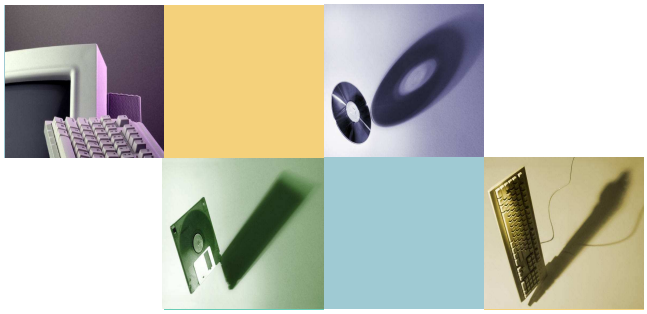
CSE123A discussion session

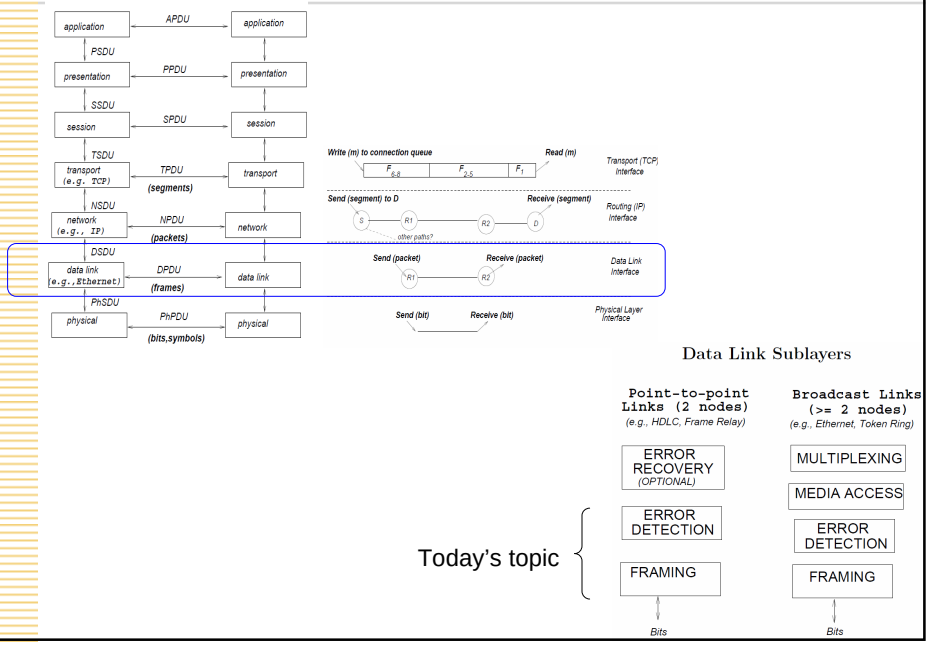


2007/02/02 Ryo Sugihara

Topics

- Review
  - Data Link layer (1):
    - Overview
      - Sublayers
      - End-to-end argument
    - Framing sublayer
      - How to delimit frame
        - » Flags and bit stuffing
  - Data Link Layer (2): Error detection sublayer
    - Parity bit
    - Hamming distance
    - CRC
      - Modulo-2 arithmetic

Where are we now?



Data Link Sublayers

Point-to-point Links (2 nodes) (e.g., HDLC, Frame Relay)	Broadcast Links (>= 2 nodes) (e.g., Ethernet, Token Ring)
ERROR RECOVERY (OPTIONAL)	MULTIPLEXING
	MEDIA ACCESS
ERROR DETECTION	ERROR DETECTION
FRAMING	FRAMING
Bits	Bits

Today's topic

Sublayers

Data Link Sublayers

Point-to-point Links (2 nodes) (e.g., HDLC, Frame Relay)	Broadcast Links (>= 2 nodes) (e.g., Ethernet, Token Ring)	
ERROR RECOVERY (OPTIONAL)	MULTIPLEXING	
	MEDIA ACCESS	"Who sends next?"
ERROR DETECTION	ERROR DETECTION	Detect errors
FRAMING	FRAMING	Make frames
Bits	Bits	

