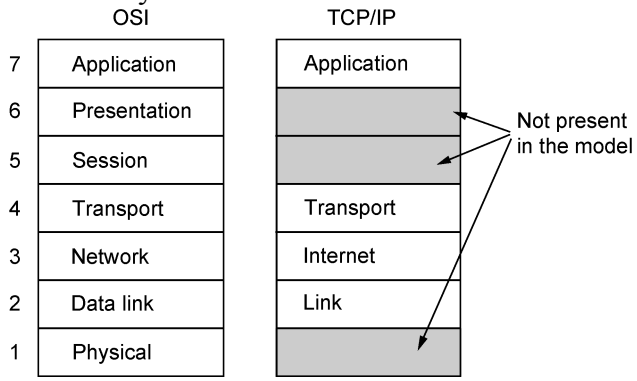
Lecture 8
Plan: cover 3.1 and 3.2
Look ahead

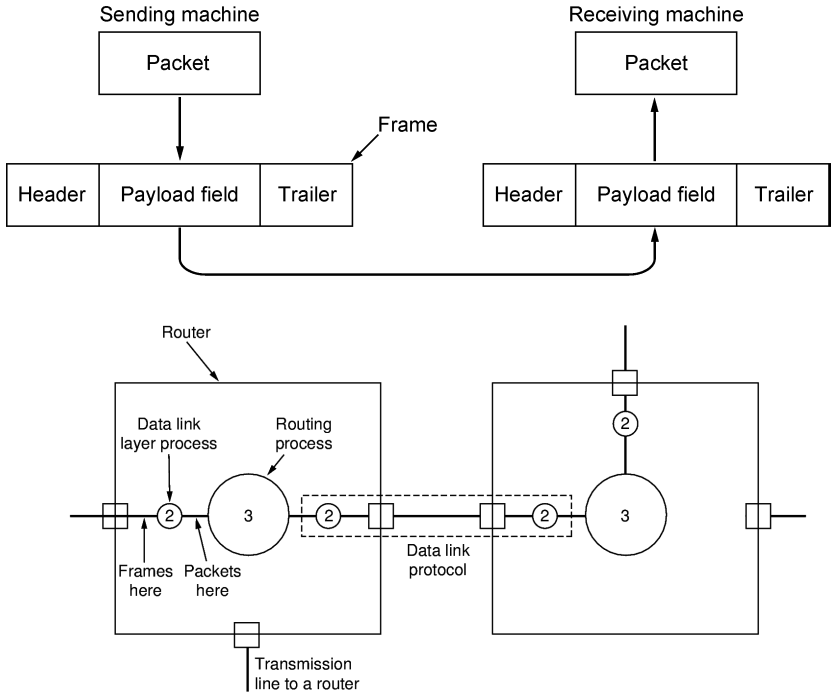
- In chapter 2, we primarily focused on data transmission (how are 0/1's transmitted over different type of networks/medium. What technologies are available to us (modulation, multiplexing, packet switching).)
- The layered structure:



  - physical layer
  - data link layer
  - network layer
  - transport layer
  - application layer
- Chapters 3 and 4 covers the data link layer – switch to data communications. (don't worry about how exactly 0/1's are transmitted. We know it's there).
  - C3 deals with point to point communications, I.e. host to host with a dedicated link.
  - C4 deals with additional issues in broadcast networks (channel access protocol)

1. Data link layer

- Objective: provide an "enhanced" bit stream service to network layer with the support of bit stream service from the physical layer
  Q:/ How enhanced?
  A:/
  - Framing: aids error checking and synchronization
  - Error Control (error detection/correction and automatic repeat request (ARQ)
  - Flow Control (regulate the flow of data between hosts running at different speeds)
- Possible service to the network layer
  - unacknowledged connectionless (less reliable with less work and less overhead) – most LANs use this
  - acknowledged connectionless
  - acknowledged connection-oriented (more reliable with more work and more overhead)
    Q:/ Which service should we use?
    A:/ If the medium is inherently error-prone (such as wireless). It's better to do it here and save work at the transport layer
    If the medium is inherently error-free (such as fiber or high quality copper based medium) better not to do it.
    Q:/ Why do we want to acknowledge in datalink layer instead of the network layer?
    A:/ Because frames usually have a size limit and packets don't

