow a trail by using multiple messages via the “breadcrumbs” (state) left by previous message
- TCP machine ready to send another message
- Alternately, it is a TCP machine that is currently held up because of congestion control mechanisms
- There is another type of “breadcrumbs” to use
- We shared a slide where the HTTP layer talks to the HTTP layer, TCP layer talks to TCP layer, etc.
- In comes a message containing a bunch of bits, we don’t have any magic and that message we got when we came in.
- We have to have these breadcrumbs in it.
- Once it’s down to the TCP level, it will go down to the HTTP level
- Various layers have been inserted to allow the receiving end to do the right thing.

Breadcrumbs inside the message?

- Breadcrumbs are inside the envelope and we need to do processing for that layer
- Put information into the message and have data provided to the lower layers
- Typically, it will build it’s own message by wrapping what you gave it with more stuff.
- Put more layers and each layer will create a brand new envelope with “breadcrumbs” you care about.
- Pass everything subsequently to the lower level
- Continue this process recursively
- Wrap it up and pass it down
- All of this happens recursively
- Gets built up in your messages
- Take incoming messages and see what the envelope is saying
- Who should you give that interior envelope to?
- Breadcrumbs tell me that this is a TCP message
- TCP layer says we have a message coming in and look at the TCP layer.
- What do we mean by “breadcrumbs”
- All we are doing is moving bits
- **They are bits in a header in a message**
- Create a message with a header, and create a bigger message that encompasses this smaller header
- This allows the guy on the other end to figure out what we need to do with it.

The DAG looks complicated

- Conceptual, NOT real

- If you build the DAG based on what is actually in your machine, it would look very complicated
- We want dynamic path selection, empower you to choose your path through the network as you move through each connection

More accurate than ONE hourglass

- Various other places in the stack because we can have high-level protocols

Dynamic graph path selection

- Communication across some other node, you want to build a stack
- The stack does whatever it does to try to move that message
- This is NOT done in Internet technology.
- Try to send the message and things didn't work out.
- Return an error message upon failure
- When things get injected at the top, they are two different messages
- If it comes in at the top, it is different from what I saw before
- Things don't work out at the UDP layer but the network layer will decide if it is a good choice or not.
- Switch to TCP if necessary
- Nothing fundamental about networking that prevents you from moving up or down
- Even in a world of real, useful technology, there might be options that let you do this.
- Rather than going back to the application, if I the data center have an alternate link, we will go there and fiddle around with the stack in the middle.

Q. Things slipping in between layers. Is this a fan-in/fan-out aspect?
A. This is included in the DAG. This is very general. IP/Sec (?) provides cryptography but it doesn't move packets across the network. This lets you put another Hourglass going out to a common layer. The way people have talked about this is that it is a 3.5 layer (layer in the middle).

- Went to the DAG, added a new layer, more complex DAG, and all kinds of recursion
- It all fits in very naturally
- Talk a great deal more about the DAG and exactly how it works

Summary

- A good way of generally thinking is to treat them as recursion
- Move up and down through layers
- We need multiple choices of possible layers
- We need some way of choosing which layer to connect up with
- We need a map to govern what is the next recursive step
- Which one shall I do?
- Pick one recursive call or the other
- The map will manage the FSM (protocol) state
- What you keep locally

- What breadcrumbs you keep so the guy on the receiving end knows what to do with the following messages.

W 7 R Lec 2-16-16
Using the Network Layering DAG
Outline

- What do you do to determine which network layer interacts with another network layer
- What is the meaning of the DAG and how do we use it to interpret what happened?
- There are nodes that represent FSMs and optimizations
- How do we walk through the DAG to understand what is going on.

Reminder

- <u>Conceptual approach!</u>
- For the most part, the code is designed as a fixed structure
- They worry about their own layer of code and their own layer of network functionality
- They don't really think about it as a whole.
- Often, they bundle different things in different places.
- You can program networks in this way i.e. Click Router

Another Reminder

- The DAG is motion through network layers
- A lot of it goes up and down using network functionality through its nodes.
- Multi-hop path to communicate between source and destination
- Connect together the DAG of different elements for a more complex DAG

The DAG

- Describe different layers of different functionality
- Combination of Nodes and Arcs
- Set of components for these components
- How we layer things, DAGS get built in a particular way

10,000 ft view

- You will see that we have alternating nodes here
- Some of them, the ovals in orange will describe finite state machines and the states of these FSM's
- We will have some kind of lookup of the name and translation
- Directed arcs
- Describe direction in which things flow
- Rules that govern what we can connect
- Outgoing links connecting up to incoming links
- Alternate between the two.

Components

- Tables
- One example of a table from a DAG
- Does name translation and we are going to have a name for things such as a source or destination
- Name changes as we go from layer to layer.
- As we go up to the IP level, we will have a name change to have an IP name
- Part of what is going on is indicating which layer we will go to next.
- Going from DNS to IPv4 name, which implies that the next layer down will have IPv4 functionality
- This in turn lets us pick the path for particular choices.
- For example,
- Lecture 13, Page 9
- this node translates the DNS name to an IPv4 name
- This in turn lets us take different paths through the graph to do different things
- Lets us employ different data through the network.
- We could go through the Internet or through the Wi-Fi network.
- FSMs and their states
- Lecture 13, Page 10
- Each protocol can be described by an FSM
- Describes which state you are in at the protocol
- Waits for input at a particular state.
- That input can be tape-in (input that is not part of the message)
- It could be a message from the FSM above it.
- If you are getting a message, you are coming from the FSM below you
- The message has arrived and you are passing it up to the stack.
- In our previous discussion, we can't afford to wait forever, so we need to change states via a timer
- This particular FSM can also emit things and put things out
- One, it is going to send messages
- It will go to the "lower layer"
- Another possibility is that a message came in and we did our processing on it and we should push it up to the next layer
- Finite state machines have tapes in and tapes out
- Rules
- Associated with this FSM will be rules
- For example,
- Lecture 13, Page 11
- A TCP finite state machine
- Lower layer will provide a DNS message to the server
- We want IP + 4
- What we actually get is IP + port (the appropriate transport name)
- This does whatever processing it needs to do
- We will get a response in the opposite direction

- Directed links
- Show the relationships between running FSMs and Translation tables
- Running FSMs
- Includes “users”
- Translation tables
- Links describe relations between FSMs and translation tables
- For example,
- Lecture 13, Page 13
- This said take the output of the translation and provide it for the lower level state machine.
- You do your TCP thing with that.
- Describes the relationship between DNS -> IPv4 table
- When we go up in the other direction, the things it gets from lower layers will go to the DNS layer

Rules (graph constraints)

- Node rules
- Table rules
- <u>Root</u> tables
- Table just below the highest level oval (the one without writing in it)
- This is the “user FSM”, whatever is on the computer performing network operations
- Provides the inputs for us with the network messages and the communication we would like to make.
- The messages that ultimately get delivered.
- Not really part of the system and it isn’t necessarily a human user.
- Translate name to the protocol level.
- In order to get to Google.com, we have to go through multiple layers and this gets us out on the actual link and this will have other finite state machines.
- The user-provided names are local, inside names
- Things understood by the OS here.
- The translation we will perform is performed locally.
- For example,
- Lecture 13, Page 17
- One of these are the ones we are looking at before.
- The user or application will provide a DNS name
- The purpose of this table is to do a translation that gets us to an IPv4 name
- Could be an actual table
- Could be a whole set of other things on the table
- The purpose of doing this translation cannot work with DNS names
- It needs transport layer names
- <u>Leaf</u> tables
- Roots at the top, and leaves are at the bottom
- Just above the actual, physical link that lets us move data across a physical link

- This represents input/output to a physical link
- **Only part that is "real" (physical and you can touch)**
- You can see the cables coming out of the machine
- The other things will be virtual (provided by software in the machine in question)
- The purpose of the leaf table is to translate
- Will translate protocol names and changes from the realm of doing computation and doing computational tasks to the point of putting bits out to another place
- For example,
- Lecture 13, Page 19
- This table translates an IPv4 address to an Ethernet Mac address
- From this point onward, we will dump it into an Ethernet card and we hope everything goes well from that point onward.
- Intermediate table rules
- These intermediate tables represent messages that come into the FSM from below or go in from above
- Translate names
- FSM + state rules
- Finite set of states and a predefined arrangement of states
- Operates on a protocol, so there is a single namespace for that protocol at that layer
- We are already working on a fixed namespace that belongs to the protocol
- This matches names in the table above AND the table below
- There is also virtual user FSM at the top
- Represents what is going on outside the communication
- For example,
- Lecture 13, Page 23
- These are roots that represent user programs
- Link to root tables
- Leaves
- FSMs represent physical links
- Represent a physical namespace and there is an alternation between FSM and tables

Paths

- Linking nodes
- Orientation
- Directed in one way
- Acyclic: no cycles (loops)
- Head/tail
- Head and tail always connect to different types of graph nodes **from each other**
- Head/tail rule
- The tables and FSMs "**alternate**"

- Depending on what we are doing, we may be following the direction of the arrow, or we may be going in the opposite direction.
- When we have something coming in from below, we are going in the opposite direction of the arrows.

Path meaning

- Creates communication from networking
- We are going to have layered capabilities that lets us build networking to perform communication when we don’t have the direct ability to.
- Describe the “stack” of layers
- All communication is pairwise
- Nested composition
- Has a linear path
- Described by the FSM
- Each message that we send traverses one **single** path through this acyclic graph

Path rules

- The different host types
- User hosts
- Guys at the end
- Initiate messages
- Consume messages
- Relay hosts
- Machines that sit in the middle
- Not the machines that are supposed to get the message
- Rather, they move things from one place to another place
- Used for transit purposes
- Path in a user host
- Start at the top, go down to the bottom, goes across via relay hosts
- If OTOH, you have a message coming into the user host, start from the bottom and go up to the top.
- Pop layers off when you come through the other side

Implications of host path rules

- Need to connect via a physical link
- Start with a name that the user knows
- Translate a name that is understood by the user and then we need to get down below.
- The user has to understand, so we cannot start with an IPv4 name unless the program knows about IPv4 names (compatibility needs to be present from the start)

DAGS and links

- Links are physical network connections
- If we have a DAG at various relay nodes, we can connect them up to create an overall acyclic graph

- When we consider totality of the DAGs, everything that happens from the message goes down a stack and up through a stack
- The process continues until we get to our end goal destination.
- Use this graph to describe the overall network path.
- Not the way that real networks are built.
- We are describing a way to understand what is going on in a deep manner.
- We want to give you a deeper understanding of what's going on in networking.

Path in a relay host

- Moving from node to node. As we move through each relay node, what's going to happen?
- Enter through the bottom, and then we go down again.
- Not necessarily in the same path as you go up and down
- Eventually, you get a common stack
- Ultimately, it gets to a certain FSM
- Figure out what you should do to figure out the next hop on the destination node.
- FSM understands the name of the host at a destination level and it should figure out what happens between multiple nodes and multiple links
- Ethernet doesn't understand IPv4
- What you have to do is go down through more tables and FSMs until you reach the FSM at the bottom that lets you get to a physical link

Path matching

- Situation where we go across multiple machines
- Source machine is on the left of the dotted line (Lecture 13, Page 34)
- We keep going down and eventually we reach that green oval.
- Use the link and we aren't going through any more protocol layers
- Goes to the next node in the system.
- The relay node is in the center
- Each one represents a layer of network protocol, and eventually, we reach a node where we know it doesn't belong to us and needs to be transmitted.
- We have to go down through protocol levels and this is what happens on the right-hand side of the area of this diagram.
- Go through each translation table and we eventually hit the pink ovals
- Represents the physical link we are going to on our next hop.
- Destination will use the next link and we have to go up through the stack
- Working well above the link layer and it has to be an HTTP message.
- Something understandable so we can hand it off to our web server.
- Go from pink physical link and go through a translation table that represents a higher level protocol.
- Certain aspects of any path that are represented by these FSM graphs
- Links have to match their tails
- We have to have the green oval match up with another green oval

- Pink oval matches up with a pink oval
- Do more translations and translate it to different protocols
- The guy at the top wants to do HTTP and the guy on the other end also wants to do HTTP
- Up and down through multiple layers of the stack
- At the end, you have to end up with the same FSM at the top and bottom.
- Put a striped link and pretend that is what you got.
- Introduce this by the program and directly transfer that to the destination end at the same protocol.
- Emulating purple striped link with all the other pieces of this graph.

Pushing and popping

- It is going to use the matching FSM on the way up
- Doing DNS translation, we have to remember it is for this particular thing
- When we get a response, we have to send something back up related to the application.
- This makes all of these multiple FSMs act as if they were one distributed FSM

Tables

- Name translation
- Some domain of names
- Domain: a set of possible names that all belong to each other
- Map from one name to a name in a different domain
- All kinds of things you can do
- Translate from IP domain to the Ethernet domain
- Translate from IP-5 to Eth-85
- Essentially, the table says go from a name in Domain A to a name in Domain B
- The graph can be more complex than 1:1
- Could have aliases and they all translate to the same thing in the two domain
- Be a little careful regarding “to” and “from”
- In actuality, I may end up translating multiple names for the same message
- As we go down the TCP level, we will translate them both into IP names and this will be a fairly trivial transformation
- Maybe we don’t need to translate the source name at all
- At least, we will probably translate 1 name though.
- We can translate via aliases (N:1), proxies (1:N) or 1:1
- We can keep info about how long a particular translation is good for and under what circumstances are they good to be used
- Relay support
- When can we relay things?
- Gives info about where entries in our table came from.
- Not necessarily, implemented by an actual table

- Could be implemented by a process or look up at a different data structure
- If you get yourself too tied up into bits and bytes, you will probably misunderstand something

When can you relay?

- You could do it and say there is only going to be a translation possible if it is cached or permanently kept somewhere
- Otherwise, if we cannot access it, we will throw an error condition
- Else, we can grab it from another place
- Step aside and do whatever else to get that name
- Can even go to an entirely different protocol stack
- You could also say you don't have the name, but you generally want to try to grab the name from another server
- Each message has a FSM that we have to go down from
- Different FSM from ours but it still incurs some overhead
- Important to remember that when we are dealing with a node, it can have a lot of disjoint computer communications going on.
- The table entry may have something that indicates we have it here.
- If it isn't, return an error
- Follow the following procedure to pull the following procedure

Does an entry ensure connectivity?

- Not necessarily!
- A DAG inside one host should in theory match to another if the want to communicate
- What if we want to talk to someone who doesn't speak the same protocol?
- We have a situation where we have a source, relay, and destination without appropriate nodes for communication
- We will get to a point where we don't know what to do with the message.
- Eventually, we will get an error, and at a high-conceptual level, there is no virtual connectivity
- We won't be able to communicate like we hoped to.

Other table information

- It may be that there are probabilities where we translate from one versus another.
- That is one thing we can do in terms of resolution context.
- Table maintenance, how do we keep it up to date and correct?
- We have a lease on our IP address, and it may be that we are hanging around in the network when the lease is about to expire
- There are various other types of things that we can keep in the translation table to do more effective networking later.

Table origins

- Not necessarily explicit table data structures
- Where does this info come from?

Manual table configuration

- You have a boot file that means you don't have to go to any boot file
- You can look it up and it is manually configured
- This is the way I want you to translate it to a lower name.
- Ultimately, we get the table from outside
- DHCP: Grab it from another address
- DNS has the translation, ask them for it!
- Dynamic update
- Things that are going on disjoint from actual communications that senders and receivers have.
- The table should be sitting here, waiting to go with the correct information.

Directed links

- You could go this way or that way
- Could define protocol reuse or sharing

Links in the table

- What names need to be here?
- Something that indicates how to get to the next FSM
- You have to have something to decide under which particular circumstance which choice you have to take.
- Do a translation from A to B
- B to A means to go the other way in terms of translation
- It could be a separate identifier
- For this whole set of names, use a protocol. For another set, use a different protocol

Alternates

- Proxy where we can say that we have multiple things that can be associated with the work.
- Translate it to various different proxies
- Can be done via waits, etc.
- Try one of them and hope it works out
- If it doesn't work out, keep trying until you run out of alternatives or find one that works.
- Send the same message along both paths and hope one of them goes through.
- Down at the bottom, we can hope to have alternatives
- Try something at the **bottom half of the hourglass**

Reuse

- Several FSMs point to the same table

- In effect, one translation table that is useful for several different things
- Essentially the top half of the hourglass
- Merge the down to a single point of IP
- All of them use the translation table to translate you to IP

FSM + state

- Soft vs hard
- Soft state
- The FSM could recover even if the state weren't there
- **Would eventually achieve the right result even if the current state is incorrect**
- For optimization; thus, we need state that may or may not be correct.
- We do have to arrange that the FSM comes out to the right result if the soft state info was incorrect.
- We need to have some recovery mechanism to get the right address and go to the right place.
- If there was no soft state at all, we would end up at the same result if we had bad soft state
- Hard state
- We really have to have this right
- Pretty damn sure that our FSM is in this state
- **If we are wrong, we are screwed**
- The best we can do is throw everything away and start from scratch
- Persists between messages
- Provides context for our FSM operation
- What happens if you can get persistent information and the information keep coming in.
- You want it such that there is no pause period in the middle
- Soft
- Fault tolerant if lost
- More efficient to recover
- Hard
- Needs guarantees or backups
- If you lose your state, you're screwed until you find a way to restore from scratch
- May not always be possible, so you don't want to lose hard state

The base case as an FSM

- No FSM that is Ethernet
- Physically, it is a channel
- We aren't actually implementing an FSM there
- We just assume it behaves in a particular way

DAG walks

- Information that goes to the big outside world
- Go through the graph in many different ways

- Either one can go through Ethernet or wireless
- Different paths through the graph
- If you are sending a message, which path are you taking.
- Choose some path to go through it.
- That is the path that I am using.
- All subsequent paths will go through this path and we made an early bound decision
- This essentially does the same thing in sockets
- Use the same path forever
- Kind of outside the FSM
- We are talking about selection of a path through the graph

Late binding
- Every time we have a message, make an entirely different decision at the level you are at.
- We may have no concept of what we did at the last message, and every message can have a different path.
- 1st message can choose its path and network
- The next message has its own freedom to choose, so late binding does NOT bind you down to common path or network
- Not something that relates to a FSM, but rather how you choose your path

Early vs. late
- Early (static)
- Stable
- Always going to do the same thing
- Not very flexible for changing conditions (suboptimal, fragile)
- Late (dynamic)
- Quite adaptive
- Using a wireless network but if you get a terrible throughput, switch to Ethernet
- This can be expensive
- Make decisions every time you go through the graph, make the decision properly
- Can be unpredictable and leads to all kinds of uncertainty in outcomes

Processing
- The recursive block walks the tree
- Uses the message as part of the context of the FSM

Graph traversal
- Goes to translation table, gets necessary translations, then replaces necessary values

Summary

- This is conceptual
- Not typically the actual implementation method
- Assuming they implemented it correctly, we can think of it in this way
- Assuming they did do layering properly, you can build your own layering and think things through using this way
- Different layers relate to each other.
- DAG encodes the protocol stack
- Describes the stack we are using at a particular place
- Also uses the network architecture
- Portions must match as we said earlier in this class
- Not a strict architecture that we have to follow
- Actual implementation that people use will vary and they vary a lot.
- Groupings and partitioning will vary quite a bit and you won't have any guarantees here.
- If you look at how this DAG works, it will be helpful under most network circumstances.

W 7 Dis 2-19-16
Q. Should you send a struct across a socket? A packet struct for example
A. Not a bad idea because you need to design your own header and put some information in front of your data.

Q. Since you send a pointer and it is local, it might have a bit of an error. Is there a better way of doing it?
A. Either way is fine.

Q. For sequence #'s, if you are going up to the max of 30,000 but rather you get to

like 29,763 and there is an additional added information, could you arbitrarily dump your sequence #'s.
A. As long as the sequence # doesn't exceed 30,000, you are good.

- Demo sign-up sheet and procedure will be distributed in Week 8
- Demo date: 3/10 (Thu), 3/11 (Fri)
- Test files will also be distributed that day
- The files will be only text based
- One file will be very small, the other file will be very large
- You want to see, how the sequence # works
- Distribute tasks in the demo procedure document
- Figure out what you need to do in the demo

Principles of congestion control
- Congestion: What is the difference between flow control?
- Congestion is from the perspective of the network from many sources and the receiver
- What is the flow here?

- The flow control stems from the rate (connection between a source and a destination)
- How can source adjust the speed of transmission?
- Congestion control: Even if their transmission speed is good for the flow control, it can trigger network congestion since there will be too many sources.
- If there is packet loss and long delays of the packets, it probably indicates there is network congestion.

TCP Congestion Control
- Assumes it uses the best-effort network
- IP makes an effort to transmit a packet but does NOT guarantee reliable data transmission
- Each source determines transmission speed by itself
- No explicit feedback from the network

Two challenges:

0. Determining the initial available capacity
0. Adjusting changes in capacity

Q. How do we determine if there are problems with congestion control?
A. Packet loss indicates there is congestion in your network. In this case, you are probably going too fast and want to reduce the transmission speed.

- If effective window size is smaller, you cannot handle more data, so you have to decrease your transmission speed
- If congestion window size is smaller than receiver window size, you have to choose a smaller value from these two possible window sizes.

