CSE 461: Introduction to Computer Communications Networks
Autumn 2006

Module 3
Direct Link Networks – Part A

John Zahorjan
zahorjan@cs.washington.edu
534 Allen Center

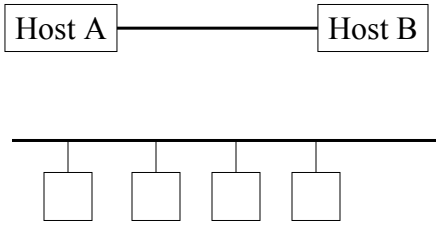
This Module's Topics

Overview of Computer Networking

1. Overview – Scope of today's discussion
2. Encoding / Framing / Error Detection
3. Reliable Transmission

Direct Link Networks

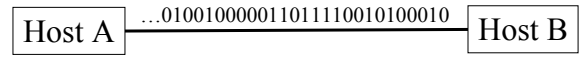
"Direct link" ⇒ no switching/routing



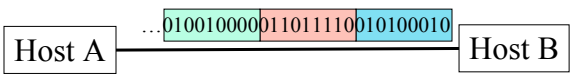
Point-to-point

Broadcast / shared

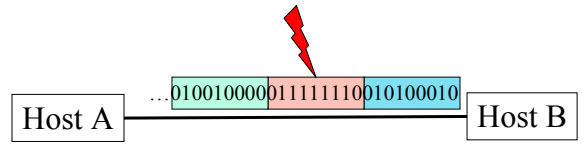
Direct Link Networks



Encoding

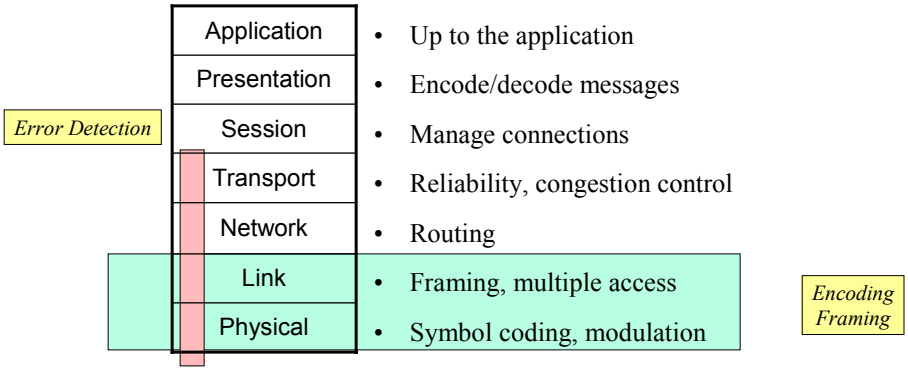


Framing



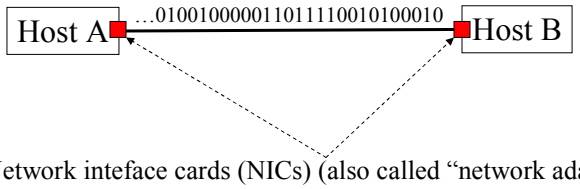
Error Detection

Relationship to the Protocol Stack



Remember, this is an idealization of what actually goes on (and the organization of the book is explicitly non-layerist).

Relationship to the hardware

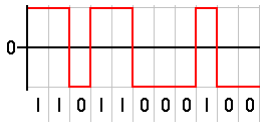


Network inteface cards (NICs) (also called “network adaptors”)

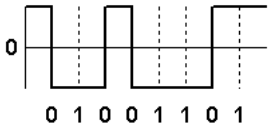
e.g., Ethernet card or 802.11g card

Encoding

- Modulate something – amplitude, frequency, phase
- A key issue is clocking
  - Higher transmission rates require better synch
- Some example encodings (thanks, *wikipedia)*:



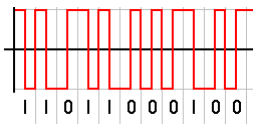
NRZ
(RS-232)



NRZI
(CDs, USB, Fast Ethernet)

Encoding: Self-Clocking

- Receiver can derive clock from the data signal
- Example 1:



Manchester
(10Mbps Ethernet)

- Example 2: Use NRZI, but make sure there are transitions
  - 4B/5B multi-level transition (MLT)
    - 100Mbps Ethernet, with 3 levels of signal
  - 8B/10B MLT
    - 1000Mbps Ethernet, with 5 levels of signal
  - (MLT is used to limit the required signal bandwidth to what can be carried on cheap, CAT 5 cable (100MHz).)

