<u>LANS</u>

- **WHY LANS?** Cost: connect up to 200-1000 users, save wiring costs to share one wire; Generally higher bandwidth because traffic exhibits locality; uses Statistical Multiplexing
- **Strict Multiplexing:** users are given fixed allocation regardless of whether user has data to send or not
    - Better with predictable bandwidth guarantees and less burst traffic, i.e. voice calls
- **Statistical Multiplexing:** Each user gets access to entire LAN bandwidth when other users are idle (more users = more bandwidth) ➜ efficient
    - Better for bursty (high peak / ave ratio) traffic, like data transfer
- **(Binary) Exponential Backoff:** Generate a random number between 0 and N and transmit if you got the lowest number. Start w/ small N (like 1) and double after every unsuccessful collision. Reset N after success.
- **ALOHA:** Ethernet Predecessor; used carrier sense to grab channel; Problem ➜ vulnerable to collisions, which are only handled in software
- **Slotted Aloha:** reduces vulnerable period by half but requires a common clock; No detection of frame corruption, Frequent collisions, Semi-reliable
    - In collision, Aloha sends entire frame; ethernet aborts after 64 bytes
- **Min Packet Size:** 64 bytes (avoid finishing transmission b4 collision detected)
- **Jam:** Transmit bits after detecting collision to ensure other hosts detect it too
- **Collision Detection:** Detects increase in DC voltage level.
- persistent CSMA is where you retry immediately with P prob when channel becomes idle...non-persistent CSMA is when you retry after random interval

<u>802.11</u>

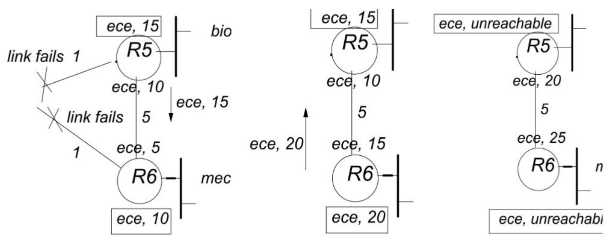
- **Carrier Sense:** If channel idle, transmit. Else, defer (persistent will retry)
- **Hidden Node Problem:** A and C talking to B at once.
    - A sends RTS to B with packet size and B sends CTS to all nodes so they synchronize transmissions accordingly
- **Backoff Interval:** With many contending nodes, RTSs will collide. So, when transmitting a packet, choose backoff interval in range [0,CW]. Wait length of interval when medium is idle (countdown suspended when dest is busy). Transmit when backoff interval reaches 0.

<u>Bridges</u>

- Forward based on DEST, learn based on SRC in ethernet hdrs, flood when no info, timeout to handle stale learning info
- Avoids loops using spanning tree algorithm (state is timed out and redone)
- Multicasting (do it for autoconfiguration and efficiency) generalizes broadcasting (sending to all) by sending to a subset of stations.
    - Multicast v broadcast: multicast better because only stations that listen to multicast pass it up
- **Why Bad?**
    - Address / max packet size / bandwidth incompatibility
    - Bridges have to learn all addresses in an extended LAN while routers only learn addr within each level of hierarchy
    - Spanning Tree inefficient and doesn't route packets on shortest path, latency up and throughput down, flooding wastes throughput
- **Why Good?**
    - Generality: allow stations w/ diff routing protocols to use same LAN
        - Bridges can be used to connect small # of compatible LANs to form Extended LAN. Routers connect Extended LANs to form routing network. Multiport devices these days, with some ports being bridges and some ports being routers
    - Cost performance: do less so cost less (than routers)
    - Control traffic: smaller amt of routing control traffic

<u>Distance Vector</u>

- Format: Each node has a DV database (port/forwarding + central).
- **Count-to-infinity problem**: When two links crash. R5 notices it has update saying it get to ECE with distance 10. Therefore, R5 updates to 15. R6 thinks it can get to ECE through R5 with distance 15+5=20. This continues.
    - Fix: if distance > 16, you stop



