ECE438: Communication Networks

Powered by Red Bull GmbH

Pradyun Narkadamilli

Contents

1 Introduction

- Loose definition of a **computer network** - collection of network devices connected together that implement protocols and othe rfeatures to communicate data between devices
  - Fundamental job is to move data from point A to point B

2 Internet Design Goals

- Some primitive network performance metrics
  - Bandwidth (number of bits send per second) - dependent on hardware and network traffic conditions
  - Delay (total time to get a bit from point a to point b) - dependent on distance, congestion, drops, etc.
- Two approaches to sharing networks
  - *Reservations* (also called circuit switching)
    - Reserve bandwidth needed in advance, set up a path (*circuit*) and send data
    - Must reserve bandwidth for *peak* bandwidth
    - Can lead to wasted bandwidth (might not always be transmitting at the same bandwidth)
    - Low overhead in terms of maintaining the link (not much metadata to send regarding link maintenance)
    - Reliable bandwidth
    - Circuit switching does not route around failures - if a link on path fails, you must establish new circuit (but this is going to happen frequently)
  - *On Demand* (packet switching)
    - Break data into packets
    - Full bandwidth dedicated to a single packet
    - We now regulate the *number of packets* being sent per connection rather than separating a single connection between connected nodes
      - e.g if you have a packet, send it, YOLO
    - Instead of insufficient bandwidth problem for multiple clients, you now have an arbitration problem between multiple clients (when passing out to a single connection)
    - Arbitration solved by buffering packets if connection inavailable, but buffer space must be carefully architected such that overflow is unlikely
    - Packets carry a header (that network will use) and body (for end destination/application)
      - Header contains routing info, error correction information, size, etc.
      - Transmit destination/source info to send ack (in cases where order number if not required)
      - No resource underutilization, blocked connection, per-connection state, no setup cost

        - Unpredictable bandwidth, unpredictable delay/latency, packet header overhead means extra bandwidth needed
    - Circuits have better application performance and are more reliable, packets offer better "bang for the buck" in terms of resources, better failure recovery, faster startup-to-packet latency

- *Statistical Muxing* - we can assume that peak of aggregate load is much less than sum of each connection's peak load. Thus, we should share resource instead of partioning

3 Internet End-to-End Overview

3.1 The 4 Fundamental Problems

- Some fundamental problems of newtorking
    - *Naming* - how to find destination?
    - *Routing* - how do I find a path to destination?
    - *Forwarding* - how do I send data along that path?
    - *Reliability* - how do I handle potential failures while traversing this path?

3.1.1 Naming

- Network address (e.g IP address) says where the host is located
- Hostname identifies the host (domain name, for example)
    - Needed for human readability, host relocation (changing IP address), and improved user service
- The name->address translation is done by the *Domain Name System (DNS)*

3.1.2 Routing

- When a packet arrives at router, a routing table determines which outgoing link to send the packet from
    - This is computed via routing protocols
- Routing protocols are *distributed* algorithms (no single router sees entire network topology)
    - Messages exchanged to gather enough info about the network topology to determine a route
    - Compute paths via known network topology
    - Store forwarding info (use link N for destination X) in each router - this is called the *routing table*

3.1.3 Forwarding

- Refers to queueing and forwarding packets at switches and routers
- Switches will need to have buffers internally to account for simultaneous accesses to the same output port
  - *Input Queueing* buffers all the packets per input port if there is contention on an output port
    - Can lead to head of line (HoL) blocking (if there is a contesting packet followed by an uncontesting packet, then the second one is needlessly delayed)
  - *Output Queueing* put the queues at the output port - no unnecessary blocking, but you need much faster memory access speeds
  - *Virtual Output Queueing* has virtual queue per-output port per input port (is usually a shared memory with per-output FIFO)
    - Suffers a speed penalty but it's the most used nowadays
  - Switch does two things when a packet enters (on VOQ model for example)
    - *Queueing*: when a packet arrives, store it in input queue, look up destination address from header, use routing table to find correct output port, store it in that virtual output queue
    - *Forwarding*: when outgoing link free, pick packet from the VOQ, then forward the packet
- In order to forward, packet requires (minimally) destination/source address and a data load
- Switch latency separates into two components
  - *Processing delay* comes from the switch deciding where to put the packet (needs to unpack header, write to memory, etc.)
  - *Queueing delay* comes from network characteristics (like high load on a single outgoing link), and is dependent on network load
    - Very high network loads or certain network patterns can cause packet drops

3.1.4 Reliability

- Packets can be dropped along the way - buffers can overflow, routers can crash while buffering, links can cause bitflips and whatnot
  - How to guarantee safe packet delivery on unreliable network?
  - We need to have *no* chance of false positive, recovery procedures in the event of a negative, and a high success rate
- Need to ask two questions about reliability
  - From an architectural perspective, reliability responsibility can be assigned the host or the network - which should it be?
  - From an engineering perspective, *how* do we engineer the reliability methods?

3.2 End-to-End

- Outside of the network design questions outlined above, we still need to consider how the network interacts with host
  - How does host send packet to correct process? (mechanically, but also how to know which process)
  - Who puts source/destination info on the packet header?
- Above questions answered by *end-host* stack
  - Process opens a socket when it wants to interface network, and socket associated with a port
  - *Socket* is an OS-level mechanism to connect process to network, *Port* is the number identifying a socket
  - Port number of destination used by OS to direct to a socket
- The overall flow is as follows
  - Application opens socket, connects to network
  - Name of website is mapped to address via DNS
  - Network will embed source/destination address and port for packet header
  - Router constructs a routing table via distributed algorithm
  - Do switch magic to route the packet to destination

4 Internet Architectural Principles

- We want network to be modular - add/remove devices (in the case of the internet) without needing to modify the state or overall architecture of the network
  - How to break system into modules? (*layering*)
  - Where to implement modules? (*end-to-end principle*)
  - Where to store state? (*fate-sharing*)
- Ideally our modularity is done such thtat we have long-lasting interfaces

4.1 Layering

- We consider our network's functionality to be separated into 5 layers
  - *Physical* - Transmit data from point A to point B over some communication medium
  - *Link* - forward data packets between neighboring network elements
  - *Network* - routing data across global network elements
  - *Transport* - (reliably) deliver data between host processes
  - *Application* - use ze data

- Fundamentally, we have layers interact with adjacent layers *only*
- Layers simple with single machine, but layers across machines require clean and smart partioning of responsibilities
  - A router must get the data off wire, forward to next router/switch, and is a part of the global process of network routing
    - This means router implements the physical, link, and network layers. There is no involvement with host processes, so no transport layer implementation.
    - By contrast, a switch only receives data off wire and sends to next router/switch *without* taking part in global routing
      - Switch only has physical and link layers

4.2 End-to-End Principle

- **Definition**: if a function can be fully/correctly implemented *only* with the state of the application at communication endpoints, then that function cannot be a feature of the communication system itself
  - Maybe we can have an *incomplete* version of the service in the system for performance, but system cannot do the whole thing
- Just know the high-level definition and trust it - most people don't get it, and that is okay (quote from Saksham garu himself)

4.3 Fate-Sharing

- **Definition**: we want to collocate state with the entities that use it. That way, if the entity fails so does the state. If the entity is alive, so is the state. *They share the same fate.*
  - This approach minimizes dependence on other network elements, and improves reliability on a per-node level

5 Transmission Foundations

Goal: discuss physical layer behaviors, what is a signal, etc.