Separate Clock Distribution

- Self-clocking consumes bandwidth
  - Manchester: two transitions per bit
  - 4B/5B and 8B/10B: overhead of additional bits
- Alternative: send explicit clock
  - SONET (Synchronous Optical NETwork)
    - Clock can be carried explicitly from one network element to another
    - Nodes can all use clock from GPS
    - Various fallbacks

Table A: Stratum Clock Requirements and Hierarchy

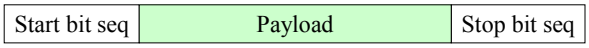
Stratum	Accuracy/Adjust Range	Pull-In-Range	Stability	Time To First Frame Slip *
1	1×10^{-11}	N/A	N/A	72 Days
2	1.6×10^{-8}	Must be capable of synchronizing to clock with accuracy of $+/-1.6 \times 10^{-8}$	1×10^{-10}/day	7 Days
3E	1.0×10^{-6}	Must be capable of synchronizing to clock with accuracy of $+/-4.6 \times 10^{-6}$	1×10^{-8}/day	3.5 Hours
3	4.6×10^{-6}	Must be capable of synchronizing to clock with accuracy of $+/-4.6 \times 10^{-6}$	3.7×10^{-7}/day	6 Minutes (255 in 24 Hrs)
4E	32×10^{-6}	Must be capable of synchronizing to clock with accuracy of $+/-32 \times 10^{-6}$	Same as Accuracy	Not Yet Specified
4	32×10^{-6}	Must be capable of synchronizing to clock with accuracy of $+/-32 \times 10^{-6}$	Same as Accuracy	N/A

Framing

- Need to know where a frame starts
  - Special bit sequence marks start of frame
- Need to know where frame ends
  - Special bit sequence, or
  - Length of frame is transmitted, or
  - Fixed length frame

Framing (cont.)

- The generic view



- Because the payload may contain the start or stop sequence, may have to "stuff" payload at sender, and unstuff at receiver
  - Something like putting a quote inside a quoted string in a programming language
    - Suppose start bit sequence is 0x7E.
    - Replace 0x7E in payload with 0x7D 0x5E
    - Replace 0x7D in payload with 0x7D 0x5D
    - At receiver, 0x7D 0x5E replaced with 0x7E
- We'll see more frame formats when we look at specific link level protocols in a bit…

Problem: Transmission Errors
Solution: Redundancy

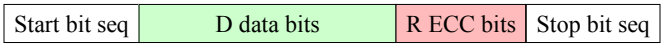
- Noise can flip some of the bits we receive
  - We must be able to detect when this occurs!
- Basic approach: add redundant data
  - Error detection codes allow errors to be recognized
  - Error correction codes allow (some) errors to be repaired too
- Questions we'll delay for a bit:
  - What should happen if an uncorrectable error is detected?
  - Which layer(s) should do whatever it is?

Patterns of Errors

- Q: Suppose you expect a bit error rate of about 1 bit per 1000 sent. What fraction of packets would be corrupted if they were 1000 bits long (and you could detect all errors but correct none)?
- A: It depends on the pattern of errors
  - Bit errors occur at random
    - Packet error rate is about $1 - 0.999^{1000} = 63\%$
  - Errors occur in bursts, e.g., 100 consecutive bits every 100,000 bits
    - Packet error rate ≤ 2%

Error Detection/Correction Codes

- Detection/correction schemes are characterized in two ways:
  - Overhead: ratio of total bits sent to data bits, minus 1
    - Example: 1000 data bits + 100 code bits = 10% overhead
  - The errors they detect/correct
    - E.g., all single-bit errors, all bursts of fewer than 3 bits, etc.
- A scheme maps D bits of data into D+R bits – i.e., it uses only 2^D distinct bit strings of the 2^{D+R} possible.



