Data Link Layer (1d)

Specific services to carry out:

- *framing*
  - determining how the bits of the physical layer are grouped into frames
  - addresses used in frame headers to identify destination are different from IP addresses!
- *dealing with transmission errors:*
  - *error detection*:
    - receiver detects presence of errors:
    - signals sender for retransmission and drops frame
  - *error correction:*
    - receiver identifies *and corrects* bit error(s) without resorting to retransmission
- *flow control:*
  - pacing between adjacent sending and receiving nodes
- *separate MAC sublayer*
  - controlling channel access if the link is a shared medium

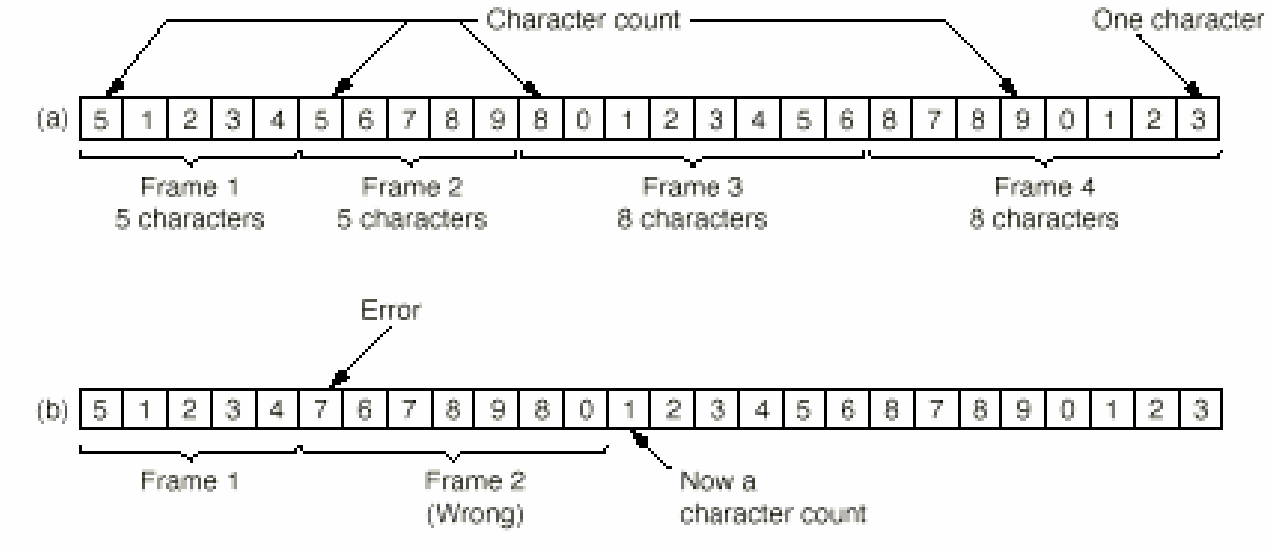
Data Link Layer(2)

Grouping into frames

Methods for framing:

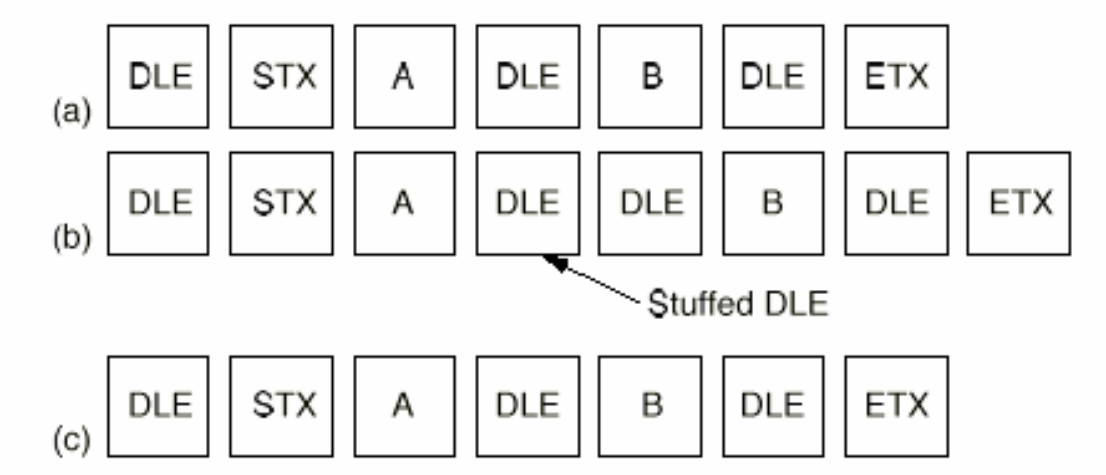
- character count
- character stuffing
- bit stuffing
- exploiting redundancy in the physical layer (Manchester encoding)