- The congestion window size is changed by time intervals
- Slow increases to absorb new bandwidth
- Quick decreases to eliminate congestion

- Sender limits the transmission speed:
- LastByteSent - LastByteAcked

- Formula:
- rate = cwnd/RTT [Bytes/sec]

- If there is loss, the sender needs to reduce the rate (condo)

Three Mechanisms:
- 1. AIMD Rule: Additive increase, multiplicative decrease
- Increase transmission rate and the y-axis has to indicate the congestion window size
- If there is NO packet loss, you can increase your window size
- You have to decrease your window size as 8 kBytes
- Increase window size by adding 1 MSS

- In general, the MSS is 1 packet size
- If window size is 1, we can increase 1 MSS until it reaches 2
- MSS is a packet that can increase and send 1 more packet
- Q. Is this common for a lot of downloads and torrents? The rate at which you can receive?
- A. BitTorrent is an application that uses TCP, so it is a TCP thing and won't apply here.
- Actual value of the window is in units of bytes
- If a single packet is 1 kByte, why do we need to decrease it by 50%
- Why AIMD? TCP Fairness
- Many source hosts and we want to fully use the network and also keep the fairness among the many hosts
- If there are k sessions that share the same link, each should have an average rate of R/K
- Why is AIMD fair?
- Two competing sessions
- The graph shows the transmission rate bandwidth
- x-axis: Connection 1 throughput R, y-axis: Connection 2 throughput R
- Same transmission speed and if the point is around here, this Connection 2 has a much larger transmission speed
- If there is no packet loss, you can increase the congestion window size, which means the bandwidth will be increased
- Both connection 1 and connection 2 will increase its bandwidth.
- Packet loss at this point, Connection 1 and Connection 2 have to decrease its window size
- Throughput would be here and after decreasing, there will be no packet loss.
- It increases its window size and when the packet loss happens, it will decrease again.
- Eventually, you will reach around the intersection
- You can fully use the network and you can guarantee the fairness
- This is why we use AIMD!
- 2. TCP Slow Start
- The idea is that when connection is established, you send a single packet first
- If you are using the slow start, whenever you receive an ACK packet, you can double your window size
- Host A establishes connection to Host B
- Initial window size is 1
- When it receives a single ACK packet, it can send 1 more packet (you can send 2 segments)
- Here you receive 2 ACK packets!
- AIMD is fair because if there is a loss, the window size just goes back to the previous size.
- When it reduces by 50%, it just goes back to the previous one
- ssthresh: slow-start threshold

- Q. Is threshold set before connection begins?
- A. Yes, it has a default value
- 3. Congestion Avoidance
- Occurs when you try to avoid a congestion by adding a single 1 packet
- Do this rather than doubling your window size
- If you keep doubling your window size, your network will get congested
- You can probably avoid network congestion
- Increases window size by 1 MSS per RTT
- At this point, you are receiving 4 ACK packets
- Even if you are receiving 4 ACK packets, you are just receiving 1 MSS
- This is all under a single RTT
- Q. When does the RTT begin?
- A. The sending point to the receiving ACK packet encompasses the whole RTT
- Formula:
- cwnd += (MSS/cwnd)*MSS(bytes) upon every incoming non-duplicate ACK
- Whenever you receive an ACK packet, you increase your window size by 4 + 1/4

When loss occurs

- If there are 3 duplicate ACKs, there is a packet loss and if the timer times out, it also indicates the packet loss
- For example, 3 duplicates means that you lost 1 ACK packet, but you received the other ACK packets
- You only lost 1 packet but the next packets are successfully transmitted
- This means the network is NOT that congested
- OTOH, if the timer timeout, you lost everything and there is definitely network congestion
- When the timer times out, you reset everything
- If there are 3 duplicate ACK packets, we will NOT reset everything
- Just decrease your window size and cut it in half (reduce by 50%)

Fast Retransmit/Fast Recovery

- When you receive 3 duplicate ACKs, you retransmit
- Use fast retransmission: set ssthresh = condo/2
- cwnd = ssthresh + 3MSS
- Retransmit the lost packet
- Fast recovery: governs the transmission of new data until a non-duplicate ACK arrives

TCP Congestion Window Trace

- x-axis indicates time
- y-axis indicates congestion window size
- Increase window size by doubling (slow start)
- Window size will be increased by 7T

- When the timer times out, you have to reset everything
- The window size will be 1 again since there is a time out.
- You can also use the slow start period again
- Blue line should be the threshold and when the congestion window size meets this threshold value, we are NOT using slow start anymore.
- Congestion avoidance:
- If there are three duplicate ACK packets, you just cut your window size in half
- Q. How is the threshold NOT necessarily the threshold, and the first time around, the threshold is a different number? How did it affect the threshold?
- A. The threshold value is the current window size. Threshold value HAS to be > 1. Initial threshold value allows the sender to use slow start and the sender can double its window size up to 65

Fast Retransmit/Fast Recovery
- Window size divided by 2
- The other # is the cwnd which was just 1
- If the packet loss is caused by 3 duplicate ACK packets, the window size will be cut in half.
- If window size > threshold, there is an additive increase
- If it times out again, the window size becomes 1 again.
- By adjusting the congestion window size, the network can transmit all the packets from many hosts.
- Q. Why is the penalty for a timeout > penalty for duplicate packets?
- A. When timeouts happen, this implies you lose A LOT of packets! OTOH, for 3 Duplicate, you only lose 1 packet in this case. You don't want to decrease your window size as 1.
- Q. What happens if you receive 3 duplicate ACKS, but the timer still times out?
- A. You have to reset the timer when you retransmit it.

Best way to divide the project?
- Design your selective-repeat first together. One of you guys can implement the sequence #, the other can implement the data transmission without packet loss or corruption

W 8 T Lec 2-23-16
Network Routing
- We use networking when we don't have a direct communication link to our other party
- We have to make decisions about which route to take (left or right)
- Requires network routing

Background
- Think about how routing fits in with general approach
- What information is required

What we're doing

- We have a network that allows us to communicate from one place to another
- Need to share state to achieve general communication
- How do we get that information from where it needs to be?
- Usually, we will use our network in order to run our network.
- Move pieces of information around and use this underlying capability
- What can we assume?
- Who can talk to who else?
- Who is in charge of setting this up?

Relaying and routing

- If there is no direct channel, we need to relay
- Send from one point to another point and we have to keep doing this until the message reaches its destination.
- If we have one choice on a ring or linear network with no decisions, it is easy to decide who to relay to.
- Just go to whoever is next
- If we have a more complex topology, some relaying must involve a choice to make and how do we make this choice?
- We make this choice by doing routing and decide if it's correct to move left or right.

I'll do it myself!

- All of the routes that I care about and we can decide if we go to where I am to any of the points I want to get to.
- Perhaps, I will have my network router do it for me.
- The other thing I can do is to use defaults.
- Basic ways to get things and just fall back on those defaults.
- I just have to throw my packets out the door
- This is actually a very common scenario!
- Just send things out to network interface but we don't have to worry.
- Eventually down the line, we need to set up routes because this can only take you so far.

Limits of going solo

- Reconfiguration: Whenever a node joins i.e. node X joins, we have to figure out what the route is to node X.
- Understand the topology and see which link to rely on and see which link to use.
- It is also difficult to bootstrap this
- Assume incremental deployment for pretty much anything.
- Gradually, more and more stuff gets added all the time.
- When we are saying to do all the static configuration yourself, it is difficult to do bootstrapping to get started.

- Some people might not be available
- Some nodes might not be available
- Generally speaking, you are assuming that other people are going to do things correctly and make sure the people do what you want them to do

Automated routing

- We probably want to do some automated routing for anyone who connects up to the Internet
- Any node on the Internet can connect to any other node without external intervention
- You assume the Internet will do automated routing for you.
- There will be a lot of nodes coming and going.
- Needs to be adaptive in an automated way!
- No big network administrator in the sky who will save you, we need this automatically!
- It also has to bootstrap automatically
- Needs to route to others and route it back to me without any system or network administrator

Collecting our thoughts

- DAG describes the stacked protocol we will use
- Map between names in one protocol to names in another protocol
- Describes how we organize the layers and what options we have available
- This is what we have in the graph, what more do we need?
- Who is connected to whom?
- Who can I reach and if I have proper routing which nodes in the network could I send to?
- Assuming multiple ways to get places, which should I choose?
- Understand the properties of the paths
- This is useful for determining how to route our messages

Terminology

- Relaying
- Process of moving a message hop by hop.
- It has to be sent on somewhere else
- We do this based on the DAG tables
- They say what we do with a particular message in a protocol for a particular destination
- Can also be done by forwarding (IP)
- Can be done by switching (which uses more hardware)
- Routing
- NOT the same as relaying
- It is setting up all the information you need to make the relaying decision
- These sets of tables say here is what I do with it and there is how I relay it to another node.

- All these things will be involved in a routing decision.
- Path is the overall choice and we need to have tables that decide what to do and possibly through 8 different nodes in the middle

More terminology

- Two approaches
- Link state
- Distance vector
- Both depend on link state
- You are going to make your decision based on the state of the link.
- You are always going to be calculating distance vectors to decide if this is a good link to use or NOT a good link to use.
- You also always need to compute distance metrics.
- You use a bit of both each time you calculate one of these two metrics

How do we collect that info?

- Every node that has the potential to relay messages will have to have some kind of information that lets them choose the routing.
- Need some simple default to send messages out and decide where to send them.
- Only a source but I also have some choices.
- How do I get information?
- Ask your neighbors!
- Ask them about where they are.
- Works if you only need to talk to neighbors but won't get you far.
- Flooding
- **Make sure what you are sending gets to EVERY node in the network**
- You ask each and every receiver to send the node down to all their other connections,
- As you can see, this will eventually reach everyone.

What do we flood?

- The topology
- We know who we are and who we connect to.
- We know who our neighbors are
- Perhaps, a little more information can be provided about how you connect with them.
- Send it to everyone who is your neighbor, and they send it to their neighbors
- Once every knows that, we have a comprehensive idea of who can connect to what and figure out paths to get to people who build our routing tables.
- Our decisions need to be decided by who we can get to.

When do we flood?

- When we first join the network, that is probably a reasonable time to do this.

- We can do some relaying potentially for you, so here is my link state.
- Having just joined the network, you presume no one knows about you.
- You can change your routes to certain places based on your information.
- We can also try to compute a route and we won't proactively figure out ahead of time.
- If we need to route to someone, we could do some flooding to know who to reach.
- We have to send a message to node X and hope that other people are also flooding out information.

Goal

- Information to go through the DAG and go step by step to translate names as we go.
- Get messages off our node and eventually it will reach our destination.
- What is the next layer we want to go to.
- Go down to physical 802.11 protocol and eventually reach the wire network protocol.
- Which do we choose?
- We need a way of choosing a way to go down through our alternatives.
- If there are proxies, where multiple people choose a single thing for the destination.
- Be aware that the nodes in the middle are serving as proxies here.
- Result in different paths through the network and make a choice based on information and the context of what we are going in.
- Put it into our tables and make use of it to get to our destination.

Optimization

- Reachability is the ultimate goal of the Internet
- Given multiple alternatives, we want to optimize our routing and make it cheaper than the other choices.
- Faster bandwidth, more reliable, etc.
- Get there in the best possible way, rather than not getting there are all.
- If we aren't careful in the network, we will run into loops.
- **The network (Internet) is NOT acyclic!**
- We can go in loops!
- Avoiding routing loops is very important when talking about relaying and routing in a network, and eventually, the message never gets there.
- Particularly bad because without another mechanism, then not only does the message NOT get to the destination, but also resources are used in a useless way.

Information requirements

- Node name
- Referring to node X
- Who are we?
- Who are our neighbors?

- Assuming multiple links attached up to our machine and we have five links going out, we have to name each of our links
- We need to figure out each particular destination and it should go out to link 3
- We need to identify properties of the link with the name of the link
- Cost is a general term
- Refer to ones related to delay
- Perhaps if we take a link and we are going via some lengthy path, how long will it take me to get to node X?
- Typically, costs are expressed in terms of delay units.

Key algorithms

- Basic flooding
- We tell everyone everything
- Distance vector
- Links

Basic flooding

- If you run the algorithm correctly, it eventually cascades down to everyone.
- Every node in the channel has to be prepared to receive information and be ready to send it to all their neighbors!
- Relay it out to every other interface.
- Most flooding algorithms know if it came in on interface A, everyone else in the hierarchy above you already received it.
- There is an important question here about this distributed algorithm.
- That question is when are we done?
- We don't want algorithms run forever!
- We want whoever is most interested to know when it is done.
- The guy who has to get the result knows the algorithm is done!
- Message comes in on interface A and the guy who started the flood needs to know when the flood is complete.

Goals of flooding for routing

- Get to everybody reliably and get a copy of the information
- Further, someone sent the flood out for some reason and routes need to be able to be communicated.
- Maybe that's the guy who started the flood, maybe it's someone else.
- If we are sending out a request, and please tell me about your links, everybody should send out information and we need to know when we have gotten all our responses.
- I would like to know when I get a complete set of information and start building out my routing tables.
- I want to minimize cost
- True flooding is NOT cheap but we want to minimize whatever cost we have to pay.

- This assumes connectivity
- Obviously you can’t reach nodes you aren’t connected to.

Limiting the flood
- Make sure we don’t send an unlimited # of messages
- Try to track messages OR state of the nodes

Hopcount in messages
- Inject messages into network with our own link state and we want to have some control over this.
- Look at hop count and keep flooding
- Decrement the hop count by 1 and it will go out to all my neighbors
- Sooner or later, the node will see the hop count is 1 and they will decide NOT to flood anymore.
- Eventually, this leads us to stop flooding.
- This only works if you know the # of levels of the tree
- If we want everyone to know, we need everyone to hear about the message.
- We cannot have the hop count go to 0 without everyone knowing.
- Going from a flood starting at A and we want it to reach X, with n hops, we need to get the message from A to X
- As long as the longest path is less than n hops, we will reach X
- Q. Does the hop count measure how far away it is?
- A. Each will decrement and many nodes will get the same hop count. We are hoping that using a hop count of say 32, we can reach all the nodes in the Internet
- If I knew that it was fewer than 32 hops to get to me, I would already have had a fair amount of information and the presumption is that we don’t know that.
- The best you can do is guess
- Set it to ridiculously high (very wasteful)
- No flood would disappear until the end.
- People would hear that this message would have been flooded millions of times and NO new info is provided
- The cost would be immense
- If it’s too big, it is expensive, if it’s too small, it doesn’t reach everyone

Checkbox at nodes
- Check if I have received the message and use a boolean variable
- Either the box is NOT checked or it is checked to determine if it has heard it’s message before.
- We don’t need to flood to all our neighbors and we can assume that we send out the flood, don’t worry about the hop count business
- Don’t decrement hop counts, but the message has to check off a box and we have to remember that.
- Look up the list of check boxes and say if we have received this one before.
- This doesn’t guarantee that each node receives a message just once

- Each link won't be used more than once and eventually it will have gone out to all possible links that are reachable.
- Assuming connectivity and we are looking these checkboxes up in the proper way, it will work!
- How do we know when it is done?

Controlled Flooding
- Algorithms to have better control over our flooding
- Make sure we don't have multiple copies of the message, make sure everybody gets the message, and make sure we know when to stop
- Chang's Echo algorithm
- Send a copy on every interface except A because A already knows it.
- When I am done, send a message back to the incoming interface once I know that everyone I sent the message to has received the message

A picture of Echo
- Lecture 14, Page 26
- Look at the diagram
- The flood will send the message on and look around and see which interfaces I have gotten to.
- These messages can send these messages to each other and they have no way of knowing we have received this message to another path.
- The content of those messages directly and the nodes and the top could be sent at the top, bottom, and middle.
- People will get multiple copies inevitably with this algorithm.
- I have heard the message and whichever came in first will choose the incoming interface.
- Keep track of which interfaces you have heard.
- Tell it to everyone EXCEPT the source of information where you heard it from.
- We have gotten something on this particular link and we should go to this particular interface except the one it comes in on.
- The flood cannot flood it any further and the only people he can talk to can hear this message.
- Bidirectional links here.
- We are going to send the message back on the one interface that I marked.
- Send it out and shown in blue nodes in the diagram.
- This is the 1st guy he has heard it from and this guy over here is going to talk about the Marked interface and have I heard this flooding message from both of my other interfaces?
- Yes!
- We now know for a fact that each neighbor has heard this message.
- We don't need to participate any further in this flood and we need to send back the message.

- At this point, that guy knows he is done and he had heard from all of his neighbors and the others were NOT marked.
- He too was able to understand that he was done with his flood.
- Two points simultaneously done with the flood and we have to remember that the other points may NOT be done as well.
- Points do NOT necessarily know the global state of the path.
- Did it flood only to the top two nodes back and this is when the node above turns blue.
- When it turned blue for all the other neighbors that reached, we need to indicate that we got these incoming links and flooded to a couple of links
- Flooded to guys at the bottom because he marks it as interface of guy who didn't flood on.
- Get a copy from him and we only have three interfaces.
- Everybody who I connect to has received this message and also told me they received the message
- Send it back and this is what the blue arrow means
- Q. Are the blue and green messages the same?
- A. Yes! They go at different points in the algorithm and we decide when they send the message.
- Q. Whenever any nodes hear the message, does it turn blue?
- A. Yes! It needs to send its final message and send it back to the interface it marked.
- The whole goal is to flood it out to everyone and have the original sender know that the flood finished.
- The blue node just sent his blue message to the next node over in the middle
- He heard about the original flood and uses a star to denote it as a marked link.
- All the neighbors have echoed back and he can say I am done and send it to the marked link and say I am done.
- This guy has 3 links for the original flooder and he can hear from the original flood.
- This link is still marked and he doesn't have to hear from the guy at the top.
- He has heard it from all three of his interfaces, turn blue, and send it back to the original flooder.
- I flooded out on two links and echoed out on two links
- Everyone who is reachable has heard my message.
- If I would like to send a message to him, send it to the guy at the top and the guy at the bottom
- Gradually, this information has gone out.
- We could be trying to flood out in the opposite direction and gather up information in that way.
- Algorithm completes, and we talk about this flooding algorithm in the context of routing.
- This is a flooding algorithm, NOT a routing algorithm.

- This assumes EVERYONE followed the algorithm.

Properties of the echo algorithm

- Assumes
- Bidirectional links
- Connected graphs
- Trivial but we have to remember that networks are dynamic
- Links go up or down
- In that case, the algorithm might not work perfectly.
- What happens in the case of failures?
- If there are E/2 links, the message will go outbound on each link and inbound on each link exactly once.
- Once forward, once back.
- We know how much this algorithm costs and it will cost 2X messages.
- You don't have to have hop counts and you don't have to come out to anything.
- At a billion - 1, you don't have to do any counting.
- This is a scalable amount of information at each node.

What did all that get us?

- Flooding
- What do we do with that capability?
- **Use two phased flooding**
- This message goes out to all the phases

Two phase flooding

- Phase 1
- Build up an entire list of everyone who heard
- All the messages go back to the original source and build up this big old list of everyone who heard that message
- If we threw about more information about the starred links, we can have complete information at the initiator and throw out information about the properties of the links
- Phase 2
- We have built a complete build with comprehensive information
- Be a good neighbor and send out information on the map to everyone.

What map do we flood?

- A map with a billion nodes
- We have a big ass map
- Get to the shortest path
- Assuming we want to route by the shortest path, how do we get that.

Link state

- If we were doing this entire flooding, we were going to use the Echo algorithm

- Get the shortest path using Dijkstra's algorithm

Dijkstra's algorithm

- Not a distributed algorithm!
- Runs entirely from one node
- Does NOT gather information from anywhere
- Derives useful stuff from the information it has
- For every single destination in this map, what is the shortest path to get from source to destination?
- Every source and destination pair needs the information we have just talked about
- Grab big set of information and grab one node out of it, this is our current node
- Usually you choose yourself.
- The node needs to know for myself the cost to getting to any other node.
- Are there any links going to outgoing nodes?
- Keep a couple of sets of nodes in addition to the invested node.
- What set do other nodes belong to.
- Could belong to the UNVISITED set or the FRONTIER set
- Everybody starts in the UNVISITED set originally.
- The cost of a link is whatever the CURRENT cost is + the cost of the new link
- If the node is already in the FRONTIER, we put a cost on it and we need to compute a new cost.
- Somebody else needs to compute that node and is my way cheaper than that of the other guy
- Is this cheaper than what I have already gotten?
- When I am done with all the nodes, mark this node as VISITED and keep track of what I computed for it.
- Go to my FRONTIER set and out of all the nodes on the FRONTIER set, choose the one with the smallest cost.
- One by one, they get added to the FRONTIER set and everyone will get removed from the FRONTIER set.
- This algorithm is NOT complete until I consider every connected node in the graph.
- Presume we got this information by flooding in the first place.
- We will eventually remove everyone from the FRONTIER/UNVISITED set and we will have a cost associated with each node.

Dijkstra' Algorithm at work

- Lecture 14, Page 47
- Choose one node at a current node and we want to calculate our path
- Can we move any of the ones that connect to the current node and those can go into the Frontier set
- Computation of everyone in the frontier set and add a link value to that frontier node

- Compare that with the existing cost of the frontier nodes
- Those are considered infinity
- Take unvisited node and put it into the Frontier set
- What we put into the Frontier set will take the current cost and add the link that takes it to the node in the Frontier set
- Goes in based on the original 1st set
- Bottom node's cost dropped at this step
- Notice that node's costs never go up, they only stay the same or go down
- We ignore higher costs because we know there are cheaper paths
- We only have one node on the frontier set now and we need to see if he has links to anyone who is NOT on the frontier set
- We put him into the frontier set and say what his value is.
- We have a current node value of 5 and his value should be 7
- This was the only guy with a frontier set and he goes into the visited set
- We check until we have no one in the UNVISITED set and the FRONTIER set
- We know we are done because we do this computation at a single node!
- When he gets to the point where we have removed the current node, he knows he has done the entire graph
- He has a complete set of costs and knows that to get to the node here in the middle, the cheapest way is to go from 0 to the node at the top to going down

Which paths are used?

- Lecture 14, Page 63
- We now know that for node 0, all the paths he should use to get to any other node in the network
- Uses a cut-set from CS 180

What does Dijkstra compute?

- Shortest path between two nodes
- Shortest rooted tree
- This will be the lowest cost tree

Dijkstra: pros and cons

- Pros
- Simple to implement
- Broadcast to everyone
- Everyone runs the same algorithm
- Everyone ends up with a set of routes
- Cons
- Requires broadcast flooding
- Not everyone might compute the same tree
- It isn't that cheap if we are referring to a billion paths
- **This would be a high computational expense for something most people do NOT need**

- End up with a lot of stuff that is quite similar without getting any notable benefit from it.

Distance vector

- Do not necessarily work by flooding
- Bellman-Ford algorithm will be covered

Basic distance vector algorithm

- Tell your neighbors that you can reach the following people and here is my cost for each
- Do a transitive closure for each (analogous to CS 143 closure)
- Not everybody gets everything

Example of Bellman-Ford

- Lecture 14, Page 69
- Keep a table at each node and you know how to get to each item
- We have infinite costs to get to D and E
- Similarly, everyone else has a similar kind of thing
- We don't know how to get to E and the only thing we know how to get to is the one that he connects to someone else
- Set up the table using purely local knowledge
- These tables aren't correct yet, so we need to run the Bellman-Ford algorithm to get them correct
- We aren't going to build up a huge table, but rather we need to do a transitive closure
- A could go to itself at 0 cost, to B at 1 cost, to C at 4 cost
- A could actually get to C at 3 cost using an alternate route!
- We have found a better way to get to C, so we will now route it through B

A look at A

- Lecture 14, Page 73
- Observe that if you go through B, there are places you can get to cheaper than if you go in a different way
- B is the appropriate route here and we can update our tables

Bellman-Ford

- Converges over time
- Keep track of each low entry and the link for each entry
- Keep track of each outgoing link that we use
- What is my path to C?
- I know it starts with B, so I will send it to B and have B worry about it from there
- Each step is O(N)
- You don't need a complete and perfect routing table immediately, but it just keeps getting better and better
- Dijkstra's won't tell you the best path until the very end

Bellman-Ford Pros and Cons

- Pros
- Fewer and smaller messages
- We only have to send out what we changed
- If we didn’t change something, we don’t have to tell them
- The only things that change are the things that changed
- In order to find out more information, this link’s cost changes
- Cons
- Decentralized
- Errors at various times or we can make malicious attacks
- People can lie and we will believe them even though it won’t get there very cheaply
- NSA does this crap!
- Link failure has a slower convergence

Link state vs. distance vector

- Link state lets everyone see the entire graph
- Provides you with a complete path because you know exactly which path you want to take.
- Always floods
- Large local table
- O(N^2) computation
- Distance vector tends to affect things locally and does NOT propagate beyond its local area
- We don’t care if your link changes further properties
- Smaller tables of O(N) computation
- Reacts faster to certain types of changes
- Only provides the next hop (rather than the complete path)
- No global optimization

Other algorithms

- Hierarchical routing
- Go up when you don’t know
- Towards the root
- Go down based on what you know
- Move through the graph of nodes
- How Internet routing is done

Geographic

- Plane of whatever we are working with.
- Use geometry to get you there
- You have to map all the nodes and all the geography that we are using

Landmark

- Partially geographic and partially hierarchical

- Scatter landmarks through the geography of my system
- Know how to get to nodes close to them in their geography.
- This is used in cases of wireless networks and mobile computing.