- The sender computes the ECC bits based on the data.
- The receiver also computes ECC bits for the data it receives and compares them with the ECC bits it received.
  - Detection occurs when what the receiver computed and received don't match
  - That is, detection occurs when the D+R total bits are not one of the 2^D messages valid using the code

The Hamming Distance

- Hamming distance of a code is the smallest number of bit differences that turn any one codeword into another
  - e.g, code 000 for 0, 111 for 1, Hamming distance is 3
- For code with distance d+1:
  - d bit errors can be detected, e.g, 001, 010, 110, 101, 011
- For code with distance 2d+1:
  - d errors can be corrected, e.g., 001 → 000

Specific Schemes

- We'll briefly touch on the three schemes mentioned in the book
  - They're organized from least to most expensive to compute
- Scheme 1: parity
  - A single parity bit is associated with each K bits of the data, for some K. It is set so that the XOR of the data bits + the parity bit = 0 (for even parity)
    - Example: K=8, one parity bit per byte
      - Detects all odd numbers of errored bits
    - Example: 2-dimensional parity: one parity bit for each bit in a byte, another for each of the eight bit positions in 8 consecutive bytes
      - Detects all 1-, 2-, and 3- bit errors, plus many >3-bit errors

2-d parity example

0101001	1
1101001	0
1011110	1
0001110	1
0110100	1
1011111	0
1111011	0

Specific Schemes

- Scheme 2: checksum
  - General idea: Sum successive blocks of K-bits of the data, as though they were integers
  - Internet checksum: K=16, use 1's-complement arithmetic, take 1's complement of result as checksum
    - Example: data is 01 00 F2 03 F4 F5 F6 F7
      - 0100 + F203 = [0] F303
      - F303 + F4F5 = [1] E7F8 = E7F9
      - E7F9 + F6F7 = [1] DEF0 = DEF1
      - Checksum is 1's complement of DEF1: 210E
      - Transmit 01 00 F2 03 F4 F5 F6 F7 21 0E
    - Why use 1's-complement is a bit arcane (e.g., endian-ness of machine doesn't matter), and not terribly crucial

Specific Schemes

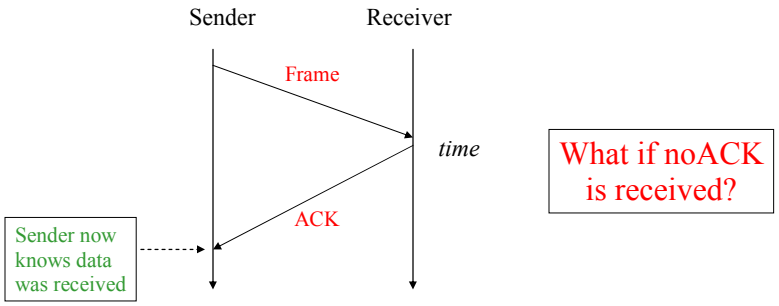
- CRCs (Cyclic Redundancy Check)
  - Stronger protection than checksums
    - Used widely in practice, e.g., Ethernet CRC-32
    - Implemented in hardware (XORs and shifts)
- Based on mathematics of finite fields
  - "numbers" correspond to polynomials, use modulo arithmetic
  - e.g, interpret 10011010 as $x^7 + x^4 + x^3 + x^1$
- Algorithm: Given n bits of data, generate a k bit check sequence that gives a combined n + k bits that are divisible by a chosen divisor C(x)

How is C(x) Chosen?

- Mathematical properties:
  - All 1-bit errors if non-zero x^k and x^0 terms
  - All 2-bit errors if C(x) has a factor with at least three terms
  - Any odd number of errors if C(x) has (x + 1) as a factor
  - Any burst error < k bits
- There are standardized polynomials of different degree that are known to catch many errors
  - Ethernet CRC-32: 10000010011000001000111011011011 1

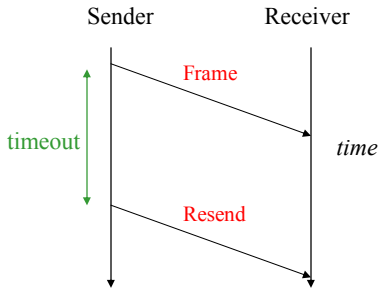
Reliable Transmission

- Because there may be uncorrectable errors (no matter what ECC scheme is used), how can the sender be sure that the receiver got the data?
  - The sender must receive an acknowledgement (ACK) from the sender



Timeouts / Automatic Repeat Request (ARQ)

- If no ACK comes back, the sender must re-send the data (ARQ)
  - When is the sender sure that no ACK is coming back?
    - Because as a practical matter delays are very difficult to bound, in some sense it can never be sure
    - Sender chooses some reasonable timeout – if the ACK isn't back in that much time, it assumes it will never see an ACK, and re-sends



What if original frame arrived, but ACK was lost?

