



EEC-484/584 Computer Networks

Lecture 5

Wenbing Zhao

wenbing@ieee.org

(Lecture notes are based on materials supplied by
Dr. Louise Moser at UCSB and Prentice-Hall)



Outline

- Review of lecture 4
- Data Link layer
 - Data Link Layer Design Issues
 - Error Detection and Correction

15 September 2005

EEC484/584

Wenbing Zhao



Physical Layer

- Public switched telephone network
 - Modems, Codecs, DSL, FDM, TDM
- Mobile telephone system
 - CDMA
- Cable television

15 September 2005

EEC484/584

Wenbing Zhao



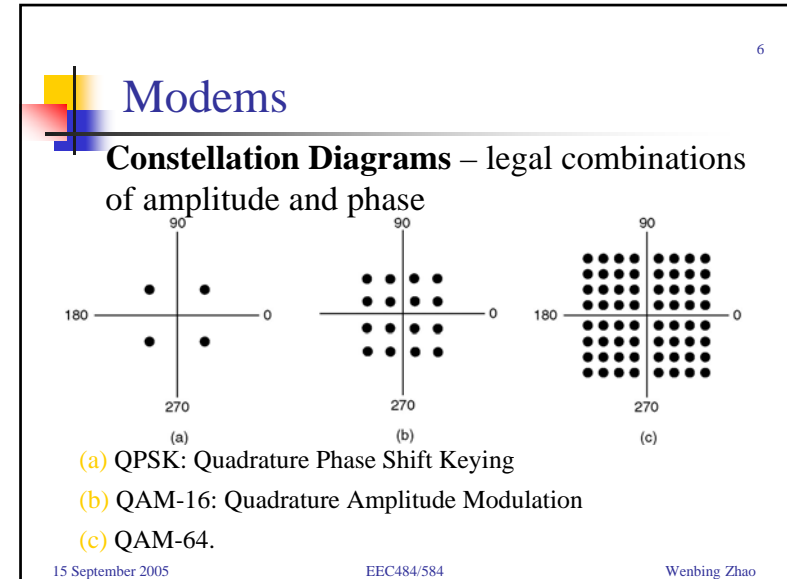
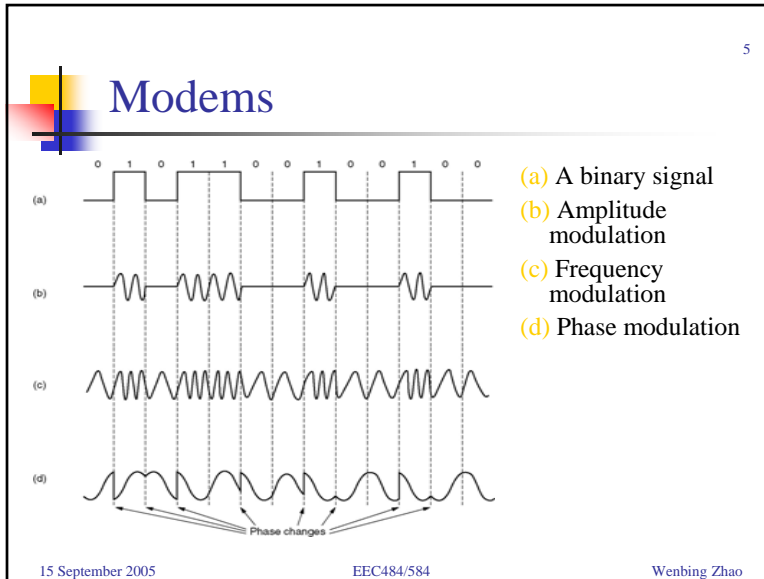
Modems

- **Modem** – device used between digital computer and analog telephone system. It converts digital bit stream into modulated analog signal and vice versa
- **Codec** – inverse of a modem. It is a device that converts a continuous analog signal into a digital bit stream
- **Baud** – number of samples per second. During each baud, one symbol is sent. One symbol can carry multiple bits