Who uses what?

- Link state (Dijkstra)
- OSPF
- IS-IS
- Distance vector (Bellman-Ford)
- RIP
- BGP
- EIGRP
- We don't know the entire topology of the Internet, but we know the entire set of nodes we can specify and put it in a path vector.

Split horizon

- DV algorithms converge slowly
- They only keep track of the cost
- When you lose a link, it may take quite a while to figure out the other link you should use instead.
- Run the entire protocol again and take into account that the entire link got lost.
- Send info as bad (poison reverse)
- Very bad place to go and you have to recalculate

Loop avoidance

- **Try to avoid loops as much as possible**
- You are likely to get the loop forever
- Some of the things dynamically might run into a loop
- What will we do then?
- Try to find the loops and once we have found them, we will remove the loop and make sure there is no longer a loop.
- Probably I won't have loops in my routing and no message should take more than 32 hops.
- Loop count keeps getting decremented and eventually we will drop the message.
- Every message I send will get dropped since we have hop counts that will go down to 0.
- Steady state loop implies we will have unreachability problems
- Users shouting to their system administrator is an issue
- Ensure the message isn't wasting resources and getting nowhere.

Cost metrics

- We need a metric to describe path
- Lowest propagation delay?
- How long it takes a bit to get from here to there.

- How much information can we should down for this link.
- Lowest price?
- Some fixed dollar price

How to compose cost
- Various equations
- Highest possible capacity
- For the path we are considering, what is the minimum capacity of any link
- Take the delay for each link and that is the overall cost of the path.

Algorithm performance
- Time
- How long does it take to get to the place where nothing changes
- What happens if a new node comes in?
- Bandwidth
- Passing out a bunch of information and this is overhead on my network.
- Actually moving bits around and use the network to control the network.
- Use fewer messages for smaller messages.
- For everyone who wants to run the component of the algorithm, how much do they have to do it?
- Don't have compute power running all these algorithms

Solutions to performance
- Use simple topologies
- Original Ethernet
- Token rings
- Wireless LAN
- Whoever is listening can hear it
- These won't help us with scale
- Compartmentalize
- Break down graph into regions and try to route within regions
- Just worry about getting from region to region

Compartmentalization and Internet Routing
- Name each of the subgraphs and use an "Autonomous system" (AS)
- OSPF is typical but there are others
- In order to deal with things NOT in your subgraph, you can use BGP

BGP and autonomous systems
- Does NOT route between nodes
- Routes between autonomous systems aka collections of nodes
- Phone services
- Verizon, AT&T, UCLA
- Each autonomous system (AS) will contain multiple routers
- BGP will tell you how to get from AS to AS to AS, but it doesn't tell you what is inside each AS

Graphically,

- Lecture 14, Page 93
- Consists of five AS's and each cloud will have a whole lot of stuff with individual routing nodes
- We can do the AS and these nodes will connect to each other using BGP
- These links wouldn't exist and you are just routing from AS to AS to AS
- For the destination we are trying to get through, this is the path to the next AS we should take
- Route from AS47, AS7, to AS55
- Use any algorithm, routes internally as it pleases
- BGP knows nothing about that detailed route that is internal to AS7

BGP and policy-based routing

- Tens of thousands of AS on the Internet
- A lot simpler than going from node to node
- Networks can move data in their own sets of routers
- Control their own internal thing
- Core of the Internet that connect a whole lot of stuff are run by businesses
- Businesses want to negotiate agreements between partners and control the amount of traffic and routing decisions
- BGP routing decisions express policies that describe business goals
- BGP routing is made purely on the basis of shortest path and all these things
- Made primarily on the basis of which contract that people are connected to.
- Use a protocol that passes information around at the AS level
- We have some local policy and use that policy to determine if the new option is better than the one I was using before.
- Everybody gets to run their own policy and they don't need to tell anyone else what the policy is.
- Complex and difficult to figure out what the hell is going on.
- We need to get to choose the way to get decisions of how to route things.
- We need to know why though!

Building BGP paths

- IP prefix is what it is called
- 15.33.124.0/24
- Control these nodes and these are addresses belonging to machines I have in my AS
- If you want to get traffic from that prefix, give it to me and I can deliver that traffic
- If I choose to, the neighbors can tell my neighbors that this is a way to get to that prefix.
- To get to that AS, you go through me.

For example,
- Lecture 14, Page 96
- His AS sends out a message saying that we can get to that prefix!
- The neighbors know that and they now consult their local policy.
- Tell me neighbor that if you want to get to that prefix, I'll send it to that neighbor

Some BGP implications
- No centralized decisions
- No central authority or single algorithm
- ASes don't even know every single possible choices
- Decisions are changeable dynamically
- You can withdraw an advertisement and it is purely up to the AS
- Based on business arrangements
- Only a partial description of the routes
- Makes a promise that if you deliver a message to me, you will get that next AS and it will call out the door to the next AS

W 8 R Lec 2-25-16
- Routing through the network and spoke about a # of possible algorithms to use.
- A little bit more to say about BGP

For example,
- Within the system, they decide on some method to route internally.
- In BGP, this is standardized to routed between autonomous systems.
- Internally, you can use whatever method you want, but you could also do something else.
- Each of these things will have fewer nodes than the autonomous systems on the Internet.
- They wouldn't work for the entire Internet
- Entirely up to the people who run the AS
- BGP is used for Internet routing tot get information around in the Internet
- Used universally throughout the Internet so let's go through a few more examples.
- Instead of a few tens of thousands of systems, we only have 5
- The links represent NOT just physical links but also some kind of agreement under certain circumstances
- For practical purposes, they are NOT there
- Autonomous systems may not link a one-to-one correspondence leading from one system to another.
- Autonomous system level is a higher conceptual level than talking about a router and a link.
- Forwarding is done simply to replace the router and the packet comes in the link

- Send it out on some other link
- At the higher conceptual level, we have to decide which path to go through an autonomous system
- Left-side, we have a particular local area network.
- Prefix of 15.33.124.0/24
- We would like everyone else to send packets to that range
- We can advertise information on saying how to get to me.
- The only way to get to it is through AS 47
- Whatever else you do will inevitably go through AS 47
- Afterwards, you have some choices to make
- AS 47 says okay I take care of this sub-network
- Business level contract between AS workers and network administrators
- We need to make sure everyone can reach this sub-network through this new AS
- This causes BGP to leap into action and you can get to this particular address range through AS 47
- 15.33.124.0/24 -> AS47
- Get your packets to this particular sub network
- Goes out to AS 91 and AS 7
- BGP works on the basis of policy
- Every AS gets to decide what it does with advertisements on its own.
- Works on the basis of advertisements it has got.
- AS 47 will take care of the rest and but we don't now the inner details of AS 47
- Not the problem of BGP and AS 91

Q. Moving through Dijkstra's algorithm, is there any advertisements of bandwidth that is available? How does this work with AS happening over that?
A. Each AS knows about each of the physical links it has because there is a cable leading into their building and they know exactly what the cable does.

- That much they know. They do NOT necessarily know much about AS 47 as well as the sub-network there.
- What they do is they don't magically create cables and AS relations are contractual.
- Lawyers show up for each of the two sides and one of the things they talk about is how much bandwidth I can send through you per minute.
- Are there limits to each kind of transit I can take?
- The people on AS 47 know what is happening and know exactly what they got since they run it.
- The lawyers can argue and debate about the advertisement of bandwidth.
- We don't have a global view of how much traffic moving through the network in any sense.
- It is all purely local

Q. With Dijkstra, we kept echoing. Doesn't this NOT give a local look?
A. This isn't using Dijkstra's. They are NOT going to be listening and to run Dijkstra's algorithm, you need every node in the network.

- They will participate, but they won't play your game.

- They will use BGP, not Echo and not Dijkstra's.
- You need to keep running things every time it changes.
- Not feasible and has to deal with minor changes in the network.
- Let's go through a few more steps and then we will talk about that.
- Those two AS's will remember all their advertisements and in terms of giving someone else there ad, they will remember it and tuck it away into their database.
- This is information I might want to use in the future
- They would know how to get to this particular prefix.
- One thing to decide is for a particular prefix, do I wish to tell my neighbors that I can send a packet to that prefix?
- AS 91 will be willing to carry the packets and it will send an advertisement to its neighbor.
- This advertisement will look a lot like the original advertisement but with an additional AS 91
- AS 158 will now know how to get there from AS 91
- AS 7 can do the same thing, but it can also say it doesn't want to send traffic through its network.
- He may send packets to me, but I may not send packets to him.
- Don't say to any other network to send traffic to another network.
- If you don't want to carry the traffic, **do NOT advertise**!
- I could carry traffic to this prefix and everyone could make a decision
- AS 55 could say that it knows something else NOT in BGP advertisements, it knows something from AS 7 but it won't do it (or it's not supposed to)
- You have to obey the advertisement
- Believe the other person and do NOT use external knowledge
- Just because AS 7 has information doesn't mean you should use router tables and set things up in that way.
- I heard an advertisement of AS 158 going to that prefix, which will then send them on the path that was advertised.
- Believe the advertisement and work in the context of that advertisement.
- How do we know if everyone is obeying BGP?
- We cannot find out for sure!
- Can we deliver packets directly to the network from there?
- Everyone would believe him!
- There would be another way to get to the network, but this isn't true.
- The BGP protocol itself has nothing to prevent that.
- NOT a single thing.
- This occasionally causes a little bit of trouble
- Some years ago, there was controversy of Youtube that some Islamic groups felt was blasphemous
- Certain countries where Islamic fundamentalists had influence
- People who lived in Pakistan created a route to Youtube
- Sending packets to Pakistan to get to Youtube was a bad idea!

- We don't want our people to see these blasphemous videos, so we will drop packets form Youtube.
- Everyone getting Pakistan's advertisement was no longer able to go to Youtube, so the packets would get dropped.
- Because they advertised from BGP, they lied about their ability to reach Youtube to a large portion of the world.
- China can advertise a fast way of sending packets from LA to NY through Beijing, but people notices this happens and they fix it up.
- Network says "Oh what the hell, it is the Chinese again."
- One machine figures out what to do with the advertisements and they say that is junk
- This things can also happen purely by accident and it is NOT an act of malice or an attempt to spy on anyone.
- When you see an advertisement, check against the table
- Authenticated your message is more effective.
- CS 136 teaches message authentication
- This is basically how we move packets in the Internet.

Some BGP implications

- No centralized authority
- Moreover, there is NOT only a single entity that does this, but there is also NO single algorithm that does this.
- Here is how you distribute things and here is how you create new advertisements.
- This is a matter of your local policy.
- It may very well be the case that the AS does not know all the possible routes
- Let's say the AS hears about two possible routes
- AS will look at factors like path lengths through AS and determine if X is a better path than Y
- Go through me and I will get you through this destination and NOT advertise why.
- NOT done through a people level
- People design a policy that is implemented by an algorithm or an input to an algorithm
- Chunks through the advertisement and this goes out through local routers through all the AS and this will go through the forwarding table
- Build up an advertisement and add an AS to the end of the path and send it off to a place where I want to send it to.

Q. If an AS knows two paths to a final destination, will it every split a disproportionate amount i.e. 80% to A and 20% to B?
A. It can do that, it allocates based on the forwarding table

- It would be up to the AS in the middle and we can design the split of this traffic
- NOT something downstream to two AS's can deal with this.
- They would choose one or two AS and pass this along.

- They sure hope that when you get to a particular destination, that AS can get to that destination and they will end up in the right place.
- AS will do whatever it wants and both go to the same destination but going through different AS will affect this.
- Neighbors say great and we will set up the AS advertisement and it will go to the AS in question.
- Supposed to pick up the new X advertisement, but they can simply start routing through the other path without taking the original.

Q. How do I know the change going around me?
A. Assuming no congestion of the Internet, you probably wouldn't notice because you are still moving packets at such a speed that you wouldn’t notice that.

- They are dynamically changeable decisions made here
- Can change its decision anytime it feels like it.
- Told you about this route, it isn’t good, don’t send me packets this way anymore.
- They can choose to advertise it now if they still feel like it.
- As long as it doesn’t change their reachability, there is NO responsibility to change anything internally.
- They just do it!
- How we route things through the Internet is NOT based through physical connectivity; it is also based on business arrangements.
- Since we aren’t telling everyone everything, NO ONE knows all possible routes thorough the Internet!
- A bunch of people running BGP agree to tell you about every advertisement I hear.
- Those all go through a database called the route views database and it has a huge connection from quite a few autonomous system that gives information about routing through the Internet.
- Gives a better picture of what is going on and NOT used in a dynamic way.
- Used by people writing a research paper about the length of paths through the Internet.
- Very important because this is how routing is done through the Internet!

Backups and then some

- Various things are done in the routing infrastructure to deal with issues of failures and alternatives.
- Two links that go from Point A to Point B
- In case of failure, I have the second link and I might have to reconfigure the ends of the two links.
- Can be set up so it is quite automatic.
- Get more through from src to dest
- Another common thing is having alternate paths
- If there is another one, they can quickly switch to the other one.
- Bad to be too aggressive to switch things based on parent failures

- You can get a dynamic condition where the route fails and then it is good again or vice versa
- Spreading this info out on the network takes a certain amount of time.
- You might get into bad conditions where you temporarily cannot get from Point A to Point B simply because people change their routing and it isn't consistent
- Not too good an idea to perform **route flapping**
- User at A, user at B, user at A, user at B, …
- Wait a little while to take action to resolve user switching
- If a link fails, we won't instantaneously change to another path
- This leads to inconsistency, so for a little while, we might for a network to settle down
- Can be a little flaky
- When it is a little flaky, it can fail for a short period of time.
- Don't want to get into a situation where you keep changing your routes.
- Could result into transient inconsistencies and we don't want this happening too often.

Summary
- Variations of transitive closure
- Link state protocols where we pass back information and choose info based on the state of those links
- How do we compose several different costs and deal with transients.
- Usually done through a centralized computation or distance vector

Q. Is it just a timeout?
A. Combination of timeout and you may want to change your path. If it looks in 5 more microseconds, you may NOT switch back. It may determine on how well you understand your link failures.

Q.If you don't want to be tracked, how does that work?
A. Tracking isn't that simple! You might use estimates and you do NOT necessarily know except by observing a packet's actual path of how it is going. Very few parties who can do that but the NSA is one of them.
- If I don't care about how I got there, but I care who you are, I will look at your IP address!
- You don't have to tell the truth of your IP address, you can create a bogus IP address and the guy who gets that will think it came form the bogus IP address. You won't get to see your responses.
- Talking about onion routing and Tor
- Multiple hops in the network in different ways.
- DAG and recursion helps you understand Tor in many ways.

Optimizing Layers
Where are we at?
- By transitive closure, we can go from the DAG description of the layering and understand movements through that graph through recursion as you go down the layers.

- We can understand how to relay from one place to another.
- Routing algorithms to set up our relaying table.
- Sending an HTTP request from A to E

For example,

- Lecture 15, Page 3
- Get off of A and use name translation
- Go from 802.11 -> physical
- Do this through the name translation and relay through C
- C now uses its DAG
- This will be an IP message and we can take the IP address and translate it.
- Go down to the physical layer which goes through the ATM
- D now has his very own DAG and this message coming in as an IP <- ATM
- When we go from ATM to the physical layer, it will lead to the IP
- Go through a translation through your table and go from D to E
- This involves actually sending the bits of the packet through the Internet, and I will now say what I have gotten.
- It is a TCP packet and it is an HTTP packet.
- Deliver the message to my web server
- This is what is going on as I go through the network.

Where are we and where next?

- Use networking to go from one place to another.
- This is the most important things from the Internet's perspective
- You need a way of getting from Point A to Point B
- We need to provide the basic connectivity!
- There are other things that are important too!
- Performance
- Reliability
- The website needs to be reached
- How do we optimize to go beyond simple things such as networking and relaying?
- Give some background on this
- Deficiencies
- Emulation

Optimizations we can make
Networks and optimizations

- You could have one layer used everywhere and just one layer
- Expensive!
- There are all kinds of interesting optimizations that can make everything work better.
- This could be done at things in the low level and the question is where to optimize.

- There is a fundamental different location and we could be optimizing in a layer
- We could optimize across a communication path and this could be optimized.
- Do we optimize for some layer or some connection?

Intra-layer vs. intra-communication?

- We might have some shared context and someone at the other end would have to know what we did to work properly with whatever optimizations we performed
- Either one can support optimizations
- Explicit coordination
- If you are over here and he is over there, we are going to explicitly send some information to indicate what we did in terms of optimization.
- Saying I happened to know to send something from IP to 802.11
- If I understand that, when I have these two layers capable of being stacked, it will be in the way layers connect to each other.

If layers, which one?

- Optimization can occur at many possible layers.
- However, they are more likely to have optimizations at certain layers.
- Optimizations can interact across layers
- If you ever look at TCP variants, people have worked out many versions that behave slightly differently
- Involves optimizations down at the Link layer as well at the TCP layer
- Isn't the IP layer supposed to provide the narrow waist?
- Sometimes it makes sense to do optimizations across any layer.

Living at a layer

- Optimizations can be run at any possible layer
- They will differ by the set of optimizations at several layers
- Some optimizations may occur more often at certain layers
- The physical layer is where most of the errors occur
- Problems with electric bursts or the wire
- This occurs quite often!
- Assuming the signals are gone and the link layers properly determine the signal, they are NOT too common.
- **If you don't correct errors at the lower levels, they propagate upwards to the higher layers.**
- It makes sense to optimize for the problem at a lower level.
- TCP provides ordered delivery of messages; UDP doesn't
- Ordering optimization usually happens only at the transport layer and never below the transport layer
- At an application layer, there are layers of optimizations that TCP hasn't given you, so if it is NOT Provided, you have to get it from UDP

What connection?

- We are optimizing across the connection
- Across the entire path at which things are going
- The guy at the sending end has to share an end at the receiving end
- Hard state vs soft state
- State information that has to be correct in order to get correct results.
- You can get better results in some sense even if it isn't
- Caching is a soft state type of optimization
- You can fall back on wherever the information lives permanently.
- Optimization is NOT required for correctness
- Belongs to a state in order to achieve the optimization across a connection
- Not everything is associated with an explicit, stageful connection
- You may still want some form of optimization
- DNS caching
- Every DNS request is its own separate connection
- No history maintained and now he is asking for this thing.
- DNS is NOT stateful in that sense
- Getting a reasonable performance requires caching

The End-to-End principle

- Networking principle depending on Internet
- Requires networking and doing the application specific stuff at the endpoints, NOT in the middle of the network
- What goes in the middle of the network?
- The kind of thing that every application using the network needs.
- Need stuff that goes from Point A to Point B
- Some people want reliable delivery of the messages
- These shouldn't be put in the middle of the network; they should be put at the endpoints!
- Not a hard and fast rule
- The end-to-end principle says we shouldn't do this
- Think about doing an optimization at some point in the network
- Do it at the endpoints but consider it in the middle
- Highly influential principle
- Many optimizations are particular to a network layer
- Should occur a the endpoints of that layer!
- The 802.11 network layer can go from here to there across an 802.11 link
- Optimizations need to perform the optimization at the endpoints of the HTTP layer and NOT in the nodes in between
- The endpoints are NOT the endpoints of all the layers in question

For example,

- Lecture 15, Page 20
- Go from A to C to D to E
- A to E are the endpoints

- The fact you are going from C to D in the middle isn't relative
- IP -> ATM
- ATM -> physical
- Destination says you should do ATM and go down to the ATM physical layer and send it out to that link
- ATM message should go up to the IP layer and figure out it has to go to E
- D -> E is NOT Ethernet and should work
- For the purpose of an ATM optimization, C and D are the endpoints.
- What if there was another node inside in the middle?
- Where should you make the optimization?

Dr. Kleinrock is badass, he would sit up until 3 AM and move things from the animation!

Now let's explore how to optimize:
- Have some networking system that has these characteristics, can we make what we've got look like what we want?

Deficiencies
- We want integrity
- We don't want our data is tampered with.
- This is the message that is sent through the guy
- Privacy
- No one but the intended receiver should receive the content of the message

Why do you care about deficiencies?
- We cannot share the information in the way we want to send it.
- Combination of getting in the way of networking
- Manage the things we want to do.

Integrity
- Quality of being who or integral
- What you got is what you sent
- Nothing more, nothing less
- Don't split things up
- Whatever information you put out at the source end is what you get bit to bit at the destination end
- Integrity can be compromised for a # of reasons
- Accidental alterations
- This means a symbol is changed due to noise and there is equivocation that we don't know which portion is sent.

Integrity Loss
- Just disappears; gone forever!
- Receiver has no idea he got a message

- Sender can put it in a networking system and assumes that it goes off his node.
- If there are some problems on his own node, it might NOT have gotten sent at all.
- Somewhere along the line, it got lost or dropped.
- Message could have gotten from source to destination and a whole lot of bits have gotten lost.
- No lookup for the next protocol up and it appears to be a bunch of junk.
- Even if not all the bits got corrupted, it becomes a useless message

It’s 2:20 pm, do you know where you are?

