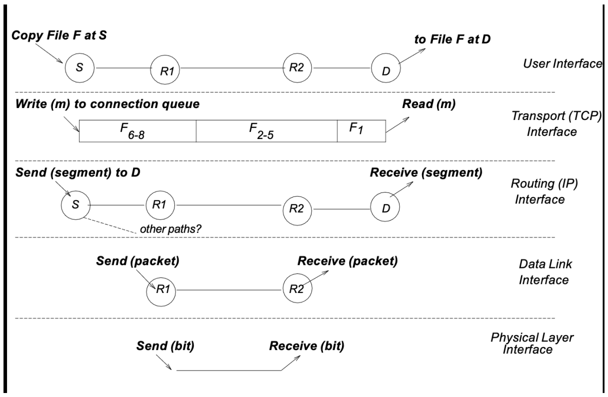
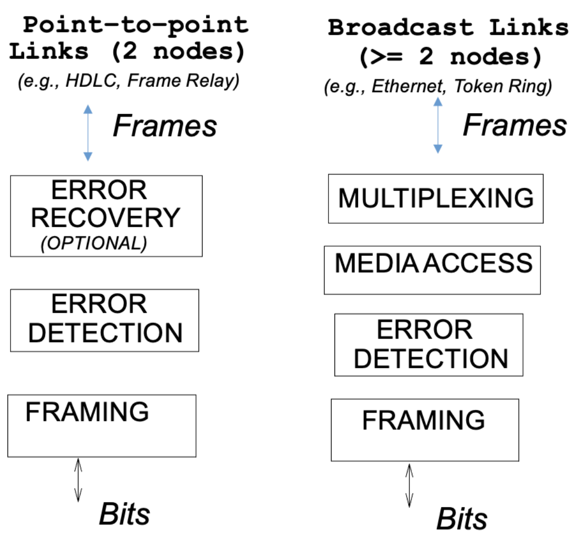
03 - Data Link Layer

Role of Physical Layer Now

- abstracted to a semi-reliable 1-hop bit-pipe (modem to modem)
- bits - transmitted at physical layer
- frame - transmitted at data link layer (ip packet with ethernet header)
- packet - transmitted at ip/routing layer (tcp packet with ip header)
- 

Data Link Sublayers

- quasi-reliable 1-hop frame-pipe (router to router)
- EOD, output from data link is a frame - a group of bits



- **Framing**: breaking up a stream of bits into units called frames so that we can add extra information like destination addresses and checksums to frames. (Required.)
- **Error detection**: using extra redundant bits called checksums to detect whether any bit in the frame was received incorrectly. (Required).
- **Media Access**: multiple senders. Need traffic control to decide who sends next. (Required for broadcast links).
- **Multiplexing**: Allowing multiple clients to use Data Link. Need some info in the frame header to identify the client. (Optional)
- **Error Recovery**: Go beyond error detection and take recovery action by retransmitting when frames are lost or corrupted. (Optional)

  - not usually done in modern routers, assumption is already that not all routers do error recovery, so you can't trust hop-to-hop error recovery
  - but modern storage area networks implement hop-to-hop error recovery to ensure low-latency recovery end-to-end by implement recovery at each hop

Data Framing

Why

- frames allow multiplexing and prevent infinite streams
- frames allow for better error detection and recovery

How

- flag and encoding (HDLC) - add a flag (bit pattern) to delimit frame boundary and encode data to ensure the flag is not preemptively found in the data
- flag and char count (DDCMP) - add flags and a character count, only look for flag after char count
- physical layer flag - supply a special symbol from physical layer to dellimit

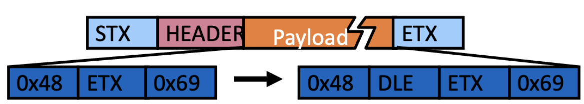
Fixed-Length Framing

- good for receiving router but bad for variable size frames
- usually used within router code to fragment large payloads

Length-based Framing

- variable length frame with length pre-pended
- still needs a flag to demarcate beginning
- bad if data corrupted b/c requiress reading and must be done while reading transmission
- e.g., DECNet, DDCMP