<u>Link State</u>

- Format: Each node contains LS table with LSP packets from every other node. Generation -> Propagation -> Dijkstra Shortest Path.
- **Intelligent Flooding Algorithm:** Normal flooding forwards to every node except the source node. IFA adds sequence numbers to LSPs so we don't flood duplicates we have previously received. Seq_num += 1 for update.
- Recovery after node crashes (jumping): send update. When another node receives update w/ seq num lower for LSP it currently has, it sends current LSP number *z* to sender. Sender jumps to that number *z+1* to send update.
- Avoids loops, more responsive to failure, but uses more memory and performance is low during initial discovery.
- **Challenges**: How to scale ➜ minimize control messages and routing table size. Adapt to failures/changes? ➜ aging (time out every 30 min, 64 bit seq)

<u>Naming</u>

- **DHCP:** how to get IP address to get started/assigned
    - automates host boot-up process. Given MAC addr, asking unique IP addr and tell host other stuff about LAN
- **DNS:** how to map from user-friendly names (ccle.ucla.edu) to IP address
    - given host name, provide IP addr. Given IP addr, provide host name.
- **NAT:** how to build large private network with only 1 assigned public IP addr
    - NAT's end hosts not reachable from Internet; all connections must be initiated from within private network (solved byNAT traversal protocols)
    - When NAT's end host sends message, NAT replaces private IP with its public IP and assigns unused port for connection to avoid duplication with other end hosts talking to same destination
- **IP Addr** (for scalable routing): used by routers to forward packets, unique topologically meaningful locator, hierarchical namespace of 32 bits
- **MAC Addr** (for unique ID): used by network adapters to identify cool frames, unique hard coded identifier burned into network adapter, flat name space

<u>IP Addressing</u>

- Class A: First few bits started with 0 (network # = 8bit)
- Class B: First 2 bits started with 10 (network # = 16 bit)
- Class C: First 3 bits started with 110 (network # = 24 bit)
    - Good for organizations with small networks, allowed large # of networks with 256 hosts each
- Depletion of Address Space => NAT (uses TCP port src & dst #s to expand address space from 32 ➜ 64bits)
- **Supernetting:** routes to multiple networks with similar network prefixes are combined into a single routing entry, with the routing entry pointing to a Super network, encompassing all the networks; referenced by CIDR
- **Subnetting:** a single big network is divided into multiple smaller subnetworks; i.e. UCLA's network has subnets for each department
- **Neighbor Routing:** 1. Routers need data link addresses of end nodes [ARP for MAC addr of dest]. 2. End nodes need MAC addr of 1 router [DHCP gives IP addr of closest router]. 3. End nodes on same LAN should comm w/o router [End Nodes know then on same subnet by comparing masks, and ARP]. 4. When router gets packet whose source is an iface, it sends source a REDIRECT so that it sends to that router in the future

<u>BGP</u>

(1) Highest LOCALPREF (customer>peer>provider when importing)
(2) Shortest ASPATH
(3) Lowest MED (prioritize path to AS wit multiple entry points)
(4) eBGP > iBGP
(5) IGP path: lowest IGP path cost to next hop
(6) Router ID: smallest router IP address (breaks ties)

Operation: establish session ➜ exchange all active routes ➜ exchange incremental updates; KEEPALIVE timer to keep connection open

- Node learns multiple paths to dest, stores all of routes in routing tables, applies policy to choose single active route, & may advertise neighbors' routes; announcement (upon selecting new active route, add own node id to path and advertise to each neighbor) & withdrawal (if active route no longer available, send withdrawal message to neighbors
- D <u>unreachable</u> if all the routes to D have AS PATH w/ this routers AS #

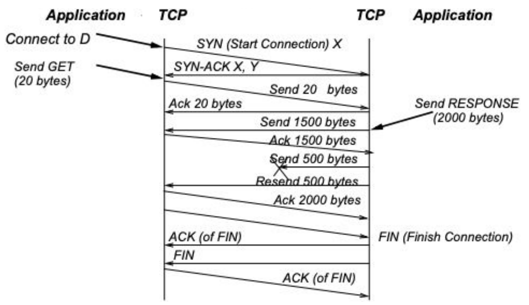
iBGP: Speaking routes exchange external route info with other eBGP routers

- Every eBGP router sends iBGP update to every other router (mesh)
- iBGP over IGP because IGP does not scale, and relies on periodic routing announcements and doesn't have a rich set of attributes
- **Route Reflectors** (to solve scaling): Selects best route and sends it to all clients. If route from client, route is sent to all clients and iBGP, else client

- **Multi-homing** (customers may have more than 1 provider): Extra reliability, survive single ISP failure, financial leverage through competition, better performance with better path.

