Framing

Three approaches find frame and idle fill boundaries

1) Character oriented protocols.

2) Bit oriented protocols.

3) Length counts.

Character oriented framing (e. g. ARPANET)

	SYN	SYN	STX	Header	Packet	ETX	CRC	SYN	SYN	

SYN = synchronous idle

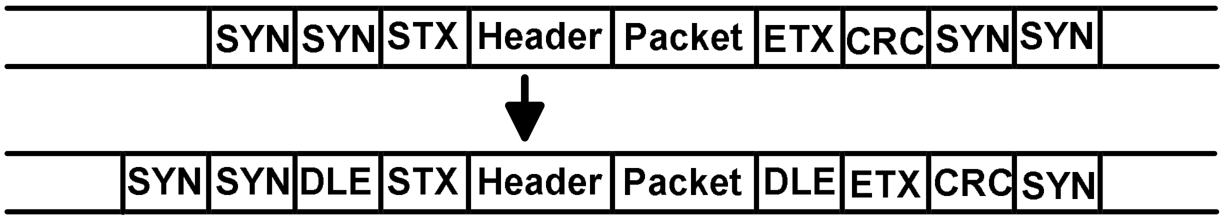
STX = start of text

ETX = end of text

Standard character codes such as ASCII and EBCDIC contain special communication characters that cannot appear in data.

Transparent mode

In transparent mode, a special character, data link escape (DLE) is used to distinguish real communication character from accidental appearances within an arbitrary data

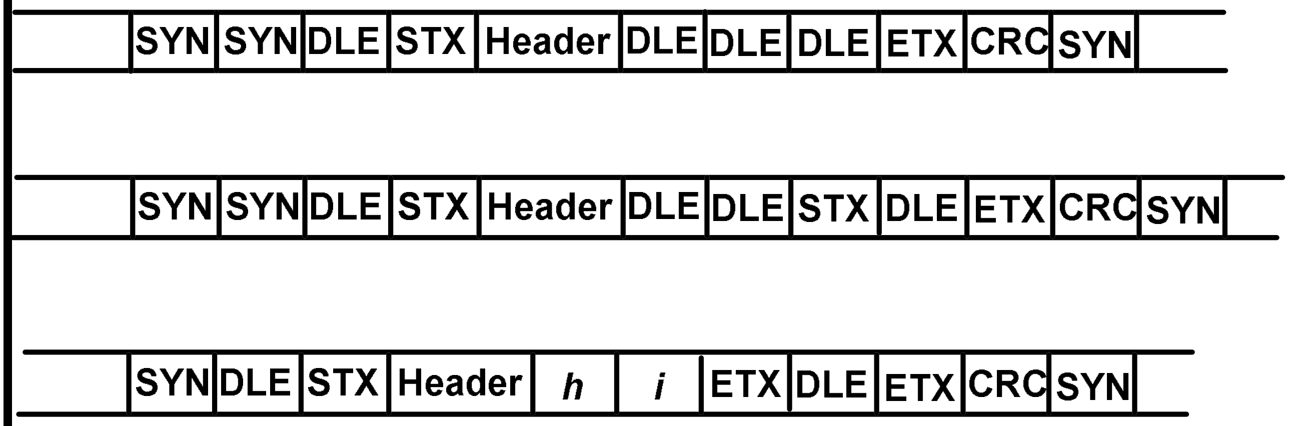


SYN SYN STX Header Packet ETX CRC SYN SYN

SYN SYN DLE STX Header Packet DLE ETX CRC SYN

IF a DLE appears in the data, it is doubled at transmitter (and undoubled at the receiver).

Examples



SYN SYN DLE STX Header DLE DLE DLE ETX CRC SYN

SYN SYN DLE STX Header DLE DLE STX DLE ETX CRC SYN

SYN DLE STX Header *h* *i* ETX DLE ETX CRC SYN

Problems with character based framing

1) It is character code dependent.

2) Frames must have an integer number of characters

3) Require 6 framing characters.

Bit Oriented Framing (flags)

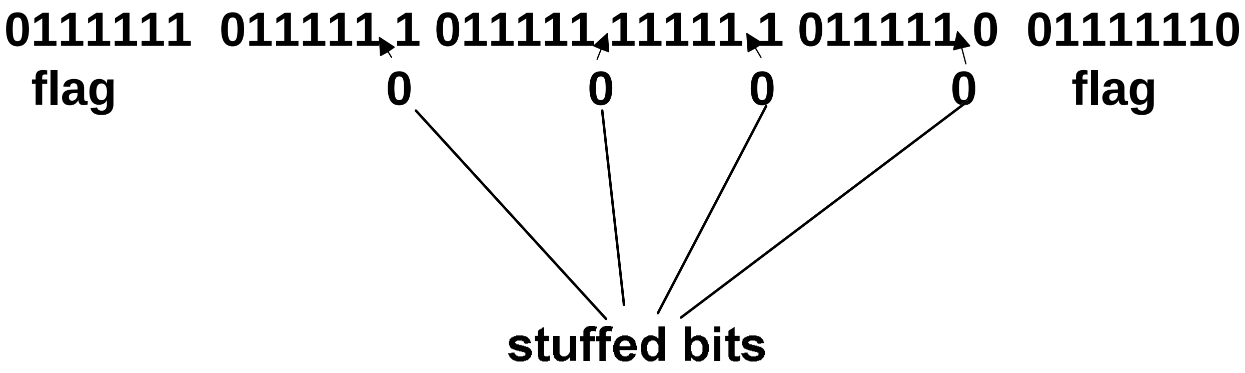
- **A flag is some fixed string of bits to indicate end of frame.**
- **Any string can be used, in principle, but appearance of flag must be prevented somehow in data.**
- **the standard protocol (HDLC, SDLC, etc.) use 01111110 as a flag; they also use 01111111 as a terminator under error conditions.**

Thus 0111111 is the actual bit string that cannot appear in data.