- Takes a certain amount of time for a message to get delivered
- Maybe it will come if you wait a little bit longer.
- How do we detect loss?
- After a certain amount of time, I will declare a loss.
- Therefore, it isn’t coming
- You never know when you use time as a basis to see if a loss really occurred

Integrity: Corruption vs loss

- Corruption
- Packets come but have an unrecoverable error
- Loss
- No message arrives and the timer indicates it isn’t coming
- Indistinguishable
- When destination name or source name is corrupted or missing
- Difference
- How much of the message is gone:
- Corruption
- Some of it
- Loss
- All of it

Integrity: Tampering

- Intentionally, change the message in a way to cause an effect different from how he sends the message
- This is what the protocol says they should do
- Change the message in a way NOT expected from the person who sent the message
- Lay the electronic hands on the bit of the message
- Change some of its content or headers and do this during relaying.
- Guy who has access to a medium can overlay a signal and this would be most simple to do for a wireless communication mechanism
- Send noise to change bit values when you can introduce a competing physical signal.

- For many communication media, this isn't as common of a way to tamper with a message.

Integrity: Corruption vs tampering

- Corruption
- Detect it by doing an error check
- Checksum
- Reasonable probability that an error has occurred
- Property of the universe
- Tampering
- Use an integrity check
- Checksum won't work because it is attached to a message
- Not going to help if tampering occurred
- With an attacker who has gotten you message, he will change everything in the message to make it look like an entirely different message was sent.
- Match the new value that he wants.
- Difference
- Intention
- If you don't have reasonably good integrity checks, tampering will create a message that is perfectly plausible and you cannot tell the difference

What about order?

- Order is not a property of the message!
- Not a good thing to happen from the point of view of the connection!
- It is the integrity of the channel that has NOT been preserved for the ordered channel!
- For general communication, we need to see if we are using layers of network protocols
- Particular set of communication protocol needs to account for this

How much integrity protection?

- Hop count issue and we need to discuss issues that we see in routing
- Different tables in different places unless we are pretty careful and we need routing loops.
- That message isn't going anywhere and it will circle around until we do something.
- We need a hop count on the message before we decide that this message cannot be delivered.
- When the hop count cannot be changed, we have a routing loop!
- We cannot relay more than n # of hops, overtime you go from one node to another, hop count will eventually reach 0
- Every hop that the message takes will have slightly different header information
- If we did a checksum, which we started at the beginning
- Content + header will cause the checksum to fail because we change the hop count.

- We won't consider the hop count and the checksum
- Consider fields that might change in this way
- If we have an integrity problem, we won't detect it.
- That is life!
- Other examples of protecting hop count.

Authentication

- Ensures that particular information was created by a particular party
- Integrity component to authentication

Authentication: Name

- Message comes form John and goes to Ben
- Created by whoever sent the message (assuming no corruption)

Authentication: Control

- How do we know if hop count is correct?
- Count the layers
- Expect that by looking at the info, the receiver knows which message to use
- If we are able to authenticate this information based on the control, we can use the proper state machine!
- Purpose is to use the right content and information
- Be absolutely sure that these are as that particular sender wanted it to be.
- A little different from simply providing integrity.
- Provide integrity without providing authenticity.
- What is the difference?
- We know for a fact that the bits we sent are the bits we received.
- We want to make sure they were actually sent by John and NOT anyone else.
- Authentication will require some extra work.
- We often do want authentication of a message because we will do something on the bass of that message.
- Some people we trust more than we trust others
- We will do things that we will NOT do if we got that message from an untrusted sender
- If we do sensitive things, we want it only done for known people
- Authentication comes in very handy here.

Why bother with all three?

- Why not have three different parties sign it?
- Go up layers and down layers along with up and down relays
- We may want to authenticate what is going on a the other end of the relay.
- The party who represents the party at the source end and a bunch of parties that work at the source end.

- Have it signed by the overall party and have it signed to many people who are at the source end.

Why?

- Some of the content will come from the top of the stack
- We can forward a message from someone and this can authenticate for someone else.
- We may need to authenticate information at the bottom, and we may be able to authenticate this in multiple places
- We want to NOT provide an authentication check and we want to try to attest to things we know that are NOT supposed to change.
- Change the hop count and the IP
- Hard to make authenticity guarantees.
- Probably won't talk about IP seconding in this class (136 material)
- How do we authenticate that message?

Privacy

- Throwing info out across a network and we are likely concerned if we have maintained privacy using the shared resource.
- Privacy relayed to control
- Info related to content
- Hiding info from everyone except the intended receiver
- People who are in the shared network who observe the message going by cannot know who sent the message
- We want to hide the fact the message was sent if possible
- Anything we don't send across the network anything that can be decrypted easily

Privacy: Identifier

- Hide the endpoints
- We don't want anyone in the network to know except the sender and receiver
- If you have heard about discussion the governments snooping metadata, this is part of the metadata we are talking about.
- Who talked to who at what time
- Can be derived if the rest of the message is unreadable
- This is very VALUABLE information that we don't want other parties to know
- Hiding identifiers is challenging

Privacy: Control

- Don't hide from the receiver
- If he cannot perform the correct instructions, he cannot do what the sender wants him to do
- Any information about control signals that we are sending i.e. metadata will tell someone what parties are doing within their communication

- Attacker is trying to interfere with your communication to compromise your machine
- If he knows info about your state on your FSM on your site, he may be able to attack your machine in a more efficient way
- You do want your receiver to see your content in a particular way and you want everyone else to NOT be able to do that.

Impact of deficiencies

- Different perspectives on what is going on
- Each relay has a perspective with various preferences
- Work within the limits of those perspectives
- Expect each relay to do a certain amount of processing
- Try to minimize the processing
- Ask ourselves if the relay can afford to do that
- Will he NOT be able to pay that cost?
- Other types of deficiencies that are related to performance.

Time

- The network doesn't necessarily work the way we want it to work
- One thing is rate
- How to get form Point A to Point B at any given point in time
- Latency
- Single bit faster than that
- Move a whole lot of bits and it takes quite a while
- Jitter
- The derivative of the latency
- Is the latency constant or does it go up/down
- Better to have constant latency than a changing latency
- NOT good because receiver cannot predict how long it will be for the next message to arrive
- It might take a long time to get there or a short time to get there
- A lot of jitter == significant problems getting your communication to work well.

W 8 Dis 2-26-16
Project 2 Demo

- You have to check the port #
- Program has to use UDP
- If you don't, Seungbae won't grade the rest
- Sender and receiver should have the same file
- diff send_file recv_file
- He is going to upload a copy of this demo procedure on CCLE
- Retransmission, Window, Timer (15%) is the most important step of the Reliable file transfer (60%): If you fail this part, he will skip steps 2b and 2c

15 minutes to demo the Project 2

- Small timer so we don't wait to timeout.

Project 2
Q&A

- Implement window size to transmit 3-5 packets at once
- Timer should start after sending a packet
- One of the timer times out, we should retransmit the packet

TCP Congestion Control Basics

- TCP controls how many packets you should send
- Sender decides the # of packets to transmit by using this situation and the minimum # of receiver side is for flow control
- ACK packet receives window size info from its header
- It should choose the minimum # so it can control both congestion and its flow to the other side
- Sender's actual window size - unACKed segments that haven't received the ACK packets
- Subtracting this value will be the current sender's window size

You cannot implement congestion control if you send them all at once

- You need feedback from the user!

Slow Start

- We are sending a single packet to there receiver side and the window size will be the current window size + 1
- Eventually we can double our window size in slow start pace

Q. If we send two and only get one ACK, then does it go to 3 rather than 4?
A. Yes, if you lost this ACK packet it can only send 3 at a time! The current window size will be 3 and it has to time out, so it will wait for it to time out and then we can do fast retransmit

Congestion Avoidance

- You cannot double your window size forever
- If your window size > threshold value, you will increase your window size at 1 RTT
- Avoid the network congestion
- Whenever you receive an ACK packet, your window size will be + 1
- Here you only increase 1 / window size
- If your current window size is 4 and you receive 4 ACK packets, you can only increase 1 window size
- In congestion avoidance pace, when you receive all ACK packets, you add 1 each time

Impact of Timeouts

- If you lost any packet, maybe your timer times out, in a single timer timeout, your window size should go back to 1 and you should slow start again

- Packet loss indicates network congestion and you should decrease your window size and start from window size 1 again.

- In Project, we will not consider duplicate ACK packets
- Q. Wasn’t fast retransmit something we had to do for congestion control?
- A. It is a feature for extra credit and we have to build a way for sender to send extra packets to the receiver. Sender should recognize that the ACK packet is duplicated. We can build it such that the receiver sends 3 ACK packets for the same sequence.

Fast Recovery - 1 Drop
- Increase window size by 1 whenever you receive a single ACK packet
- You can send a packet # from 15-28 and then the ACK # here is 13 again
- You lost packet # 14 and all of these will have the same ACK #
- When you receive the 3rd duplicate ACK packet, the threshold value is your current window size / 7cx

Problem 1
- Look at handwritten notes
- Sequence # is in units of bytes

W 9 T Lec	3-1-16
Performance Deficiencies
- Time
- Things take too long
- Space
- Can’t get as much into the channel

Time
- There is also the fact that getting to 1 point in a network to another point takes a finite amount of time
- You cannot get there faster than the speed of the channel
- Latency is not always constant for many channels
- Keeps changing (up and down)
- Never goes below the minimum but there will be some **jitter**
- Expects a steady stream of information flowing it at some rate

Rate
- How many messages can you send at one time?
- However long it takes vs how long it takes too send concurrently
- Use the equation to try to fiddle with some of the terms to get a more favorable answer
- Lecture 15, Page 50
- Reduce the amount of time to send a message
- More messages in the channel concurrently means we will do better
- Receiver at the other end with a machine consuming the other messages

- It isn’t the physics of the channel itself but we should change the things handling at the receiver.
- We can again change that bottom term in the # of terms sent concurrently at the bottom.
- We can also say that there is something in the network that isn’t a physical characteristic of the network.
- We aren’t getting as much into the network per unit time.

Flow Control

- Messages arrive at the receiver’s rate
- Avoid overwhelming the receiver
- Avoid using excess storage resources
- Do something at that message and do some higher level computation
- Takes some time
- Don’t overwhelm the receiver; otherwise, he cannot handle our message
- If we store the messages that have been received, we can look at certain messages and catch up.
- If we are trying to make sure we don’t overwhelm our receiver, we have to do something at the sending end to make sure he can handle it.
- Let’s only put messages into the system at a rate we know he is capable of handling
- If he can handle 5 messages per second, we put those 5 messages in
- If we aren’t sure how much he can handle, have the receiver tell me when he is finished with the message.
- In that case, we can send a bunch of messages into the network, but we have to wait for future acknowledgements
- We will be certain his limit of acknowledgements that he can receive
- We need some sort of feedback and know how many messages at a time the receiver can receive.
- How do we know if a message is accepted?
- Typically, for flow control, we need explicit feedback coming from the receiver

Congestion control

- Moving packets from LA to London are shared by many people
- They use the channel for the same source and destination
- We could in the whole, combine everyone
- This creates more traffic, which causes congestion
- We might have to start dropping messages (which is good)
- We need to find which messages to drop though.
- Similar to flow control
- Flow control involves us knowing who we are talking to.
- For a # of reasons, involving scaling, we don’t know when we put a message in the network which link it goes to.
- Cannot have the guy in the middle telling us it is okay.

- Have some other kind of feedback that will tell us there is a problem in the middle of the network.

Latency

- If we do NOT like our latency what can we do?
- Reduce distance between sender and receiver
- Even for a multi-hop network, there is delay between each hop.
- Not always a practical thing to do.
- Take a more efficient path
- If you can take a path going from 6 hops, it could potentially be quicker.
- More bandwidth lets us send bigger messages or have more outstanding messages
- It used to take you 10 seconds to move your file; now you can move it in 3 seconds
- Decrease latency by increasing BW

Space

- How much space does a message take?
- Expressed in bits and everywhere the message is
- It is taking up a certain amount of space as long as the messages are going somewhere.
- Use less space on the line, less space in the buffers
- Compress the message
- Works by looking for redundancy in the data and replace it with shorter patterns
- We need a key to organize our replacement
- We can do this within a single message and try to find all the redundancies
- We will be sending 10,000 messages as part of this overall, high-level activity we are performing
- Look at the entire collection and do compression on the whole thing
- Generally speaking, compression works better on large collections of data compared to small collections of data
- We can also encode efficiently
- Among those encodings is putting headers on the message
- If we encode efficiently, it takes up less space
- Usually, we want less space

Energy

- CPU is capable of running some # of instructions per second
- Burning actual energy (Joules)

CPU

- Message arrives at a computer
- Binary instructions
- GO up through the DAG and make use of the message

- All of this requires the CPU to run instructions
- The fewer instructions required, the better.
- Every layer we are going through runs through a FSM
- The more layers you go through, the more cycles you will spend
- Other ways of handling minimization of the use of CPU's
- Do the same thing over and over and over again.
- Why redo the entire set of work and recognize that we did duplicate work?
- If we say that we want to send to Amazon.com, if we don't know Amazon's IP address, we do a translation as we go down to the IPv4 layer.
- Send another message down to Amazon.com
- We can expect the same result if we do it again, so let's just cache the result
- Caching is one of the big tools we use throughout computer systems
- We can say to look at a system and take a look at your code or algorithm
- Can we write a better algorithm?
- Take a look at the opcodes
- It does take more effort to write in machine code
- Maybe we don't need to do quite that algorithm, so it isn't perfect.
- Gets you results that are cheaper

Actual energy

- How much energy to process a message?
- Electrical power, heat to dissipate, etc.
- There are other advantages i.e. paying for your electricity
- Whenever we run any kind of processing in a computer, we burn the electricity and this generates heat.
- We will have air conditioning and this must be paid for.
- Reduce the amount of heat and reduce the amount of cost to process this set of messages that we are dealing with.
- Even with air conditioning, we have to see how to cool down components with the hot air.
- How do we save energy?
- We burn energy by doing things on our computer.
- We want to do less!
- Make use of caching and make use of what we have done before.
- Slow down our computer
- The faster our computer runs, the more energy it uses
- Not a linear relationship, more like an exponential
- Run fewer instructions per unit time but this is bad for latency
- Avoid conversions
- Translations are an example of this

Emulation

- At the level of what it looks like to me and the guy over there is the receiver

- What do I have between me?
- The Internet
- Something a whole lot less complex and at least at a high level, should be simpler to deal with
- Wires
- It would be a lot easier to work with because it is essentially a Shannon channel
- Emulation
- Some other kind of useful entity

Wires

- Circuits had some good properties
- We have packet switching, but we want circuit switching
- Wires are much more simple than circuits
- If we don't have wires between us, can we look at wires?

Circuits from packets

- Put something in at one end and that would be desirable
- Have some state kept at endpoints and provide an illusion of the circuit and try to maintain that illusion
- Limited capability of doing anything
- We can change what happens at the endpoints but not at the middle
- Keep all to of info about order and received messages
- Produce the illusion we have a circuit
- We won't necessarily make this exactly a circuit, but we are trying to get as close as possible
- Use TCP above IP, which gives us reliable, in-order delivery
- Examples
- AAL 1-4 from ATM

Pseudowires

- A lot easier to deal with this kind of channel
- Gives you something else like the properties of the wire
- You know how long it takes to send a bit from one side to another
- You can measure it on the wire and it will be accurate for sure.
- We want a performance guarantee and achieve some performance
- Examples
- SONET

Order

- Circuits and pseudo wires emulate channels
- We need to test things out of order and put things back into order
- We usually only have control over our endpoints
- Cannot really do it at the source since we have no control of what happens in between
- Destination deals with this sort of ordering

Boxes and bundles

- Think of the boundaries you put around things
- Dealing with items you receive from the network

Boundaries

- We can say we are doing a whole lot of communication and this would require a whole lot of messages
- We can have a lot of little tiny items and perhaps they share some characteristics
- Pack a whole lot of items into the messages
- When we send an IP message across the network, we may run into a link that is capable of moving packets of that size
- When it hits that think, we can split up those bigger packets into smaller packets
- IP protocol would work, you divide it up into pieces and from that point, those subpackets move independently
- Ultimately, you need them coming back together again
- Once at the IP level, you don't do this
- Each fragment travels independently
- All of these go to the designation and they will arrive at the destination
- Defragment and put things back together again.
- HTTP level
- Handle an HTTP request and your request is potentially very large
- Multiple packets that represent the entire thing you want to do
- When we know that we will get a whole bunch of stuff back, we are going to save up everything and build it up into one big piece.
- Then we will know everything and give it to a higher level.
- We want low user latency rather than saying we will wait for everything
- HTTP can gather everything and it will arrive at the very 1st packet
- We will move things reliably and in order
- 500 packets can be part of a webpage.
- Movement of a big thing entails that we need to organize how the pieces are arranged so we can put them back together again later on.
- This will be the piece I am talking about for this particular sub-element

Flows

- We sometimes want to say that we have a bunch of connections and they all need to be related to the same thing that is happening
- Maybe to the same place, maybe to different places
- Because of the technologies in question, we can have a particular bandwidth
- Whether it works or not depends on underlying factors
- Cooperative web session, and some shared screen that is supposed to show up for everyone
- Require some degree of control

- We need to say we have N participants, and we need to order the results in the appropriate way.
- Don't confuse it based on the parts of the checkmark.
- We can also have alternates
- We can take an important piece of data, but what if the destination crashes
- This is why we need backups!
- We want the flows to be coordinated
- TCP does control block sharing
- Multipath TCP, one amount of data from point A to point B
- Perhaps we can get more data through

Transactions and beyond

- Higher level of control that we want out of this network operation
- A bunch of stuff can happen in a coordinated fashion
- Other types of things that we care about
- Do more coordinating things
- May want the networking at some layer to support a degree of coordination
- Why?
- The purpose of software reuse.
- Moving files i.e. file A and file B
- Do some conjunction (AND) types of things:
- A AND B required to send C
- Disjunction (OR)
- A OR B required to send C
- Many variations
- Make sure there are three copies and there are six places that might hold them.
- Please make a copy of this and don't worry about the other redundancies
- Be aware there are 6 and keep track which of the 6 we are using.

Translation

- Do this within the recursive block and this will go up and down through the stack
- Language translation
- English -> Chinese
- Format conversion
- HTML to ASCII
- Display conversion
- Going from a desktop resolution to mobile resolution

Other services

- You have particular limitations and if it doesn't map well, you are NOT going to get the effect you are hoping to in some way.

Summary

- Deficiencies need to be fixed first
- Fix the deficiencies that prevent parties from communicating effectively
- The major goal in the Internet is to communicate to someone else
- See if this is good enough and if it is fast enough to support what we actually want.
- Compress all of our data and maybe use multiple paths
- Want it to go fast and cheap
- This will already go to some extent and we can want our network to look like this.
- We want reliable connection with guaranteed bandwidth and guaranteed reliability
- We go to the 3rd step and emulate wha the user wants on top of the less capable network.

Layer Optimization: Handling loss
Detect or correct?

- Cannot fix error by just looking at the mistake
- Not a trivial error

What do errors look like?

- Errors in destination address
- Drop messages if they go to the wrong place
- There can be an error int he source address as well
- FSM that represent individual layers of communication
- Many different source addresses and it will go to the wrong FSM
- It can mess up this FSM's behavior and it will go to the existing FSM and we never heard of this particular source address
- The source destination can go to the right destination and goes to the right FSM
- What will happen is we go up through the players and somebody is going to use the erroneous message to do the wrong thing.
- You can see the effects many years down the line

Making errors visible

- Consistency checks
- Header fields within the content of our message and understand our content.
- If we understand things in a range, we can say this should NOT be in that field and that will be an error
- This is a range of valid things that should be in a particular part of the message
- There should be an error in that case.
- NOT everything can be consistency checked
- Redundancy check
- Add more to detect errors

- Adding redundancy back in (opposite of compression)
- Redundancy simply says a pattern occurs X times
- We will probably have more information added to the message
- Redundant info can affect multiple places and it can change the redundant information in one place to another
- One of these can change and we will have an error

Consistency checks
- Each layer has a header and other info related to the content of the packet
- Field is supposed to be all 0's
- If it is NOT all 0's, we know something wrong happened
- If we get to a router, then there will be an error?

Q. Why is there a padding field in the TCP header?
A. When they figure out everything they wanted, it will be terms of bits. We could use additional padding, but we never did.
- Needs at least 40 bits

Redundancy checks
- We can use consistency with the expectation we can detect errors in the message
- We don't need two copies of the field but rather certain mathematical relationships with this portion of the message and other portions of the message
- We can always say to treat it in the moment and consider portions to be a # and set up some mathematical relationship that won't hold if there is an error
- Common methods
- Matching: A = B
- Parity: (Summation of a_i) % 2 = p
- Checksums
- Cyclic Redundancy Check (CRC)
- Do some kind of mathematical computation, send a message, repeat the computation

Example - IPv4 checksum
- Content takes most of the header field and does ones complement
- May result in a carry over the 16 bits
- This is advantageous because it is really easy to build hardware or software that does this fast.
- Maybe we want to do something really simple to implement in software
- Check things at various places in the network
- **IPv6 does NOT have a header checksum.**

TCP checksum
- It is NOT considered to be related to the IPv4 checksum.
- Covers a whole lot of fields
- IP header files includes a hop count field

- It would never match because we would have decremented through several relays and the value of that field would NOT have been the same.
- We don't look at that! We use a method like the IPv4 checksum.
- Different headers and alignments
- TCP checksum is computed at different ways depending on if we are looking at IPv4 vs IPv6
- The way it is performed depends on another layer
- We aren't always going to be really careful related to a layer's behavior
- Translate things up and down
- We have to remember that a translation occurs from IPv4 to IPv6

Loss
- Bits got flipped and somebody picked the wrong values in the field
- Packets didn't get delivered and it didn't come out.
- It might have been put into the network but it hasn't arrived.

One way to lose a message
- Lecture 16, Page 12
- We could create the message, send it, and something bad happens.
- The message is gone and it is never going to get delivered
- Timeout
- Indicate there is a problem
- Message starts going across and something changes in the message.
- There will be some kind of corruption that isn't correctable
- Such an ill-timed corruption that the guy cannot really tell what the message was.

Another way to lose a message
- Lecture 16, Page 14
- Error detection/correction
- Used to identify the problem and fix it

A 3rd way to lose a message
- No room and there is no buffer space to store the message
- The message will have to be dropped
- Issue for flow control

A 4th way to lose a message
- Really busy with messages that came from other nodes
- There is a way to fill up other buffer space
- He will be too busy to store the message here and he will have to drop the message.
- This is a significantly different situation from when the receiver was too busy
- Network is busy in this case
- Use congestion control!

Detecting loss

- Use time as our metric and keep track of it to detect loss.

The key issue: what time?

- We are expecting a time, and we are expecting it at T + delta
- Send a message and we need to wait and wait.
- Let's move on and do something else
- How long can we afford to wait?
- Sit here as a receiver and expect to get messages.
- Maybe time is how long we have to wait.
- Time since last message was sent.
- Time since last message received.
- We can use that to detect loss.
- One of these types of times can be used to indicate if there is a loss.