5.1 Signal Overview

- *Signal* is just a value/measurement that varies as a function of an input - usually time
  - We consider a *symbol* to be a signal that carries information
- *Baud Rate*: fancy way of referring to symbol rate (symbols/time)
- Signals have both time domain and frequency domain representations
  - Use a fourier transform to project signal onto frequency basis

- *Frequency Band*: area of the frequency spectrum that we send our signal on
  - *Bandwidth* is the length of this region (in Hz)
  - Larger signal bandwidth means higher frequency signal transmission possible
- Data rate might be referred to as data bandwidth in some literature
- *Spectrum* of a signal refers to aggregation of different active frequency bands
- Frequency domain data transmission preferred for two reasons
  - Noise resilience (time-domain signals can easily be corrupted in short-term)
  - We can send different types of information concurrently using different frequency bands

5.2 Basic Signal Transmission

- Even if we use high frequencies as carriers, we modulate the frequency domain coeffs down to a lower-frequency
  - e.g we do not need a 5GHz CPU to use 5GHz WiFi
- *Modulation*: Alter a signal at the carrier frequency to carry the specified baseband signal
- We consider a f_c carrier signal and a f_b baseband frequency - baseband signal can be keyed onto carrier signal in two ways
  - *Amplitude Shift Keying*: ASK means we use varying amplitudes of the carrier frequency to represent baseband signal
  - *Frequency Shift Keying*: FSK means we use varying frequencies in the carrier band to represent the baseband signal

5.3 Noise and Error

- *Bitrate*: target bits to transmit per second
  - Calculated as `baudrate * (bits per symbol)`
- *Bit error rate*: also called BER, refers to the corruption rate of a single bit
  - Caused by noise and interference
- We can use two metrics to determine signal strength relative to noise
  - *Signal Noise Ratio (SNR)*: $\frac{P_{signal}}{P_{noise}}$
  - *Signal to Interference and Noise Ratio (SINR)*: $\frac{P_{signal}}{P_{noise}+P_{interference}}$
- Signals decay when transmitted over a medium - power of signal at receiver: $P_R = \frac{P_T}{r^2}$
- **Shannon Capacity**: Best bits-per-second we can do given a certain SNR and transmission frequency B

  - Described by $C = B\log_2(1 + \text{SNR})$

- *Packet Error Rate* or PER will refer to the probability that there are $N \geq 1$ unrecoverable bit errors in P bits
  - Some bits are recoverable because of ECC (coding) - e.g redundant bits
  - Coding can recover or detect bit errors
- *Throughput* - number of correctly delivered bits per unit time
- *Goodput* - number of correctly delivered application layer bits delivered per unit time

5.4 Characterizing Transmission Latency

- *Inter-Packet Time (IPT)*: Time between end of packet 1 and start of packet 2
  - Packet gaps can be created by link business or application layer delays
- We can break down packet tranmission latency into some sub-components. Suppose we transmit a packet of L bits at R bps.
  - Transmit Time - Time for transmitter to write packet to the medium
    - Time for Tx to write packet to wire is L/R
  - Propagation Delay - Time it takes for packet to cross the medium
    - Example: delay for data to propagate down a wire, or across the air
  - Processing Time - Time it takes for the receiver to process your packet
    - Need to read header, decide the route, etc.
  - Queueing Delay - Time that packet needs to wait in queue because of link busyness. Happens for two reasons:
    - Outgoing link slower than incoming link
    - Receiver is routing multiple sources to same destination (contention)
    - With an arrival rate of a packets, we expect delay $\frac{La}{R}$

5.5 Multiplexing

- Fundamentally, link needs to be shared by multiple users. Need to pick a method to do so.
  - Methods will be different for circuit switched vs. packet switched networks
- **Circuit-Switched Network Methods**
  - *Time Domain Multiplexing*: One user gets the link at a time, and rotate link possession between users
    - Simple, but requires synchronization
  - *Frequency Domain Multiplexing*: Split the frequency band for this link between the different users

        * Harder to implement, but requires no synchronization

- **Packet-Switched Network Methods**
  - *Statistical Multiplexing*: Dynamically allocate bandwidth to each channel ad-hoc. Total bandwidth of link will be less than aggregate peak demand. Because of the way burst-y traffic works, this can lead to congestion and oversubscription for the link if multiple clients hit peak traffic at the same time.

6 Application Layer

- Some big things to consider when writing a network app
  - Programs should run on different end systems, communicate over *a* network
  - Not much software written for devices in the network core
  - Application being mainly in the end devices allows for rapid development and propagation

6.1 Principles of Network Applications

- Want to consider 3 main application architectures
  - *Client-Server*: Server is a network edge with a static IP address, in some cases a server farm for scaling. Clients will communicate with the server. Clients can be intermittently connected, and may have dynamic IPs. Clients do not communicate w/ other clients.
  - *P2P*: Serverless architecture, arbitrary number of end users (not clients, technically) can communicate directly. Peers are intermittently connected, with dynamic IP addresses. More scalable than C-S arch, and better overall security, but can be hard to synchronize.
  - *Hybrid*: Merges the above architectures. Can use a server for "matching" then make the end-to-end communication direct.
- Napster utilized a traditional C-S architecture, with a central index server, while Gnutella had a fully P2P architecture
  - Napster has potential reliability issues and bottlenecking issues, Gnutella has stability issues and broadcast storm
- *Process*: Some host program
  - *Client Process*: initiates communication
  - *Server Process*: awaits contact
  - P2P/serverless architectures implement both
- Two types of process communication
  - Same host, processes will use *IPC*
  - Inter-host comms done with *messages*

- Process uses a *socket* to send and receive messages
- Process needs to be addressable via a *identifier* to receive messages
  - Consists of host's IP address and the process port no.
- Applications need to have a *protocol* defining message types, syntax, and semantics
  - Public domain protocols include HTTP and SMTP
  - Proprietary protocls also exist, like KaZaA
- Applications have constraints on bandwidth, data loss, and latency - need to pick *transport protocol* appropriately
  - *TCP* is a protocol geared towards preventing data loss: it offers reliable transport, flow control, and congestion control. Does not provide latency or bandwiidth guarantees
  - *UDP* makes no reliability guarantees, nor does it provide any of TCP's features. It also does not have a latency guarantee.

6.2 Application Layer Protocols

6.2.1 Web and HTTP

- Web page has a set of objects that need to be retrieved to render
  - Images, CSS files, HTML file, etc.
  - Every object being retrieved for render can be addressable via a URL
- *HTTP* or hypertext transfer protocol is web's application layer protocol
  - Assumes a C-S arch, and uses TCP for transport (usually via port 80)
  - Protocol is stateless - each request is treated independent of any past requests
- Two big flavors of HTTP connections
  - *Nonpersistent* HTTP will transfer no more than 1 object per TCP connection
  - *Persistent* HTTP can send multiple objects over a single TCP connection
  - HTTP/1.1 uses persistent by default, HTTP/1.0 is nonpersistent
- Persistent HTTP means that the server will leave the TCP connection open *after* sending the response, and wait for subsequent messages over that same connection
  - Less OS overhead, since you don't need to keep creating TCP connections for new objects
  - Allows for *pipelining* - while parsing the current response, we can send new requests for referenced items
- Different methods supported by 1.1 and 1.0
  - **1.0**: GET, POST, HEAD

  - **1.1**: GET, POST, HEAD, PUT, DELETE

- Defining non-obvious methods
  - POST: uploads some data to the server in the body of request
  - PUT: uploads a file in entity body to the path specified in URL
  - DELETE: deletes file specified in URL field
  - HEAD: leaves specified object out of response

- Cookies can be used to coordinate user-server state
  - Has four big components: cookie header line in HTTP response/request, a cookie file on user host, and some backend database on the server
  - Allows for authorization, inter-session state (like shopping carts), etc.