15 September 2005

EEC484/584

Wenbing Zhao



- 7
- ## Trunks and Multiplexing
- Idea: multiplex many conversations over single physical channel with high bandwidth
 - **FDM - Frequency Division Multiplexing**
 - Frequency spectrum divided into logical channel
 - Each user has exclusive use of own frequency band
 - **TDM - Time Division Multiplexing**
 - Time divided into slots each user has time slot
 - Users take turns in round robin fashion
- 15 September 2005 EEC484/584 Wenbing Zhao

- 8
- ## Analog to Digital Modulation
- Encoding systems for digitizing analog signals - use statistical techniques to reduce number of bits/channel (signal changes slowly compared to sampling frequency)
 - **Differential pulse code modulation** - output difference between current value and previous value rather than digitized amplitude
 - **Predictive encoding** - Extrapolate previous few values to predict next value. Encode difference between actual signal and predicted one
 - **Delta modulation** - Requires each sampled value to differ from its predecessor by +/-1
- 15 September 2005 EEC484/584 Wenbing Zhao

Data Link Layer Design Issues

- Services Provided to the Network Layer
 - Point-to-point, source-to-destination
- Framing
 - Physical bit stream divided up into frames
- Error Control
 - Acknowledgements (acks) and negative acks (nacks)
 - Sender has timer –within timeout get ack, or send again
 - If frame transmitted multiple times, receiver may get several copies - Use sequence numbers

Data Link Layer Design Issues

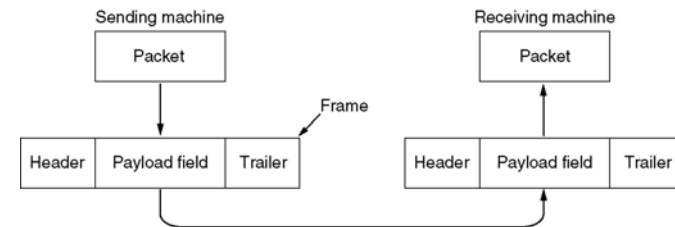
- Flow Control
 - Sender may transmit frames faster than receiver can receive them
 - Throttle sender so sends no faster than receiver can receive them

Functions of the Data Link Layer

- Provide service interface to the network layer
 - Virtual source-to-destination communication channel
- Dealing with transmission errors
- Regulating data flow
 - Slow receivers not swamped by fast senders

Functions of the Data Link Layer

Relationship between packets and frames



13

Services Provided to Network Layer

(a) Virtual communication (b) Actual communication

15 September 2005 EEC484/584 Wenbing Zhao

14

Services Provided to Network Layer

- Placement of the data link protocol
 - When frame arrives at router
 - hardware verifies checksum
 - Passes frame to DL layer software
 - DL layer software checks if frame is one expected
 - If so, gives packet in payload of frame to routing software
 - Routing software chooses appropriate outgoing line
passes packet back down to DL layer software
 - DL layer software transmits it

15 September 2005 EEC484/584 Wenbing Zhao

15

Services Provided to Network Layer

Router

Data link layer process Routing process

Frames here Packets here

Data link protocol

Transmission line to a router

15 September 2005 EEC484/584 Wenbing Zhao

16

Services Provided to Network Layer

- Types of service
 - Unacked connectionless
 - Ok if low error rate, real time applications
 - Acked connectionless
 - Connection-oriented
 - Each frame sent is received exactly once in right order

15 September 2005 EEC484/584 Wenbing Zhao

Services Provided to Network Layer

- Three phases of connection-oriented
 - Connection established, variables and counters initialized
 - Frames transmitted
 - Connection released, buffers, variables, etc. freed

Framing

- DL layer divides physical bit stream into frames
- Checksum computed by source included in frame
- Checksum recomputed by destination and checked against checksum included in the frame
 - Discard/recover bad frame, notify source

How Does DL Layer Form Frames?