Sentinel-based Framing

- variable length frames with flags delimit at both beginning and end
- add stuffing (stuff some bits) where the flag is found in the data/frame
- receiver "unstuffs" data e.g., sps flag is 111, then any time we see 111 in the data, add a 0 after → 111 ... 1110 ... 111
- irl use byte-stuffing and denote the escape char to prevent false flag assign byte `DLE` and stuff it whenever STX/ETX found in payload if you get `DLE` in data then do `DLE_DLE`
- denote flags as STX/ETX (start/end) 

Physical Layer Solution

- because 4-5 encoding produces 16 possible symbols but there are 32 possible encoded, pass 2 of those unused as SOF/EOF

Error Detection

- B/c TCP doesn't require end-to-end checksums, data link undetected errors must be so small close to 1 undetected error per 20 years of data

Types of Errors

- **Random Errors**. A noise spike or inter-symbol interference makes you think a 0 is a 1 or 1 to 0. Fiber: 1 in 10 10
- **Burst errors**: .A group of bits get corrupted because of synchronization or connector plugged in. Correlated!
- **Modeling Burst error**: Burst error of length k à distance from first to last is k – 1. Intermediate may or may not be corrupted. Burst error of 5 starting at 50. Bits 50 and 54 are corrupted, bits 51-53 may or may not be corrupted
- **Goal for quasi-reliability**: Like to add checksums to detect as large a burst (say 32) and as many random (at least 3)
- **Comparison**: Imagine a frame of size 1000 and an error rate of 1 in 1000. If random, all frames corrupted on average. If we get a burst of 1000 every 1000 frames, only 1 is lost!

Parity error detection to Checksums

- parity - parity of the number of 1s in the data
- doing XOR of bits to detect parity may ddetect up to 2 bit error, but how to check error for >3 bits
- instead use checksums
- goal is detection not correction, detection = some bit in frame bad so drop frame vs flip bit (correction)
- CRC32 - use mod 2 division (XORs) for checksum instead of sum

Simple Divide Checksum

- Consider message M and generator G to be binary integers.
- Let r be number of bits in G. We find the remainder t of $2^r M$ when divided by G. Why not just M ? So that we can separate checksum from message at receiver by looking at last r bits.
- Thus $2^r M = k.G + t$. Thus: $2^r M + G - t = (k + 1)G$. So we add a checksum $c = G - t$ to the shifted message and the result should divide G.
  Example: M = 110010 (50), r = 3, G = 7. After shifting 3 bits, we get 400. Remainder t = 1, checksum = 6. So we send 110010 110 which is divisible by 7 (406)
- Has reasonable properties. However integer division hard to implement. Prefer to do without carries.
-

Mod 2 Checksum - CRC (Cyclic Redundancy Check)

- No carries. Repeated addition does not result in multiplication. e.g. 1100 + 1100 = 0000; 1100 + 1100 + 1100 = 1100
- Multiplication is normal except for no carries: e.g. 1001 *11 = 10010 + 1001 = 11011. Shift and Ex-or instead of Shift and Add as in normal arithemtic.
- Similar algorithm to ordinary division. Again let r be number of bits in G. We find the remainder c of $2^{r-1}M$ when divided by G. Why only shift $r-1$ bits this time?
- Thus $2^{r-1}M = k.G + c$. Thus $2^{r-1}M - c = k.G$. Thus $2^{r-1}M + c = k.G$ (addition same as subtraction). Send c as checksum

- background
- example, observe remainder is less than generator

Generator ***Shifted Message***

```
Generator      Shifted Message
1 1 1          1 1 0 0 0

               1 1 1
               -----
                 0 1 0
                 0 0 0
                 -----
                   1 0 0
                   1 1 1
                   -----
                   0 1 1
```

- For CRC, we need to repeatedly add (mod 2) multiples of the generator until we get a number that is $r-1$ bits long that is the remainder.
- The only way to reduce number of bits in Mod 2 arithmetic is to remove MSB by adding (mod 2) a number with a 1 in the same position.
- While no more bits

  If MSB = 1, XOR with generator (RED)

  Shift out MSB and Shift in next bit (BLUE)

CRC-16: $X^{16} + X^{15} + X^{2} + 1$ = 11000000000000101

We skip proofs of these properties this quarter but they are in your notes, Not required for HWs and tests.