Estimating time

- Dealing with a sender and receiver separated by a wire
- Short time between when it is sent and when it is received
- In that period of time, you can detect loss because you know it should happen quickly.
- We can have a 15 hop path using an overloaded wireless link at the end
- In that case, it will take a whole lot longer for that message to get through
- This must indicate loss and this can indicate incorrectly if the message is still being sent.
- Time has to be related to a particular circumstance that I am using.
- We have to wait a much longer time and we will have inefficient loss detection
- If I say we have to get through it very quickly, we are going to claim a lot of losses and take actions in response to losses.
- Look at 8 hops between me and the destination
- Another thing I can do is send a bunch of packets and measure the transmission time.
- There is certainly a limit to how fast I can get the message there.
- Limit on the longest time and give you an idea of the range.
- Figure out a # of hops and figure out a per relay cost.
- We can say it doesn't take the dynamics of the network into play.
- Measure what is happening in my collection.
- If I base the measures on what I got, this could be the typical delay that I saw.
- Throw 8,000 packets into the network.
- We come out with what we should figure as the 8,001st message.

Measuring time

- Have some timer that is a clock of some nature

- If I have a clock that ticks every second, and we want to check every 100 ms, that is way too SLOW!
- We have to have clock increments that fit within our desired results
- We are looking for effects that cover more than one place
- If I send a message at time X and it goes across to the other side, we want to have timestamps for those messages.

Reordering
- One reason we care about this is because we care about messages delivering in order.
- Both arrive and they both get here in a reasonable amount of time
- We can fix things by getting things at the destination end
- We can deliver the message right now and we can save the message
- Deliver the messages in the right order
- Lecture 16, Page 21
- Wait for blue to arrive and deliver the blue first
- Deliver the red after event though it arrives first
- Different layers do NOT understand ordering
- Transport layer is probably the only one that understands order
- How we save messages depend on which layer detects the disordering

What's hard about reordering?
- Need a lot more state - a buffer to save information that did NOT arrive in the appropriate order.
- The way we figure out is what is the maximum mis-ordering we can get.
- We need a buffer big enough to store the space
- The only place we can really do this is at the receiver

TCP receive window
- TCP segments and header information built-in.
- Is what I see here between 1,000 and 2,000?
- This won't make it possible to handle misorderings
- There are downsides as well and I want to receive things to come in. This is the biggest thing I am expecting to see and it should fit somewhere inside the window.
- This implies we will get a buffer to hold this much information
- Handle a buffer and we can get a 4,000 byte buffer.
- We have NOT received information and we need to see something up to byte 1,000.
- Make sure it is within that range I am expected to see.

Receive window management
- Keep window at the same size
- Move from 1,001 to 2,001
- Keep moving the window as we get things in order
- What if we get something that is NOT in the left-side of the window.

- Ones from 1,001 to 4,000
- Cannot move the window since we have NOT seen the stuff on the left-hand side.

Receive window
- The green is his current window and we are expecting that the next message

Receive window with disordering
- We can deliver both messages to the next higher level in the protocol to the next window

Hmm - what about the sender?
- Senders can be within a certain range and we shouldn't send within that range
- The receiver will send acknowledgements and what the receiver buffer is like.
- Since he is paying the received buffer, I will now know what is in an in-order message.
- We can figure out what is happening at the receiving end.
- Buffers are taking up space and we can see a relationship between time and space
- This leads to some problems that we are going to deal with.

Flow control I
- Both networks are potentially going to be at different speeds
- In particular, the receiver may be slower than the sender
- The sender can send a lot more messages than the receiver can process
- Sooner or later, the buffer space will get entirely filled unless we do something
- We cannot be buffered because there is no more buffer space
- We assume that if we are a sender or receiver, the receiver will be able to send one more message.
- We cannot send a 2nd message safely because we don't know how long it will take
- If we wait for the receiver to tell us, we can send another message and we know he is NOT busy with another message
- Therefore, we just keep doing that and we need to retransmit things
- If the sender is capable of handling messages as fast as the sender, this isn't an issue
- We don't have certain knowledge and we cannot be sure of that.
- Send one message at a time and we have to be sure it is ready to handle.

Stop and Go
- Steady flow resulting in neither party being overwhelmed.
- Usually what we care about is the sender NOT overwhelming the receiver

- Only one outstanding message at a time
- Stop when you sent a message and you go again when the message has been properly received
- Sender and receiver work in lockstep
- Wait for a message to be received and keep going that way one message at a time.
- Sender will NOT be able to send messages unless the receiver acknowledges that they got it.

Let’s take a walk

- Lecture 16, Page 31
- Wait for 3 and 4 to be asked to send 5

Limiting the walk

- Send and receive ACK linked by a limit
- Next message should be ACKed and wait for an ACK on 5

A look at the exchanges

- One message per round trip
- ACK indicates received and ready for next

A look at the numbering

- Packet 0 and packet 1, we need to wait to get a 1 bit counter and go from 0 to 1
- Alternate 0’s and 1’s in addition to protocols and alternating bits
- Alternates from 0’s and 1’s, 0’s and 1’s

Why do anything else?

- What if the receiver is faster than the sender/
- The receiver instantly handles it and he has to deal with more messages.
- Process the ACK and send a slow message that will eventually get to the receiver.
- ACK takes time to get through the network
- Until the ACK arrives, the receiver will NOT be busy and he isn’t processing a message, however.
- During that time, the channel is NOT being used at all.
- This is perhaps a bigger window than 1
- This gets back to a problem of misordered message
- Send message 1 and 2 can arrive before 1.
- Do some of the stuff with having a buffer that will save messages that arrive out of order.
- We can still have that out of order message in that case.
- If a sender can send more than 1 message a t a time, that receiver can actually be busy and message 1 arrives instantly while message 2 arrives right after.
- When a message comes into a receiving node, it must be able to process this message.

Flow control II

- We can use this buffer to handle the other problem too
- Maybe the buffer is saying that it wasn't in this order and it came in the right order.
- The previous message cannot handle the message sitting in the buffer.
- Sooner or later, the message is NOT busy anymore.
- Deal with disordered messages and they might arrive faster quicker than the receiver is handling.

Go Back N

- Misordering now possible
- We assume we have a sequence # sitting on the messages
- Each message has a sequence # associated with it
- They are going to work in lockstep and both will "walk" the # space
- We can expect a pair of messages between sequence messages
- They have to be playing by the same rules and know which messages the sender will receive.
- Similar to stop and go but instead of one outstanding message, we will have N outstanding messages.

Let's take another walk

- We cannot move the sequence # until we have have gotten an ACK on 3 or 4

Limiting the walk

- Limit of what the sender can send
- Go-Back-N, N > 1

A look at the exchanges

- Sends them out at whatever speed he is capable of sending them out at
- The receiver can say it is time to ACK the messages
- The 3 ACKs go back across and now the sender can keep sending more messages
- He has 3 outstanding messages and we should wait for more ACKs to come in.
- Keep things in the buffer and use the buffer space he expects the sender to say.
- We could have very short sequence #'s and we cannot have 1 bit sequence #'s since we have to tell if it is message 1, message 2, or message 3.
- Sequence # that goes up to a certain level
- We only have to have sequence #'s of up to 3.
- We have to have up to 2N - 1

Why 2N values?

- N outstanding values

- Each RTT, the window can move forward by N
- How many messages am I willing to wait to get my very first ACK
- He might send out all N of those messages
- He can send the next one when he is ready
- Every ACK/N + 1 pair will act like stop-and-go
- Effects related to the buffer for a large N
- We have to have a buffer capable of holding large messages
- We are occasionally going to have loss
- Lose 1 of the N messages because if it bigger than N, it cannot be sent.
- If a message is lost it is within that window of N possible messages.
- It could be ANY of them!
- The sender will have to retransmit lost messages
- We have to send a copy of that message and the receiver has to have a window buffer and the sender does too.
- The sender can generate a bunch more messages and he will have to save those because he cannot send them yet.
- The bigger the N, the more messages he may generate.

About the receive window

- What if the receiver isn't fast enough?
- This means that as the message comes in to the reliable delivery layer.
- We have to make sure the FSM are working fast enough to release the messages to an upper layer
- We probably cannot ACK it and move on.

Recall receive rules

- Anything moved across will be passed up to the next layer
- If the FSM is fast enough, we cannot move the bytes until we have properly passed up to the higher layer.
- Free the 50 bytes until the FSM is done with that work.

Left and right

- If the left side doesn't move, you cannot move the right side.
- This buffer is going to be sitting at some layer like the TCP layer
- Ensure that we are going to coordinate with the sender as well.
- ACK a message with the sequence # and it has to contain the sequence # to know what is ACK'ed
- Controls the sender as well as the receiver back and forth.

Coordinating SND and RCV

- Lecture 16, Page 48
- Categories of what happens with the receiver that is sent
- Not done from a sender's perspective
- Category #2 is his view of what the received window is and it is between 32 and 45
- He may not send anything else if not necessary

- If the window is up, we can still have some space in the receiver window.
- In that green area, he can send a few more bytes since the receiver changed this.
- The sender has more bytes that he would like to be able to send.
- This is NOT synchronous; not magically changing, so it is possible that in actuality, the receiver has gotten bytes 32 through 45
- Sender hasn't gotten the ACK so there will be a delay so the sender can match the receiver's window.

Lecture 9 R Lec 3-3-16
Coordinating SND and RCV

- Window that is kept in one state at the sender and one state at the receiver
- Permitted to send a certain amount of data to the receiver, until the receiver lets the other guy know
- Wait for all the earlier information to come in
- The receiver needs to be capable of holding a certain amount of data until the FSM is capable of catching up and dealing with this data.
- This will ensure that you will never overflow the receiver and he will never have to drop any packets.
- I, the sender, will have a similar sized window and we will never have outstanding, unacknowledged data
- Lecture 16, Page 48
- Maybe the receiver can still be chunking things in but the sender doesn't have to worry anymore.
- Window is the black box around the numbered boxes
- This categorizes information that is already sent out to the receiver
- It might get dropped or corrupted, so it might be necessary for the sender to send it back.
- It cannot be used to use those slots for any other information.
- He cannot send more of the data that he wants to send.
- Covers part of the sender's window and he cannot send out all of the data.
- He will send bytes 46-51 and it won't cause any flow control problems at the receiving end
- Data outside the window that the sender is perhaps ready to send.
- No guarantee that the receiver can handle it then, so he will wait.
- Until he receives an ACK from the receiver, he will have to wait for an official confirmation.

Combining loss + windowing

- Positive feedback (ACK)
- Whenever some information comes in on the channel from the sender, we can have that particular information and we can say that I have got it.
- That portion of the stuff on my send window is taken care of.
- Negative feedback (NACK)

- Can be the case if you have some out of order delivery
- Sees 7, sees 5, doesn't see 6
- What can he do?
- He can send a negative ACK and he didn't see anything.
- This is a way to more aggressively indicate to the sender that I am NOT getting the things I am expecting.
- The sender could live without negative ACKs
- Can survive with just positive ACKs

Loss/windowing variants

- Stop and Go
- Sender goes for a little while and then he stops, and he stops, and he stops until he is told by the receiver to go again.
- Go back N
- Whenever you have a timeout on the receiving end, ACK the lowest missing sequence #
- I have seen everything up to that point.
- It can be just 11 that you are missing, but this is a simple way of handling a loss of a whole sequence of packets.
- Once you are told to go back, assume nothing ahead of that point has been received, so resend them all!
- Selective ACK
- ACK whatever you get
- The lack of an ACK will indicate to the sender that particular packet did NOT get received
- Resend only packets that you don't see an ACK.
- If you are using Go back N, eventually the receiver times out and the receiver will be asking for a resend of N.

Congestion control

- Receiver might be ready, but we cannot overload the receiver.
- Q. If you use NACk, will you get resets sooner?
- A. Yes, you don't worry about what the sender chooses to do, but you still need half the round trip time before the sender responds.
- It may be faster to wait for the sender to time out himself.
- Q. What happens if you lose the NACK along the way?
- A. You will get another time out.
- If you get to a stage when you don't see packet 16, sooner or later, the implementation will say that we give up, and this channel is dead.
- You may find it somewhat frustrating and we do a secure shell on my laptop. If you go home and you are still connecting to the machine at work, it will keep trying and it will eventually say you are NOT connected anymore.
- Server has been saying why don't I hear from my client so it quit a long time ago.
- Number of links and outgoing links and it is one of our relays.
- It can have a capacity and that link can have some kind of problem.

- Eventually a buildup of information buffers and the links themselves aren't necessarily congested in this way.
- It can simply handle a bunch of switching back and forth for a period of time.
- This can happen for any of the links in our messages.
- Only known about at the sender and receiver.
- Cannot use the same techniques to deal with overload at the middle of the network.
- Congestion control: essentially, you have congestion problems when the sender is okay and the receiver is okay but the channel in the middle is busy.
- To handle congestion control, add another window!

Solution: Congestion window
Receive window

- As large as reordering max and allow multiple messages at the same time

Layer Optimization: Congestion Control and Other Space Issues
We can lose packets for many reasons

- Poor flow control
- Receiver gets overloaded

Congestion control

- Network in the middle is too busy, so we have to do something.

A network problem

- We now switch to a problem that is more general and is covering a bigger portion of the elements and congestion is particularly about the network path and a particular element is that it is caused by the combined actions of different senders and receivers who share the same relay link or relay node.
- The parties communicating with each other now share the same link and between them, they cause congestion.
- The only shared information is the endpoints
- Hasn't received certain packets, etc.
- What are all these other people doing?
- What are the states of the relay nodes in the middle of the network?
- We have to be something of the form of less traffic and it has to be in the overloaded portion.
- This is what the solution has to be and the only way to have less traffic is to send less.

How to address the congestion control problem?

- Global problem with anyone who shares the network
- All of them are failing to get nice, smooth performance.
- Sounds like we want a global solution.
- This link got corrupted, so send less here, and send less there.

- This is a problem because we don’t have anything with an inherent global view of our network.
- You can make a phone call, but at network speeds, this is only useful at certain speeds.
- Send messages and you want a global traffic cop.
- Describe the congestion levels at each of these places.
- Some central point that does that, and this doesn’t sound like a good idea.
- Do some kind of analysis, so what would he do?
- Reduce the amount of traffic in the network so the congestion can go away.
- Drop packets since he cannot magically make it handle more packets.
- Doesn’t work out to have any single thing that is centralized and do things at the network speed.
- Routing algorithms that have global authority and build a lovely global routing table and pass it out to everybody.
- Each piece does some local stuff and we work out a solution by subdividing pieces
- Each solves by local entities
- We cannot use global solutions, so we use distributed solutions!
- Have each party deduce if there is congestion on the channel it is using
- Then, each party will try to help out and if there is congestion, the yawl each try to do something to make it better!

But what can I do?

- You can only change your own behavior
- You cannot affect other people’s traffic
- If you are a sender, you can choose to send less
- I think there is congestion in the network and if everyone backs off a little bit, it seems like life would get better for everyone.
- Everyone pays a little bit of cost but they each benefit from it.

Send window maximum

- Consider the maximum size in terms of # of bytes
- What you want is to say what is the bandwidth I can get on the path between source and destination
- It takes some amount of time to propagate bits and over a period of time, we have some # of bits on the wire and if we multiply it by the bandwidth, then we get a bandwidth delay product
- Maximum amount of data you can shove though the channel
- Wait for ACKs to come back and this indicates your receiver is ready for more data.

TCP and congestion control

- Doesn’t help us directly with congestion control
- Let’s talk about it in the context of TCP

- Most traffic is carried over TCP
- Unlike some other protocols, TCP cares about congestion control
- Cooperative approach
- We assume that everyone using the congested link will do a similar thing and we each slow down.
- Assuming everyone slows down, less traffic goes to the congested link, and eventually, we free up the link's congestion

TCP's CWND

- Handles congestion control
- Two important parameters here

TCP MSS and RTT

- MSS - Maximum Segment Size
- Every link you will go through will run some link level protocol
- This many bytes and we cannot send anything bigger than that.
- Cannot send a message more than 1000 bytes
- If at the sending side, we send messages that are NO bigger than that size, then no other link can handle it.
- For TCP's purposes, we have to care about how much TCP stuff we can handle the headers.
- IP header, link level header, etc.
- How many bytes can you shove?
- 536 "octets" (essentially bytes)
- The reason for this is because previously, bytes were 7-bits, not 8-bits!
- For all practical purposes, bytes nowadays are octets
- In TCP, if you don't know anything else, assume maximum segment size is 536 octets/bytes
- You might be able to shove more or less in.
- Essentially, each link has some maximum segment size and we need to find out what is the smallest size.
- Sooner or later, it will have to be split up, which is undesirable.
- You don't go out and query people and you don't know which links are carried in TCP connection.
- RTT - Round Trip Time
- How long does it take in clock time for the sender to send a TCP packet and receive an ACK
- Indicate change in time, and round trip time is whatever I measure.
- Q. Why is it undesirable to split things up into links?
- A. Fragmentation and generally, you will have higher overheads.
- Split up a packet at higher levels
- One of the problems with splitting things up; we get into situations where we get mis-orderings of fragments of the packets.

Adjusting the congestion window

- TCP CWND management

- We say that we have some window which is the maximum amount that I can send
- The window we were using before meant we would agree that this is what the window should be and this should be the beginning of a TCP connection.
- This will be our window and it will remain the same in our connection.
- He would always have a buffer that big and the congestion control window is based on dynamics.
- We do NOT want full control over this.
- Window that gets bigger and smaller as we handle dynamics.
- Default, you start it out at some size and send it up to 10 packets and it depends on decisions made.
- Network can already be pretty congested, and we do NOT want to make it worse.
- Usually, we don't want congestion and we can handle a lot more traffic than we offer.
- We don't want packet drops, so we don't want congestion in this case.
- Start out TCP connection with very low window since we are concerned there is congestion somewhere along our path.
- Let's add a little bit to the size of the window.
- What if we have added too much and we have tipped it to a congested state.
- What are we going to do?
- If you have congestion, send less traffic yourself.
- When you observe conditions suggesting congestion, decrease your window.
- You liked what you saw and you raised it a little.
- Lower the layer A LOT and drop it SIGNIFICANTLY!
- Very common approach to dealing with congestion in a network.
- When things go bad, drop back a long way.
- Slowly probing the network and overlay the fact we are sending data from the sender to the receiver.
- We need a form of probing by getting acknowledgments from the receiver's.
- There wasn't a TCP connection, so we need to start sending in data.
- It makes sense to have two different phases of congestion control; no info about if we are congested or not.

The slow start phase

- We have a pretty good idea of what the slow start conditions are in the network.
- Try to maintain a fairlly stable position
- Done in the slow start phase
- Enter a phase of behavior where we stay here.
- We have a parameter called SSTRESH
- This says keep increasing your window and we reach the slow start threshold.

- Enter the other phase of behavior.
- Each time we get an ACK, grow our congestion window by 1.
- Now, remember what we are doing here is starting it with 10 packets and one right after the other.
- Assuming no congestion and it has got plenty of bandwidth and buffer space, shortly after it is received, back comes the later ACKs.
- In little more than just one round trip time, the sender can get back 10 ACKs for 10 messages.
- For every ACK he gets back, he gets 10 windows.
- Now his window is 20, and he can send out 20 messages.

Why's that exponential?

- Send out N packets
- If everything goes well, you get N ACKs back
- Next time, you will send out 2*N packets
- If all goes well (no congestion), you get 2*N ACKs back, so we send out 4*N ACKs
- We are sending 2^i * N and this becomes exponential
- Q. Send window doesn't change?
- A. Send window will NOT change throughout the TCP window
- It is possible to have a send window smaller than receiving window, but there isn't really an advantage of that.

Why does it stop?

- We have reached the slow start threshold.
- We have a lot of data without getting ACKs.
- Keep doing doubling each round trip time and we get congestion pretty damn soon.
- Stop doing the slow start behavior in this case.
- Otherwise, something wen wrong and we would care about congestion.
- Some element in the network cannot handle all the traffic it was given.
- When it runs out of buffer space, it starts dropping packets.
- If you have increased and increased the amount of data sending down your path, then one of your packets will get dropped.
- Time out waiting for an ACK and it does't come
- Resend that packet and deduce there is congestion.
- We don't exactly leave the slow start phase but now we have our threshold
- Divide SSTHRESH by 2, so probably, the result tis that we will start sending less
- The TCP protocol is thrown around loosely
- There are dozens of variants (in fact, probably hundreds of variants!)
- Instead of having SSTHRESH, let's divide by 10, take 90%, etc. (lots of variety!)
- There isn't a whole lot of guarantee that he is necessarily going to be doing what you think.

- All the things are compliant and mechanically work together.
- Endpoints can vary a whole lot depending on which TCP variant they are using.
- Which TCP variant the other guy is using.
- Q. How is it a slow start if we double each time?
- A. We are starting at a low level i.e. 10 packets.
- Multiplicative decrease == exponential backoff

Congestion avoidance phase

- We keep doing this forever and there is a limited amount of bandwidth in the network.
- Let's increase it pretty quickly during this phase and this is a high-level of bandwidth.
- We are on the edge of what we want to do without congestion.
- Let's exit this slow start phase and reach SSTHRESH
- There will be no congestion so far and what we can do is go up to a high SSTHRESH
- All calculations are based on a lower value of SSTHRESH
- At any rate, if you go over that value, you are at a slow start.
- Keep inching up and push just a little harder and push a lot less aggressively.
- We are going to increase much more slowly now and this is like a linear increase rather than an exponential increase.
- Let's sum a period of RTT past and then add a little bit to the congestion window.
- It does NOT go up at the same speed, and visually this is the kind of thing we may have.

Visualization

- Lecture 17, Page 18
- We can see we initially have an exponential increase and we hit an SS Threshold
- Then we see we have to go up 1 for each roundtrip time and then we only see an additive increase
- We can still push up too far and the ACK doesn't come in
- Reduce congestion, go back to the slow start phase and then repeat the process.
- Q. Why does it go all the way back down rather than start off at the threshold?
- A. Congestion is a dynamic problem, and no matter what, they will inch up too high inevitably.
- Takes a while to clear up and do our best to clear out our congestion.

Details

- Popular variants don't actually double per RTT in slow start

- We increase by 1 for every ACK we get, but we only ACK every other message.
- ACK of message 2 indicates that we got message 1.
- We still only add 1 so we don't really go up by 1 per message sent, so we go up by every 2 messages sent.
- Compare 50 different TCP variants for different purposes.