- Insert time gaps between frames
 - Too risky, no timing guarantees, not used
- Character count
- Flag bytes with byte stuffing
- Starting and ending flags, with bit stuffing

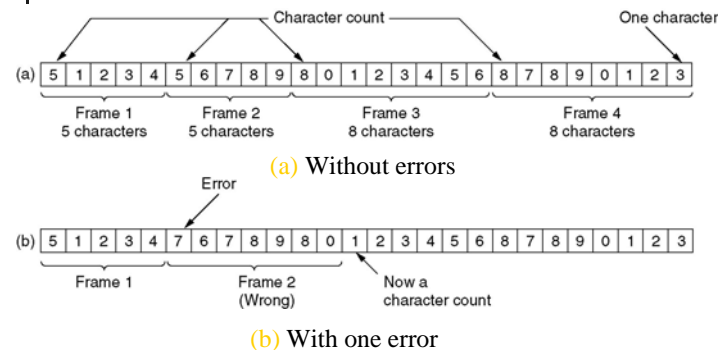
How Does DL Layer Form Frames?

- Physical layer coding violations
 - Applies only if encoding on medium contain some redundancy
 - Example: encode 1 bit with 2 bits
 - 1 => 10
 - 0 => 01
 - Can use 00 or 11 to delimit frames

Framing – Based on Character Count

- Use field in header to indicate number of characters in frame
 - Count might be garbled, not used

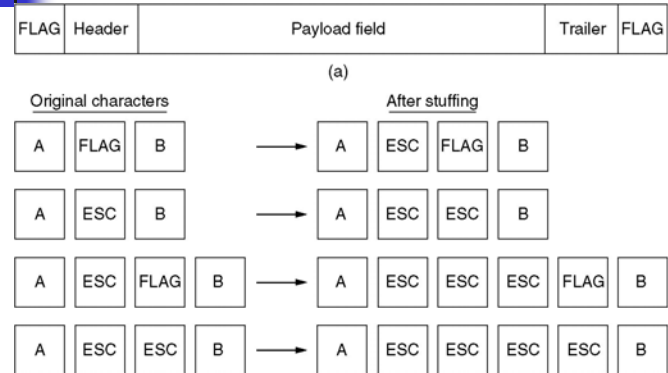
Framing – Based on Character Count



Framing – Based on Byte Stuffing

- Each frame starts and ends with a special flag byte
 - Problem: flag byte might appear in data
 - Solution:
 - Source inserts ESC (DL escape) before each flag byte; ESC before each ESC
 - Destination removes inserted ESC bytes
- Disadvantage: depends on 8-bits characters in ASCII

Framing – Based on Byte Stuffing



Framing – Based on Bit Stuffing

- Each frame begins and ends with special bit patterns, 01111110 (in fact, a flag byte)
 - When source's data contains 11111, stuff 0
 - When destination receives 111110, deletes 0
 - Advantages:
 - Allows arbitrary number of bits per frame
 - Allows arbitrary number of bits per character

Framing – Based on Bit Stuffing

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0

Stuffed bits

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Error Detection and Correction

- Causes of errors
 - Transmission errors on phone lines due to thermal noise
 - Data transmission errors due to impulse noise
 - Signals are separated, distorted, recombined
 - Crosstalk between physically adjacent wires
 - Compression and decompression
 - Receiver out of synch with sender
- Errors usually occur in bursts

Error-Correcting Codes

10001001
 10110001
 00111000

- n -bit **codeword** - an n -bit unit containing data and check bits (m bits of data, r bits redundant/check bits)
- Given any two codewords, it is possible to determine how many corresponding bits differ, using exclusive OR and counting number of 1 bits in the result

Error-Correcting Codes

- **Hamming distance** - number of bit positions in which two codewords differ
 - If two codewords are a Hamming distance d apart, it will require d single-bit errors to convert one into the other