- *Proxy Servers*: Intermediary between client and the origin server. Can be used to fulfill client request without bothering origin server
  - Can be called a *web cache*
  - Proxy will return the requested object if it has it, otherwise it requests from origin server
  - Helps reduce response time, traffic on outbound links, and can improve connection quality to "poor" providers

- Benefits from a "conditional GET", or "CGET", which is an HTTP request that only sends back the file if the current cache information is invalid (based on the timestamp in the request)

- *CDN* or Content Delivery Network is a large-scale cache
  - Can be intercontinental, and is more distributed than our simple proxy server idea
  - Lowers latency, less bandwidth requirement at source, better availability

6.3 P2P

- *Central Directory*: server that every peer will connect to - informs server of IP address and content
  - Other peers can then lookup "contact info" for desired peers
  - The "napster architecture"

- By contrast, Gnutella has no "local server" - everything gets ping-ponged across other peers

6.4 DNS

- *Domain Name System* is a distributed database implemented via many name servers
  - Used to perform name/address translation (e.g resolving names)
  - This is implemented as an application-layer protocol

  - Provides hostname-to-IP translation, host alises, and load distribution (multiple IP addresses per name)

- We want a decentralized DNS to avoid single failure point, better distribute traffic, and to minimize distance
  - Ping a "root" DNS to find suffix-based DNS server, then find domain-based DNS server from that one
  - Domain-based DNS server is queried to find IP

- *Root Nameserver* will contact authoritative nameservers to find an unresolved name
  - There are *13 total root nameservers* globally
  - Contacted by *local* name servers when a name cannot be resolved

- *Top-Level Domain Server* (or TLD) is the suffix-based server we referred to earlier

- *Authoritative DNS Server* is the the domain-specific DNS server (technically organization-specific)

- *Local Nameserver* does not directly belong to the above hierarchy - basically acts as a proxy server for DNS lookup

- DNS querying can be done *recursively* or *iteratively*
  - **Recursive**: The DNS server will make the next required request, chains until it sends back a resolved name
  - **Iterative**: The DNS server will tell us which server to request from next - we keep going until resolution

7 Transport Layer

- Four goals for reliable transfer
  - Correctness (TBD)
  - Fairness - every flow must get a fair share of resources
  - Flow Performance - latency, jitter, etc.
  - Utilization - want to maximize bandwidth utilization

7.1 Correctness

- We define a two-point transport reliability criterion
  1. The transport layer should *attempt* to make progress
  2. The transport layer resends any dropped/corrupted packets

- If transport "gives up", this should be known to the application
  - Transport requires a feedback system to indicate packet was received
  - Either an *ack* or *nack* (nack can be used to say corrupted data)

7.2 Reliable Transport Methods

- We can devise a simple single-packet solution
  - Send packet, set a timer
  - We expect an `ACK` from the receiver when packet arrives
  - If no `ACK` seen by the time timer expires, we can resend packet
  - The timer ensures fairness without blocking on this packet
  - Latency better w/ small timeout, Throughput better w/ large packet
- Single-packet solution cannot trivially scale to multiple packets - one packet in-flight at a time
  - This leads to low throughput
- We can use a *window-based* approach for multiple packets
  - Some W in-flight (no `ACK` yet) packets at a time
  - Swap out packet for a new one when `ACK` is received
- Window does three big things
  - Takes advantage of large link bandwidth
  - Limits bandwidth used by a single flow (congestion control)
  - Limits buffering needed at receiver (flow control)
- Ideally, we align the size W such that first `ACK` is received right when all W packets sent
  - This translates to RTT $\times B \approx W \times$ size for minimum link BW B
- We define a new constant *Bandwidth Delay Product*: BDP $= B \times P$, for propagation delay P
  - Note that $B \times$ RTT $= 2$(BDP)

7.3 Feedback Types

- Need to consider 3 big things when designing feedback systems in reliable transport
  - What should `ACK` indicate when many packets are in-flight?
  - How can we detect packet loss, outside of just using a timer?
  - How do we respond to packet loss - is retransmission needed?
- There are a couple different flavors of `ACK`'ing mixing and matching these attributes
- *Invidual Packet ACKing*
  - Each `ACK` indicates fate of a single packet
  - Packet retransmitted if `ACK` not received
  - Every loss of an `ACK` requires retransmission

- *Full Information Feedback*
  - `ACK` lists all received packets (highest cumulative `ACK` + any additional packets)
  - Packet losses are obvious, retransmit any gaps in the `ACK`
  - If `ACK` is dropped, the next `ACK`'s info will encompass this one - offers redundancy
  - More complexity on receiver's end, since constructing the `ACK` takes more work
- *Cumulative ACKing*
  - Compromises between the two previous forms of feedback
  - `ACK` contains the number of the highest consecutive transmission (no additional packets)
  - Duplicate `ACK` indicates a packet loss, triggers retransmission
  - To be robust to reordering on the receiver end, we wait for some k duplicate `ACK`'s before retransmission
  - Receiver can store more complex state, so filling the "gap" in sequence can fast-forward `ACK` number
- *Go-Back-N Protocol*
  - Receiver will send cumulative acks but will not buffer OoO packets (only buffers next expected)
  - Sender sets a timer on transmission, on timeout *entire window is resent* (any packet drop implies all subsequent packets dropped)
  - Lower memory requirement from the receiver, since you aren't coalescing packets anymore
- *Selective ACK Protocol.*
  - Receiver sends acks per-packet (not cumulative), and will buffer OoO packets
  - In addition, receiver keeps a window of size W (in lockstep with sender), acks packets that are within W of current seqnum
  - Sender uses a *fixed size window* - on timeout, retransmit only for the packet that caused the timeout
    - Sender advances window based on largest *inorder* ack

8 TCP

8.1 More on Transport

- Transport layer offers "pipe abstraction" - process-to-process communication. Two big pipe types
  - *UDP*: Unreliable Packet Delivery - messages are all single-packet, since we cannot guarantee packet ordering

    - connections are not tied to the destination, you basically just shoot darts at a wall and hope they land
  - *TCP*: Reliable Delivery - bytes are inserted into pipe by sender, magically all come out in order at the other end
    - Transport protocol has to do the magics

- **User Datagram Protocol**
  - Source sends packets, destination receives them
  - Does not care about loss, reordering, etc.
  - Corrupted packets can be dropped with no real issue
  - This is a *minimal extension of IP* - we are simply getting packets from point A to point B

8.2 Overview

- **Transmission Control Protocol**
  - The GOAT
  - Sends/receives a *bytestream*, not discrete messages like UDP
  - Delivery is assumed to be reliable (no drops) and in-order
  - Might be delayed sometimes, but bytestream will *always* reach the other side
    - Reordering, corruption, delay, loss, apocalpyse
- TCP packets are sequenced based on *byte number* - this is essentially your sequence number
  - Window size is expressed in bytes intead of in no. of packets as a result
  - TCP doesn't really care how big the packets are, just that the bytes sent are received as is
  - TCP is *full duplex* - forward and backward communication can happen simultaneously
- TCP has 4 big components
  - *Connections* - need set-up and tear down
  - *Segments, Sequence Numbers, Acks* - splitting stream into chunks, then sequencing/handshaking chunks
  - *Retransmission* - if data is lost, we need to retransmit (based on dupack and timeout)
  - *Congestion Control* - need to dynamically restrict bandwidth to match network path caacity
  - *Flow Control* - need to make sure we do not "overwhelm"the receiver
- **Connections**
  - Each bytestream or "session" requires state on the host
    - What packets have been sent and are unacked?

        * What packets were received, and was there any out-of-ordering?

- **Segments/Sequence Numbers**
    - TCP will dump a segment to the receiver when you either reach the maximum segment size or reach a timeout
        * Timeout starts when the first piece of data in current segment is put into the bytestream
    - Connection needs to be set-up with a 3-way handshake to synchronize sequence numbers
        * *SYN* prompts a *SYNACK* which contains the sequence number of the other side, then *ACK* to ack the *SYNACK* with correct seqnum
        * *SYN* is sent by the connection initiator (usually the initial sender), and it contains the ISN of the initiator
    - Since port can be reused, the initial sequence number (ISN) is persisted between uses instead of reset to 0
    - Teardown happens via a *FIN*, prompting a *FINACK*, which in turn prompts an *ACK*
    - TCP uses *cumulative acking*
- **Retransmission**
    - Loss can be detected with either *duplicate acks* or with *timeouts*
    - Duplicate ack means that *a* packet was dropped, but overall stream is still making it across
    - We also set timer for *Retransmit Timeout* - if this fires, then we retransmit the packet for the "next byte"
        * Linux uses a 200ms RTO
        * RTO is usually *very expensive* - want to do our best to avoid this, and we prefer dupacks
- **Flow Control**
    - Receiving window of W bytes past the first expected sequence number
    - Receiver will advertise W to prevent sender from overflowing the buffer
    - Sender will want to send at $\min(\frac{W}{RTT}, B)$, where B is the network transfer speed in bps
    - *Does not take into account network conditions*
- Congestion Control is a very loaded topic, needs its own section