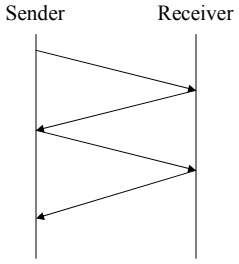
Duplicate Detection: Sequence Numbers

- So that the receiver can detect (and discard) duplicates, distinct frames are given distinct sequence numbers
  - E.g., 0, 1, 2, 3, …
- When a frame is re-sent, it is re-sent with the same sequence number as the original
- The receiver keeps some information about what sequence numbers it has seen, and discards arriving packets that are duplicates

The Stop-And-Wait Protocol

- Basic idea:
  - Sender transmits n^{th} frame only after it is sure that $n\text{-}1^{st}$ was received – i.e., it has received an ACK for the $n\text{-}1^{st}$

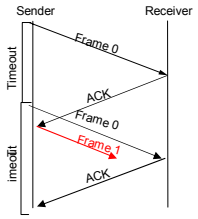
Stop-and-Wait Protocol



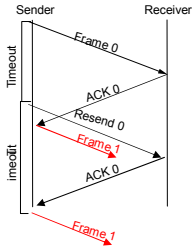
- Sender doesn't send next packet until he's sure receiver has last packet
- The packet/Ack sequence enables reliability
- Sequence numbers help avoid problem of duplicate packets

Stop & wait sequence numbers

- Simple sequence numbers enable the receiver to discard duplicates
- ACKs must carry sequence number info as well



The Problem Scenario



The Solution

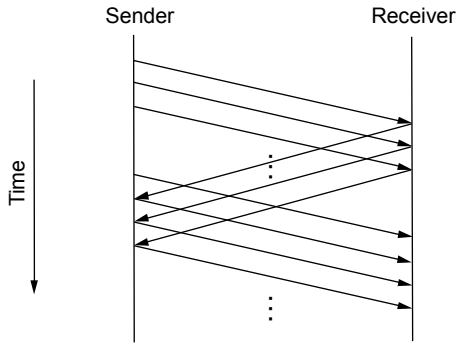
- Stop & wait allows one outstanding frame, requires two distinct sequence numbers

Problem with Stop-And-Wait: Performance

- Problem: “keeping the pipe full”
  - If the bandwidth-delay product is much larger than a packet size, the sender will be unable to keep the link busy
- Example
  - 1.5Mbps link x 45ms RTT = 67.5Kb (8KB)
  - 1KB frames imples 1/8th link utilization
- Solution: allow multiple frames “in flight”

Solution: Allow Multiple Frames in Flight

- This is a form of pipelining

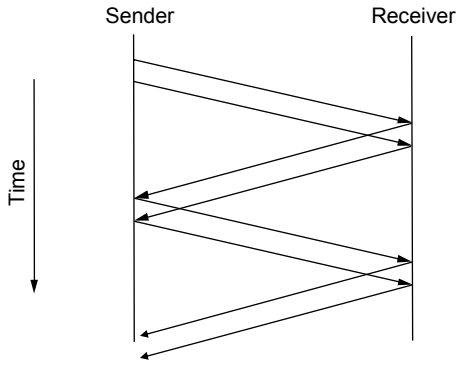


Flow Control

- Why can't we allow the sender to send as fast as it can, timing out and re-sending each frame as necessary?
- Flow control:
  - Receiver needs to buffer data until it can be delivered to higher layers
    - If the sender is much faster than the receiver, it will overwhelm it, causing the receiver to run out of buffer space
  - Additionally, if a frame is lost, the receiver will receive frames "out of order". It wants to buffer those frames to avoid retransmission, but cannot deliver them to the client until the missing frame is re-sent and received
  - Flow control is the notion that the receiver must be able to control the rate at which the sender is thrusting frames at it
- A common, important approach to flow control is the sliding window protocol