Error-Correcting Codes

- In general, all 2^m possible data messages are legal, but not all 2^n possible codewords are used
- Given the algorithm for computing the check bits, it is possible to
 - Construct a complete list of legal codewords
 - Find two codewords whose Hamming distance is minimum
 - This distance is the Hamming distance of the **complete code**

Error-Correcting Codes

- To detect d errors, need a distance $d+1$ code
 - No way that d single-bit errors can change a valid codeword into another valid codeword
- To correct d errors, need a distance $2d+1$ code
 - Legal codewords are so far apart that even with d changes, original codeword is still closer than any other codeword, so it can be uniquely determined

Error-Correcting Codes

- **Parity bit** - a single bit is appended to the data
 - Parity bit is chosen so that number of 1 bits in the codeword is even or odd
 - Given 1011010
 - with even parity => 10110100
 - With odd parity => 10110101
 - A code with a single parity bit has a distance 2
 - Since any single-bit error produces a codeword with wrong parity => can be used to detect single bit errors

Error-Correcting Codes

- Example for error-correcting code
 - Consider a code with only four valid codewords
 - 0000000000, 0000011111, 1111100000, 1111111111
 - This code has a distance 5 => can correct double errors
 - If 0000000111 arrives, receiver knows the original must have been 0000011111
 - However, if triple error changes 0000000000 to 0000000111, the error will not be corrected properly

Theoretical Lower Limit

- Want to design a code with m message bits and r check bits that will allow all single errors to be corrected
- Each of 2^m legal messages has n illegal codewords at a distance 1 from it
 - Formed by systematically inverting each of the n bits in the n -bit codeword formed from it
- Thus, each of the 2^m legal messages requires $n+1$ bit patterns dedicated to it

Theoretical Lower Limit

- Since total number of bit patterns is 2^n , must have $(n+1)2^m \leq 2^n$
 - $n = m+r \Rightarrow (m+r+1) \leq 2^r$
 - Given m , this puts a lower limit on the number of check bits needed to correct single errors

Hamming Code

- This lower limit can be achieved using a method due to Hamming
 - The bits of codeword are numbered consecutively, starting with bit 1 at the left end, bit 2 to its immediate right and so on
 - The bits that are powers of 2 (1,2,4,8,16,etc) are check bits
 - The rest are filled up with the m data bits

Hamming Code

- Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.
 - Position 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1,3,5,7,9,11,13,15,...)
 - Position 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc. (2,3,6,7,10,11,14,15,...)
 - Position 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc. (4,5,6,7,12,13,14,15,20,21,22,23,...)
 - Position 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc. (8-15,24-31,40-47,...)
 - etc.

Hamming Code

- Set a parity bit to 1 if the total number of ones in the positions it checks is odd. Set a parity bit to 0 if the total number of ones in the positions it checks is even (assuming even parity is used)

Hamming Code - Example

- Data (character 'H'): 1001000
- Create the data word, leaving spaces for the parity bits:
 - `__ 1 _ 0 0 1 _ 0 0 0`

Hamming Code - Example

- Calculate the parity for each parity bit
 - Position 1 checks bits 1,3,5,7,9,11:
 - `? _ 1 _ 0 0 1 _ 0 0 0`. Even number of 1's, set position 1 to 0:
`0 _ 1 _ 0 0 1 _ 0 0 0`
 - Position 2 checks bits 2,3,6,7,10,11:
 - `0 ? 1 _ 0 0 1 _ 0 0 0`. Even number of 1's, set position 2 to 0:
`0 0 1 _ 0 0 1 _ 0 0 0`
 - Position 4 checks bits 4,5,6,7:
 - `0 0 1 ? 0 0 1 _ 0 0 0`. Odd number of 1's, set position 4 to 1:
`0 0 1 1 0 0 1 _ 0 0 0`
 - Position 8 checks bits 8,9,10,11,12:
 - `0 0 1 1 0 0 1 ? 0 0 0`. Even number of 1's, set position 8 to 0:
`0 0 1 1 0 0 1 0 0 0 0`
- Code word: 00110010000