$\overbrace{01111110}^{\textit{flag}}$ **[bits of frame 1]** $\overbrace{01111110}^{\textit{flag}}$ **[bits of frame 1]**

$\overbrace{01111110}^{\textit{flag}}$

Bit Stuffing



0111111 011111 1 011111 11111 1 011111 0 01111110

flag 0 0 0 0 flag

stuffed bits

A 0 is stuffed after each consecutive five 1's in the original frame. A flag, 01111110, without stuffing is sent at the end of the frame.

Destuffing

If 0 is preceded by 011111 in received bitstream, remove it.

If 0 is preceded by 0111111, it is the final bit of the flag.

Example: Bits to be removed are underlined below

$$1011111\underline{0}110001111011111\underline{0}0\ \underbrace{01111110}_{flag}$$

Why is it necessary to stuff a 0 in 01111110 ?

If not , then

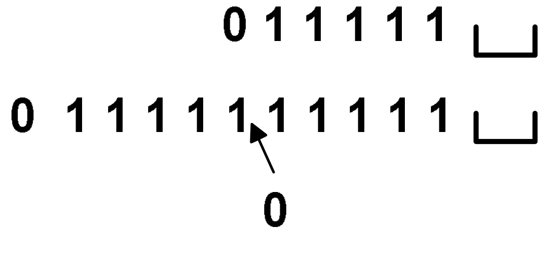
0 1 1 1 1 1 0 1 1 1→ 0 1 1 1 1 1 0 1 1 1

0 1 1 1 1 1 1 1 1→ 0 1 1 1 1 1 0 1 1 1

The overhead per frame in the flag scheme is one byte for the flag, plus about 1/64 times the expected frame length (efficient).

Why 1/64 ?

case where we stuff in a given position



0 1 1 1 1 1 ⌴

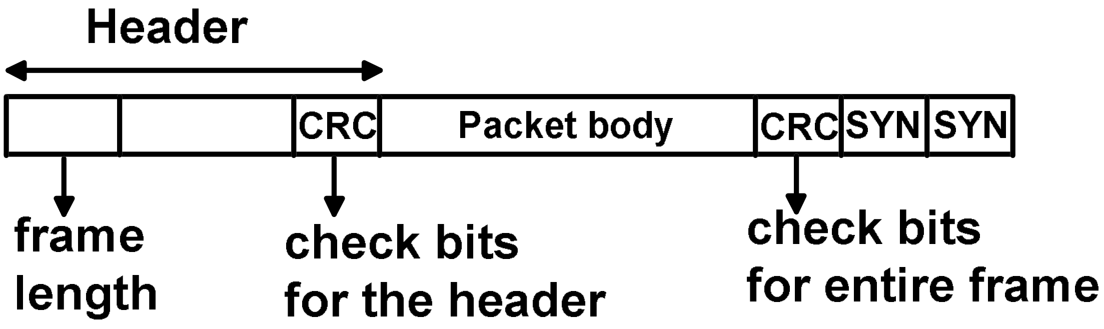
0 1 1 1 1 1 1 1 1 1 1 ⌴

0

Length Counts (example: DECNET)

Some DLC protocols use a header field to give the length of the frame (in bits, bytes,.....)

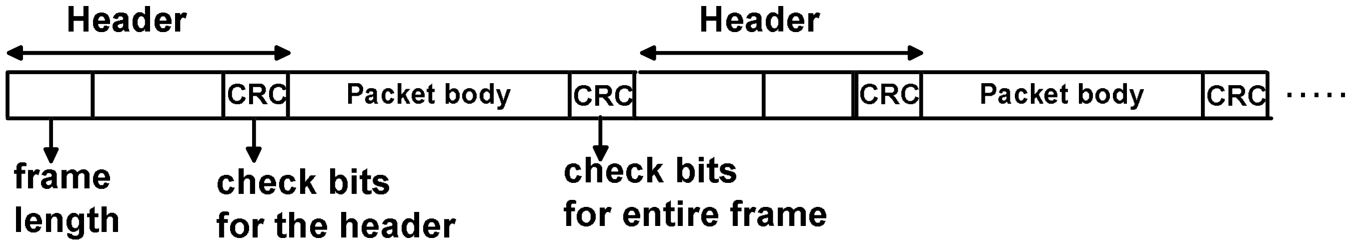
This conveys the same information as the flg scheme.



$K_{\max}$ =maximum frame size

Overhead $= \lfloor \log_2 K_{\max} \rfloor + 1$**(per frame)**

Resynchronzation is needed after an error in length count.


Header
Header
CRC
Packet body
CRC
CRC
Packet body
CRC
.....
frame length
check bits for the header
check bits for entire frame