8.3 Congestion Control

- Want to make sure that we size the sending window/bitrate in response to network conditions
    - In low traffic, send at the flow control limit
    - In high traffic, cut back on window/bitrate to ensure fair use
    - Packet drops act as an implicit piece of feedback on network congestion

- In reality, there are two windows - the flow control window (receiver window) and congestion window (max ok for network condition)
  - Sender window should be the minimum of the two
- Windows usually tracked in terms of no. of bytes
- Principles of window adjustment
  - On successful transmission, probe for unexplored performance benefit
  - On congestion detection, cut the window to fit estimated network availability
  - We prefer to have an *undersized* window - this reduces throughput, but we have a lower risk of timeout/retransmission
  - Gentle increase (additive) on success, rapid decrease (multiplicative/scalar) on congestion detection
    - Refer to this scheme as *additive increase, multiplicative decrease*, or *AIMD*
- We can never go below 1 MSS of window size - this would violate our obligation to always make progress
- Basic AIMD
  - Increment window by $\frac{1}{CW}$ on an `ACK`, cut in half on dupack , reset to 1 on MSS
  - Results in a slow start behavior and sawtooth-esque oscillation in window size
  - We would prefer a faster ramp-up on cold/slow start to find available bandwidth
- We can add a "slow-start" phase to AIMD - on a cold start, we allow for exponential window increase *until* a threshold
  - At this threshold, we use the normal additive increase paradigm
  - On timeout, we find that our slow start was too aggro - we reduce the threshold at which it transitions
  - Slow start *is exponential with time* - window increases by 1 per `ACK`, so window doubles every batch
- TCP uses AIMD + slowstart for the most part, but has some different flavors of congestion control
  - Tahoe: reset cwnd to 1 on triple dupack
  - Reno: reset cwnd to 1 on timeout, halve on triple dupack
  - newReno: add fast recovery to Reno - the default TCP flavor in this course
- *Fast Recovery* - a third mode in TCP window management, uses dupacks to increase window size
  - Exit this state once you receive a *new ack* (the packet loss is fixed on receiver end)

8.4 Fairness

- Want to make sure that all flows over a link get a "fair" amount of the bandwidth
  - If all flows want equal rate, easy enough - just even division of bandwidth
  - *What happens in uneven bandwidth case?*
- We shouldn't just apportion bandwidth based on requested bandwidth - incentivizes lying!
- *Max-Min Fairness*
  - Give all flows asking for less than fairshare ($\frac{C}{N}$) their full request
  - All other flows will receive a fair share of the remaining total bandwidth
  - Only fairness method that is symmetric and incentive-compatible (no point in overrequesting)
  - TCP will guarantee max-min fairness in a stable state

9 Network Layer

- At a high level, this is the "host-to-host" layer
  - Protocol should be deployed in every host/router
- Sender will convert segments into "datagrams", and receiver delivers unpacked segments to the transport layer
  - Router will look at the network headers of all datagrams to figure out where it should go
- Network layer has the least flexibility in the network stack
  - Many different application level protocols, and transport has the TCP/UDP options
  - Link layers have a couple options (lot covered yet), and physical layer has many mediums you can transmit over
  - *Only IP protocol is used at the network layer* - every device in internet uses this, we need intercompatibility

9.1 Routing and Forwarding

- *Routing* is done at a global scope, determines route for packets to take from source to destination
  - We say that routing algorithms form the *control plane*
- *Forwarding* is a local construct, moving packets from router input to correct output based on a forwarding table
  - We say that forwarding tables form a *data plane*

- Every router will run a routing algorithm to pick the route and modify the forwarding table
  - Routing algorithm is usually decentralized, to prevent "single point of failure" issue
  - Routers will communicate with each other what the cost to send a packet to some destination is via that router
  - Forwarding table basically records what the "optimal" hop is from the perspective of this router
- Instead of decentralized routing, we can also use *software-defined networking*
  - A remote *centralized* controller will install the routing tables into each router
  - Instead of a routing algorithm at each router, each router only implements a "data plane"
  - A control agent per router will dump information from the remote controller into tables
  - Software-defined network can offer performance benefits (better routing efficiency) due to global view
  - Software-defined network incurs the risk of single point of failure/congestion, because routing decisions are centralized

9.2 Virtual Circuits and Datagram Networks

- We want some guarantees from the network
  - Each datagram should be *delivered*, and we can make some guarantees (from ISP) about packet latency
  - A *flow* of datagrams should be in-order, and we want some guarantee on the bandwidth/datarate we are sending at
- Today's internet routes each datagraph *separately* - e.g they're transmitted connectionlessly
  - The above guarantees may not be supported by default - we use a technique called *virtual circuits*
- IP Layer is *best effort* (no guarantees on performance)
- Remember circuit switching?
  - TL;DR: Reserve part of bandwidth of a link instead of rotating link access
  - We usually prefer packet switching since it requires much less state
  - Has better guarantees - can we emulate via packet switching?
- Mentioned earlier, but we can make a *virtual circuit network*
  - A series of datagrams with same source/destination form a *flow* in our VCN
  - Our VCN is like TCP but it is *host-to-host* instead of process-to-process, implemented by the router, and *the host cannot control this part of the layer*
    - Host would have control over TCP vs. UDP, it cannot control this though

  - The main goal of our VCN is that a *flow* of packets should follow the *same route*
- VCN router forwarding tables will contain a mapping from incoming interface+VC to outgoing interface+VC
  - VCs are mapped at each *link*, so the VC number might change from hop to hop
  - Different flows are *guaranteed* to have different VC numbers
- Sender selects a valid VC number for the flow in its network (IP) header - the router will perform translation accordingly
- Current internet *does not use Virtual Circuit Networks*
  - It has dedicated resources and guaranteed services, but it is not robust to network condition changes and takes a long time to setup
- Datagram network (stateless, guarantee-less) is used by current internet since it is more elastic
  - Route can change for two back-to-back packets depending on network state
  - Much simpler state per-router (no VC forwarding tables), but now host complexity is higher to guarantee ordering (hence TCP)
- Datagram forwarding different from VC forwarding
  - Destination IP is mapped to an *outgoing port* - how to map down the 2^{32} into a small enough space for table?
  - *Longest Prefix Match*: IPs are grouped by IP prefix - router will map IP prefixes to specific interfaces, so nearby devices should have similar IPs
    - Different mappings can use different prefix lengths
- An IP forwarding rule will be given the format `N/P` - match port N with an interface P
  - Try to match IP with all rules, pick whichever rule gives our IP the longest match
  - No two rules will match prefixes *of the same length* and match to two different interfaces

9.3 Routers

- Router split into a software-based control plane and hardware-based data plane
  - Software control plane operates on the scale of milliseconds
  - Hardware data plane operates on the scale of nanoseconds
  - Routing processor (control plane) communicates with switching fabric (data plane) to alter operation

9.3.1 Input Ports

- Input port has 3 big parts
  - Line termination is parth of the physical layer, converts physical layer into ”bits” for the link layer
  - Link layer receiver protocol will determine if there are corrupted bits, and do other link layer things
  - Link layer receiver outputs into the network layer queue - performs lookup of output port via forwarding table, will queue packets if the forwarding speed is insufficient for the bitrate
    - The fowarding component can either do destination based forwarding (based on IP addresses, the traditional method), or generalized forwarding based on some fields in network header
- After these 3 sections, the input port will dump into the *switching fabric*
  - Transfers packets from input link to correct output link
  - Has a *switching rate*, which is how fast packetes can transfer to input to output
  - We prefer the switching rate to be NR, where R is the linerate and N is the number of input ports and output ports (not cumulative)