BGP Problems:

- Instability: route flapping, long AS path decision criteria defaults to distance vector-like behavior, not guaranteed to converge
- Scalability: > 500k network prefixes in default-free table that wanna manage traffic to very specific networks but also wanna aggregate information
- Performance: not optimal, doesn't balance load across paths
- Suboptimal: local knowledge only, AS Path Length doesn't measure real distance or latency

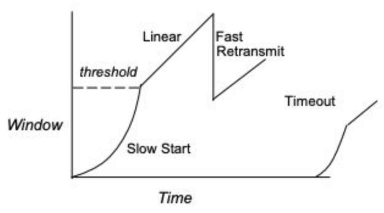


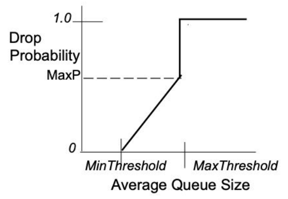
TCP Protocol

- Provides abstraction of a shared queue (also called a socket)
- X, Y = #s that client/server have not used recently (these help detect duplicates); also used as initial sequence numbers
- ACKs are based on bytes and not packets (allows receiver to ACK portions of data if it has limited buffer space)
- TCP connections separated by: Source IP, Dest IP, Source Port, Dest Port
- Detecting Duplicates: use sequence numbers!
  - **(1) Timer-based:** server remembers a seq # for max time a packet can live in network; bad if you have lots of calls w/ large packet lifetimes
  - **(2) Clock-based:** number packets w/ latest time when sent; bad since u need to keep memory to detect dups w/in timeout T
  - **(3) 3-Way Handshake:** Pick validation (sequence) numbers(X,Y) you have never used before when sending SYNs; Only need to remember most recent validation number & increment by 1 next time (or use rand #s)
- Open Issues: TCP designed around the premise of cooperation; there are a bunch of magic numbers
- TCP + Router Scheduling: dynamically allocate resources when multiple flows compete and use deficit round robins
- Deficit Round Robin: Weighted fair queueing - Schedule round robin among queues with proportion to some weight parameter
  - Each flow given own queue and unused weight rolls over to next round

TCP Crash Failures

- TCP itself can't handle crashing (Ex. crash in middle of file transfer)
- Solution: Rely on application level file transfer ACK & idempotency





AverageQ = (1 −w) * AverageQ + (w * SampleQsize)

- **TCP Congestion Control:** match sender and network speeds
  - Slow start is used to avoid initial delays; grows exponentially till threshold
  - Congestion window (cwind) is managed with AIMD policy after
  - AIMD: Additive Increase / Multiplicative Decrease
  - Upon fast retransmit detection, cwind reduced by half
  - On timeout, window size set to 1
- Alternative: RED router (congestion avoidance) - Detects congestion early
  - Explicit: Use DECbit/ECN scheme & send congestion bit to the source (currently no space in IPv4 headers).
  - Implicit: Drop packets with certain drop probability calculated using queue size (see picture above)
  - When packet is dropped, transmission rate is halved
- **Congestion control**: Only control congestion when congestion occurs (e.g. dropping packets); reactive, worse than congestion avoidance
- **Congestion avoidance:** Make sure you never send more than bottlenecked bandwidth & queues never fill up (e.g. RED)
- **Congestion collapse:** More traffic than network can handle; network drops packets and no packet makes it through completely
- Retransmit timer = Avg Round Trip (RTT) + K*Variance
- **Flow Control:** match sender and receiver speeds
  - Receiver tells sender RWIND (max window size receiver can handle), sender takes min(CWIND,RWIND) to use as window size
  - RWIND can be set to 0 → In TCP, the sender must keep probing, every RTT time or so, to check if the window is open again
- Throttling Options:
  - Window-based (TCP; constrain # outstanding packets allowed in network. Increase window to send faster. Pro: cheap, good failure properties. Con: creates traffic bursts which require larger buffers)
  - Rate-Based (many streaming media protocols. Two parameters – period and packets. Allow sending of x packets in period y. Pro: smooth traffic. Con: fine grained per connection timers).