Odd bit errors: can handle but not a big deal as parity can handle with using just 1 bit. 1

Two bit errors specially designed CRCs can do this. Beats parity!

Burst errors: CRC-32 can catch any 32 bit burst error for sure. Further it can catch larger burst errors with very high probability: (1 - 1/ 2^{32})

Summary: So the big deal is that it can for sure catch up to 3 bit errors, and can detect **any** error with very high probability. Like a hash function with with some deterministic guarantees

- CRC is implemented via LFSR in CPU

Error Recovery (Optional)

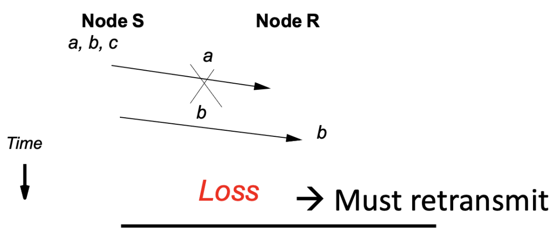
- usually not done on WAN but done on SAN
- RFC spec - english spec for network protocols

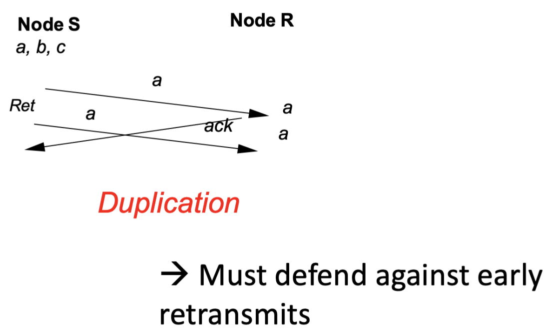
- RFC for error recovery at data link layer must ensure packets delivered without duplication, loss, or mis-ordering

Assumption

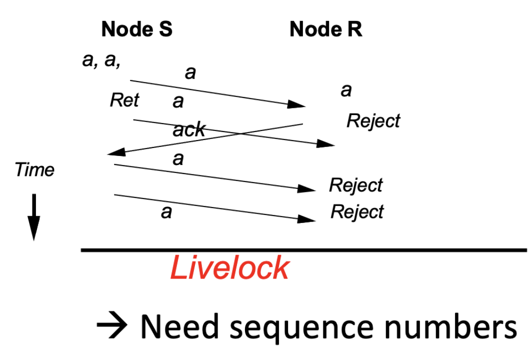
- **Assumes error detection**: Assumes undetected error rate small enough to be ignored
- **Loss as well as errors**: whole frames can be lost in a way not detected by error detection
- **FIFO**: Physical layer is FIFO
- **Arbitrary Delay**: Delay on links is arbitrary and can vary from frame to frame.

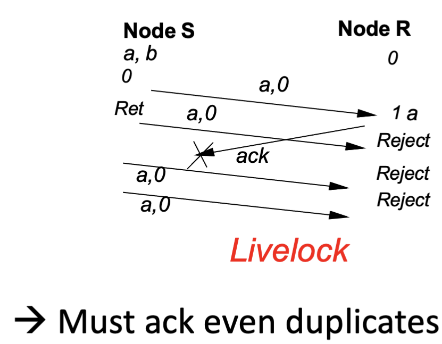
Time-Space Examples





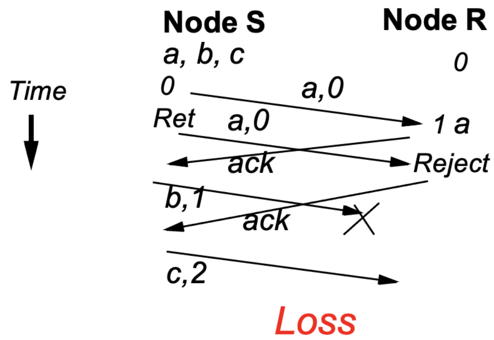
- must return ack to validate the sending of the next packet, must id the packets to