9.3.2 Switching Fabric Architectures

- *Memory Fabrics*: first-gen routers directly copied packets to memory, and switching was under CPU control
  - Speed was limited by the memory bandwidth (two bus crossings required per datagraph - input to memory to output)
- *Bus Fabrics*:- the datagram from input port to output port was passed directly via a bus
  - Single central bus shared by all inputs/outputs
  - Switching speed will be limited by the bandwidth of your bus
  - Need to manage bus contention if multiple inputs need to forward at the same time
- *Interconnection Network*: more modern option, creates a ”crossbar” or interconnection net
  - Initially developed to connect processors in multiprocessor, so you see this a lot in NoCs
  - You make a small switch (say 4x4) and construct larger switches by linking stages of these in a network
  - Can improve parallelism by fragmenting datagram, switching them through fabric then reassemble at output
    - More in-flight datagrams at one, lower backpressure on ports (on avg)
- Need to make sure fabric is fast enough - slow can cause excess backpressure on input queues, and drop packets

- *Head-of-the-Line Blocking*: Datagram at front of input queue prevents others in the queue from moving forwards
  - Ex: First packet needs OP4, that is occupied, second packet needs OP1, that is free. Packet 2 is blocked for no reason

9.3.3 Output Ports

- Fabric dumps into another 3-stage pipeline symmetric to the input port
  - Datagram buffer, for when fabric is forwarding faster than link transmission rate
  - Link Layer Protocol for *sending*
  - Line termination to send packets to the physical layer
- If there is too much congestion (excess backpressure on output buffer), then datagrams can be dropped
  - Need to have a *drop policy* to determine what packets should be dropped
  - Also want a *scheduling discipline* to determine which packet in queue should be sent on link next
    - Makes sure that important packets are sent in a timely way, get better performance
- Buffers usually sized to a "typical" RTT - $\frac{RTT \cdot C}{\sqrt{N}}$
  - C is link capacity, N is the number of total flows
  - Too much buffering means that you might have long RTTs and sluggish TCP response
    - TCP does not realize that packets are getting delayed, since they're not being dropped
- Three big types of buffer management protocols
  - *Tail Drop*: drop the arriving packets if buffer is full
  - *Priority Drop*: drop lower priority packets if there is space contention
  - *Marking*: Flag certain fields in the packet header to signal to the receiver that there is link congestion
- Four big types of scheduling disciplines to know
  - *FCFS*: simply follow the buffer in FIFO order
  - *Priority Scheduling*: Separate buffer into multiple "priority queues", sort incoming packets. Send packet from highest-priority non-empty queue
    - Highest priority queue dispatches packets in FCFS order
  - *Round Robin*: classify incoming packets into multiple "classses", where each class gets its own queue
    - Cyclically scan class queues, send packet from non-empty queue on its turn. Queues handled in FCFS order

- *Weighted Fair Queueing*: Round robin but each class has a "weight" determining its timeshare
  - Allows for bandwidth guarantees for specific traffic classes

9.4 Internet Protocol (IP)

- IP header, like TCP header, is 20 bytes - contains information on IP version, length, fragmentation/reassembly, source/destination, and what the upper level protocol was
- *IP Address* - 32 bit identifier for each host or router interface
  - *Interface*: connection between a device (host/router) and physical link
  - Routers will generally have multiple interfaces, host will have ethernet/Wi-Fi
- Interfaces for devices can be interconnected via an ethernet switch or a Wi-Fi base station
- **Subnet**: set of device interfaces that can communicate *without* a router intervening
  - IPs in a subnet usually share the same higher order bits, and then have differing lower order bits to indicate host
  - If you detached each interface/router from its host and router, the "islands" of interfaces form a single subnet
- *Classless InterDomain Routing*: Also called CIDR, subnet is addressed w/ any number of prefix bits
- Two IP types
  - *Static*: intended for servers or remote access - manually configured on device/router for consistent access
  - *Dynamic*: "plug-and-play", IP is assigned whenever device joins the network via **DHCP**
- **Dynamic Host Configuration Protocol**: Reserves next available subnet IP for the client when it joins the network
  - Client can send a "discover" broadcast to find DHCP server, which will then return its IP and what IP client should use
  - Client always sends a DHCP request message, which is acked to confirm client has been assigned the IP
  - DHCP is considered an *application layer protocol*
  - This whole thing is done over Ethernet broadcasts, since no IP has been assigned yet
- DHCP can also assign which first-hop/gateway router to use, which DNS server to use, and the network mask (subnet determination)
- *Hierarchical Addressing*: An ISP can subdivide a large IP space into smaller spaces via varying network masks of a *slightly* longer length

  - This constrained address space may not be large enough for all connected clients - motivates *IP sharing*
  - Done via *Network Address Translation*, or *NAT*: combination of public IP and port is mapped onto *private* IP and port
- NAT offers indirection between LAN and WAN (open internet) - gateway router keeps a NAT table w/ translations
  - The NAT router needs to transparently maintain the mapping, update network header in incoming/outgoing packets to do so
  - Router also needs to update the TCP headers (layer 4) to do NAT, so it's violating end-to-end principle
  - Address shortage is handled by IPv6, so technically no more NAT needed? But it's standard, so GG
- IPv6 was motivated by IPv4 address being too short, as well as enabling fast processing/forwarding w/ fixed-length header and enabling network-layer "flow" representations
  - Extends address to 128 bit and allows for labelling priority and flows
  - gets rid of checksum, fragmentation information, and no more options (implemented as an upper-layer protocol)
  - To retain compatibility, IPv6 datagram may need to be wrapped as an IPv4 datagram's payload (**tunneling**)
    * IPv6 routers will wrap/unwrap the message before sending across an IPv4 tunnel (chain of routers)
    * When tunneling, src/destination of IPv4 header is the pair of IPv6 routers

9.5 Routing Protocols

- A *good path* is considered to be:
  - The "least congested"
  - "Fastest"
  - Lowest "cost"
- Routing algorithms have 2 general points of classification
  - **Global vs Decentralized**
    * *Global protocols* have all routers maintain comprehensive state on the network topology and link costs (Link State Algorithm, e.g Djikstra's)
    * *Decentralized protocols* have all routers maintain states of only their neighbors - there is an interative computation and neighborly information exchange
      · Distance Vector (e.g Bellman-Ford) is a good example of this
  - **Static vs Dynamic**
    * *Static* protocols change the routes slowly over time
    * *Dynamic* protocols change the routes either periodically or in response to link-cost changes

9.5.1 Link-State Routing Algorithms

- Djikstra's is the best example of this - the network topology in its entirety is known to all nodes
  - Information disseminated via a link state broadcast
  - Compute least cost from one node to all other nodes (determines a forwarding table for the node)
  - Guaranteed to know least cost for k destinations after k iterations
- **How the fuck to Djikstra's**
  - On each iteration - keep a list of "unvisited nodes" and their current pathcosts
  - Pick the node with the lowest current pathcost (tiebreak with assfuck arbitrarity) and then if pathcost is lower, use the current visited node as the new "predecessor" for the node that you just tested with an edge
  - Can run in $O(n^2)$ or $O(n \log(n))$ if you aren't a fucking moron
- Note: oscillations possible - adjusting path adjusts costs which… adjusts path again
- FUUUUUUUUUUUUUUUUUCK

9.5.2 Distance Vector Algorithm

- Is literally Bellman-Ford's algorithm w/ DP
  - Minimum cost from a node to a destination is the minimum sum of a node to a neighbor and then the neighbor to the destination
- Each node knows the cost to its neighbors, and it maintains a list of its neighbors distance vectors
  - Periodically, a node will broadcast its own distance vectors to all of its neighbors
  - On a DV update from a neighbor, a node will update its own distance vectors via B-F equation, then broadcast if the cost to a destination has changed
  - Only inform neighbors if there is a change in DV, but periodically check own link costs
  - Above issue will cause "good news travel fast, bad news travel slow" phenomenon
- Using this update/broadcast method is *iterative*, *asynchronous*, and *distributed*
- *Count-to-Infinity Problem*: By default, distance vector algorithm can continue propagating incorrect information after a route failure due to a lack of DV update, so network keeps sending through bad information
  - Can think of this as a routing loop, where information needs to keep being propagated until some convergence point due to interdependencies
  - *Poisoned Reverse*: If a node is used by *this* node along a route, then the distance to *this* node from *that* node is reported to be INFINITE
    - Only reports this to its *neighbor*, e.g if the neighbor is the next-hop