Difference between Data Link reliability and TCP

- Connection management: only done for DL when link crashes/arises, clients dynamically requesting connections (HDLC didn't work here)
- Network (instead of single FIFO) link: packets can be delayed for a long time, duplicates can be created by packet looping, delayed duplicates = need for large sequence numbers, packets can be reordered by route changes
- Data link only needs speed matching between sender and receiver but here we also need speed matching between sender and network
- Transport needs to dynamically round-trip delay to set retransmit timers

UDP (connectionless)

- Provides unreliable message delivery between processes so it multiplexes
  - Source port filled in by OS as message sent
  - Dest port identifies UDP delivery queue at endpoint
- Delivery: packets arrive, get demuxed and split into various message queues and their ports which line up with different application processes
- Checksum intended as end-to-end check on delivery, so it covers data and UDP header and IP pseudoheader
- Applications: streaming media, DNS, NTP, FPS video games

Code

simple-router.cpp

- Check if MAC address is broadcast or destined to router
- If ARP
  - If ARP Request, if IP = iFace IP, send ARP reply
  - If ARP Reply, insert ARP entry if it's not in cache
    - Forward all pending packets to destination and remove request
- If IP, check packet length and checksum
  - If destined to router and is ICMP ECHO request
    - Check ICMP checksum and send ICMP ECHO reply
  - If not destined to router, update TTL and do checksum
    - If MAC address is in ARP cache, forward packet
    - Else, queue packet

arp-cache.cpp

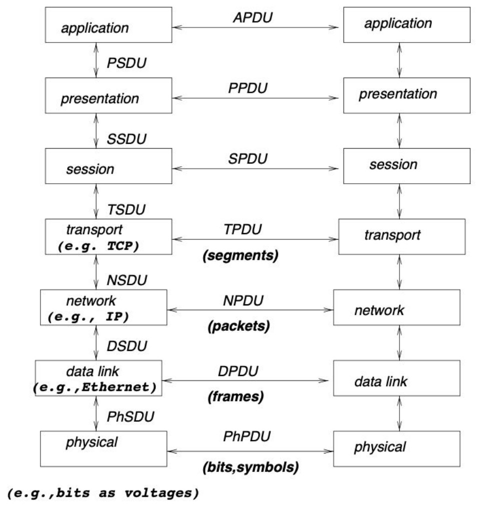
- Loop through all requests
  - If nTimesSent > MAX_SENT_TIME (5), discard request
  - Else, send ARP Request and increment nTimesSent
- Delete all invalid ARP cache entries

routing-table.cpp

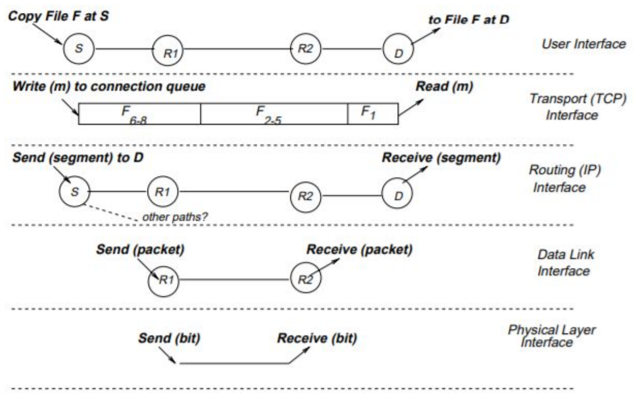
```
RoutingTableEntry max_entry;
uint32_t cur_max_mask = 0;
bool found_entry = false;
for (auto const& entry : m_entries) {
   if ((entry.dest & entry.mask) == (ip & entry.mask) &&
       entry.mask >= cur_max_mask)
   {
     cur_max_mask = entry.mask;
     max_entry = entry;
     found_entry = true;
   }
 }
if (found_entry)  return max_entry;
else              throw std::runtime_error("... not found");
```

Layering

- Layering = division of labor; each one builds on top of the other
- Why headers? Economy and easier synchronization
  - Cost of adding a few bits is cheaper than sending separate msgs
  - w/o headers, one can only send data as a continuous stream of bits; otherwise need to coordinate control and data msgs
- OSI Model (1. Physical – 7. Application)