TCP's biggest assumption

- Assumes that it understands things about what is happening in the network.
- TCP in its ordinary interpretation knows the following
- What arrived: it believes it got whatever is received.
- Timeouts: Go through a period of time without an ACK arriving, then we know we have problems.
- We know what the max receive window is and we think we know about network congestion (via timeout!)
- Timeout is NOT necessarily an indication of network congestion, but rather tell you that you didn't get an ACK
- Tries to measure congestion by getting timeout on an ACK
- Either the message that should have been ACKed was lost and there could have been corruption in the network
- Didn't ACK the message because it didn't get to you.
- What is the right thing to do if you had a problem with corruption?
- Send it again as soon as you possibly can.
- Need the data that got corrupted.
- Packet was moving through the network and got dropped because the congestion is still and it will probably happen again.
- If there is no checkout, it is considered a loss on TCP.
- You might not even know if it was destined for you!
- For two things to cause this to get lost, there are two right things we should do.
- We should wait and back off and send less.
- We will eventually have to resend it and TCP says that we need problems with loss is congestion.
- If I have lost a packet, we assume congestion and we take measures in response to that.
- In the network where corruption is high, TCP works poorly.

Impact of loss = congestion

- Use variants of TCP where loss packet is due to congestion.
- I will either take extra measures to distinguish between congestion and corruption
- Also the case that if corruption is NOT due to load, then TCP will work poorly.
- If you have TCP corruption due to high load, then your packets got corrupted but you still want to send less.

- TCP is aggressive
- Pushes until it pushes too far.
- If you have two TCP flows, they are competing with each other and there are elements that are each trying to grab as much as they possibly can.
- What TCP is supposed to do is assume congestion and we have to back off.
- We don't get as much data through and the congestion would go away because we stopped sending as much.
- People cheat and we end up NOT doing the reduction we should do.
- Assume other people will fix it for you.
- Q. What are the two TCP flows that are fighting each other?
- A. Let's say you have a link and there are two TCP connections, they will both go through the link and we are sharing a small amount of bandwidth between two flows.
- As they increase, they will try to get more and more of that bandwidth.
- This will happen to all the bandwidth.

Congestion control algorithms

- Little tiny tweaks
- What you did for the basic TCP protocol was ad hoc and you pulled it out of a hat
- We know all about equations about fluid flow, feedback theory (engineering topics)
- In many cases, TCP variants are kind of the variants we got.
- Whatever software we are running does what it does and it puts together all the right TCP variants.
- If you know enough, we slip in different layers and there were DAGs before.
- In practice, it is harder to do that if we are using a system like Linux.

Latency management

- Another thing we see that is a characteristic of networks is latency
- We have buffers at the sender and receiver
- The purpose of these buffers is to adjust to bursts in the traffic.
- We never vary and we would design that particular piece of equipment to handle the traffic.
- Varying amounts coming in than we can push out.
- Hold messages for a while and these messages are always finite.
- Once we will it up, if we cannot deal with the incoming messages, it cannot be buffered and something has got to go.
- Some message has got to be dropped!
- The most common thing is called "tail drop"
- Fill up the buffer and this has an effect on latency and deal with whatever delivery we are dealing with.
- Move to the next guy on the buffer and move him out.
- If we have a very big buffer, it can take a long time to go from end to end.

- The last message coming in can have a very long latency.
- Doesn't get ACKed very quickly.
- Timeout occurs on the sending end and it just takes forever to get through the buffers.
- We may get false signals of congestion and a busy network.
- We might say that this is very reasonable since we do have congestion and too much traffic in the network.
- If we do tail drop behavior, we need to keep the buffers full.
- If there isn't buffer space, we need to drop it.
- Once the buffer gets full, they keep getting added to empty spaces in the buffer.

Solutions to latency management

- One thing we can do is ask "Why does my network have to be so dumb?"
- Why can't it tell people "I'm congested" to the people that need to know.
- How can we tell someone that this particular link is congested.
- Another thing we can do is say "We can hold 5,000 messages in the buffer, but we would be a lot happier with fewer messages in the buffer"
- Signal somebody that you have a pretty full buffer to instead put messages into remaining empty space, you can drop a few instead!
- In the previous slide, the tail drop favors keeping buffers full and if it fills up the buffer, the buffer can have one slot empty and instantly another message comes in
- Empty slot on the buffer and because we are doing tail drop, we only drop if the buffer is full.
- If you get a high # of packets sent to you, keep the buffer filled as long as the packets keep coming in.
- For a long period of time, you will get a full buffer and this will have a long latency.
- Wait for all the messages to be delivered and when we use tail drop, it is common that the buffer will remain full.

Explicit congestion notification (ECN)

- I am a relay and I am congested, look at all these buffers filled up.
- I know that and wouldn't it be nice for everyone to know it too.
- I am communicating endpoints and I have packets coming through me and why don't I put a mark in the packet and say you got congestion here.
- Life is bad and you should do something about it.
- Mark the packet and we don't want to drop the packet.
- Full or nearly full buffers, put a mark in the packet's header (a bit saying that I am a congested router)
- Sooner or later, I will deliver that packet and assuming that destination is paying attention, we can say we are going through a congested router.
- Perhaps then we should deal with congestion on my path.
- If we know there is congestion on a path, we will still decrease the congestion window and we need to reduce our congestion window.

What if ECN isn’t available?

- Random early drop, what if we do this?
- Basically, as long as the buffer isn’t full, we aren’t dropping anything.
- It all gets dropped until a space in the buffer is freed up.
- Random Early Detection (RED)
- Randomly drop a message that you know you cannot hold.
- These messages are probabilistically chosen not to be put into a buffer
- Send a signal that they are sending too much traffic!
- What signal can I send?
- I will signal them to drop one of their packets and we have to decide which one to drop.
- We don’t keep track of what’s sent at a particular source or particular unit of time.
- It would be a good idea to back off.

What if ECN isn’t available?

- Random Early Detection
- Different equaattions to use to decide what to do.
- Keep increasing in a linear curve
- Have a more sharp and gradual curve.
- Q. Isn’t the red line prevention rather than detection?
- A. Detect in an early phase before it becomes a disaster.
- Increased latency is a bad thing so we want to cut down that as well.
- All of a sudden, you start dropping things and doing tail drops.
- We can do that in the same fashion and if we do ECN, we need some determination mechanism based on a curve like this based on a simple threshold.
- Usually congestion notifications are binary and we need to set the bit and we should set it on a random basis.
- We could do it in a plateau fashion and it’s really up to the router.
- It depends on what algorithm you choose to use to determine the ECN
- Based purely on the individual router that does the ECN
- Signal congestion based on this packet and if it chooses to, it doesn’t need actual congestion and send a marker on every packet.
- We have a marked packet and we don’t even know who marked it.
- At least on of those hops mark it and we don’t care which element was congested.
- This can cause a problem and if we do packet switching and NOT circuit switching, we can go down that route.
- Routing protocols tend to try to avoid doing fast switches of routes.
- In the short haul, the next packet will take a path and there won’t be a failure of a link.
- It will be dynamic enough to see a change in the routing.
- The packet switching can take the same path and we will see two connections to the internet.
- Send packets alternately and then split up the traffic in that way.

- Disadvantages because you will see two different paths in every situation

Better buffering

- Buffer packets in the relay and every packet can be a totally independent packet
- We know that we have these connections that are sending a whole lot from one to another
- They are all probably going through us.
- It might be better for each connection to have a separate set of buffers
- If there is one guy trying to push through us, there might be a lot of the.
- We might drop a lot of packets for everyone trying to send data through us.
- The guy sending a lot of stuff would help a lot.
- This wouldn’t help much with congestion and this isn’t really fair to them since they aren’t causing the problem.
- Set up a buffer for different flows going through us.
- Set up buffering on this basis via **fair queuing**
- Beneficial idea in some case and we can put these at a reasonable price at other costs.
- We wouldn’t have to worry about where to put the message in a complex way.
- If we are using this connection business, we don’t need identifiers or memory as each packet come sin.
- Analyze headers and see if there is a connection N
- Determine if there is a packet coming through for connection Z and put buffer space for future connections.
- Has a whole lot more complexity and cost because it is tough to route!

Space

- Handle issues relating to other space
- How big are the messages and how big are the buffers.
- How much space is taken up handling messages going through the network.
- Compression
- Series of technologies that take a whole lot of big data and compress into a smaller amount of data
- Look for redundancies
- Replace long things with shorter patterns and keep track of what the shorter patterns mean.
- Expand them back to what the really mean.
- Take redundancy and create less redundant data.
- If there are many messages beyond sent, take a set of messages and agree between send and receiver to NOT send the entire message and send just a small piece of it.
- Q. Are compression algorithms set up to whatever protocol, or does it depend on negotiation between the protocol?

- A. Pre-agreement between sender and receiver to use compression algorithm in any particular way.
- Assuming info on the sender and receiver’s part, he can agree on some initial handshake and have the receiver do the decompression in any particular way.
- It would take a long time in the short thing I send, we can decompress and expand it back out to our original form.
- Lossless compression and we compress, send , receive, decompress, and end up with the exact thing we start with.
- In image format,s we may take a high-resolution image and may compress it into a lower-resolution image
- Cannot go from lower resolution back to higher resolution

Web traffic

- HTTP 1.1
- A lot of the data is labeled and is built into the name and implicit to the way it is introduced into the application.
- HTTP 2.0
- Fair amount of redundancy in those so compression is viable

E-mail

- Postscript, word
- Compresses it by itself
- Zip folders
- Manually compress things and put it in the message
- Smart emails can do this for you

TCP/IP headers

- Compress the TCP and IP headers
- 40 bytes down to 16
- Routers in the middle need to figure out what to do within that message.
- Most of the header is predictable within a single connection.
- Header of those messages is substantially similar and make use of the inter message redundancy to handle header information.
- Used when you have very low bandwidth
- Anything it can get rid of have helped
- Dial-up lines have shitty bandwidth
- Another advantage though is that it is a point to point connection.

TCP/IP compression

- For 40B ACK packets, save 60%
- For 512B payload, save 4%
- Save 1500B segments (Ethernet), saves 1.6%
- Not that much!

Required compression information

- You have to know what are the patterns in the data

- Patterns that occur a lot are good candidates
- Long data
- Take entire set of data and it will examine the entire set of data
- What if we send network messages and we keep sending various messages.
- if we are trying to do compression based on the payloads, how can we do that.
- Use frequency information
- Morse code (certain letters are more common than others)
- If we use this channel frequently, we can move a lot of GIFs here and use some compression that is good for a GIF
- Detect patterns as things go along and things get better and better and better.
- Faxes work this way!

Compression Trade-offs

- Trade (consume)
- Not usable in that form so you have to decompress it.
- The algorithm takes a certain amount of time to do decompression.
- It will take some decompression and that will have a del there.
- The gain is that you will have smaller messages
- Takes up less space in memory
- Move fewer bits across the wires
- There is a limited amount of bandwidth and there is a propagation delay across the links
- The fewer bits you send, the fewer propagation delays you get until you end the entire message.
- You can afford to say we will take more time comprising because we get more benefits.
- Not always going to be a win since NOT all data is compressible.
- Anything you have encrypted is incompressible data.
- Random data doesn't have patterns in it so compression cannot compress in this scenario.
- Try taking a file, encrypt it using AES, and try compressing it
- You either get a same sized or even bigger version
- NEVER compress after encrypting!
- If you want to encrypt, compress it first!

Compression caveats

- works once
- Nothing more to be gained and it would only work with one layer
- Compress, decompress, (rinse repeat)
- Will work in the sense of cutting down how much you actually move
- Compression is NOT encryption but makes it incomprehensible until the decryption

- It cannot do the right thing and we have to if the firewall makes it a safe or unsafe packet.
- HTTP header compression is controversial and was put in HTTP/2

Caching

- Save things if it is being reused
- Do this over time in a single stream
- If you figure out the DNS translation, don't do it again.
- Cache it!
- If you are doing the same thing each time, we can do web browsing and we can look at a webpage that corresponds to that.
- Cache the copy of it and it saves the network traffic.
- ARP
- Process of translating Ethernet address to another address.
- Web caching is common and we can use control block sharing.
- Figure out what is the buffer size of that guy's destination.
- Roundtrip time and figure out the maximum segment size.
- If you communicate on the same destination's, that part doesn't change and we could say to set up TCP connection.
- Store away somewhere a good congestion window for this guy.
- Next time I want to start connection, look for if the value is cached.
- Same principle can apply to peers
- MSS is probably the same for all of us.
- If he has already seen it, we can save the trouble of figuring it out ourselves.

Why reuse?

- All this other information will tend to remain the same.
- Path (routing) tends to be stable
- Endpoints tend to be stable
- Why infer information when we can share?
- This has an advantage if we are all trying to get simultaneously and share a bunch of stuff.
- Fight over congestion windows and do something more cooperative.
- Less blind probing and do maximum segment size
- Try something smaller and do something that doesn't get split.
- The estimate of roundtrip time and do the lowest thing possible.
- May take a while to share this and get a better estimate.
- Somewhat safer to predict transience to a certain extent and he can tell you about it.

Project 2

- Q & A

0. Please read project requirement carefully
0. Test your sender and receiver by following the demo procedure and test files

0. Seungbae may ask us to change some parameters such as window size, timer value, or sequence number size during the demo
0. Demo will be in BH 3803 on March 10 & 11
- Use diff to check the receiver and sender so that if the diff command does NOT return anything, then file transmission is successful

Congestion control - change window size and receiver has to use duplicate ACK for the fast retransmit
- Selective repeat doesn't use duplicate ACK
- We have to make one more threshold which is slow start threshold
- When the window size meets the threshold value, you have to use congestion avoidance instead of slow start.

W 10 T Lec 3-8-16
Information delineation
- Various ways we can optimize protocols which implies optimizing the layer of the protocol
- # of shortcomings
- Deficiencies related to space i.e. congestion control, caching, etc.
- Q. End-to-end principle?
- A. If there is some thing you need to have related to a particular application i.e. voice over IP, you should put optimizations at the endpoints of the network, NOT the middle of the network.
- To ensure things get there on time, you build it at the endpoints, NOT the middle of the network. This can be used to try to identify the limited common set of everyone needs and put that in the middle of the network.
- Deal with congestion in the middle of the network, and you don't have to deal with things like compression. More things will get done on every single packet => wasting work and meaning a lower bandwidth.
- Boundaries
- Compression to shrink down what we are sending down the network.
- Use the copy he has got instead of using a new copy.
- How do we delineate the information?
- We want to receive information so how do we send this pile of information and put out limits and breakpoints to divide this information up.
- Would it make sense to aggregate things together and move it as a group rather than moving things together.

Boundaries
- Start at out the high-level where someone says I have a message I want to send.
- Can I fit one packet into multiple messages?
- At the application level, deal with general messages and the application designer doesn't worry about matching packets and message length.
- Put things into frames of certain size and the medium can only hold wires of particular fixed size.

- Dealing with boundaries
- Messages can span multiple packets
- You have a longer packet, so you will need to divide it up.
- Make sure the message and packets are the same length
- You might have little tiny packets and big packets
- You can send a message packing multiple messages together
- None: no boundary support (i.e. TCP)
- No boundary at the top level other than this is the beginning of the data and this is the end of the data

Adding markers is easy…

- We can do things like putting a length indicator
- How long is the next piece of data I want to send?
- Divide things into IP packets with a maximum size and a lot of data you want to send.
- Implicitly there is a length associated with that.
- Efficient (rapid jump)
- Will have a fixed max
- We need special symbols
- You need to look at packets and identify which part of the packet deals with what part of the message.
- You can say for example 1st 100 bytes are message 1, 2nd 100 bytes are message 2, …
- You can also use special symbols to indicate the beginning, end, etc.
- Escape sequences
- Can be as long as you like but does mean you have to scan data.
- Length fields are explicit in terms of length, so they aren't as flexible to change.

Deciding marker use is hard

- Costs
- Getting two little tiny things and you only have 10.
- With these small chunks into big packets, you have to have some delay in these cases.
- If you choose wrong, you get a lot of overhead.
- It turns out that little tiny messages are 5 bytes long (very small), so there could be header overhead.
- To merge things into one packet on both the sending end, there will be extra overhead than if you weren't trying to set those boundaries.
- Multiple packets don't have to share fate and the packets that don't take the 2nd remaining path get there.

Marker examples

- HTTP, email, SCTP
- Doing things at multiple levels where we pack a whole lot of things into a single message.

- At the level we are dealing with, we put things into one email and it all gets packed into one message.
- Another thing we can do is preserve it.
- Every message is independent and we will have a maximum length on the maximum packet length.
- UDP doesn't do multiple packets related to each other.
- Whatever you put in at UDP will come out at the bottom at the same size.
- IP is going to fragment that packet and it will do that and divide it up into pieces.
- 1,000 byte IP packets become two 500 byte IP packets
- There will likely be extra overheads so it will be more than two 500 byte IP packets.
- Preamble on the Internet is a special symbol
- Ethernet preamble was the business of sending meaningless stuff to ensure that we were using the channel and no one else was.

Definitions

- Span: messages longer than a packet
- Preserve: message matches packet
- Pack: packet carries multiple messages
- None: no boundary support (e.g. TCP)

Flows

- Like a channel….
- Have something way up at the top and we need to figure out how to pack it into one transmission and divide it up into pieces of transmission.
- Different FSM that deals with that data
- If you are running TCP and each packet gets handled by a different flow, if you have 5 TCP connections, each has their own FSM that are completely independent of one another
- Could have one channel going across the same set of links or you could have multiple senders and multiple receivers
- Share some portion of the overall paths
- Go through some set of links that are common

Examples of multiple flows

- Multiplexing
- One flow but I would really like to make it look like one flow
- Fewer headers on the messages
- You only want one thing coming out and this basically says to try to make more efficient use of channels than the flows I really have.
- This could be used in web connection (HTTP)
- Moden webpages have many connections via a server.
- They can all be related to the same HTTP request, so we can combine these flows into one TCP connection.
- This can be a better way to do things and it does give you more efficient optimizations

- If they have little to send, you can even out the information.
- How are you going to share in the information?
- Fair-sharing
- Multiple parties creating multiple flows that get combined, who goes next?
- Who do we put in there?
- Whoever has the smallest info, we put that in.
- Round-robin strategy, proportional, largest-first, shortest-first
- Whichever way you do it is dependent on how you define “fairness”
- The fair thing to do is to give everyone exactly the same thing.
- How is “each” defined?
- Would you give each person an equal amount? How are you dividing things.
- Multiplex things down to one link.
- Head-of-line blocking
- Any of you have ever shopped at the market know this phenomenon.
- Guy with 70 items is followed by guy with 2 items.
- Guy with 70 items blocks the system and the guy with 2 items has to wait a lot of time.
- Priority is an issue here!
- How do we deal with this?
- Say we have a limited chunk size.
- If it is your turn, give me 100 bytes, no exception.
- He hasn’t finished checking, so if it’s your turn, you have to pay for items 11-20.
- Separate connections
- A bunch of different lines with varying amounts of head of line blocking
- Striping
- One thing you like to do and you could have multiple channels used to do it.
- Presumably at the destination end, you will put them back together again.
- You could have a greater bandwidth since links match up to different channels
- If one of the channels has a failed link, other channels can be used in place of it.
- You somehow figure out to send 1/5th of traffic on one channel and 1/5th of traffic on channel 2, …
- Set multiple sets of hops to send different packets of TCP flow to your destination.
- You will probably have multiple connections to the Internet and when you want to put things out, you can go in multiple directions.
- Results in different paths through the Internet
- Partitioning
- Split an information stream into separate ones
- One thing into one queue and another thing into another queue.

- You can partition an information stream because different parts have different characteristics, and you want them to be treated differently.
- Teleconference (video vs voice)
- Bundle together all the bits representing video and all the bits representing voice and merge them together.
- Maybe we can split up the flow and send the audio down another stream.
- Prioritize things and we can tend to drop the video rather than bits and pieces of both.
- FTP control vs. content
- Moving data (bits to the file move)
- You have at a particular semantic level, two information streams and you can combine things
- Alternately, you can choose NOT to combine those and put those in separate streams.
- Moving the data will be of lesser importance.

Recall:encodings
- There are some other encoding issues
- Variance we have to worry about.
- Bit order issues

Bit order and formats
- Many channels exchange sequences of bits
- At the upper layer, we put them into words and we start with bits.
- We first think of what order?
- We can do it either way.
- Least significant bit going in first or most significant bit first.
- You don't know about the carries until you get the significant bits

On holy wars and plea for peace
- Gulliver's Travels
- Big endian vs little endian!

Endianess
- Big-endian: ABCD stored as A, B, C, D
- This is what some other kinds of processors use
- Telephones use this.
- Little-endian: ABCD stored as D, C, B, A
- Intel processors
- Both
- ARM
- There will have to be translation for Intel processors

Conversion
- There were still people who built 7-bit machines.
- Conversion that had to be done to make conversions.

- 32-bit words -> 64-bit words, how do we deal with that.

Marshaling

- There were two different formats for encoding English language letters
- ASCII
- EPSIDIC (?)
- Created by IBM, the dominant computer company in the world at the time.
- Over the course of time, ASCII won out over EPSIDIC (?).
- Packing and unpacking what you got in a sensible manner for others.
- Similar to what we are doing for a function call
- A way of indicating to another party that the party in our network is what we are given you and its types.

Why is marshaling hard?

- Expensive
- Conversion takes time
- Has to be done for people who want to communicate with different formats

Summary

- Details matter and they don't
- Do it right and it's quick, do it wrong and it's slow.
- You don't have to put things in ASCII; you can put it in EPSIDIC (?) but this doesn't mean ASCII is universally better.
- There are communication details that do need to agreed on to ensure effective communication.

Layer Optimization: Security and Privacy

- Change various layers to ensure the privacy effects and make it look more like it does.

Another type of layer deficiency

- The mere fact there is a copper wire says there is a fair amount of security and privacy.
- When you start moving up and saying to do HTTP across the network, we won't get those privacy settings anymore and we need to provide it for you.
- What optimizations can we perform?

What do we mean by "security"?

- Informally, we usually talk about some combination of three different properties (CIA)
- Confidentiality
- Integrity
- Availability

- If there are no bad people preventing us from achieving these properties, we won't do anything about security.
- In the face of adversaries, we are looking at people who are trying to screw us over.
- Smart bad guy trying to cause us trouble.

Security on a single link

- A relatively simple problem
- If this picture is accurate.
- Lecture 18, Page 4
- Nobody else can hear your information
- Confidentiality is good
- Nobody else can alter your message
- Integrity is good
- Nobody else can interfere with your messages
- Availability is good
- The sender and receiver want to work together

Security in a complex network

- Get to an Internet that is broadcast medium within the Ethernet.
- No longer so simple from a security perspective
- Various points where they might hear the messages.
- Hear the signals like that access point will hear the antennas.
- Each one of those autonomous systems contain multiple routers.
- If I do NOT trust the autonomous systems, it might get more conflicts.
- Others can hear messages
- Confidentiality is bad
- Others can alter your messages
- Integrity is bad
- Others can interfere with your messages
- Availability is bad
- Many of the other parties can get here as well.
- People don't have to be on the path to get this done either.