9.5.3 Comparing LS and DV Algorithms

- LS requires a higher message complexity
- LS will always converge in $O(n^2)$, but DV can run into count-to-infinity or routing loops (variable convergence timing)
- LS is more robust to router failures
  - A node can advertise incorrect link costs, but each node computes its own path in LS
  - In DV, if an inaccurate path cost is advertised, then that can propagate through network

9.5.4 Scalable Routing

- There can be *billions* of destinations - all destinations will not fit in the routing tables
  - Furthermore, links would be swamped with DV updates
- Network admin may want to control routing of a subnet - after all, internet is a network of networks
- **Aggregate Systems**: An AS separates routers into "regions", so to speak
  - *Intra-AS Routing* is homogenous in protocol, with a gateway router linking to other AS's routers
  - *Inter-AS Routing* is done via gateway routers - gateways need to perform both Intra and Inter routing
- Intra and Inter-AS algorithms cooperatively determine the forwarding tables for routers
  - Inter-AS routing tells nodes which gateway router to use to reach a different AS, propagates reachability information effectively
- By separating intra and inter AS routing we get:
  - Policy flexibility for an intra vs inter AS admin
  - Table sizes are smaller, and hierarchical routing can reduce update traffic
  - Inter-AS can prioritize policy, intra-AS can be performance-centric

9.5.5 Intra-AS Routing Protocols

- Also known as *interior gateway protocols* (IGP)
- 3 big ones
  - Routing Information Protocol (RIP) - essentially just DV
  - Interior Gateway Routing Protocol (IGRP) - some cisco proprietary bullshit until 2016
  - One more…

1. Open Shortest Path First (OSPF)

- Open refers to the algorithm being publicly available
- Uses link-state algorithm (Djikstra's), floods link-state advertisements to all other routers in the AS
  - Messages are carried over IP directly, no transport protocol involved
  - All messages are authenticated, and multiple same-cost paths are allowed
- Has support for multicast, with hierarchical OSPF used for larger domains
- **Hierarchical OSPF**
  - Separate AS into "backbone" and "local area"
  - LS advertisements only in the area, each node knows area topology fully
  - Area border routers (overlap between backbone and local area) summarize distances to nets in rea, advertise to other ABRs
  - Backbone routers will run OSPF, but only within the backbone
  - Boundary routers are connected to other ASes

9.5.6 Inter-AS Routing Protocols

- **Border Gateway Protocol**, or BGP, is the default Inter-AS routing protocol
  - *eBGP* obtains subnet reachability from neighboring ASes
  - *iBGP* propagates reachability information to the routers in the AS
  - Subnet can advertise its existence to the rest of the internet via BGP
  - Determines good routes to other ASes based on reachability and policy
- Gateway routers will run *both* of the BGP protocols
- Two BGP peers exchange messages over a semi-permanent TCP connection
  - Advertises presence of a path to some destination network *prefix*
  - Essentially advertises what subnets it can reach
- Prefix advertisement includes BGP attributes
  - Important attributes: *AS-PATH* is which ASes the advertisement has flowed through, and *NEXT-HOP* is which gateway to use
- Gateway receiving the route advertisements uses an **import policy** to determine whether to use path or not
  - AS policy also determines what to advertise/not advertise
- How to choose between different advertised paths to an AS?
  - Local preference value attribute (policy decision)
  - Shortest AS-Path
  - Closest NEXT-HOP (Hot Potato Routing)
    - Choose local gateway that requires the lowest intra-AS cost, DGAF about inter-AS cost

10 Link Layer

- Link layer is responsible for transferring a datagram to a physically adjacent node over a link
  - Links - wired, wireless, LANs, etc. - are channels that connect adjacent nodes
  - Nodes are simply hosts and routers
- The fundamental packet of the Link Layer is a *frame*, which encapsulates a datagram
- 3 big purposes of Link Layer
  - Sync devices with different interfaces
  - Guarantee transmission in case of bit *corruption*
  - Share a medium between multiple links
- To do this, Link Layer offers a couple of services
  - *Framing + Link Access*: Wrap datagram into a frame, adding header and trailer
    - Link layer will arbit channel access if there is a shared medium involved
    - MAC addresses are used in the frame headers to identify source/destination
  - *Reliable Delivery* when transmitting between adjacent nodes
    - Rarely used on low bit-error links like fiber-optic, but useful on wireless links
  - Additional:
    - Flow control - syncs sender/receiver data rates
    - Error Detection: detects presence of errors from signal attenuation/noise, signals sender to resend
    - Error Correction: receiver can potentially correct bit errors w/o retransmission
- Link Layer usually implemented in the Network Interface Card (NIC)
  - NIC implements the link/physical layer, attaches onto host's system bus

10.1 Error Detection

- When sending a datagram, *Error Detection and Correction* (EDC) bits are appended to the datagram
  - When the datagram comes out of a bit-error prone link, the bits in the datagram are checked against the EDC bits
  - If the bits can be corrected from EDC, great, we have a datagram. Else, trigger retransmission
- A single parity bit can be used to detect single-bit errors
- 2D bit parity is used to detect and correct single-bit errors
- *Internet Checksum*

  - Sender treats segment as sequence of 16-bit integers, takes addition of these integers, puts checksum into UDP checksum field
  - Receiver computes checksum of received segment, compares to UDP checksum - can definitively detect errors, but even if it's not detected there *might* be an error
- **Cyclic Redundancy Check**
  - Data bits D are used as a binary number
  - Given a bit pattern G, CRC bits R should be chosen such that `{D, R}` is mod-2 divisible by G
  - CRC can detect burst errors less than `r+1` bits, where R is r bits and G is r+1 bits
  - CRC used in Ethernet, WiFi, ATM, etc.
- Generally - $R = \text{rem}(D \cdot 2^r, G)$

10.2 Multiple Access Protocols

- Two link types
  - Point-to-Point - one sender, one receiver
  - Broadcast - shared wire or medium (ex: shared RF, WiFi)
- In broadcast channels, simultaneous transmission by two nodes will cause interference
  - A node receiving multiple signals at once is called *collision*
- We use *multiple access protocols* to arbitrate broadcast channels between nodes
- 3 broad classes of MAC protocols

10.2.1 Channel Partitioning Protocols

- **Time Division Multiple Access (TDMA)**
  - Channel access given in "rounds", each station gets a fixed-length slot per round
  - Unused slots are simply idle, which is fine
- **Frequency Division Multiple Access (FDMA)**
  - Channel spectrum divided into bands, with each station given a fixed-frequency band
  - Unused frequency bands go idle, which is fine

10.2.2 Random Access Protocols

- When node has a packet to send, transmit at full channel datarate
  - No coordination - if a collision is detected, then we can recover/retransmit later
- **Slotted ALOHA**
  - Makes a fuck ton of assumptions
    - All frames are equally sized
    - Time is divided into slots to transmit a single frame
    - Nodes always align transmission time w/ slot
    - Nodes are synchronized
    - All nodes can detect a collision
  - When node gets a new frame, it transmits at start of next slot - continues transmitting in next slot if there is another frame
  - If there's a collision, node will probabilistically retransmit the current frame until no collision
    - Probability p^k after k collisions
  - *Pros*: single active node can transmit at full channel datarate, very decentralized, and simple
  - *Cons*: collisions can cause slot wastage, potential idle slots during collision recovery, requires clock synchro
  - At best, channel is used for *useful* transmissions only 37% of the time
- **ALOHA**
  - Way fewer assumptions, most importantly *no fucking synchronization*
  - When frame arrives, start transmission immediately
  - Collision probability does increase because of no synchronization (partial overlaps possible)
    - Tanks the efficiency to 18% - what the shit?
    - Which moron thought this was a good idea
- **Carrier Sense Multiple Access (CSMA)**
  - Listen before transmit - if idle, transmit the entire frame
  - If channel busy, defer transmission (simple enough)
  - Collisions can still occur due to propagation delay, this can waste full transmission time
- **CSMA + Collision Detection (CSMA/CD)**
  - Collisions detected in short time, transmissions immediately aborted
  - In wired LANs, can simply measure signal strengths, compare transmitted/received signals