Hamming Code

- When a codeword arrives,
 - Receiver initializes a counter to 0,
 - It then examines each check bit, k , to see if it has the right parity.
 - If not, adds k to the counter.
 - If counter is 0 after all check bits, the codeword is accepted as valid.
 - If counter is nonzero, it contains the number of the incorrect bit
 - e.g., if 1, 2, 8 in error, bit in position 11 is inverted
 - **Can only correct single-bit error**

Hamming code

- Hamming code simulation (Java applet)
 - http://www.frontiernet.net/~prof_tcarr/Hamming/applet.html#APPLET

Hamming code

- Trick to permit Hamming codes to correct burst errors
 - A sequence of k consecutive codewords are arranged as a matrix
 - Transmit a column at a time
 - Matrix is reconstructed at receiver
 - If a burst error occurs, at most 1 bit in each of the k codewords will have been affected

Char.	ASCII	Check bits
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	01111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	10101011111
d	1100100	11111001100
e	1100101	00111000101

Order of bit transmission

Error-Detecting Codes

- If a single parity bit is appended to a block, error detecting probability is only 0.5 if burst error occurs
- This can be improved by treating a block as a matrix, n bits wide and k bits high,
 - A parity bit is computed for each column and affixed to the matrix as the last row
 - The matrix is transmitted one row at a time
 - Probability of accepting bad block is 2^{-n}

Error-Detecting Codes

- Polynomial code, also known as CRC (Cyclic Redundant Code)
 - Treat bit string as polynomial with 0 and 1 coefficients
 - M-bit frame: $M(x) = b_{m-1}x^{m-1} + \dots + b_0$
 - Example: 11011010 $\Rightarrow M(x) = x^7 + x^6 + x^4 + x^3 + x^1$
 - Use modulo 2 arithmetic
 - No carries or borrows - XOR

Cyclic Redundant Code

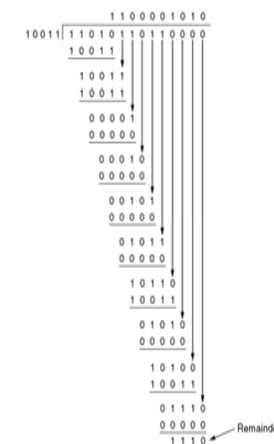
- Sender and receiver agree on generate polynomial $G(x)$, with high and low order bits = 1
- To compute checksum for some frame with m bits corresponding to $M(x)$
 - $m > \deg G(x) = r$
- Append checksum to end of frame so polynomial $T(x)$ corresponding to checksummed frame is divisible by $G(x)$
- When receiver gets checksummed frame, divides $T(x)$ by $G(x)$
- If remainder $R(x) \neq 0$, then transmission error

Algorithm to Compute CRC Checksum

- Let $m = \deg M(x)$, $r = \deg G(x)$
- Append r 0 bits to lower-order end of frame to obtain corresponding polynomial $x^r M(x)$
- Divide bit string corresponding to $x^r M(x)$ by bit string corresponding to $G(x)$
- Subtract remainder $R(x)$ from bit string corresponding to $x^r M(x)$
 - Result is checksummed frame
 - Let $T(x)$ be its polynomial
 - $x^r M(x) = Q(x)G(x) + R(x)$
 - $x^r M(x) - R(x) = Q(x)G(x) = T(x)$

Compute CRC Checksum

Frame : 1101011011
 Generator: 10011
 Message after 4 zero bits are appended: 11010110110000



International Standard Polynomials

- **CRC-12** $G(x) = x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
 - Used for 6-bit characters
- **CRC-16** $G(x) = x^{16} + x^{15} + x^2 + 1$
CRC-CCITT $G(x) = x^{16} + x^{12} + x^5 + 1$
 - Used for 8-bit characters
- **CRC-32** $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$
 - Used in IEEE 802
 - Detects all bursts of length 32 or less and all bursts affecting an odd number of bits