Sliding Window Protocol

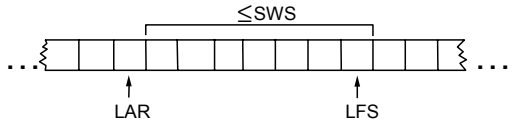
- Here is some maximum number of un-ACK'ed frames the sender is allowed to have in flight
  - We call this "the window"
  - Example: window size = 2



Once the window is full, each ACK'ed frame allows the sender to send one more frame

Sliding Window: Sender

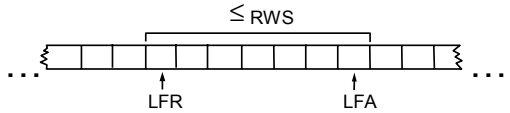
- Assign sequence number to each frame (`SeqNum`)
- Maintain three state variables:
  - send window size (`SWS`)
  - last acknowledgment received (`LAR`)
  - last frame sent (`LFS`)
- Maintain invariant: `LFS - LAR <= SWS`



- Advance `LAR` when ACK arrives
- Buffer up to `SWS` frames

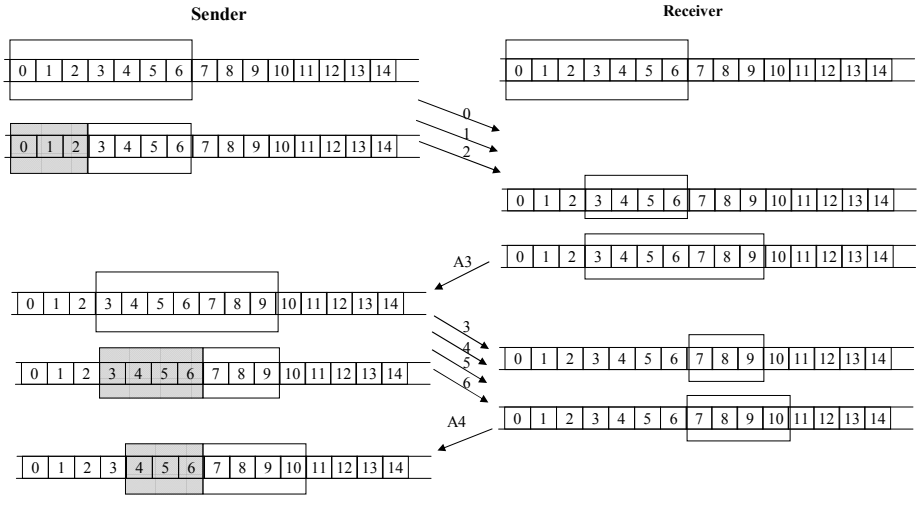
Sliding Window: Receiver

- Maintain three state variables
  - receive window size (**`RWS`**)
  - largest frame acceptable (**`LFA`**)
  - last frame received (**`LFR`**)
- Maintain invariant: **`LFA - LFR <= RWS`**



- Frame **`SeqNum`** arrives:
  - if **`LFR < SeqNum ≤ LFA`** ⇒ accept + send ACK
  - if **`SeqNum ≤ LFR`** or **`SeqNum > LFA`** ⇒ discard
- Send *cumulative* ACKs – send ACK for largest frame such that all frames less than this have been received

Sliding Window Example



Sequence Number Space

- `SeqNum` field is finite; sequence numbers wrap around
- Sequence number space must be larger then number of outstanding frames
- `SWS <= 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 the original incarnation of 0..5
- `SWS < (MaxSeqNum+1)/2` is correct rule
- Intuitively, `SeqNum` "slides" between two halves of sequence number space

Sliding Window Summary

- Sliding window is best known algorithm in networking
- First role is to enable reliable delivery of packets
  - Timeouts and acknowledgements
- Second role is to enable in order delivery of packets
  - Receiver doesn't pass data up to app until it has packets in order
- Third role is to enable flow control
  - Prevents server from overflowing receiver's buffer