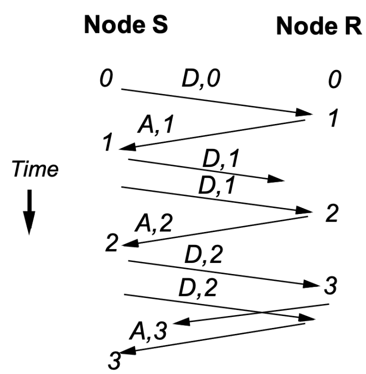
detect true duplication vs intended duplicate

- issue with sending acks back-to-back → require ack ids



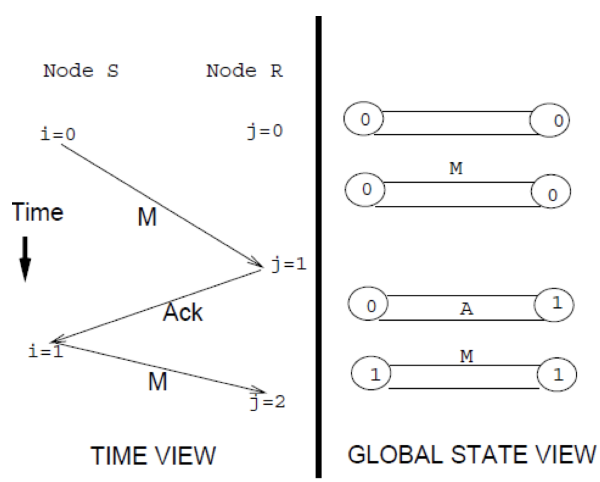
→ Must number acks

Stop and Wait Protocol (Send then wait for ack)



- When sender first gets to N, no frames with N or acks with N+1 and receiver is at N
- When receiver first receives frame N, entire system only contains number N→ only two numbers in system

- time state diagram



- global state view of messages in channels

------------Sender Code---------------------------
Sender keeps state variable SN, initially 0 and
repeats following loop

1) Accept a new packet if available from higher layer and store it in buffer B
2). Transmit a frame Send (SN, B)
3). If error-free (ACK, R) frame received and R != SN then
    SN = R
    Go to Step 1
Else if the previous condition does not occur after T sec
    Go to Step 2

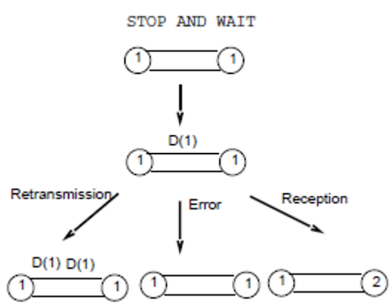
——————— Receiver Code --------------------------------------
Receiver keeps state variable RN, initially 0

When an error free data frame (S, D) is received
  On receipt:
    If S = RN then
      Pass D to higher layer
      RN = RN + 1;
    Send (ACK, RN) // Send ack unconditionally!

- code for sender and receiver

Band Invariance

- when receiver processes the message in channel and sends ack, all state are only of 1 number (the id of the latest packet acknowledged) → thus we can check for correctness of packets by ensuring band invariance
- prove band invariance by checking state transitions



- 3 other cases: Receive Ack, Send Ack, and Send new frame
- Just need to show that invariant is preserved by these 6 protocol actions/state transitions.

- alternating bit recovery code:

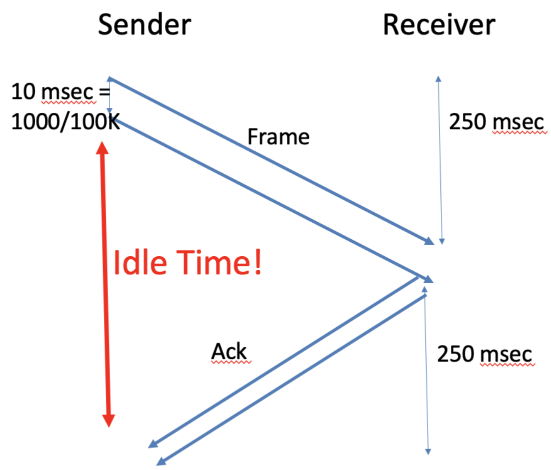
Code of Alternating Bit

```
------------Sender Code---------------------------
Sender keeps state bit SN, initially 0 and
repeats following loop

1) Accept a new packet if available from higher layer and
store it in buffer B
2). Transmit a frame Send (SN, B)
3). If error-free (ACK, R) frame received and R != SN then
      SN = R
      Go to Step 1
   Else if the previous condition does not occur after T
     Go to Step 2

Receiver Code ---------------------------------------
Receiver keeps state bit RN, initially 0

When an error free data frame (S, D) is received
  On receipt:
    If S = RN then
        Pass D to higher layer
        RN = ~ RN ; //flip bit!
        Deliver data m to client.
    Send (ACK, RN) // Send ack unconditionally!
```

Performance Measures

- throughput - jobs completed per second
- latency - worst-case time to complete a job
- 1-way propagation delay - time for the transmitted bit to reach the receiver; disregarding transmission rate, there is some amount of time it takes for the bit to travel the length of the link - this is the 1 way propagation delay
- transmission rate - the rate at which bits can be sent over a link, i.e. the number of bits per second - tells us that the second bit may come quickly after the first bit is sent
- pipe size (bandwidth-delay product) = transmission rate $\times$ round-trip propagation delay

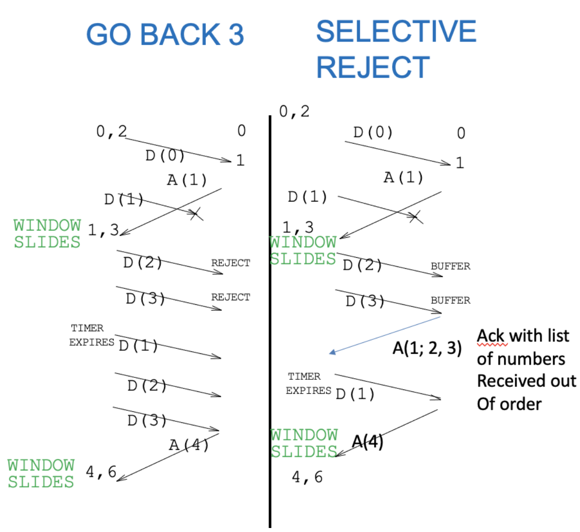
- pipe size (and prop delay) tells us our pipe/link utilization e.g., stop and wait frames
  - 1-way Propagation Delay: 250 msec
  - Transmission speed: 100 kbit/sec
  - Frame size: 1000 bits.
  - What is throughput? 2000 bits per second, which is 2% of a 100,000 bit per second link.



(send next frame after ack)

Sliding Window Protocol

- **Window**: Sender can send a window of outstanding frames before getting any acks. Lower window edge L, can send up to $L + w - 1$.
- **Receiver numbers**: receiver has a receive sequence number R, next number it expects. L and R are initially 0.
- **Sender Code**: Retransmits all frames in current window until it gets an ack. Ack numbered r implicitly acknowledges all numbers $< r$.
- **Two variants**: receiver accepts frames in order only (go-back-N) or buffers out-of-order frames (selective reject)
- we have batched rejects or selective rejects using complex acks or simple acks but resend all packets in the sliding window of frames sent



- code for both implementations

Go Back N Code

------------Sender Code---------------------------

Sender keeps state variable L, initially 0

Send (s,m) // send data message m with number s
The sender can send this frame if:
m corresponds to s-th data item
given to sender by client AND
L <= s <= L + w - 1 // in allowed send window

Receive(r, Ack) // receive an ack number r
On receipt:
L := r // slide lower window edge to ack number

Receiver Code ------------------------------------

Receiver keeps state variable R, initially 0

Receive(s, m) // receive data message m with number s
On receipt:
If s = R then
R := s + 1
Deliver data m to client.

