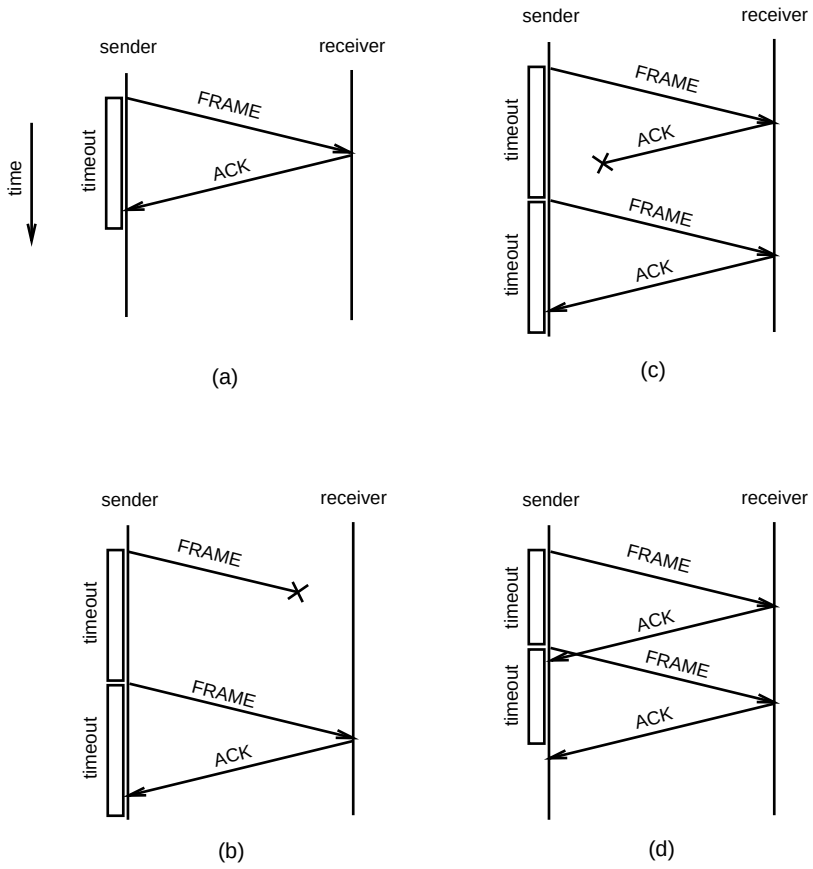
Reliable Transmission

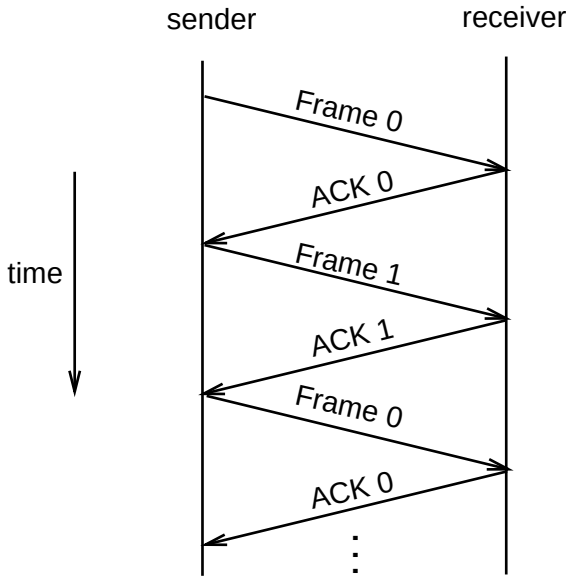
Overview

Recover from Corrupt Frames

- Error Correction Codes (ECC); also called Forward Error Correction (FEC)
- Acknowledgements and Timeouts; also called Automatic Repeat reQuest (ARQ)