- Strict layering: each layer only looks at its header + interface data to work
- Peeking at higher level headers = layer violation, i.e. firewalls
- Protocol = horizontal communication b/w layers (Transport => Transport)
- Interface = vertical communication b/w layers (Data Link => Physical)



Physical Layer

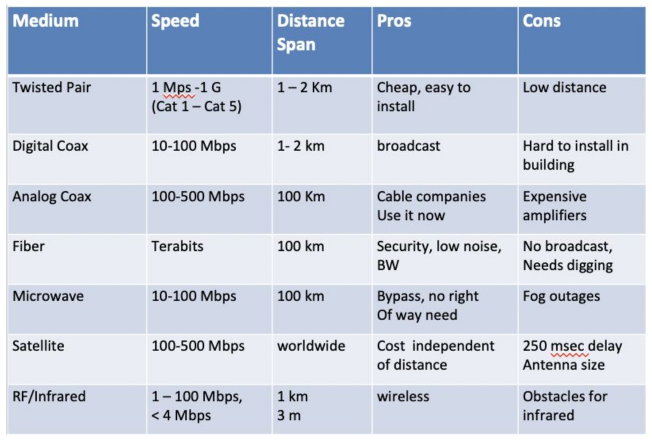
1. Transmission Sublayer (bottom)

- Signal types
  - Continuous = no sudden jumps (sine wave)
  - Discrete = can have sudden jumps (square wave)
  - Periodic = repeats itself; f = frequency, T = period = 1/f
  - Channel: physical medium that sends energy
- Fourier Analysis
  - Most channels only **scale** and **shift** sine waves
  - Bandwidth = width of freq pass band (Ex. 200Hz-1200Hz ➔ 1000Hz)
  - Lower bandwidth ➔ more sluggish; Higher bw ➔ less sluggish
  - Fourier analysis helps predict what output signal will look like
- Nyquist Limit (max symbol rate ~ speed of sending)
  - Send at rate of 2/T (or once every T/2 seconds)
  - Cannot send symbols faster than 2B per second
    - B = channel bandwidth (to prevent interference)
  - Violated if there is ISI
- Shannon Limit (max signalling rate ~ using more levels)
  - Bit rate = levels * signalling rate
  - At least 2N gap b/w levels; max S/2N levels (S: max signal strength)
    - When defining levels, arbitrarily break ties (≥)
  - "Simple" bound = 2B*log(S/2N); Shannon bound = B*log(1+S/2N)

2. Coding/Decoding Sublayer – Clock Recovery (top)

- Problem: 1 - syncing clocks, 2 - keeping clocks in sync
- Baseband coding = physical 1s & 0s using energy levels
- Broadband coding = analog coding of digital data (change amp to encode 1 or 0)
- Asynchronous coding
  - Fixed frame size (10-11 bits): send 5-8 bits data and parity bit
  - Low amp = 1 and High amp = 0
  - Frame character w/ 1 start bit (0) & 1-2 stop bits (1) ➔ guarantees transition and fixed frame size reduces error due to sampling drift
  - Cons
    - Overhead to add extra start/stop for every 8 bits
    - Overhead to start up receiver clock for each char
    - Large idle time between characters
- Synchronous
  - Use large frame size
  - Pro: Overhead of start/stop bits is amortized
  - Con: Need to encode data to guarantee clock recovery
- **NRZ** is just 1=1V, 0=0V
  - Problems: can get series of 1s or 0s, avg DC value can be not 0
- **Manchester Encoding**
  - 1 = 1.5V to -1.5V transition (10); 0 = -1.5V to 1.5V transition (01)
  - Wait for transition, then sample ¼ bit later to see if pos (0) or neg (1)
  - Preamble = 0101010101 bc string of 1's and string of 0's could look identical except for a phase shift
  - Pros: Guaranteed transitions (self-clocking) and DC balanced if levels are symmetric about 0
  - Con: Poor efficiency, 50% (1 real bit : 2 coded bits)
- **AMI**
  - 0 = 0V; 1 = alternate between 1.5V and -1.5V
  - Guaranteed DC balance, but does NOT guarantee transitions
  - Poor immunity to noise (½ noise of NRZ can confuse AMI)
- **Phase Locked Loops**
  - Gradually correct clock instead of basing everything on one transition to account for noise; implements a feedback loop
  - Phase lock = no phase difference, they are in sync
- **Eye Patterns**
  - Given input signal, compare output to output when shifted by 1 bit
  - Superimposing the 2 outputs creates eye
  - Smaller channel bandwidth => smaller eye
  - "Eye will close" with more intersymbol interference (ISI)
  - Provides visual on link quality (ISI and sampling margin)