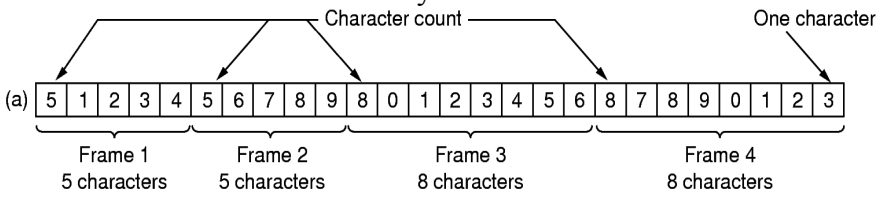


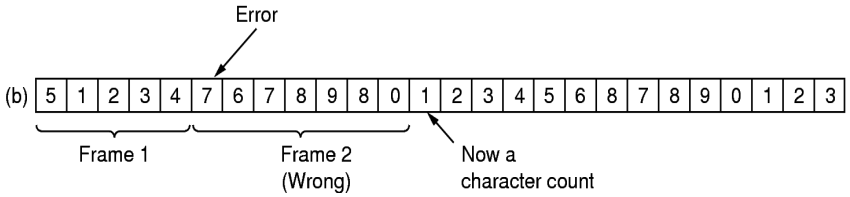
2. Framing

- Basic idea is to divide the bit stream into chunks of limited size (usually there is an upper limit) and send one chunk a time.
  Q:/ Why?
  A:/ This limits the damage if an error occur (just need to resend the frame instead of the whole data set again)
  Q:/ What kinds of errors do we expect?
  A:/ An error in computer communications is a flip from 1 to 0 or 0 to 1. Or a lost of bits or a whole frame.
  - Single-bit error: isolated bit error that does not affect nearby bits
  - burst error: a continuous sequence of B bits in which the first and last bits and any number of bits in between are received in error (the bits in between can have error or not but the first and the last must be errors)
    e.g. 1<u>10110</u>011011 → 1<u>00001</u>011011 (burst error of length 5)
    e.g. an impulse noise event of 1 microsecond occurs. At a data rate of 100Mbps, there is an error burst of 100 bits.
  - If indeed there are errors, the send can resend a frame.
    Q:/ What if a frame is lost? → the receiver has no idea that a frame is missing → no report of error
    A:/ needs a timer (if time expires without an acknowledgement, then resend

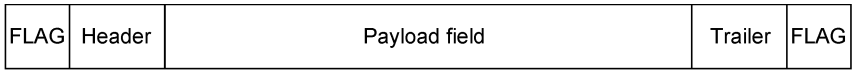
Q:/ It is possible that the acknowledgement is lost. → sender may send the same frame multiple times.
A:/ give each frame a sequence number.

- How to divide the bit stream into frames?
  A:/ need to identify the beginning and the end of a frame.
  - Byte count: give the size of the frame. → not good because the size itself can be contaminated. → hard to resynchronize

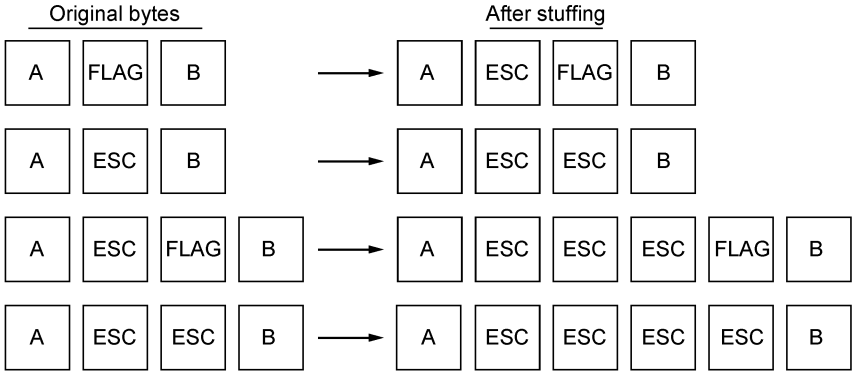




  - Character stuffing for character oriented transmission
    Use a flag byte at the beginning and the end of the frame.
    Q:/ What if the receiver loses synchronization?
    A:/ Just wait for the occurrence of the two consecutive flag bytes
    Q:/ What may cause problems?
    A:/ The flag byte happens to appear in our data.
    Q:/ How to handle it?
    A:./ Use escape sequences ESC (or any other unique sequence) and insert it in front of each occurrence of the flag byte. Also make sure any ESC occurred in data will be ESC'ed too.