Required

End-to-end argument

- About "where to implement end-to-end functionality"
  - J. Saltzer, D. Reed, and D. Clark, "End-to-end Arguments in System Design". ACM Transactions on Computer Systems (TOCS), Vol. 2, No. 4, pp. 195-206, 1984
- Points
  1. Functions should be provided close to the application that uses the function
     - i.e. Higher level in the network stack
     - Why?:
       - Function provided at low levels tends to bring unnecessary redundancy
         - » Because it can never be perfect; only possible at higher level
  2. Low level mechanisms to support them are worthwhile only for improving performance
- Most applicable to
  - Reliable transfer
  - Network security

End-to-end argument: Error Recovery

- End-to-end error recovery is required
  - Data Link error recovery cannot be perfect
    - Routers may fail
  - Anyway, Transport layer protocols (e.g. TCP) must work on both unreliable and reliable Data Links
    - Error recovery needs to be implemented in Transport layer again
- But Data Link error recovery may improve performance
  - Fewer end-to-end retransmissions
- ... or may not
  - Especially when PHY layer has very low bit error rates
  - Reasons:
    - ACK packets use bandwidth
    - Need to buffer the frames until receiving ACK

End-to-end argument: in practice

- "Based on End-to-end argument, Data Link layer does not need to be very reliable"
  - Not quite true...

- Prob. of false negative (in DL layer) should be very small
  - (BTW, no false positives if we use parity/CRC)
  - i.e. Data Link layer should provide "quasi-reliable frame pipe"
  - Why?
    - End-to-end error recovery is often ignored (in TCP & applications)
    - ex.) File download
      - Do you check md5 checksum?

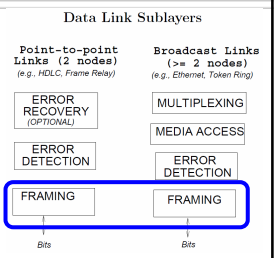
Memo

- Two kinds of error
  - Detect error in uncorrupted frame ("False positive")
  - Fail to detect error in corrupted frame ("False negative")

Berry Linux 0.71 (511.9MB)
Berry Linux 0.72 (514.8MB)
Berry Linux 0.73 (524.7MB)
Download Berry Linux 0.74 (527.2MB) MD5:b245a3d1a8425741e75d17846380becf / by
Download Berry Linux 0.75 (514.5MB) MD5:7b6d7d57a865f47c55a59876887cc6e1 / by
Berry Linux 0.76/Unstable (521.4MB) [Based on Fedora Core 6]
Berry Linux 0.77 (524.8MB)
Download Berry Linux 0.78 (524.3MB) MD5:8ea26304e0dff1f76b9e0a9524b7fd7b / by M

Framing sublayer

- Bit sequence → Frame



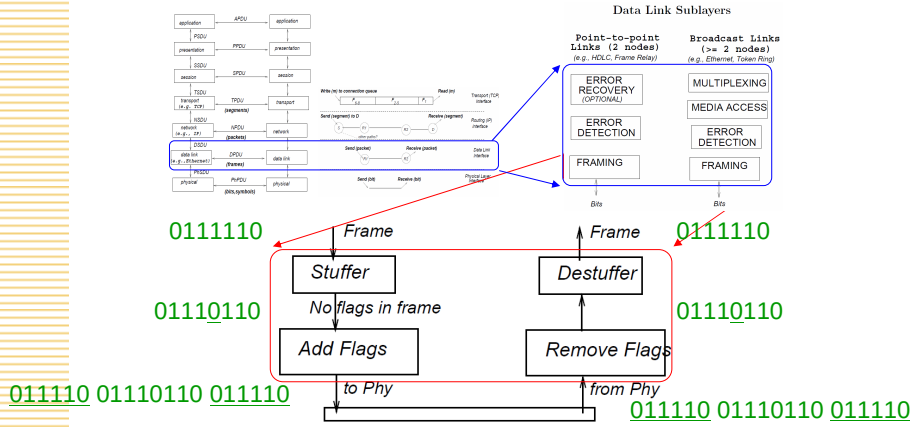
- Why do we need framing?
  - To add extra info ("header")
    - e.g.) Destination address, CRC checksum
  - Manageable unit for error detection/recovery
    - Localize the damage to small region

Framing

- How to distinguish a frame?: 3 methods
  - Flags and Bit stuffing
    - Use special bit patterns (="flag") to delimit frame
    - Use bit stuffing to encode data s.t. flag does not appear in the data
  - Start flags and character count
    - In addition to start flag, use <u>bit count</u> (=size of frame) to indicate the end of frame
  - Special symbol supplied by PHY layer
    - e.g.) "F", in addition to "0" and "1"