Example for character count:

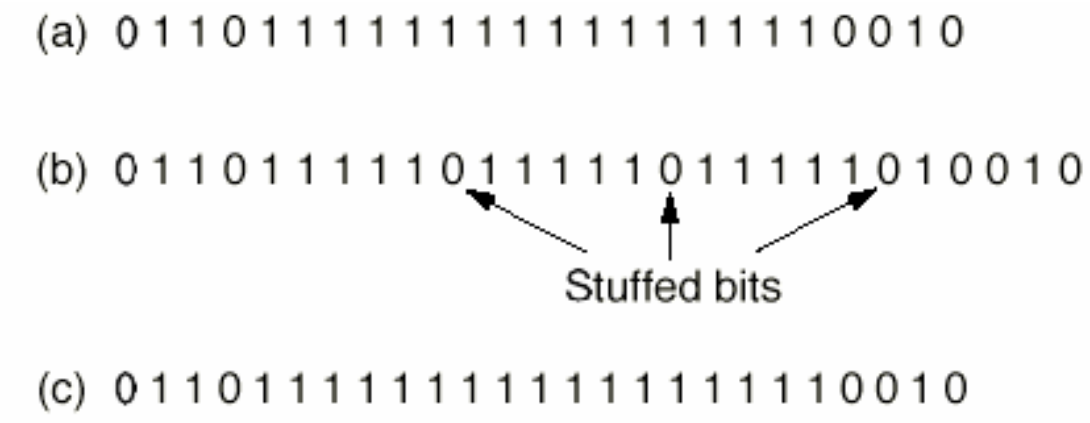


Data Link Layer(3)

Example for character stuffing:



Example for bit stuffing:



Data Link Layer(4)

Dealing with transmission errors (error control)

Error Detecting Codes (EDC) and Error Correcting Codes (ECC)

Definitions:

Codeword:= source (original) word + (redundant) check (control) bits

m:= length of the source word (number of information bits)

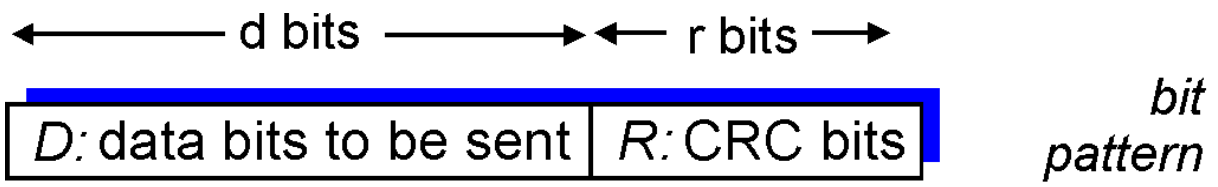
r:= number of check bits

n:= m + r := length of the codeword --->

A binary code is a subset of R^2_n . Ist elements (words) also can be considered as code vectors.

Example for Checksumming: Cyclic Redundancy Check (CRC)

- view data bits, D, as a binary number
- choose r+1 bit pattern (generator), G
- goal: choose r CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!