3. Media (middle)



Medium	Speed	Distance Span	Pros	Cons
Twisted Pair	1 Mps -1 G (Cat 1 – Cat 5)	1 – 2 Km	Cheap, easy to install	Low distance
Digital Coax	10-100 Mbps	1- 2 km	broadcast	Hard to install in building
Analog Coax	100-500 Mbps	100 Km	Cable companies Use it now	Expensive amplifiers
Fiber	Terabits	100 km	Security, low noise, BW	No broadcast, Needs digging
Microwave	10-100 Mbps	100 km	Bypass, no right Of way need	Fog outages
Satellite	100-500 Mbps	worldwide	Cost independent of distance	250 msec delay Antenna size
RF/Infrared	1 – 100 Mbps, < 4 Mbps	1 km 3 m	wireless	Obstacles for infrared

- Radio waves have smaller frequency than infrared, but can go through obstacles (however radio signals can interfere with each other)
- Single-mode fiber: avoids chromatic and modal dispersion

Wireless Networks

- 802.11b and 802.11
  - Require hotspots where there are APs (Access Points)
  - 11 partially open channels for 85 Mhz region (1,6,11 non overlapping)...can have 3 APs w/o interference
  - Move in all direction and go thru objects; but smaller radius
  - People can tune in to diff frequencies
- Bluetooth: organize as Piconet and elect a master who acts like 802.11b
- 3G: can access anywhere and good mobility availability
  - More expensive than 802.11 and exclusive to cell phone carriers

Multiplexing

Sharing multiple streams of data b/w multiple senders and receivers

- **Frequency Division Multiplexing (analog)**

Diff parts of line bandwidth for 2 data streams (ex: by pitch)
Wavelength Multiplexing: divide by wavelength, done only w/ optics

- **Time Division Multiplexing (digital)**

Time slot reserved for each user (used in T1, SONET)

- **CDMA (Code Division Multiple Access)**

Multiply each bit (1 or -1) with some code (ex: -1 1 1 -1)
Statistical: pseudo-random codes (small interferences, same error for 2N users as regular CDMA for N users) ➔ uses bandwidth more efficiently

<u>Framing</u>

- Purpose: semi-reliable bit pipe (phys layer) ⇒ quasi-reliable frame pipe
- Enables error detection/recovery and possible to multiplex
- Flags and bit stuffing (stuff bits into data to avoid bad flag)
  - Proof: flag doesn't occur in data bits + stuffed bit cannot be part of flag + no spurious flag created
- Start flag and char count (count of bits following start flag, no stuffing)
- Flags by Phys Layer: Ex: FFFFFFF or Ex. unused 4-5 encoding patterns

<u>Error Detection</u>

- Random bit errors (thermal noise)
- Burst errors (synchronization errors)
  - Size k burst error ➔ distance in bits from first to last error is k-1 (Ex. k=5, 50,54 corrupted, 51-53 maybe corrupted)
  - Intermediate bits may or may not be corrupted
- **Parity bits**
  - Calculated by applying XOR to entire frame
  - Bit is a 1 if an odd # of 1's in frame; Bit is a 0 if an even # of 1's
- **Hamming Distance**
  - Number of bits that differ in b & b'; in our case it means the smallest hamming distance between any two codewords
  - If we have d incorrect bits:
    - need hamming distance of d+1 for error detection
    - need hamming distance of 2d+1 for error correction
  - Codewords = collection of all possible coded packets or frames
- **Hamming Code:** Log P parity bits for P bits; if error in location N then N is in parity bits
- **CRC (Cyclic Redundancy Checks)**
  - G(x) needs at least two terms to detect single bit errors
  - G(x) should not divide x^k + 1 for sufficiently large k to detect two bit errors, x^i+x^j (same as burst error of size i-j+1)
  - G(x) needs x+1 as a factor to detect odd bit error polynomials because odd bit error polynomials are never divisible by x+1
  - If degree of G(x) >= k, where k = burst size, then we can detect errors of size k