Stop-and-Wait

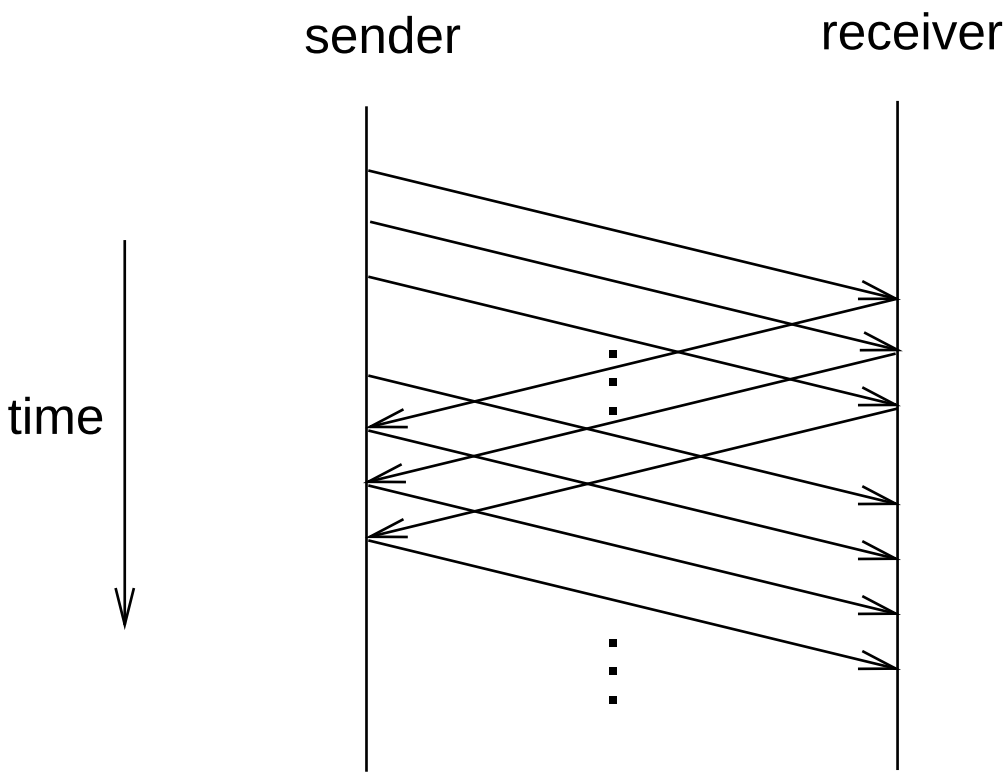


Problem: Keeping the pipe full.

Example: 1.5Mbps link $\times$ 45ms RTT = 67.5Kb (8KB). Assuming frame size of 1KB, stop-and-wait uses about one-eighth of the link's capacity. Want the sender to be able to transmit up to 8 frames before having to wait for an ACK.

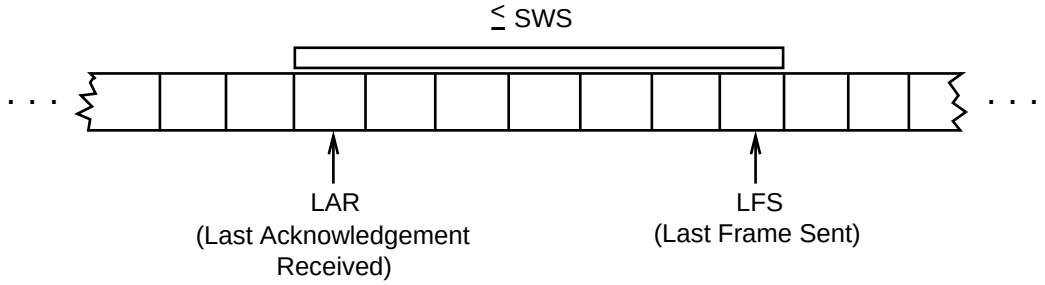
Sliding Window

Idea: Allow sender to transmit multiple frames before receiving an ACK, thereby keeping the pipe full. There is an upper limit on the number of outstanding (un-ACKed) frames allowed.