$D * 2^r \quad XOR \quad R$ *mathematical formula*

CRC Example

Want:

$D \cdot 2^r$ XOR R = nG

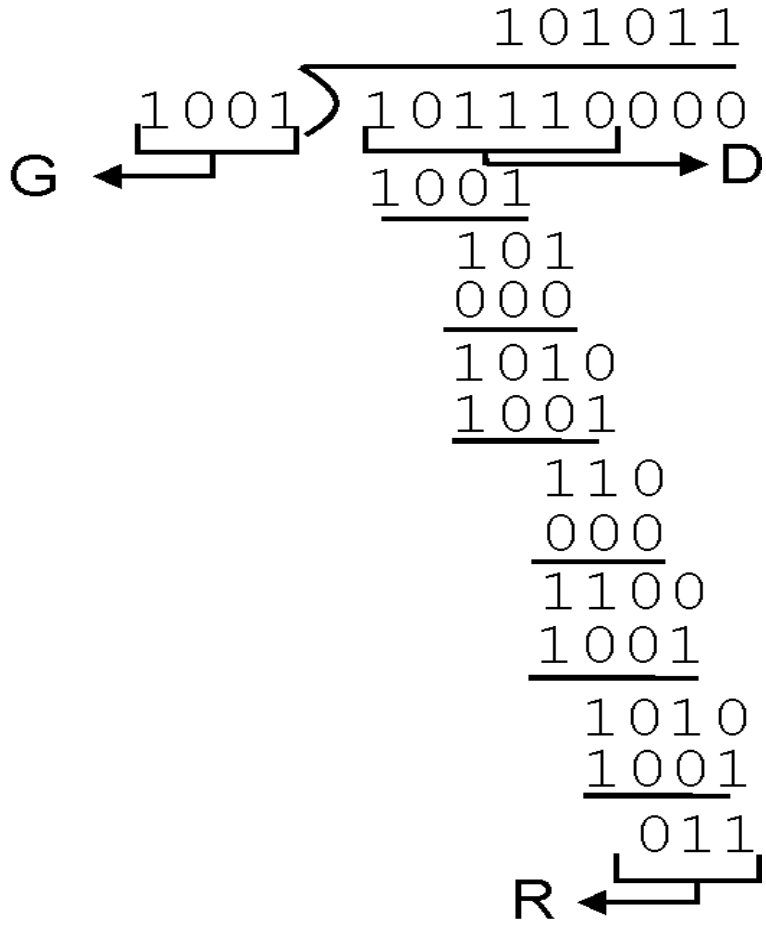
equivalently:

$D \cdot 2^r$ = nG XOR R

equivalently:

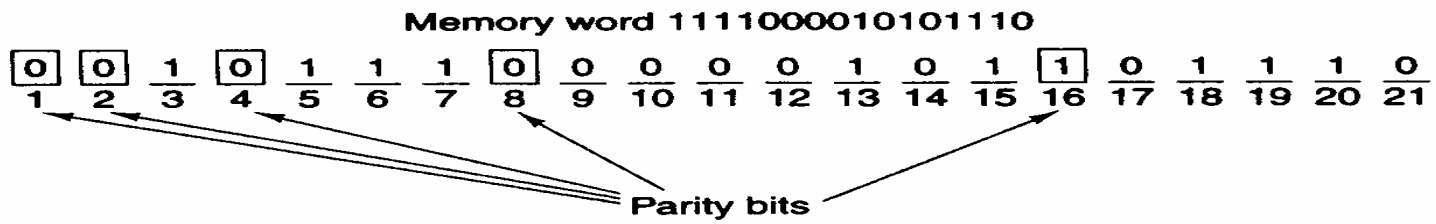
if we divide $D \cdot 2^r$ by G, the remainder will be R

$$R = \text{remainder}\left[\frac{D \cdot 2^r}{G}\right]$$



Data Link Layer(7)

Example for Hamming-Code:



parity bit 1 **wrong** (1,3,5,7,9,11,13,15,17,19,21 contain altogether **5** Ones)

parity bit 2 **correct** (2,3,6,7,10,11,14,15,18,19, contain altogether **6** Ones)

parity bit 4 **wrong** (4,5,6,7,12,13,14,15,20,21 contain altogether **5** Ones)

parity bit 8 **correct** (8,9,10,11,12,13,14,15 contain altogether **2** Ones)

parity bit 16 **correct** (16,17,18,19,20,21 contain altogether **4** Ones)

---> Bit No. 5 is wrong and has to be inverted!

