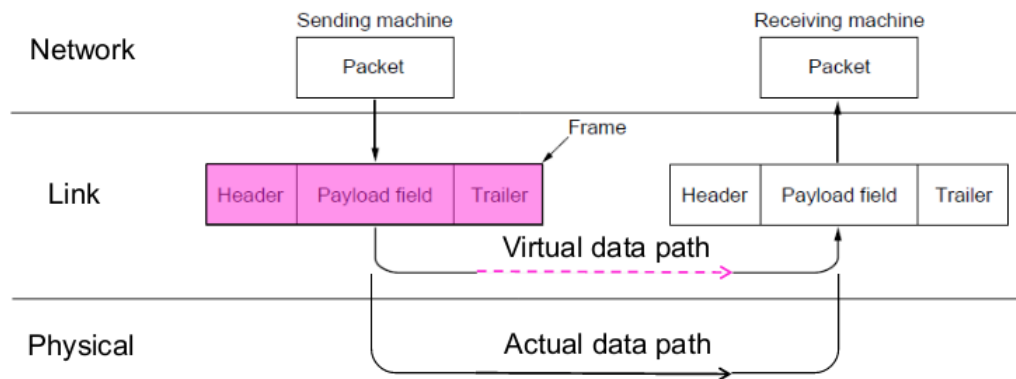# Data Link Layer

1   Frames

- Link layer accepts packets from the network layer, and encapsulates them into frames that it sends using the physical layer; reception is the opposite process

- The physical layer (below) is responsible for the transmission of raw sequences of bits
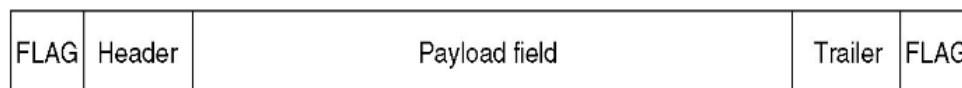

2   Framing methods

2.1   Byte count

Frame begins with a count of the number of bytes in it - simple, but difficult to resynchronize after an error

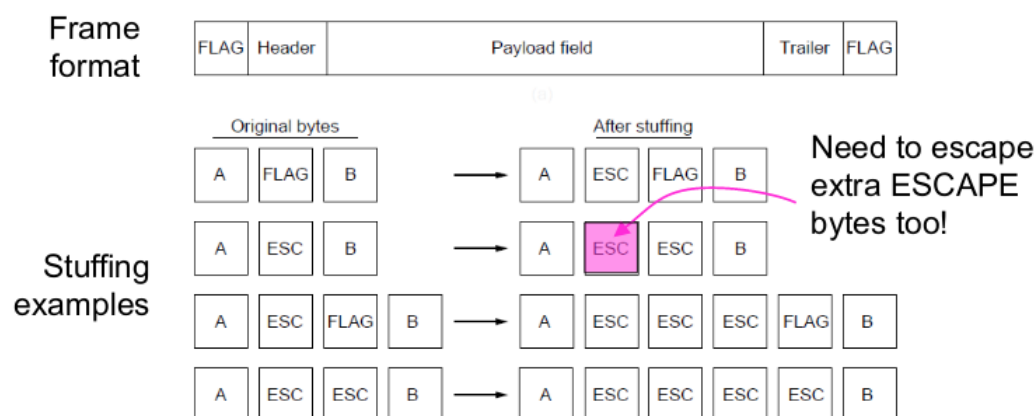### 2.2   Headers and trailers

We add a header (e.g. the length of the data, etc.) and a trailer (extra data that can be used e.g. error-detection or error-correction)
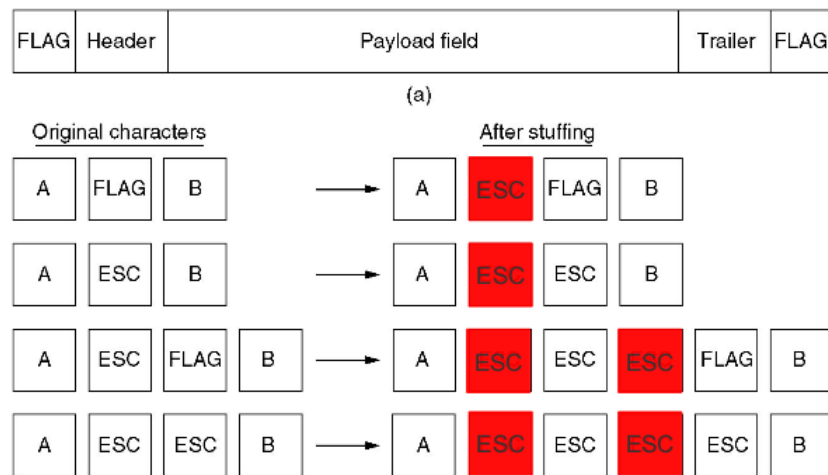