Sender:

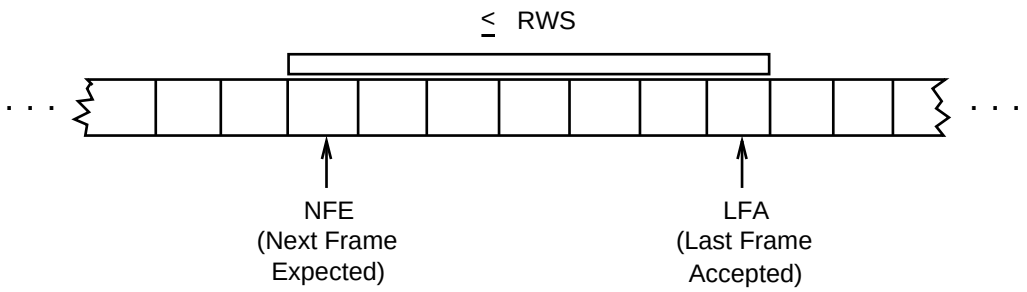
- Assign sequence number to each frame (`SeqNum`)
- Maintain three state variables:
  - send window size (`SWS`)
  - last acknowledgment received (`LAR`)
  - last frame sent (`LFS`)
- Maintain invariant: `LFS - LAR + 1` $\leq$ `SWS`



- When ACK arrives, advance `LAR`, thereby opening window
- Buffer up to `SWS` frames

Receiver:

- Maintain three state variables:
  - receive window size (`RWS`)
  - last frame acceptable (`LFA`)
  - next frame expected (`NFE`)
- Maintain invariant: `LFA - NFE + 1` $\leq$ `RWS`



- Frame `SeqNum` arrives:
  - if `NFE` $\leq$ `SeqNum` $\leq$ `LFA` $\longrightarrow$ accept
  - if `SeqNum` $<$ `NFE` or `SeqNum` $>$ `LFA` $\longrightarrow$ discarded
- Send cumulative ACK
- Variations
  - selective acknowledgements
  - negative acknowledgements (NAK)

Sequence Number Space

- `SeqNum` field is finite; sequence numbers wrap around
- Sequence number space must be larger than number of outstanding frames
- `SWS` $\leq$ `MaxSeqNum-1` is not sufficient
  - suppose 3-bit `SeqNum` field (`0..7`)
  - `SWS=RWS=7`
  - sender transmit frames `0..6`
  - arrive successfully, but ACKs lost
  - sender retransmits `0..6`
  - receiver expecting `7,0..5`, but receives second incarnation of `0..5`
- `SWS` $<$ `(MaxSeqNum+1)/2` is correct rule
- Intuitively, `SeqNum` "slides" between two halves of sequence number space

Concurrent Logical Channels

- Multiplex several logical channels over a single point-to-point link; run stop-and-wait on each logical channel.
- Maintain three bits of state for each channel:
  - boolean saying whether the channel is currently busy
  - sequence number for frames sent on this logical channel
  - next sequence number to expect on this logical channel
- ARPANET supported eight logical channels over each ground link (16 over each satellite link).
- Header for each frame included a 3-bit channel number and a 1-bit sequence number, for a total of 4 bits; same number of bits as the sliding window protocol requires to support up to eight outstanding frames on the link.
- Separates reliability from *flow control* and *frame order*.

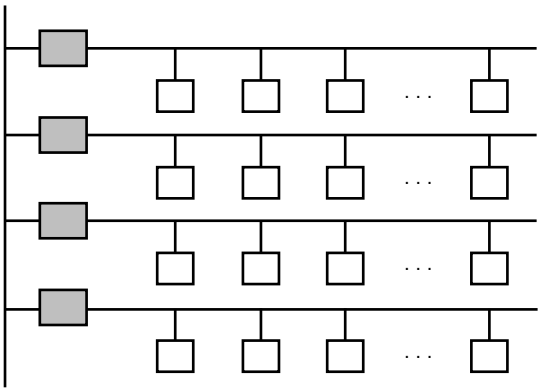
Ethernet

Overview

- History
  - Developed by Xerox PARC in mid-1970s
  - Roots in Aloha packet-radio network
  - Standardized by Xerox, DEC, and Intel in 1978
  - Similar to IEEE 802.3 standard
- CSMA/CD
  - carrier sense
  - multiple access
  - collision detection
- Bandwidth: 10Mbps and 100Mbps
- Problem: Distributed algorithm that provides fair access to a shared medium