Use of ECCs :

•Heavily disturbed transmission channels (wireless) or those with long propagation times (space missions)

•Data transmission in military (secret service) applications

•Secondary storage media (e.g. CD‘s), to correct reading errors instantaneously (without timing delay)

Data Link Layer(13)

Providing Flow Control

Simple Stop-and-Wait Protocol (rdt 2.0)

Assumptions:

- No infinite amount of buffer space available on the receiver's side (otherwise: always send (rdt 1.0))
- Communication channel is assumed to be error-free

Problem:

How to prevent the sender from flooding the receiver with data faster than the latter is able to process which results in losing frames at the receiver's side

Solution approach:

Having the receiver provide feedback to the sender by acknowledging each frame sent

---> the sender must wait until an *ack* frame arrives before fetching the next frame from the network layer

Piggybacking (if data traffic is duplex (bidirectional)) **:**

Delaying outgoing *acks* so that they can be hooked on the next outgoing data frame (having a separate *ack* field in its header)

Principal advantage over having separate ack frames:

Better use of the available channel bandwidth

Data Link Layer(14)

Sliding window protocols

Assumptions:

- Communication channel is not perfect, i.e. data frames and acks could be lost
  ---> time-out mechanism for the sender

Problem:

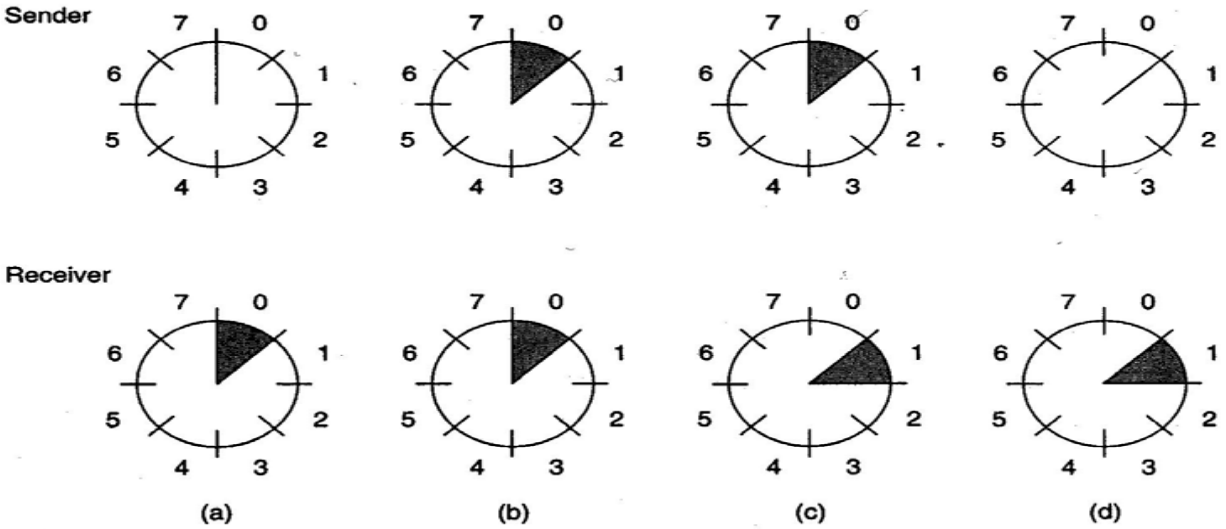
---> How long is timeout? ---> Early time-out ---> synchronization problem!

Solution approach:

Having sliding window protocols were each data frame contains a sequence number

---> for a stop-and-wait protocol it needs only 2 numbers (0 and 1), i.e. a one-bit sequence number

Illustration of the main idea of sliding windows



Data Link Layer(15)