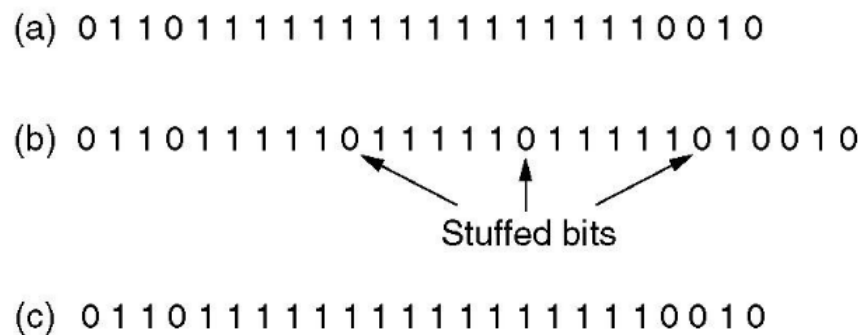Two adjacent frames are separated by a flag

2.3   Byte stuffing

Special flag bytes delimit frames; occurrences of flags in the data must (escapes) - longer, but easy to resynchronize after error

2.4 Bit stuffing

The flag is six consecutive ones. Within the data, a zero is stuffed after each five consecutive ones to void instances of the flag in the data

Bit stuffing

(a) The original data

(b) The data as they appear on the line

(c) The data as they are stored in receiver's memory after de-stuffing

Sender:

- Encloses packet (bit stream): 0 1 1 1 1 1 0

- Appends a 0 after each 1 1 1 1 1 in body (bit stuffing)

Receiver, upon receiving 0 1 1 1 1 1:

- next bit 0: stuffed bit is removed

- next bit 1

  - if next bit 0: end of frame marker
 - if next bit 1: error

3 Error detection and correction

- Errors occur during frame transmission

- Two strategies to deal with error

    – Include enough redundant information to help receivers deduce original data (error correcting)

    – Include enough information to deduce an error occurred (error detecting)

    – Neither error-correcting codes nor error-detecting codes can handle all possible errors

EDC = Error detection and correction bit (redundancy)
D = Data protected by error checking, may include header fields

Error detection is not 100% reliable, protocol may miss some errors, but rarely
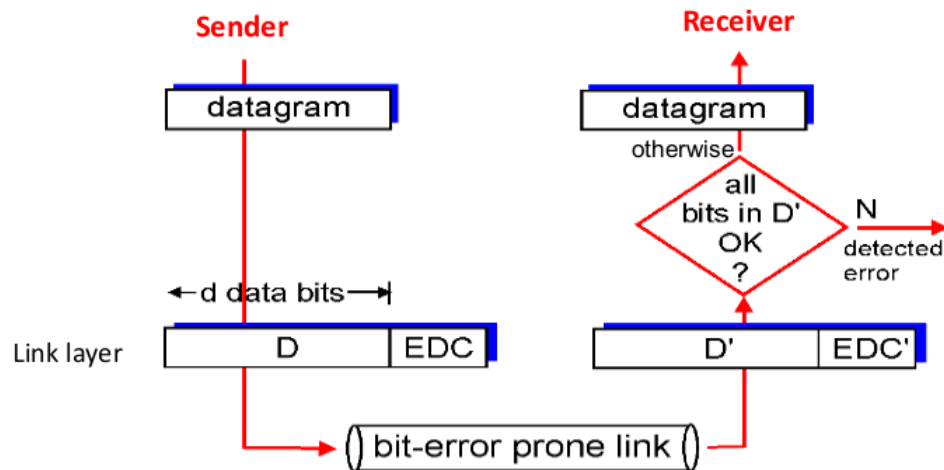
4 Codeword

- A frame consists of m data (message) bits and r redundant (check) bits

- n-bit codewords with n=m+r

5 Error codes

Error codes add structured redundancy to data so errors can be either detected, or corrected

5.1 Hamming codes

> **Definition: Hamming distance**
>
> The number of bit positions in which two codewords differ

We will see a specific hamming code that corrects a single error:

- Encoding: we number the data bits starting from one and skipping the powers of two. The powers of two are reserved for parity bits, the rest are message bits.

- Decoding: calculate all parities. If they are all OK, there was no error. If not, add up the positions of the incorrect ones - this gives us the position of the error

Hamming code gives a simple way to add check bits and correct up to a single bit error

- Check bits are parity over subsets of the codeword

- Recomputing the parity sums (syndrome) gives the position of the error to flip, or 0 if there is no error

6   Error detection

6.1   Parity

Parity but is added as the modulo 2 sum of data bits

Single bit parity - detect single bit errors

Two dimensional bit parity - detect and correct single bit errors

6.2   Checksum

The sender:

1. Data is divided into k segment each of m bits (normally 16 bit integers) and summed

2. The 1s complement of the sum forms the checksum

3. The checksum segment is sent along with the data segments

The receiver

1. All received segments are added

2. The sum is complemented

3. If the result is zero, the received data is OK; otherwise wrong

6.3   Cyclic Redundancy Check

This is a more advanced technique, which uses manipulations with polynomials. This is one of the most commonly used error detection codes

Basic approach:

- Let M1 be the message of n-bits

- M1 is padded with CRC code and send it

- CRC is generated using a given polynomial P1 (or divisor)

- P1 is agreed between sender and receiver

- Receiver gets M1 + CRC and extracts the M1 using P1

■ Sender

Receiver

M Data | n bit

↓

divisor

↓ n bits

CRC

Data | CRC

Data | CRC

↓

divisor

↓ n bits

reminder

N   Y

reject ← ⟨zero⟩ → accept