Send(r, Ack) // send ack with number r
// receivers typically send acks in response to data
// messages but our code can send acks anytime
r must equal R

-

Selective Reject Sender code

Sender keeps a lower window edge L initially 0 but also an array with a bit set for all numbers acked so far. Initially, all bits are clear. In practice, we implement this array by a bitmap of size w which we shift

Send (s, m) // send data message m with number s
The sender can send this frame if:
m corresponds to s-th data item
given to sender by client AND
L <= s <= L + w - 1 AND
s has not been acked // new for selective reject

Receive(r, List Ack) // receive an ack number r with List
// of received numbers > r
On receipt:
L := r // slide lower window edge to ack number
Mark numbers in List as acked at sender

-

Selective Reject Receiver Code

Receiver keeps a receiver number R initially 0 but also an array with a bit set for all numbers received so far. Initially, all bits are clear. In practice, we implement this array again by a bitmap of size w which we shift. In addition to the bitmap, we have a buffer for each number where we can store out of order messages

Receive (s, m) // receive data message m with number s
  On receipt:
    If s >= R then
      Mark s as acked and Buffer m   1 0 0000
      While R acked do
        Deliver data message at position R
        R := R + 1   1 0 1000

Send (r, List Ack) // send ack with number r and List
  // of received numbers > r
  r must equal R
  List contains received numbers > R

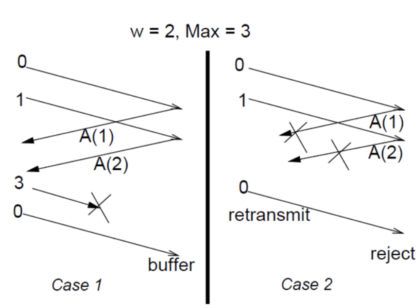
- implementation details - ONLY for FIFO packets

Implementation and Other Details

- Timers: works regardless of values, but needed for performance. So calculate round-trip delay.
- Need only one timer (for lowest outstanding number) in Go-back-n. Need one for each window element in Sel Reject.
- In selective reject, have to send an ack with R and a bit-map of numbers greater than R that have been received.
- Piggybacking: to reduce frames sent.

- Alternating bit: Modulus is 2 (just one bit)
- Go back W: Need a Modulus of W+1
- Selective Reject: Need a modulus of 2 W

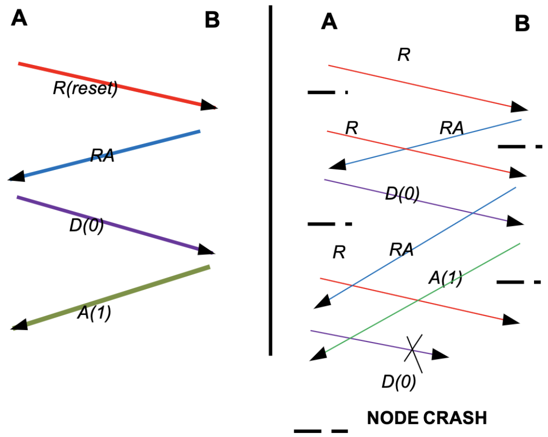
Intuition as to why the window size mu be bounded



- flow control - variable size windows, usually et by receiver to prevent backlogging frames (send the right window edge with ack)
- previously selelctive reject was not allowed as you need long listss/buffers for acks, but now RFC has some allowance

Restart signal

- although we can send restarts for error recovery, there is a deterministic protocol violation when restart ack is not sent but datagrams are already sent on line



- so we can instead number the restarts as well

Invariants

- Consider 9 queens problem: in a game of chess White can have at most 9 queens on the board, give us the invariant:

$$Q \leq 9$$

- An inducted invariant includes pawns s.t.:

$$Q + P \leq 9 \implies Q \leq 9$$

- also used in program, e.g. in bin. search. k is in R or k is not in the array

Band Invariance

- consider state of sender and receiver
- there are 2 possible overall states:
  - 1 band: sender is at x, to signal x, receiver at x
    - x band
  - 2 bands: sender at x, to signal at x, receiver at $x+1$, from signal (ack) at $x+1$
    - x band and $x+1$ band
- therefore, band invariance within $x+1$, there will never be $x+2$ in any band