<u>Error Recovery</u>

- Must guarantee that frames are not duplicated, lost, or misordered
- With high error rates, error recovery at hop is better (less retransmission)
  - Problem: Crashes and other losses at intermediate nodes
  - Problem: Transport must work over both reliable + unreliable links
- End-to-end argument: only worthwhile guarantee is end-to-end to ensure correctness (also more performant when with low error rates)

Stop and wait (alternating bit)

- General protocol
  - Send back ACKs to detect if frame is delivered (send even if dup)
  - Add sequence # to frames to detect duplicates on receiver side
  - Add ACK # to avoid ACKs getting mixed up
- Need a space of 2 for numbers
- Protocol only works over FIFO networks
- Very slow (esp. for satellite)… "bandwidth-delay" product/pipe size
- Alternating bit - optimization by sending a sequence# mod 2

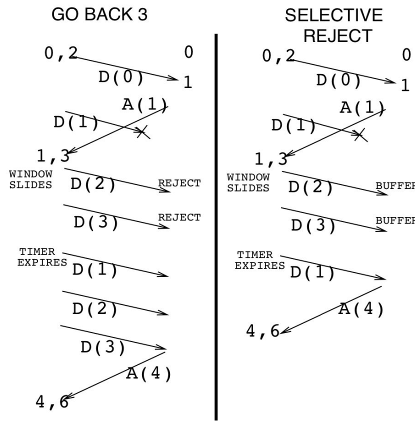
Sliding Window - Go-back-N

- Sender sends a window of frames before getting any ACKs
- Lower window edge L, upper window ledge L+w-1
- Receipt of ack numbered R => ACK for all numbers strictly less than R
- Retransmit all packets inside frame
- Receiver only accepts frames in order
- Only need ONE timer set to some multiple of avg round-trip delay
- Need a space of w+1 for numbers; If space <= w, problem would be we can't tell if new frame or retransmission ➔ could lead to duplication

```
Send (s,m)  (* sender sends or resends s-th data packet *)
    The sender can send this frame if and only if:
       m corresponds to data packet number s given to sender by client AND
       L <= s <= L + w - 1 (* only transmit within current window *)

Receive(r, Ack) (* sender absorbs acknowledgement *)
   On receipt, sender changes state as follows:
      L := R

The receiver keeps an integer R which represents the next sequence number
it expects, initially 0.

Receive(s,m) (* receiver gets a  data frame *)
     On receipt, receiver changes state as follows:
        If s = R then (* next frame in sequence *)
          R := s + 1
          deliver data m to receiver client.

Send(r, Ack) (* we an allow receiver to send an ack any time *)
       r must be equal to receiver number R at point ack is sent
       most implementations send an ack only when a data frame is received

We assume that any unacknowledged frame in current window is periodically
resent.  In particular, the lowest frame in the current window must be
periodically sent to avoid deadlock.
```

Sliding Window - Selective Reject

- Same as Go-back-N but receiver buffers frames
- When sending ACKs back, also need to send a list of which #'s receiver has already received
- Need MULTIPLE timers for each outstanding frame in window
- Need a space of 2w for numbers
- Receiver should have at least w buffers



- For selective reject, above diagram is missing the list of buffered in ack

<u>Invariants</u>

- Describe the possible states of protocol when it is working correctly

<u>Crashes</u>

- Hard to initialize data link, even numbering does not work because the sender does not keep memory after crash
- **Non-volatile memory**, sender can keep crash count on disk (RESTART msgs labeled with counter, RA accepted only if they match current crash counter)
- **Probabilistic Protocol**, use random number to label RESTART msgs
- Assume **time bound**: all messages will be lost/delivered after some time period. Sender must wait this time before sending RESTART.

<u>Network Measures</u>

- Throughput = number of jobs/sec
- Latency = worst-case (or avg) time to complete job
- Transmission rate = rate at which phys layer sends bits
- Propagation delay = time it takes for single bit to arrive @ receiver
- Pipe size = measures #bits that can be stored on physical link