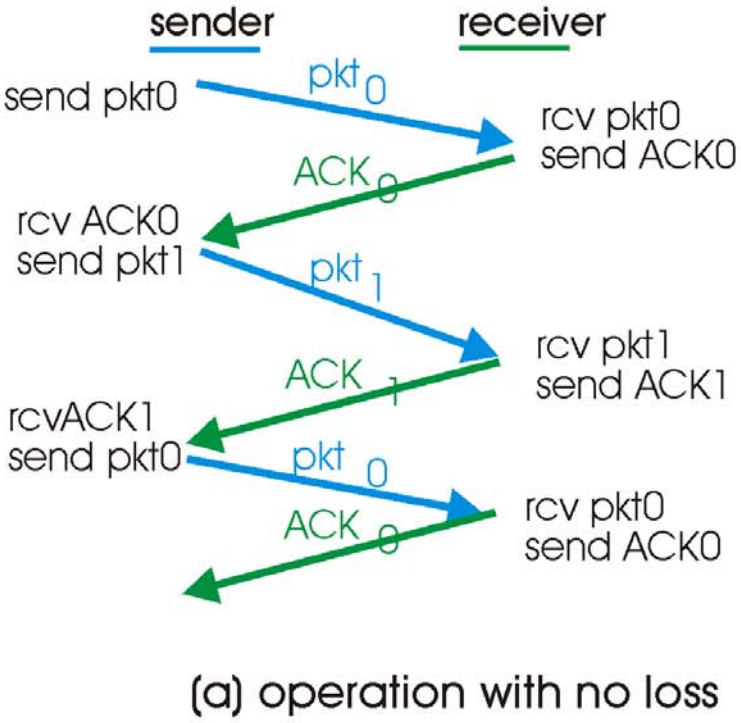
One Bit Sliding Window Stop-and-Wait Protocol (rdt 3.0)

- Ack field contains the sequence number of the last frame received without error
- Agreement with the number of the frame being sent ---> fetch the next frame from the network layer
- No agreement ---> no sending action

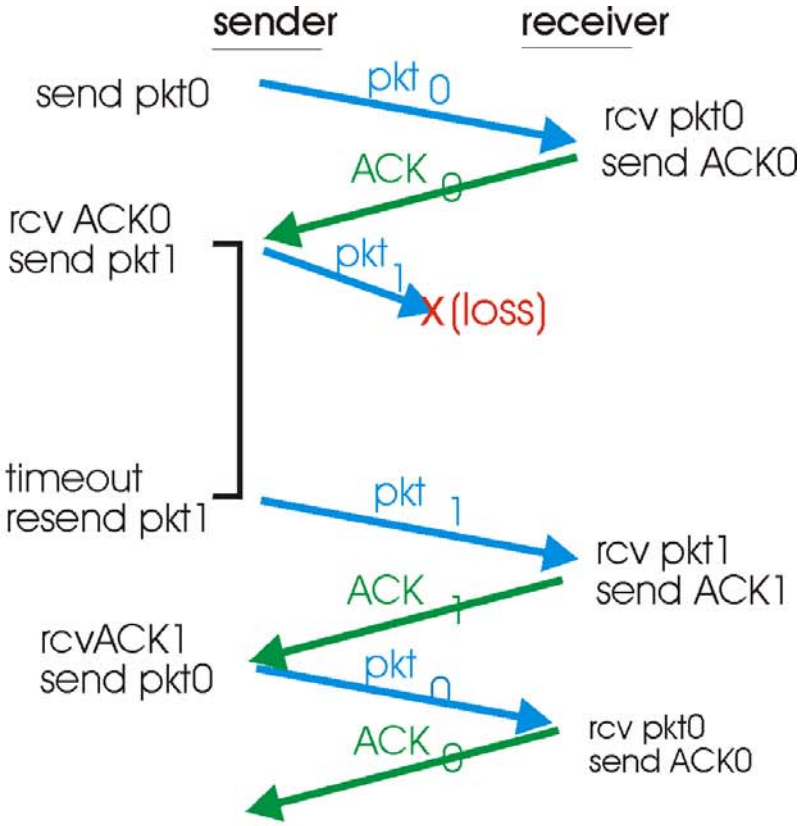
Resulting Quality of Service

- No delivering of duplicate packets to either network layer
- No packet is skipped
- No deadlock

rdt3.0 in action (1)