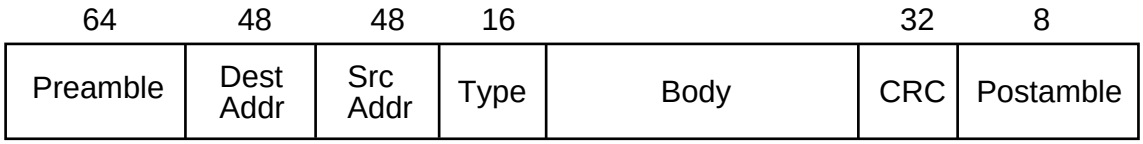
Physical Properties

- Classical Ethernet (thick-net)
  - maximum segment of 500m
  - transceiver taps at least 2.5m apart
  - connect multiple segments with repeaters
  - no more than 2 repeaters between any pair of nodes (1500m total)
  - maximum of 1024 hosts
  - also called 10Base5



- Alternative technologies
  - 10Base2 (thin-net): 200m; daisy-chain configuration
  - 10BaseT (twisted-pair): 100m; star configuration

Frame Format



Addresses:

- Unique, 48-bit unicast address assigned to each adaptor
- Example: `8:0:2b:e4:b1:2`
- Broadcast: all `1`s
- Multicast: first bit is `1`

Adaptor receives all frames; it accepts (passes to host):

- Frames addressed to its own unicast address
- Frames addressed to the broadcast address
- Frames addressed to any multicast address it has been programmed to accept
- All frames when in promiscuous mode

Transmitter Algorithm

If line is idle:

- Send immediately
- Upper bound message size of 1500 bytes
- Must wait 51μs between back-to-back frames

If line is busy:

- Wait until idle and transmit immediately
- Called *1-persistent* (special case of *p-persistent*)

If collision:

- jam for 512 bits, then stop transmitting frame
- minimum frame is 64 bytes (header + 46 bytes of data)
- delay and try again
  - 1st time: uniformly distributed between 0 and 51.2μs
  - 2nd time: uniformly distributed between 0 and 102.4μs
  - 3rd time: uniformly distributed between 0 and 204.8μs
  - give up after several tries (usually 16)
  - exponential backoff

Experiences

Observe in Practice

- 10-200 hosts (not 1024)
- Length shorter than 1500m (RTT closer to 5μ than 51μ)
- Packet length is bimodal
- High-level flow control and host performance limit load

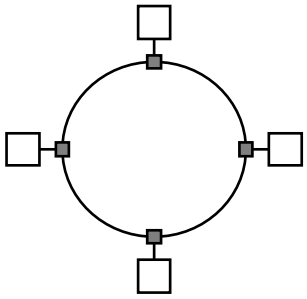
Recommendations

- Do not overload (30% utilization is about max)
- Implement controllers correctly
- Use large packets
- Get the rest of the system right (broadcast, retransmission)

FDDI

Overview

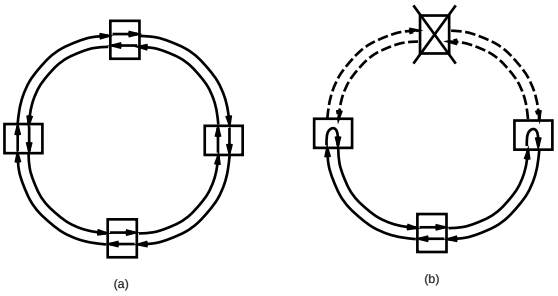
- Token Ring Networks
  - PRONET: 10Mbps and 80 Mbps rings
  - IBM: 4Mbps token ring
  - 16Mbps IEEE 802.5/token ring
  - 100Mbps Fiber Distributed Data Interface (FDDI)