  - In wireless LANs, received signal strength is usually overwhelmed by transmission strength, so gotta be careful
    * Earliest collision detection time is when the transmission reaches the source node
  - Efficiency is inversely proportional to t_{prop} - approaches 1 as t_{prop} approaches 0
  - Much better perf than ALOHA with simpler algorithm overall
- *Ex: Ethernet Algorithm for CSMA/CD*
  - NIC receives datagram from network layer, creates frame
  - On channel idle, start transmission, else wait until channel idle
  - If NIC detects collision while transmitting, abort and send jam signal
  - After m collisions, NIC selects a backoff $2^{k \leq m} - 1$, waits for that times 512 bit times, then reattempts transmission based on B2

10.2.3 Taking Turns

- Channel partitioning protocols are good at high load but shitty at low load, random access is great at low load but can run into issues at high load
  - "Taking Turns" tries to meter the benefits of both to get a nicer solution
- **Token Passing**
  - Token passed from node to node inorder
  - Additional overhead/latency due to token-passing, and if token gets dropped or lost you're fucked up the ass

10.3 LANs

- *MAC Address*: also called the LAN or physical or Ethernet address, used to get frame from interface-to-interface in the same network (from IP perspective)
  - 48-bit MAC address for most LANs is burned into the NIC's ROM, but can be set by software sometimes
  - MAC address allocations are managed by IEEE (my goat)
  - manufacturer pays to allocate addresses on some MAC address space, which guarntees uniqueness
- MAC addresses are *invariant* regardless of LAN, IPs are *not* invariant if you move networks
- **Address Resolution Protocol (ARP)**
  - Translates IP to MAC address, which a time-to-live (TTL) after which mappings are burnt
  - Each node on a LAN will have its own ARP table
- *A Dipshit's Guide to ARP*

  - If A wants to send a datagram to B on the same LAN, we need an ARP mapping
  - If nonexistent, A broadcasts an ARP query packet to all LAN nodes w/ B's IP
  - B will respond to A with another frame containing its MAC address
  - A caches this address into its ARP table until TTL expires
- ARP requires no "software support", it just kinda works OOTB

10.3.1 Ethernet

- Still the dominant wired LAN technology - single chip w/ multiple speeds, simple and cheap, and has scaled cleanly to gigabit speeds
- Two big Ethernel topologies
  - *Bus*: Many nodes connected to a single bus, nodes can all collide with each other
  - *Star*: Nodes are all connected to a central *switch*, which each "spoke" running a separate Ethernet protocol
    * Switch guarantees no collision between spokes, so this is the more popular topology nowadays
- Ethernet has its own link-layer header wrapping the IP datagram
  - 7-byte "preamble" used to synchronize clockrates
  - Contains 6-byte source/destination MAC addresses - if adapter receives a frame with matching destination, datagram passed to network-layer protocol
  - CRC bits (remember Cyclic Redundancy Check, it's important you buffoon)
    * Frame is dropped if there is a CRC error, and our beautiful transport protocols can guarantee retransmission if required
- Ethernet is inherently *unreliable* since there is no acking, and there is no *handshaking* between NICs, making it *connectionless*
  - Ethernet uses the CSMA/CD protocol outlined before
- Many different Ethernet standards w/ different speeds/PHY layers, but MAC protocol and frame format are standardized

10.3.2 Switches and VLANs

- An *ethernet switch* is a link-layer device that stores/forwards Ethernet frames
  - MAC address inspected, frames are selectively forwarded to an outgoing link
  - CSMA/CD is used to transmit on the different segments while forwarding
- Switches are *transparent* since hosts aren't aware they exist, and they work OOTB (no configuration required)

  - Since switch has dedicated connections to hosts, switches can transmit between multiple pairs of devices w/o collision
  - Links are all full duplex (bidirectional communication allowed without collision)
- Switches contain a *switch table*, where each entry denotes host's MAC address and what interface to use for it when forwarding
- Switch learns about which host is attached to which interface by snooping incoming LAN segments, storing sender/current link pair
- If a frame is received and there is no hit on the switch table, just forward to *every* interface
  - This optioal selectivity guarantees parity with typical ethernet operation on a bus, while allowing for better parallelism after switch table is populated
- Switches can be arranged in a tree hierarchy to offer even more parallelism
- **Routers vs Switches**
  - Both are store-forward devices, but routers are based on *network* header, switches are based on *link* headers
  - Both have forwarding tables, routers form them with routing algorithms, switches form them with MAC learning

11 Host-Network Stack

- *High-level Network Stack Hierarchy*
  - Applications
  - Sockets for communication
  - OS
  - Some system bus for communication
  - Hardware (NIC)
  - Network link for communication
  - Network Fabric
- Network stack incurs heavy overhead in:
  - managing packet headers
  - moving data around on sockets/ports
  - enabling application read/write to network
  - protocol-level processing for reliable transmission
- Want to minimize network stack overhead, spend more time on the application
- Modern NICs and Network Stack communicate with 2 mechanisms
  - Data transfer done via *DMA*
  - Signalling done via *doorbells* to alert NIC, and *interrupts* to alert the host

11.1 Sender Stack

TOP

1. Sender uses `write` syscall to write to a socket
    - Construct packets, keep pushing data to write queue until queue full, then block until queue empty
2. If protocol is okay with sending data, pop packets from write queue and push to next layer
    - Keep packets *somewhere* to retransmit in event of network loss
    - Delete packets from this "TCP OQ" on ack - complex bookkeeping here
3. Packets are sent to *Netfilter*, the firewall
    - Netfilter can also do network address/port translation to add obfuscation before going to another server
    - Filtration via iptable and nftable on linux, and it's lightweight
4. If they pass netfilter, these TCP packets go to *XPS*
    - Manages mapping which sockets get pushed to which TX queue on the NIC
    - Usually all sockets on a core are given to a single NIC TXQ, but some other mapping could be defined
5. XPS packets go into a queueing discipline
    - Determines socket bandwidth, how to schedule packets out of a queue - done per NIC queue
    - Each NIC TXQ has its own queueing discipline (qdisc) in the OS
    - Linux manages its qdiscs with `tc` - remember that piece of shit?
6. (b) The queueing discipline communicates with *GSO*
    - *General Segmentation Offload* is a software solution to do packet processing in a batch
    - Packets transmitted at 1500b granularity, so packets are "segmented" piror to transmission
7. qdisc sends packets to the Driver TX
    - Manages shared mem between NIC and OS w/ a ring buffer
    - Writes data to a descriptor, signals NIC that data can be transmitted via a doorbell
    - NIC fetches packets out of the ring buffer associated w/ packet descriptor
8. The packet is in the ocean of the internet

BOTTOM

11.2 Receiver Stack

BOTTOM

1. IRQ + RX NAPI
    - Packets DMA'd into OS via RX ring buffer's descriptors
    - NIC will interrupt the OS to signal handling packets
    - NAPI (new API) will disable the interrupt, start poll loop for packet-handling (only first packet interrupts)
2. GRO (Generic Receive Offload)
    - Aggregates packets w/ same connection before passing it up to reduce processing overheads
    - Software-based aggergation, so technically there is still some CPU overhead
    - LRO is an optimization that offloads GRO into the NIC for hardware acceleration
3. RPS/RFS
    - RPS/RSS choosese a CPU core to forward packet to based on packet header's hash
        - RPS is software-based, RSS is hardware-based, allows for parallel packet processing
    - RFS/aRFS chooses a CPU core based on *where the application is running*
        - RFS is SW, aRFS is HW, does not scale well if you have lots of apps running on a core
4. NetFilter
5. TCP/IP
    - Sends out acks for the packets that were received
    - Puts packets into the socket read queue
    - Wakes up any slept read syscalls
6. Read system call
    - Reads out of the socket read queue
    - Copies data into the *application* buffers