Authentication

- Optimize protocols to deal with these kinds of problems
- We frequently care about who we are working with.
- In order to have differentiated behavior, you have to know who is who.
- Is this communication coming from Mom or the Nigerian prince.
- If you cannot tell, you cannot properly make the decision.
- This is going to be a problem for people in the network, especially things at the Internet scale and it can come out at any point.
- It can come from the party we thought was sending it.
- If we are talking about the network layer, it can claim to have IP address X.
- Did it really have IP address X?

- We can have a web server from a particular IP address and it should be from a particular user.
- Should you let him buy something or not?
- In comes a bunch of bits; that's all we get!
- Based on the contents, can we figure out if it is an authentic bunch of bits and were they really created in that state.

Security

- Background
- Information protection

Basic mechanisms

- Moving bits of information
- Integrity, availability
- Protect data from being seen by other people and we want to ensure that data gets there.
- There are a # of mechanisms to work with.
- All of these involve cryptography
- Primary security tool used in computer networks.
- This is great and can protect data moving around, but we need more.
- Routers, buffers, links for certain capacities.
- We can have availability problems if we aren't careful.
- We do need to protect resources and cryptography can be of minimum use in the cases.

Security by data manipulation

- Add or change data in the packets to provide properties you want to provide.
- Add hash into the packet and see if things change similar to how we thought about bit flips.
- Detect tampering vs random bit flips.
- We can also say to set data where we want confidentiality on.
- We can convert a bit pattern to a different bit pattern via encryption
- People viewing the data will see an altered bit pattern to confuse them, but we want the receiver to be able to decrypt the encrypted bit pattern.
- We definitely don't want anyone in the **middle** to see what the hell we are doing.
- We frequently care bout who did something in the network and we want to take action.
- If you can either alter the bit pattern or provide authentication, then perhaps you can get authentication of messages.

Hash functions

- Maps a variable-length message onto a fixed-length "digest"
- More or less what it looks like in regards to hashing

Why hash?

- Scrambles messages and distributes load evenly.
- Used for many different purposes
- Distribute loads and give it to a bunch of people who have the ability to handle the load
- You can use it to hash identifies and obtain rapid lookup features.
- Hash functions can be pretty quick and it doesn't take long.
- Puts a lot of things into one bucket sometimes and allows for collisions, which could be a possible problem.
- Not a disaster if every so often, you get a collision.

Cryptographic hash

- Hash function that is useful for achieving security properties
- Takes a whole set of bits coming in and outputs a set of bits.
- Difficult to "game"
- If an attacker wants to take the song and wants to change the data, he will have a hard time coming up with something to handle that hash.
- Anti-"game" properties
- We want it hard for attackers to come up with properties to cheat the system.
- We want it hard to take the hash X and grab information or properties about A

Why not just use a checksum?

- Checksums don't protect against tempering
- Easy to generate a new message with the same checksum.
- Cryptographic hash prevents people from doing this.

Example hash functions

- Message Digest 5 (MD5)
- Recent attempt i.e. MD5, MD2, MD3, …
- Cryptographer's have a hard time of doing this right.
- Anytime someone comes up with a new cryptographic algorithm, everyone else will try to break in and learn the system.
- Weak because a birthday attack can exploit this
- Secure Hash Algorithm (SHA)
- SHA-1 was weak because of birthday attacks
- SHA-2 and SHA-3 are well regarded
- US Government designed
- In the field of cryptographer, you have the NSA who are really good at breaking things as well as creating great encryption algorithms.
- All the encryption experts have NOT seen too many issues.

What do you do with a hash?

- Publish is as-is

- This allows people to go to a place and see if data has been tampered with.
- Compare what it does with the archive site and say if it is right or wrong.
- Use it as building block in other algorithms
- Based on cryptographic hashes

Fingerprint checks

- Lecture 18, Page 17
- Each line on the left is a hash of the tar package on the right

Any alternatives to hashing?

- Protect the path
- Lock it down, seal it up, etc.
- Detect tampering
- Power loss, other physical changes
- Tampering is NOT magic either. It takes time and you need to put electrical signals that are potentially detectable.
- In certain cases, we can detect things but it is hard.
- All are very hard to do

Encryption

- Sending out original bits and anyone who is listening gets to hear our original bits.
- We use encryption to convert an easily readable bit pattern into a bit pattern that looks very different.
- Pull out the real data we want to send.
- The guy we want to do the reversal has to be able to reverse it.

Keyed encryption

- 2nd input is a secret used to perform conversion into an encrypted set of bits.
- If you know the secret, you possess the ability to convert.
- If you don't know the secret, it is extremely hard, which is what we want.
- It would be very nice to prove that no one else that is trying to get your data can get the reversal
- If I want to send data from A to B, and you are worried about people in the middle hearing the data, why do we need to do something else to keep the data secret?
- Do the same thing to get the data there.

Symmetric and Asymmetric Encryption Systems

- Symmetry here relates to the use of keys
- Symmetric crypto-systems converts using the same key to convert from encrypted to decrypted and vice versa.
- Both have the key K

- Sender uses the key K to convert to a secret form, and receiver uses the key K to get the original message.
- Asymmetric crypto-systems use separate keys for encryption and decryption
- K_E and K_D are NOT the same.
- The trick to getting a good system is that you cannot derive K_E from K_D or vice versa.

Example codes

- Symmetric
- Data Encryption Standard (DES)
- Advanced Encryption Standard (AES)
- Competition for various cryptographers
- We are using something created by a Dutch cryptographer in today's world.
- Blowfish is another example of a good one.
- Asymmetric
- Diffie-Hellman
- They just won the Turing Award (badass)
- RSA algorithm
- You use this everyday!
- Ellipsis Curve Algorithms
- Not important to know

Symmetric keys

- Also known as "shared secret"
- For the ones I showed you, it is a fact that the good symmetric crypto algorithms are faster than asymmetric crypto algorithms.
- All the asymmetric crypto algorithms known now are slow.

Using symmetric keys

- Assume plaintext P
- You want to protect the content P
- For many encryption algorithms, E and D are the same algorithm.
- Symmetry lies in the key K
- Take plaintext P and secret key K
- This pops out a bit pattern and we can give a crypto text called C
- C = E(K, P)
- P = D(K,C)
- We encrypt P with K and you decrypt the result and out comes P again.

Asymmetric keys

- Lecture 18, Page 25
- Public key cryptography
- One of those keys is NOT going to be a secret. In fact, it will be known by everyone!

- The other key will be a secret key, so only a select few know it to unlock the result.
- You have a piece of plaintext and you want to send it to someone.
- You can encrypt your message with his public key.
- Who can decrypt it?
- In order to decrypt data, you have to know the other result.
- The owner of that key knows the result.
- The only way to get the data out and get back the plaintext is to apply decryption using the private key.
- You can do it the other way around and what would happen if we encrypted plaintext with the private key.
- The other key is a public key and everyone knows it.
- Gives some pretty crummy confidentiality.
- I have another useful property: everyone knows who sent it.
- It decrypts with my public key, so it must mean that it knows my private key.
- Whoever knows the private key created the message.
- Authentication mechanism that any person in the world can know who created it.
- Anybody else in the world can determine that I was the one who created it.

Symmetric vs. asymmetric

- Symmetric: Same key used on both ends
- Achieved confidentiality among parties who know the key.
- Information that is encrypted must have been created by someone who knows the key.
- Asymmetric: keys are used as pairs
- Public key creates message only private key can decrypt.
- Confidentiality: only private key owner can read it.
- Private key creates message only public key can decrypt
- Authenticity

Using asymmetric keys

- Apply both keys to yield the original message
- $C = E(K_E, P)$
- $P = D(K_D, C)$
- Intermediaries we perform are different
- $E(K_D, P) \neq E(K_E, P)$

Digital signatures

- Let’s prove who created this pattern of bits and we are committing ourselves to these bits.
- Bits are infinitely malleable
- Encrypt private key and use it to show a signature.

- Can be done for the entire message, but asymmetric cryptography is expensive.
- What do we do to make it cheap?
- Take a cryptographic hash
- Take an arbitrary size of big data and chunk it down to a smaller piece of data
- If you encrypt a small amount of bits, it is obviously cheaper than encrypting a large # of bits
- After encrypting with the private key, you can use a Message Authentication Code (MAC)
- Same acronym as Medium Access Control (MAC)

Signatures and integrity

- Signature lets you do a cryptographic hash and it doesn't match what comes out.
- Cryptographic hash doesn't match what we computed so there could be a problem.
- Data should NOT change in transit.
- Attacker cannot generate a new hash
- If all we send is a cryptographic hash, he could have a malicious intent and modify things.
- For digital signatures, this isn't feasible, so the attacker cannot create a digital signature that matches alterations of data.

Q. You get some data and have a hash signed, what is the verification process?
A. Peter Reiher claims to have sent the message. You take the data and the signed hash, you run the cryptographic hash. You decrypt the digital signature, so now you have your own version of the cryptographic hash along with the digital signature. Compare bits to see if things match.

- Ensures authentication because we know who created the hash
- Assuming no one knows Reiher's private key.
- We try to tie it down to the guy who did it and we want to prove a guy who did it and you want him to suffer for the things he has done.
- Symmetric key prevents you from proving too much.
- He can repudiate things and say you must have sent that yourself.
- You can show it to a judge and take a public key and show the message that caused you problems.
- This assumes no one else knows your private key.

What do we have so far?

- Hash
- Integrity, fi you trust the hash
- Encryption
- Privacy if you have a key

"given a key"

- Any system that you build ultimately will be based on keys

- Certain keys must be kept secret.
- Private keys must be kept secret from everyone except those who own them.
- In a network we have a problem,
- We have to know a key and we don't trust people in the middle of the network
- We better have some other way of getting key from here to there.
- Two challenges:
- This message was created by Peter Reiher and it has a digital signature
- We can make sure it was sent by Peter Reiher so someone has to have his public key.
- How do you know it isn't an evil hacker's public key.
- Only the evil hacker knew the evil hacker's private key.
- Private keys HAVE to be kept private
- Otherwise, you are completely screwed.
- They have now lost the ability to distinguish themselves from the bad guys in the world.
- If Microsoft loses their private key, anybody in the world can pretend to be Microsoft.

Pre-shared
- This is ultimately the base of everything in security.
- Not using a computer network, we can use this to create more keys in a more secure fashion.
- Somehow, I got you a key without sending on the network.
- Everything relies on those public keys
- Out-of-band distribution: someone else' job

PKI
- Public Key Infrastructure
- Someone did work to get the keys to you.
- Get to something to deal with more common situations.
- Some kind of database used to distribute keys
- That database is used to get keys to me and we use public key cryptography to get things to you.
- We can use public key cryptography to get symmetric keys.
- Infrastructure used for distributing keys
- You can have private PKI's that are used only by a particular company.
- Database that we can access across the network with a fairly high point of scale.
- Typically going to talk about something a whole lot more distributed
- Set of in hierarchical fashion, etc.

PKI example
- PGP keys (e-mail)
- Set of servers hold public keys

- Users find keys explicitly
- Used to distribute public keys
- X.509 keys
- Handled by certificates and are the backbone of security for handling the World Wide Web
- Give you a document signed by someone else.
- My site.com is signed by public key X
- Signed by someone else's private key.
- Can only make use of that if we make use of Verisign's public key.
- Create a document that creates a document of your public key.
- Everyone who has a web browser can tell that this is your site's public key.

PGP vs X.509
- PGP
- Web of trust
- Gradually, you get relationships of people and you use this web of trust to distribute your public keys
- X.509
- Hierarchy of trust

Key Exchange
- Let's say you want to communicate and you want a new key to communicate with.
- General principle is to use keys as little as possible and change keys frequently.
- Proven to be a wise thing to do.

Key exchange using both symmetric and symmetric crypto
- What you are doing is initially using asymmetric cryptography and that key is available at both sides of communication.
- You can use this to bootstrap the cryptography
- Create a session key and send it via a secure fashion.
- He can then authenticate it came from you and determine if it is useful for communication.
- We are probably going to use the AES key

Combining Symmetric and Asymmetric Crypto
- Lecture 18, Page 41
- Alice wants to communicate with Bob
- We want to ensure that Alice created the key and that she is NOT an evil hacker
- Alice would take the brand new session key and use Bob's public key to encrypt that
- Take the secret key and encrypt it a second time.
- Send this new information to Bob.

- Only Alice could have created it because only Alice could have done that.
- Is this what should be happening or is someone trying to screw me over?
- I know that this should have been encrypted earlier with my public key.
- If I decrypt this with my private key, I should get K_S
- They will share a symmetric key that only one person could have created and the other person could have gotten it.

Q. How do we verify in the intermediate step?
A. Ultimately, you will end up with K_S and you start sending info with K_S, either information will come through and you set up a TCP segment, but you can do other things beyond just the key.

- Unfortunately, it's more complex than this (Take CS 136 to learn more!)

Diffie-Hellman key exchange
- Share a key starting from nothing
- In the clear, you shout out a # to your partner and when you finish, the two people who did the shouting share a secret key that no one else knows.
- It is based on very simple mathematics.
- Mix things irreversibly
- Multiply
- A * B = C
- Assumes A and B are prime
- Factorization is hard
- Exponentiation
- Discrete logarithms are very hard for algorithms
- You can share without having a pre-distributed key and if you share with somebody, you don't know who that person is.

What does DH do?
- DH establishes a shared secret
- A symmetric key
- But symmetric keys don't establish identity

Signed DH
- Use public key cryptography
- Prevents Man in the Middle (MITM) attack

W 10 R Lec 3-10-16
Asymmetric cryptography: public key and private key
- If you encrypt with the public key, you must decrypt with the private key or vice versa
- Make sure as many people keep it as secret as possible.
- Going to be a big long string of bits.
- 2048 or 4096 bits usually
- Private key now becomes a very precious secret for me and me only.

- I and only I know my private key, while everyone knows my public key.
- Say I have a message and you want to guarantee that I sent that message.
- How did I know I created it? They know my public key, so they will go and use their public key to decrypt what I said, and if it works, then only I could have created that message since I am the only one who knows my private key
- Secrecy since only I know my private key and only I can decrypt my message.
- If you and I each have a public key-private key pair, we can maintain secrecy by encrypting twice
- Be certain that I sent it and no one else sent it.
- Take that result already encrypted and encrypt it a 2nd time with my private key,
- You will then see that there is a message with confidentiality maintained and we know it was encrypted as well with his private key and no one else could have done that.
- Public key cryptography is really expensive; essentially, it involves complex mathematical operations i.e. exponentiation to very large prime numbers.
- Private keys are faster because they involve tables, shifts, and adds.
- Also known as symmetric cryptography

Final

- Closed books, closed notes
- Significantly different from the midterm
- Questions that require you to apply knowledge to the class to practical problems
- 5 problems
- Cumulative
- A couple of paragraphs i.e. a page of text
- No electronic devices
- Emphasis on 2nd half with readings and lectures
- No need to memorize equations
- It would be good to understand the meaning of anything mathematical

How do you guarantee that the message came from the right person?

- How do you make sure the message hasn't been changed in transit?
- Cryptography cannot do everything we want to do.
- Network is composed of machines and links that have properties
- How many cycles can you run per second?
- How many bits can you move down the pipe?
- Refers to resources.
- In addition to protecting data, we have to protect resources that let you maintain the network.
- Resources to protect
- Limited buffers
- Cryptography cannot make use of data until you decrypt

- FSM’s that represent the various protocol levels
- Unpack a message and go up through the stack takes cycles.
- Processing cycles are limited at the destination
- Content
- You put your FSM representing one of the protocol layers into some particular state
- As other messages arrive, it will change state and this can cause us to go into a bad state with undesirable behavior
- Could cause congestion at the network
- Ways to protect the endpoint
- Attackers try to overwhelm the resource
- Try to do less work!
- Get rid of some of the load being offered to you.
- We don’t want as many packets delivered to us.
- Tell the sender we cannot handle that much load and the sender will say "sure, send me less”
- If we have an attack, we need less demand for buffers, bandwidth, CPU, etc.
- If is possible for the attacker to send a bad message, you should check if this is really what I should be getting
- Verify before you ACK

Typical endpoint protections

- At the endpoint or relatively close to the endpoint
- Rate limiting
- Assuming we have an attack
- We cannot just ask our sender to send less
- Accept less and we can say if it is going to set up a connection, I am going to devote a lot of resources to this connection
- We want to be pretty sure that we have the resources and we want to service this.
- Start dropping stuff if we cannot manage it.
- Particular kind of attack made on servers called SYN floods
- Send TCP SYN’s that start up the connection
- Sends a bunch of SYN’s and never uses the connection
- This is where you want to limit your resources.
- It should NOT go through the process of setting up the connection
- If you don’t set up a connection for your customer, it becomes important to set up a connection that is real and a connection that is NOT real.
- Firewalls
- Machine that sits in the network where traffic passes through the network and to the endpoint
- The firewall’s purpose is to get rid of traffic that you don’t want the server to be handling
- Examine the incoming traffic and drop packets that you don’t like
- Packets you do like go to the server as normal

- Look at the IP address and see what port it is going to
- See which packets to drop at the firewall, but it gets more complicated at **depacket exception**
- Hard to figure out everything you should drop
- You don't want to drop anything you SHOULD NOT drop!
- There are two effects we want to achieve
- Don't drop a single packet of the stuff you do want.
- If you want TCP, we know what happens when we drop packets.
- He's going to keep sending but when he is going to back off because he sees **congestion**
- You want all those bad people who send bad things to send less packets, so we want to avoid dropping good packets.

Conditional port blocking

- NATs (Network Address Translation box)
- We don't have a whole lot of IP addresses available to us
- We will give you 1 IP address for an arbitrary n addresses
- You can route things locally in your own LAN (local area network)
- You can have an IP address for each of your own 8 machines but you cannot send anything out to the Internet
- Drop any packet that goes out to the Internet
- Assign 1 real, routable IP address to the LAN
- Anything going to the NAT box will go to 1 IP address
- This goes to machine #3 or machine #5
- Translating addresses, which is the original purpose of it.
- Obscures the fact we have 8 machines and it is good in networking terms to hide details of your network from your attackers
- Use the NAT box to filter packets as they come in.
- Conditional example
- Allow incoming packets to a port if you are expecting packets of that kind
- Done typically with UDP
- NAT box does work on this and says "This guy asked for a DNS request from the following DNS server, so I am expecting a DNS reply"
- Remember who asked!
- If I see a DNS reply coming back without a DNS request, you know that you weren't expecting that and you disregard it.
- Once you see the TCP connection closed, you say you don't want to see that port anymore.

Variable load shedding

- You might want to shed load on connections that haven't been set up yet
- This requires you to have some idea on your machine
- Frequently do this before the machine being attacked gets that message
- Usually you do this because the CPU is overloaded
- See if this is a good packet worth investing more effort in.
- Figure out if this will be good or bad work.

- Do this somewhere on a firewall.
- If I am going to have encrypted data
- Confidentiality and integrity, I will have to run cryptographic algorithms to decrypt that data.
- Overloaded and cryptography will require yet more cycles and yet more load.
- Completely opaque to everyone who decrypts it.
- What is it supposed to do with that?
- Why don't wee why IP header is encrypted on packets?
- We don't usually compress IP
- Anyone can undo compression if you know the encryption algorithm
- Take the packet and throw it out into the Internet
- If you have encrypted the header, your source and destination address has gotten encrypted
- What is the router supposed to do with the encryption address?
- Say we are going to decrypt the header before I forward it.
- Two different cases I should talk about.
- Wireless network and we encrypt our traffic on most wireless networks
- Not everything can be encrypted; the access point has to know this is a packet for me.
- I know the encryption key and we are encrypting across this one wireless link.
- When you do linked level encryption, then yes, you can encrypt across the whole header.
- If stuff comes in on the link, you know where it came from (the other guy)
- You can encrypt every single bit and that only works across a single link.
- Tunneling (network context)
- We have a protocol we want to run
- We will say we have another protocol and we will hide the protocol we want to run and more protocols we have got
- More recursion that we are putting on top of things
- Tunnel IP over HTTP
- Ironic because HTTP is above IP
- We are encapsulating a protocol inside another protocol
- Why do we want to do this?
- Maybe this is the protocol we want to run, and we can choose this by moving up and down the stack
- Security reasons: hiding our destination address
- What if I tunneled IP inside of IP?
- I am going to have my IP packet that I want to send and if I don't do anything else, I don't have to send that packet.
- Put an IP header outside of my IP packet
- Move stuff through the Internet and it is represented by the destination address
- Decrypt things and see interior packets and say this is the real destination address

- UCLA VPN is a perfect example of tunneling
- You want it to look like you are sitting in UCLA even if you are off in Starbucks in Torrance
- Get an address and if we throw this into the Internet, it should route all the responses back to UCLA and make sure we see all the results
- This can be wrapped and routed back to Starbucks
- This can be secure and we don't want random people to be able to see this
- Source address of Starbucks and a site that doesn't belong to UCLA
- When Packets come into me, they can be encrypted packets where someone would like to have delivered from UCLA and appear to have come from UCLA
- Encrypt the whole thing and wrap it with another IP header
- It gets delivered to UCLA and rip off this header.
- Decrypt the internal message and take the packet and deliver it.
- This response shouldn't be going to a site at UCLA; it should be rerouted somewhere else.
- Wrap it up cryptographically and deliver it to your machine.
- Your machine knows what we want to do here and we want to hit this layer.
- Do more work and run it through another protocol layer.
- The other layers decrypt and gets an IP packet
- The apparent UCLA address that I have
- Real one and one that I play around with for this tunneling purpose
- Look at the address and it could be really me.
- Go up through TCP or HTTP (whatever protocol)
- Q. Does this bypass any firewall?
- A. It depends on the firewall's settings. They can look at a packet, see if it is encrypted, and choose to drop it or not.
- Q. Is that how Netflix could detect a VPN? They can expect the VPN
- A. That is the kind of thing Netflix could do. They are probably going to encrypt their own packets too, but they have expectations and say if this is a VPN or not.
- Fellow in Singapore going to #5 and #18, set up enough capacity to decrypt the internal packets
- Guy in South Bay could have the following IP address and we probably want to remember what he has been up to.
- Overhead associated with this and we need to provide sufficient capacity to deal with all that overhead.
- Example of tunneling (tunneling is a much more general property)
- You can try to tunnel UDP based on TCP
- Considering network protocol layers from that point of view, it makes it relatively simple to figure out how to insert these layers
- Fiddling around with the layering and doing more recursion

- You can tunnel and if you are NOT allow do to video at a particular workplace, you want to use another protocol i.e. HTTP and put the video protocol and put it on top of HTTP
- Send the HTTP packet off to my buddy and your buddy has to pull off the HTTP header and see what the real request is.
- You can do as much tunneling as you want, but you pay a cost to do tunneling.
- Pay a cost in terms of organization and dealing with stuff for the tunnel.
- Costs in terms of bandwidth
- More layers of protocol messages
- Tunneling can lead to good abilities to achieve things that you normally cannot get across your network.