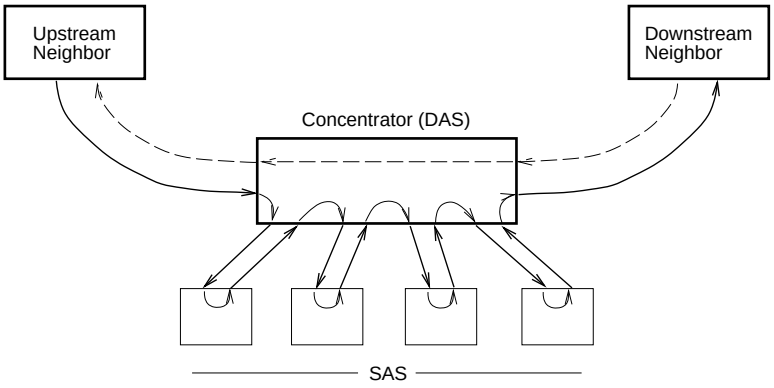
- Basic Idea
  - frames flow in one direction: upstream to downstream
  - special bit pattern (token) rotates around ring
  - must capture token before transmitting
  - release token after done transmitting
    - immediate release
    - delayed release
  - remove your frame when it comes back around
  - stations get round-robin service

Physical Properties of FDDI

Dual Ring Configuration



Single and Dual Attachment Stations



- Each station imposes a delay (e.g., 50ns)
- Maximum of 500 stations
- Upper limit of 100km (200km of fiber)
- Uses 4B/5B encoding
- Can be implemented over copper (CDDI)

Timed Token Algorithm

- Token Holding Time (THT): upper limit on how long a station can hold the token.
- Token Rotation Time (TRT): how long it takes the token to traverse the ring.

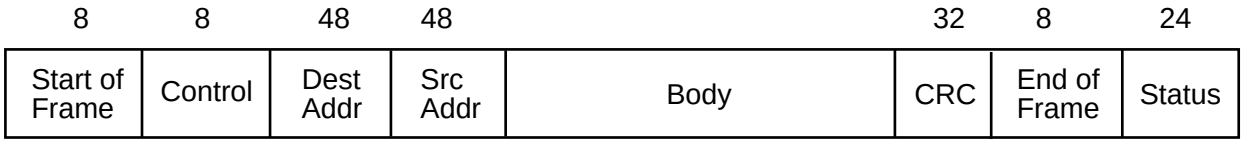
  $$\text{TRT} \leq \text{ActiveNodes} \times \text{THT} + \text{RingLatency}$$

- Target Token Rotation Time (TTRT): agreed-upon upper bound on TRT.
- Algorithm
  - each node measures TRT between successive arrivals of the token
  - if measured TRT $>$ TTRT, then token is late so don't send data
  - if measured TRT $<$ TTRT, then token is early so OK to send data
  - define two classes of traffic
    - synchronous data: can always send
    - asynchronous data: can send only if token is early
  - worse case: 2×TTRT between seeing token
  - not possible to have back-to-back rotations that take 2×TTRT time

Token Maintenance

- Lost Token
  - no token when initializing ring
  - bit error corrupts token pattern
  - node holding token crashes
- Generating a Token (and agreeing on TTRT)
  - execute when join ring or suspect a failure
  - each node sends a special *claim frame* that includes the node's *bid* for the TTRT
  - when receive claim frame, update bid and forward
  - if your claim frame makes it all the way around the ring:
    - your bid was the lowest
    - everyone knows TTRT
    - you insert new token
- Monitoring for a Valid Token
  - should see valid transmission (frame or token) periodically
  - maximum gap = ring latency + max frame $\leq$ 2.5ms
  - set timer at 2.5ms and send claim frame if it fires

Frame Format



- Control Field
  - 1st bit: asynchronous (0) versus synchronous (1) data
  - 2nd bit: 16-bit (0) versus 48-bit (1) addresses
  - last 6 bits: demux key (includes reserved patterns for token and claim frame)
- Status Field
  - from receiver back to sender
  - error in frame
  - recognized address
  - accepted frame (flow control)