TOP

12 Datacenters

- Distributed systems performance highly dependent on *datacenter interconnect*
- **Evaluation Metrics for Datacenter Interconnects**
  - *Diameter*: max hops between any 2 nodes
    - Determines maximal latency
  - *Bisection Width*: min links removed to partition network into 2 *equal* halves
    - Determines fault tolerance
  - *Bisection Bandwidth*: min bandwidth between any two equal network halves
    - Determines the bandwidth bottleneck
  - *Oversubscription*: ratio of worst-case achieveable aggregate bandwidth between end-hosts to bisection bandwidth
- The canonical datacenter interconnect has a *tree structure*
  - Root nodes are called the *core switches*
  - *Aggregation Nodes* aggregate traffic from the lower layer, are fully connected w/ the core nodes
  - *Edge Top-of-Rack* switches connect to the aggregation swtiches
  - Application servers connecto the "edge" switches
- Real datacenter interconnects will have a load balancer attached to the edge switches to ensure performance
- Puck lecture skipped, it's a lot of fluff IMO
  - TL;DR, Memory isn't keeping pace with compute improvements, need to partially offload network stack to hardware to improve datarates past 1GBPS at a large scale

12.1 Case Study: Goggle's Datacenter Interconnects

- Prioritizes low cost, centralized control, high bisection bandwidth, and graceful fault tolerance
- Has a custom control plane for easier network manageability - abstracts network as a *single switch* with a shit ton of ports
  - Anticipated many software-defined networking principles, somehow
  - This abstraction led to high congestion at even 25% utilizaton - bursty flows, oversubscription for cutting costs
- Congestion was abated with packet drops for QoS, tuning the host congetsion window, explicit congestion notifications, etc.
- Google's network had low utilization at the edge and aggregate level, with hostpots in the core links

  - 75% of traffic stays *within the rack*
- Half of the packets were very small (<200B), so keepalive needed to be prioritized in application design
- Up to 25% of core links had high utilization, needed to reduce this via bettter load-balancing and routing
- Has some radical differences from the internet
  - Single owner/entity means that *everything* can be optimized, and proprietary optimizations can be used
  - Link layer, transport layer, network layer are all far less separated in such a network, can be optimized
  - Fixed topology, with full maintenance by Goggle (plug-and-play flexibility not super important)
    * May not even need to have "routing algorithms", can simply pre-compute all the routes and distribute to switches
- Small flows and close colocation mean that TCP is not optimal - better transport protocol can definitely be found
  - Long handshake, bad performance on short flows, not good for low-latency, no "deadlines"
  - If Queue builds up from long flows, shorter flows will surely suffer
  - Aggregator nodes can run into "Incast" collisions from its children, can cause unnecessary TCP timeouts

13 Wireless Foundations

- Wireless channel-sharing is different from *wired* channel-sharing
  - Need to use FDMA or modify our existing open medium protocol (CSMA/CA)
- WiFi frequency is on the scale of GHz (2.4 or 5, typically)
- Trivia:
  - TCP does not work well with Wi-Fi, because packet loss can be both a link/physical issue *or* a network congestion issue
  - How to change IP protocol if gateway is always changing?

13.1 CSMA/CA

- Extension on CSMA/CD - Collision *Avoidance*
  - Need to account for signal decay (fading) over an open space - not an issue on wired links

- Two new problems on wireless mediums
  - *Hidden Terminal*: might transmit while another channel is busy, since the traffic for that channel doesn't reach transmitter
  - *Exposed Terminal*: channel is free, but it does not transmit - a node outside transmission range is sending to a node within transmission range
- Above problems can be solved with *RTS/CTS*
  - **RTS**: Request to send
  - **CTS**: Clear to send
  - RTS is sent, CTS acks it, Data is sent, then Data is `ACK`'d
    - The CTS being sent alerts all other potential TXes in transmission range to STFU
    - `ACK` will notify all other TXes in tranmission range that they can start talking again
    - Seeing the RTS but *not* the CTS will mean that the TX can still send - handles ET problem
- Edgecase: collision of RTS/CTS - this means that hidden terminal problem is not *completely* solved

14 Security

- **Thread Model**: defines scenario of security threat
  - *Who* might attack?
  - What is their *goal*?
  - *What* can they do?
- Three principles of security: *CIA*
  - *Confidentiality*: only sender/receiver should understand the message
    - Attack: eavesdropping
    - Handled w/ Cryptography and Encryption
  - *Integrity*: no untraceable alterations to messages
    - Attack: modification
  - *Availability*: sender should always be able to send to receiver
    - Attack: DoS or DDoS

14.1 Cryptography

- Pass plaintext through an encryption algorithm, send ciphertext to the receiver, and then decipher
  - *Symmetric Key* ciphers use same key for sending/receiving - good for low computation
  - *Public Key* or *Asymmetric Key* ciphers use a public *encryption* key and a private *decryption* key

14.1.1 Symmetric Key Ciphers

- Should be able to *quickly* generate $K_s(m)$, and $K_s(m)$ should not leak information of m, where m is the plaintext
- *Data Encryption Standard (DES)*: older standard with a 56-bit symmetric key and a 64-bit plaintext input
  - Block cipher - text is encrypted block by block in 64-bit chunks
  - DES's encrypted phrases can be brute-force decrypted in less than a day
  - 3DES, encrypting with 3 different keys, is a decent way to make DES more secure
- *Advanced Encryption Standard (AES)*: Successor to DES, processes data in 128 bit blocks with a 128, 192, or 256 bit key
  - If brute force decryption time of DES scaled to 1 second, AES will take 149 trillion years
- These ciphers run into the *mailman problem* (or the *padlock problem*) - need to somehow securely send the key to the receiver
  - Since key is shared, we cannot use it for authentication

14.1.2 Asymmetric Ciphers

- Ideally, we want a way to let *everyone* encrypt messages, but only *we* can decrypt the messages
  - Public-Private key cryptography!
- Given the public key, we should not be able to find the private key, and we should not be able to decrypt only knowing the public key
- Abuses modular arithmetic principles - critically $[[a]_n^d]_n = [a^d]_n$
- **RSA**: Rivest, Shamir, Adelson's encryption algorithm
  - Treats message encryption as if you were encrypting a very large binary number
  - Pick two large primes p, q (~1024 bits each), and then an integer $e < pq$ coprime to $(p-1)(q-1)$
  - Choose *another* integer d where $[ed]_z = 1$
  - Public key is (n, e), private key is (n, d)
  - *Encryption*: $c = [m^e]_n$ for a binary plaintext m
  - *Decryption*: $m = [c^d]_n$
- RSA encryption-decryption can be performed with the public then private key, or vice versa
- Hard to crack because factorization of big numbers is a computationally hard problem
- RSA's exponentiation is much more computationally hard than DES/AES
  - Can use public key crypto to *establish* a secure connection, then transmit an encrypted symmetric "session key"
  - Symmetric key cryptography preferred for performance reasons for session duration

14.2 Integrity

- Want to be able to confirm validity of the final message
- Can abuse the private-then-public message identity
  - Pass checksum into a hash function, then encrypt with private key
  - This encrypted hash can be sent to everyone, they can use public key to check integrity
  - Hash generates a fixed-size "message digest", much faster to encrypt and send as a verifier
    - Infeasible to find the original m if given $H(m)$, since function is not one-to-one
- Internet checksum gives fixed-length many-to-one digest, but too easy to have identical checksums for different messages that are *slightly* different
  - Susceptible to transposition, especially
- Instead, we use a proper hash function, encrypt it with private key, then attach this to message prior to transmission
  - Receiver can hash the message themselves, then crosscheck with the asymmetrically decrypted hash sent from sender
- **Common Hash Functions**
  - *MD5*: Computes a 128-bit message digest in a 4-step process
  - *SHA-1*: standard hash of the US, creates a 160-bit message digest

14.3 Authentication

- Want to somehow guarantee that the sender is who they *claim* they are
  - Attacker should not be able to "play-back" past auth messages
  - Attacker should not be able to generate a false auth message
  - Need a public way to identify an entity
- **Public Key Certification Authorities**
  - A *Certification Authority* binds public key to an entity E
  - The entity registers its public key with the CA after providing proof-of-identity
  - Any messages claimed to be from a sender can be decrypted via their public key retrieved from the CA