Forwarding resource protection

- Packets get forwarded
- Routers perform this and they can relay the message
- Messages go in and messages go out.
- We sent it to the right place
- Another thing that is going to happen and another place we are vulnerable
- Routers can run the routing protocol and find out where to go
- Similar to an endpoint, so we need endpoint protections here.
- Forwarding is something supposed to happen very fast and we want routing and control updates.
- Limited and predictable effects on the behavior of that router.

Router endpoint protection

- Block ports
- Limit rates
- Validate content
- See if the message is a good one or a bad one
- Depending on your routing algorithm, you get partial information
- Doing a lot of gossiping and from the point of view from integrity, we can protect routing protocols
- Routing protocol that controls the Internet.

Forwarding protection

- Packet has come in on the interface and we either forward it or we don’t
- Why would I not want to forward the packet?
- If a packet has passed through me in the past, maybe we shouldn’t forward this one because it can cause a problem as well.
- Decrypt every packet coming in and if we get a packet and get junk, we have wasted resources and we should maybe drop them.
- Alternately, if we get a limited buffer space and we get a poorly behaved packet, maybe we should drop his traffic.

- If I am not overloaded, someone else can have a problem (most likely someone downstream of me)
- Maybe it would be a good idea to stop forwarding those packets
- Drop them earlier
- There might be too many packets
- Guy at the end has too many packets and occurs in denial of service (DOS) attack
- If he could tell me, I don't like these packets, maybe we can do something about that.
- Destination ends don't have a very good ability to say I don't want those packets.
- Difficulties pushing back packets into the network.
- It is hard to defend against distributed denial of service attacks
- You call your system administrator and complain
- If that isn't good enough, we can call the system administrator and begs for that guy to drop the traffic.
- Defense mechanism against DDOS attacks
- Not very effective!

Where do we put security?

- Do we want to put it in a security layer?
- Do we have multiple security layers that may or may not get inserted in certain locations
- Issue of providing security is very broad and we cannot do it exactly in one place.
- True in what physical place in the network we put it in.
- Some potentially in the routers in the network, some in the firewall and some in the destination point.
- Generally speaking, we should shed load as early as possible.
- If I have a packet delivered to me, if I don't want this packet, we dump it out.
- From physical point of view, if I don't like this packet, I should drop this packet.
- Common example is based on IP spoofing
- Based on a false IP address
- Usually for DDOS attacks
- You can at your entry point deal with this problem
- I, the router sitting here in front of the LAN, know what to do
- I could look at the IP address and this is one of mine and this is a problem
- **Ingris filter**
- We still have IP spoofing because most people on the Internet do not check.

Summary

- Security is a problem that is shared

- We all have to work together and we should share at different places in the network.
- Perform things early rather than late
- # of tools i.e. cryptographic tools and analysis of cryptographic contents.
- Certain amount of black magic
- We won't always understand why things work (alchemy)
- Cryptography is based on the assumption of shared secrets
- Usually going to be some bootstrapping here.

Putting It All Together
What have we learned?

- We can communicate with another party over channel.
- Most channels experience noise
- The reason we can still use a channel is because we can use various kinds of encoding to overcome that noise.
- Given an initial channel capacity, we can figure out what will get through the channel

More things we've learned

- Scale up without having new channels
- Shared channels without more than two parties
- Once we have started sharing channels, we have entered a problem here we have to have names and slap on messages to indicate who is doing what.
- Names become an important problem when we share channels.
- Typically, someone will be in charge and nobody is in charge and we follow a protocol
- Do exponential random back off
- Limits to these shared channels
- Based on capacity of the channel
- The more people you put on the channel, the more traffic you have and ultimately the capacity becomes overwhelming.
- Difficult to have shared channels that go from China to South Africa
- Internet scale for greater networking, and all capable of sending messages from one place to another
- There has to be something else (relaying)
- Go through multiple hops in the relay and we need far fewer links than the N^2
- Scaling purposes allows us to get to a very high scale
- Sometimes, we get a message not for them and we have to pass it on to someone else.
- We have even greater naming needs once we have introduced relaying
- Everyone in the network has names and we need multiple names for each person in the network.
- New issues that are introduced by the fact we have added relaying to the network.

Things we've learned about relaying

- I am supposed to relay things and send things if we have several choices.
- I am going to figure out which of those choices are correct, and we can choose how to forward particular messages and routing protocols
- Connectivity and paths through the network.
- When we have relaying, we need congestion control
- We have a lot of people sharing links, which can have a fixed capacity
- We have to make sure these links aren't overloaded
- When we get very close tot the capacity of the channel, we lose efficiency
- We no longer have point to point characteristics
- Even if everyone is happy and wants to get a whole lot more, we can get congestion in the middle if people are trying to share that one link
- Magnifies security and privacy problems
- Relatively simple and if we don't want the receiver to know something, don't worry about eavesdroppers.
- We can identify them and here are 18 machines connected not he Ethernet.
- People who are within 300 meters of that wireless access point.
- Security problems can become a lot more difficult.

Protocols

- They agree that you will send a message and I will receive a message.
- Describes a set of fixed rules
- Everyone agrees to follow this and you can use the same protocol and the same set of rules
- Everyone communicates with each other as if they all used TCP
- We expect under most circumstances for people to follow a subset of rules for the protocol
- If they don't, we failed in our communication, which is generally not acceptable to anybody.
- Each person involved in the communication can be running their own component of the protocol that we are trying to use here.
- Track if you are the sender or receiver
- What is my state?
- Expecting to hear a reply or acknowledgment
- Otherwise, we cannot figure out what to hear next in the protocol
- The common way of doing this is to encode it in the form of some FSM
- Predefined as one kind of state and here is one way to move from one state to another
- We want to keep track of what is happening from one protocol to another

Protocols and layers

- No single protocol can handle something like the Internet
- Requires multiple protocols
- How will multiple protocols work together?
- Would we require all the routers in the world to understand HTTP?

- We need some way to understand in some particular circumstance to add functionality of particular protocols to functionality of other protocols
- Layering -> build protocols up in layers
- Linear stack of protocols used for any layer of communication
- Translate and we get off the machine relay up and down.
- Get to the destination and get down to the protocol that we started.
- We can go up the stack and two parties can communicate via HTTP
- Communicating == sharing a high-level protocol
- They are either both running DNS or they are both running HTTP
- Lower-level protocols which aren't as common
- We can have a DSL line a the bottom, yet they can both talk HTTP to each other.

Layering protocols
- How do we put these together?
- Recursive type of operation
- Start at the top and unwind on the other end
- In a real use, this is NOT necessarily the case.
- We have chosen an hourglass shape for an organization of protocol layers.
- Many choices at the top
- As you go down, you begin to have a few other options
- Eventually, you get down to a point that is the waist (IP) of the hourglass
- You have narrowed things down to IP
- The reason we don't do that either is because there are many hardware technologies used to provide wireless capabilities
- Sometimes we use Ethernet, sometimes we use a token ring.
- From the IP layer, we have to go down through the hourglass to layers that are much more specific.
- We have many technologies that we use at the hardware layers
- Goes from every possible layer at the top to every possible choice at the bottom

Layers and optimization
- Have the greatest assurance that our communication is actually achieved.
- The underlying network that we are working with aren't really doing everything we want them to do.
- Need raw hardware and connections we get in the network.
- Change the appearance of the network we got into the network that we want.
- We want to make our network better and we want to compensate for that deficiency
- Deficiencies
- Capacity
- Latency
- Reliability

- Security and privacy
- These are all things we can improve by choosing some layer and we provide compensation
- We do not have one single optimization layer; we optimize at multiple layers
- Anything having to do with actual signals might be easier

Putting it together
- HTTP request from client to the server

Our example
- Lecture 19, Page 12
- Carol wants a web page from www.funsite.com
- Clearly, messages have to go from Carol to the server
- To get to the fun site from her wireless LAN, she goes through two networks

Getting started
- Carol is running a web browser
- System call grabs the call and translates it to something that requires something
- We might need to send messages across the network and move into networking

Starting at the top
- Build an HTTP request message
- Like any other forms of messages, they want header and control information
- Body and what the application wants to do
- Header and body that is semantically oriented to what we want to do.
- We want to create packets of many different types and requesting a webpage
- Specifying a set of bits that are translated into electronic signals
- Request line and other header lines that require you to do other control types of things
- Request line indicates it is GET request
- Indicate whatever URL we are after
- A form of network address!
- Many kinds of addresses in the network
- Higher-level and lower-level addresses
- URL is higher-level than the IP address but it is still a network address

Now what?
- Can I handle this message at my layer?
- Am I this website?
- No!

- Thus, I cannot handle this message, so I have to forward it.
- I don't know forwarding so I need help from a lower-level and hand off my work to that layer.

Moving Down

- Lecture 19, Page 16
- Who is helpful here?
- TCP can help!
- TCP is connection oriented and we can set up a connection and use it.
- For the moment, let's assume we have our TCP connection to this server

Getting the TCP address info

- TCP doesn't understand URL's
- Consists of two parts
- IP address
- Obtained by taking part of the URL and looking up some translation
- We might have a cache (likely because we have a connection set up)
- Otherwise, we have to go into the DNS system
- DNS, create a new protocol stack and the only purpose is to get us this translation
- Let's assume we are lucky and we have a cache
- Port #
- If this is an HTTP request, it is always port 80
- Encoded into a method for calling it
- Static naming
- Relatively easy to figure out the port # based on the protocol type

Now something a bit funky

- We now build a header and this consists of the port # and the IP #
- Go down to IP which is the narrow point in the waist.
- What do we need at the IP level?
- We could of course have a header where we put the IP address in the TCP header, but why bother doing that?
- TCP always lives on top of IP!
- Just use the one you put on top of the IP header anyway.
- IP header doesn't know about ports, so we have to put it on the TCP header
- **Headers are easy to look up**
- Be able to deal with headers and look up what the protocol is like.
- At any rate, we need an IP address to deal with the TCP header
- Sometimes, we optimize across layers and we save a # of bytes by not duplicating the IP addresses

Other TCP operations

- It does all kinds of other stuff and this consists of info from our send window

- Based on our flow control, can we afford to send out more stuff?
- Congestion window, check what we could send
- We need to wait for some acknowledgments and sit on the wait state until we can get back our acknowledgment.
- Let's say our window is okay.
- Now we can move ahead and we can do optimizations in network layers and do space-related optimizations.

Now what?

- We have done all the work in the TCP layer but we have to decide if we can do it locally
- TCP does NOT know how to forward or do anything with bits on the wire!
- We move down to another layer if necessary

Moving down

- Lecture 18, Page 21
- Go with IP since we don't have many options
- When we say drop to the IP layer, we have two choices
- IPv4
- IPv6
- IPv4 is the more common choice
- We need to do something to deliver these
- Let's go with that one

Doing IP's work

- IP is a packet protocol
- TCP is a stream oriented protocol
- We don't need to set up connection for IP

Adding the IP header

- Used for network transit
- Pretty good idea of what is going to happen
- Put some bits out on the wire
- What are we going to do?
- Specify people who see this packet to know what to do with it.
- This IP header is going to contain source and destination address
- Assume we are going to put it onto the IP header

Getting the IP address

- We could get info from a local file
- If we see www.funsite.com, use 131.179.192.47
- Caching is yet another space-related optimization
- More IP work than just the headers in the addresses

More IP work

- Decide how big is the packet and do flow control and congestion control

So what have we got so far?

- Start with the HTTP request and deal with the HTTP body
- Add a TCP layer and the packet should just keep growing
- We now have HTTP Body, HTTP header, TCP header, and the IP adder (from right to left)
- Put each header at the front of the message and copy what we have before right before the header.
- Keep putting information right before the other headers.
- We can also put footers at the end but we NEVER put things in the middle!

Now what?

- From protocol stack perspective, can we deliver this locally?

Q. What layer does it keep translations?
A. Caches are available for anyone who wants to use it. It isn't so much one layer but rather another optimization that will still work.
Q. At what stage does it try to get to the destination IP address?
A. It will typically be done at the IP layer. Theoretically, we should always assume we are using IP. We would NOT call the DNS and if we do, it gets done in the IP layer.

- Build an OS where we know we need the DNS layer translated. Ask for a DNS translation and get some other work.
- IP doesn't relay and it isn't a network layer.
- It doesn't know how to use a channel and how to make a wireless network on the Ether.
- It cannot get it off the node and this isn't the right node.
- It moves down the stack.
- IP is isn't part of the stack!

Moving down

- Lecture 19, Page 28
- We need something that gets bits off my machine
- A matter of what pieces of hardware you have
- We want to move down to a lower layer that can help
- We know it has to go off the node and 802.11 is our only choice

Doing 802.11's work

- There can be a wireless access point that receives anything sent by the hardware
- MAC address is built into 802.11 address in my machine
- We know our MAC address of that particular router

So what have we got so far?

- Lecture 19, Page 31
- We have added the 802.11 address

Now what?

- From a protocol perspective, we can get this packet off the network.
- It can request a piece of hardware to get a packet of the network.
- Everything gets done with hardware, and we aren't going to worry about the signals for the wireless links.
- CS M117 covers this!
- We could give it another packet and do something until our wireless hardware is ready to send another packet.
- Sooner or later, the buffers are empty and the wireless hardware is busy
- Not a dedicated link but rather a shared link.

One more wrinkle

- CA is collision avoidance
- We don't want to have collisions
- It is hard and wireless cards typically don't listen when sending.
- When sending they don't listen!
- Listen for a little while and make absolutely sure that you don't hear anybody transmitting
- See if somebody else is transmitting
- Someone else might clobber your packet.
- Your antenna might not be good enough to listen as you send.
- If the wireless access point gets your message and it should send you an ACK
- Do some random backoff.
- This is functionality done for this particular layer and we wouldn't do this on Ethernet or a T1 line.

The next step

- Lecture 19, Page 34
- Ask if the message is for me?
- Pop off the 802.11 header form the stack and we move up to the IP header.

Now what?

- How do we choose?
- We have a table that tells us and we pop up the layer.
- Go out to the DSL link and check
- This is a point to point link
- Frequency defined and a unidirectional channel
- A lot like a Shannon channel
- We might need some CSMA/CSA stuff
- Get by with stuff that is a lot simpler

Over the DSL link to router A

- Mostly moves packets and we don't want to go up and down stacks here
- Use whatever link protocol and say this is an IP packet

- If we are not going up the stack, deal with it quickly

Where did that table come from?

- By running a routing protocol at some point.
- Go up a stack and up the BGP level
- Deal with stuff at the BGP level

What's being moved?

- Lecture 19, Page 39
- We are going to move through without some alteration and make sure we don't go to another level.

Up through the stack

- Internal addresses we mentioned in an earlier lecture

Moving up

- Lecture 19, Page 42
- Went through IP header and got it to the right place
- Deal with TCP packets and handle receive window and handle the ACK
- Congestion control, ordered delivery
- What does TCP header tell us to do next.

Continuing to move up

- We figure out who gets HTTP messages and deal with the HTTP protocol
- Web server code will deal with the HTTP protocol
- Figure out what to do with the header

And then?

- ACK the TCP packet and ACK the 802.11 frame.
- Create a response for the HTTP message

What else might be going on?

- The real world networking is very complicated.

Are we happy with our own transmission?

- What more could we want?
- We might encryption, quality of service guarantees, etc.

Other things we might want

- Shove in new protocol layers
- Add encryption

Summary

- Modern networks are very complex
- Barely talked about multicasts at all
- Deal with details coming out

- Limit problems to particular pieces of code and we want to be able to handle all the problems in all places
- Recursive composition - handle heterogeneity in huge networks
- We have to have optimizations at different layers
- Introduce a great deal of more complexity

W 10 W O.H.
Lecture 10 Questions
Q. What is the difference between nodes and interfaces?

Q. What is the difference between strong and weak endpoint models?

Lecture 11 Questions
Q. What are POTS on Lecture 11, Page 16

Q. What is *PROM (EEPROM)

Q. Compare IPv4 and IPv6. When would you use either in a situation?
There are a # of things in IPv6 that is better, but in actuality, we use Pv6 for address space. IPv4 has 32 bit address space and IPv6 is 64 bit address space. What is the biggest space and it turns out you needed a lot more than that (billion nodes) out there.

- IPv6 allows you to have more addresses
- Characteristic that told you whether you want to use IPv4 or IPv6
- Divide things into pieces and we will tell various people you will have this piece vs that piece
- Hand out pieces of addressees in various ways
- Handed it out to friends and other companies.
- Some people set aside some address spaces for each of these different companies
- U.S. and Euro-centric.
- America got a bunch of address space and this starved the rest.
- It was foreseen that this problem was arising, so let's have a version of IPv6 and everyone was aggressive about going to IPv6.
- Each country wants all things being equal so we want large blocks of an address space.
- We want them to more or less have access to the same physical location.
- When they were thinking about physical contiguity, we want packets to probably end up going to some place nearby.
- Rather than dividing into longitudinal and latitude.
- IPv4 and IPv6 are protocols for sending datagrams across a network.

- ARP is a protocol for a LAN. Just a way to check if a name is unique. You hook your machine up to an Ethernet and you need to find where the other IP addresses are.
- Need a table where you Ethernet connects outwards.

- If you see an IP address coming in, we don’t want him to have to know that.
- What is the Ethernet address for that?
- It seems they serve different purposes, does this mean both protocols interact at the IP level?
- One of the nodes sitting at the Ethernet also has connection to a different point on the Internet.
- If this is the only guy connecting on the Ethernet, it’s

Lecture 12 Questions
Q. What are the breadcumbs in the envelope example?

Lecture 13 Questions
Q. Why do we need a leaf table and a link FSM in addition to a root table?

Q. How does the process of path matching work?

Lecture 14 Questions
Q. A high-level overview of the Bellman Ford algorithm

Q. What does BGP refer to in routing?

Lecture 15 Questions
Q. What does the end-to-end principle refer to?

Q. How can we optimize the end-to-end principle?

Additional Questions
- If that is the case, he might be running some kind of routing protocol and this could be our routing information and we know how to get to E and F
- At that point, A knows what path it needs to take in two paths
- Build layers for a particular transmission
- Build the DSL protocol?
- The DAG table is to tell you how to change your names into the other layer!
- Compose together protocols and do name translations
- Is it a per case basis or is it a general conversion algorithm?
- More conceptual than real!
- Things built into the OS and effectively do the same thing rather than the ad hoc ways.
- What is a proxy in this case?
- If I went to send to node W (middleman) it is somebody who can take care of it for me.
- I could go to node M or node N
- Node W doesn’t have to be a node at all! It could be either node M or N
- Maybe if I go to one of them, I don’t have to go elsewhere.

- Is this the same proxy that emulates some sort of behavior on the Internet?
- Q. DSL is a type of hardware connection used to connect up, and it is a twisted pair. This is a 0 or 1 and this is not necessarily the same from a fiber optic cable.
- Fiber is beginning to take over. We don't just have fiber optic cables connecting up centers, and now Verizon leads into your house just like any other. Much higher capacity and when you buy a plan at your home to get a certain capacity, the fiber optic cable can carry a whole lot more for that.
- It could carry a whole lot more than it does.
- Get a request from interface A and if we are doing flooding, what is F trying to do?
- We need to send it to a bunch of people and if F has heard it.
- The way the starting node finds out is if it floods.
- You can do something where a guy is interesting in flooding everyone's information.
- He would flood to B and he would be finished and hear it from B.
- I have X and Y and let's send it to both of them.
- Q. It terminates because there is no more non-adjacent nodes?
- A. There is a problem with this algorithm. If there is an acyclic graph, it is a tree and there is only one path and thus, we will use every link once and we will never do it again. If we do very dumb flooding, it will never stop.

In this algorithm itself, how do we get the answer back to J or F?

- If it is something I am telling about me, if we are asking for a query, tell me something and we have to do some kind of reverse message.
- Flooding is not specific to building your path?
- You can do all kinds of things with flooding and it kind of makes sense if you want everyone to know how to get to everyone.
- Hop counting - dropping the count: Usually there is a practical question and you could start wherever you want
- There may be a path that long and there aren't any paths longer than you got.
- Take a guess and optimize space.
- Drop the count for every link you cross and you have a hop count of a million.

W 10 Dis 3-11-16
Distance vector : link cost changes

- Z will advertise its information to its neighbor.
- Node y will also receive node z's update
- Node y can reach node x with cost 1, and node z's information can reach node X with cost 2.
- Node y does NOT update anymore
- Distance vector, when it is only 2 that it goes up to, we just need to go cheaper

Dijkstra's Algorithm

- There are all nodes v
- If v is directly connected to its neighbor, we can have D(v) = c(u, v)
- This will be the shortest path value
- If it is not a neighbor, D(v) = infinity
- It tries to find a shortest path and chooses only the D(w) minimum value
- By using node w, we can reach node y
- Updated as 11 and the link cost is 3.

Small networks have to use ISP's to reach the other network

Interconnected ASes

- Inter-AS tasks
- Let's say AS1 can reach AS3
- To advertise this information to AS2, it can have its routing table reach AS3 by using AS1.
- Example: setting forwarding table in router 1d
- Network x is connected to AS3
- Router 1d is sending a message to network X.
- AS3 also advertises its reachability to AS1
- It can reach network X by using this

Example: setting forwarding table in router 1d

- Forward a message and 3a can forward the message
- Decide where to forward and this is not just one disk job
- Gateway ahs to decide which path to use and that is what inter-routing protocols do i.e. BGP

Example: choosing among multiple ASes

- In general, internal routers have different port destinations

ASPATH Attribute

- Add your own AS # on the path
- Advertise on to AS 1239 and AS 3549
- These advertisements can be reached here to AS 12654
- This AS receives two advertisements from two different AS's

Q. P21. Suppose that your department has a local DNS server for all computers in the department. You are an ordinary user (i.e., not a network/system administra- tor). Can you determine if an external Web site was likely accessed from a computer in your department a couple of seconds ago? Explain.

A. If we know the cache contents in the CS department.

Q. P20. Suppose you can access the caches in the local DNS servers of your department. Can you propose a way to roughly determine the Web servers (outside your department) that are most popular among the users in your department? Explain.

A. DNS cache will save it using temporal locality, so you can get it for fast access instead of running through the DNS caching. Cache just has a mapping for a particular thing. You don’t have a fall-through cache, whatever is kicked out is gone. You cannot keep track of things that are gone. If you access it again, it will be rewritten again.