(a)



(b)

    Q:/ What if the data is not 8-bit characters, such as UNICODE (16 bits)
    A:/ character stuffing needs to introduce a new symbol. → disadvantage.
  - Bit stuffing for bit oriented transmission
    bit flag: 01111110
    the transmitted data: after any 5 consecutive 1's, insert a 0. The receiver will remove them.

What is the result of big stuffing a?

(a) 0 1 1 0 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 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 0 0 1 0

2. Error Detection/Correction:

- Basic Idea: adding redundancy to the transmitted message
  - detection: the receiver knows that something is wrong with the received data. Ask the sender to re-send
  - correction: the receiver not only knows if something is wrong but also is able to correct the mistakes.
    Q:/ Shouldn't we always use correction scheme?
    A:/ correction needs more information included. Thus more redundancy → use more bandwidth.
    Q:/ When to use correction?
    A:/ in wireless network, the error rate is high. To avoid large number of retransmission, correction is preferred.

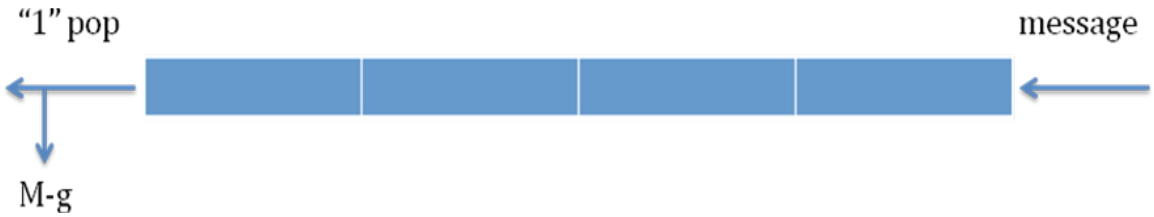
1). Error Detection

CRC (Cyclic Redundancy Coding)

- Basic Idea: Based on the arithmetic of polynomials with coefficients taken from Z_2 (ring of integers mod 2). In this domain, 2 is the same as 0.
  - 1=-1, so addition is the same as subtraction
  - carries and borrows are never necessary. They are both the same as exclusive-or
    e.g. multiply $x^3 + x^2 + 1$ with x^7+x^5+x+1.
    result is $x^{10}+x^9+x^8+x^5+x^4+x^2+x+1$
    e.g. divide 1011101 by 1101
    result is 1100 with remainder 1
  - CRC scheme: define a generating polynomial (known by both the sender and the receiver) g(x) of order r (the highest degree is r). Suppose we want to send a message of m bits, represented by a polynomial m(x) of degree m
    - Append r zeros to the message (the resulting polynomial is $x^r m(x)$)
    - Divide $x^r m(x)$by g(x) and determine the remainder r(x)
    - The binary representation of r(x) is the CRC code. The sender sends the message followed by the CRC code (corresponds to $x^r m(x)$-r(x))
    - When the message is received, the sender divides it by g(x). If the remainder is not zero, error has occurred.
      e.g. g=10011, message = 11001011010
      - the g(x) is of order 4, so append 4 zeros to the message
      - divide it by g. the remainder is 1011.
      - The sender sends the original message followed by the 4 bit remainder. 110010110101011
      - When the message arrives, the receiver divides the received message by g. If the message is changed, the remainder will not be zero.
  - Why does it work?
    g(x) should divide the $x^r m(x)$-r(x)completed since the remainder is already subtracted.
  - Division is not fun in calculations. → we can use a shift register to compute.

Receiver algorithm:

```
M = 0; // it is the length of the remainder, i.e. CRC code length
while more bits {
   lshift(M);
   M = M | next bit;
   if(M_high == 1)
      M = M xor g;
}
if(M!=0) error
```



- This might look a bit messy, but all we are really doing is "subtracting" various powers (i.e. shiftings) of the poly from the message until there is nothing left but the remainder. demonstrate the previous example if time allows.
- In the real engineering world, IEEE 802 uses a generating polynomial of degree 32.
  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
  It can catch all single or double bit errors; all erros with an odd number of bits; all burst erros of length 32 or less