Flags and Bit stuffing

- Design by sublayering



ex.) Frame data = "0111110", Flag = "011110"

stuffing rule: "add 0 immediately after three consecutive 1s"

(destuffing rule: "remove 0 immediately after three consecutive 1s")

Flags and Bit stuffing

- It can be tricky, though
  - Choosing a "good" flag
  - Determining correct bit stuffing rule

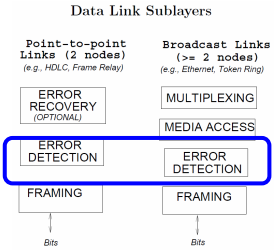
- 3 different cases for spurious flag
  1. In data
     - Easily avoided
  2. In stuffed data
     - ex.) Flag = "0111", Stuffing rule: "Add 0 after 011"
       - Data = "011111" → stuffed data = "0110111"
  3. On the boundary between data and end flag
     - ex.) Flag = "010101"
       - stuffed data = "001" → transmitted data = "010101 001 010101"

Error detection sublayer

- Why?
  - Because we want to dispose corrupted frames
    - Remember the argument in "quasi-reliability"

- How?
  - Add some redundant bits to messages
    - Parity bit, CRC checksum
  - Use that redundancy to detect errors

- It's all about coding
  - "Coding theory"



- Terminology: (probably not very standard)
  - **"Message"** is encoded into **"codeword"**
  - Sender sends a codeword and receiver receives **"received bitstring"**
    - Due to noise, codeword may or may not be identical to received bitstring

Parity bit

- 1 bit to mean "parity" (even/odd) of the number of 1s in the message
  - As a result, there are even number of 1s in any codeword
- Examples: 3bit message + parity bit (= 4bit codeword)
  - message = 000 → codeword = 0000
  - 010 → 0101
  - 110 → 1100
- Every odd number bit error can be detected
  - ex) message = "000", codeword = "0000"
    - If you receive "0100" (1bit error) or "0111" (3bit error), you can say they are corrupted, because number of 1s is odd.
    - However, if you receive "0101" (2bit error) or "1111" (4bit error), there's no way to tell if they are corrupted or not
  - (Assume noise only causes bit flipping)

Hamming distance

- Definition (from wikipedia: Hamming distance)
  - Hamming distance between two strings of equal length is the number of positions for which the corresponding symbols are different
  - i.e. Number of substitutions required to change one into the other
- Examples
  - Hamming dist. between 0000 and 1001 is 2
  - Hamming dist. between 0101 and 1010 is 4

Error correction/detection capability

- Minimum Hamming dist between any two codewords determines error correction/detection capability
  - Undetected error happens ONLY when one codeword is received as another one due to noise
  - If minimum Hamming dist between codewords is *n*, any errors up to *(n-1)*bit can be detected
  - (As of now, forget about correction)
- Example
  - Parity bit: min(Hamming dist.) = 2
    - i.e. "Every codeword is different from any other codewords in at least two bit positions"
    - Thus we can detect any 1 bit errors

CRC (Cyclic redundancy check)

- Much stronger than parity bit
  - CRC-32 adds 32bits to the message and detects
    - Any odd number bit errors
      - same as parity bit
    - Any burst errors up to 32bits
    - and many larger bit errors, with high probability
- Basic idea
  - Make all codeword divisible by G ("generator")
    - If a received bitstring is not divisible by G, it's corrupted
  - How? By subtracting the remainder of division
    - ex) msg=45, G=7: 45 % 7 = 3, so (45-3) is divisible by 7
    - In CRC, everything's done in "modulo-2 arithmetic", though

Modulo-2 arithmetic

- No carries in addition
  - ex.) 1+1=0, 11111+11111=00000
- Subtraction is same as addition
  - ex.) 11-10 = 01, 11+10 = 01
- Multiplication is normal (but no carries)
  - ex.) 1101 * 101 = 11011

```
   1101
 x  101
 ------
   1101
1101
------
111001
```

- Division is NOT normal
  - Always remove MSB (Most significant bit)
  - ex.) 101000 = 1101 * 110 + 110
    - i.e. quotient = 110, remainder = 110
  - When divider is r bit, remainder is (r-1) bit

```
        110
1101)101000
     1101
      1110
      1101
       0110
```

BACKUP