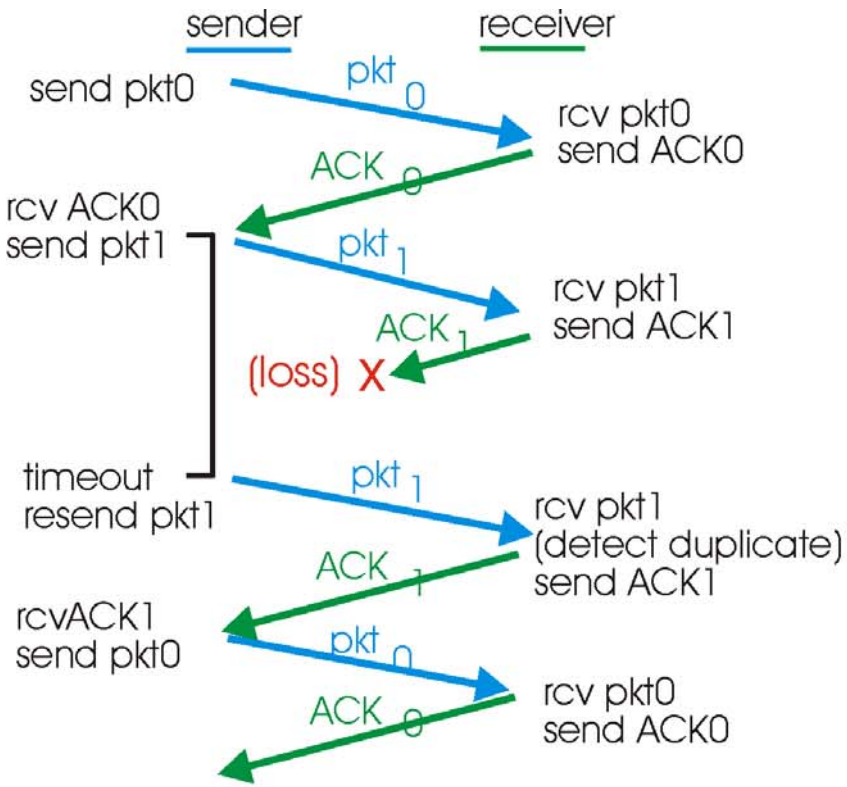


(a) operation with no loss

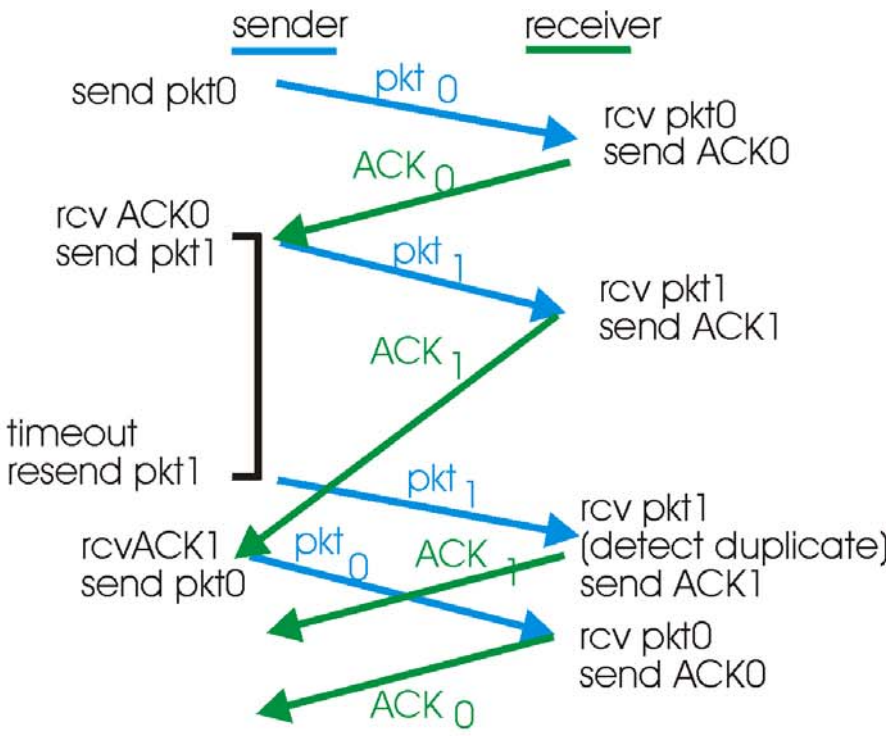


(b) lost packet

rdt3.0 in action (2)



(c) lost ACK



(d) premature timeout