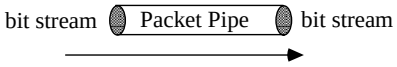
# Lecture 6 - Framing (BG 2.5)

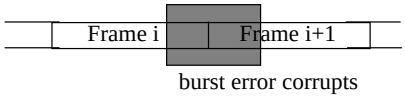
**Introduction:**

How in the world can a loss occur in DLC?




1) Someone puts a big magnet next to wire

2) Burst-error that spans two or more frames:




Today's Lecture:
As little as a single bit error can corrupt frame boundaries!

**Problem Statement**

Receiver DLC gets a bit sequence, and must find *start of frame*.

*(Note: Problem is no simpler even with intermittent synchronous physical layer [10Mb/s Ethernet]):*

- *When transmission starts, sending DLC starts by sending idle fill, so receiver must still find start of frame*

- *During continuous transmission, successive frames are sent back-to-back, and receiver must still decide where to break successive frames.)*





Example of problem: DLC receiver gets this:

10111000101011100110000

Where is

- frame break?
  (could use special bit sequence)
- CRC?
  (could precede by special bit sequence, or put in header)
- header?
  (could follow frame break bit sequence)

Problem is non-trivial if errors corrupt special bit strings!

**Five methods:**

1. character-based framing
   - DLC sends characters
   - Packet lengths must be integral multiple of character length
   - maybe bad for binary data, such as images

2. Bit-oriented framing
   - DLC sends bits
   - Flexible:
     - Permits arbitrary length packets
     - Works fine for binary data

3. Let layer 1 frame (not in Bertsekas/Gallager)

4. Length count in header

5. Used fixed length packets at DLC; frame at layer 3

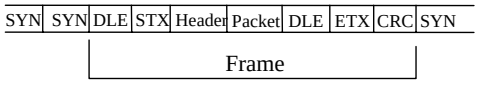




**Method 1: Character-based**

Consider some character set (ASCII). It defines:

- DLE = data link escape
  ("following char is special", like "\" or "^Q" in UNIX, C)
- STX = start of text
- ETX = end of text
- SYN = synchronous idle




SYN's are not in any frame

*Simple:*
Receiver looks for DLE/STX and then DLE/ETX to locate frame boundaries.

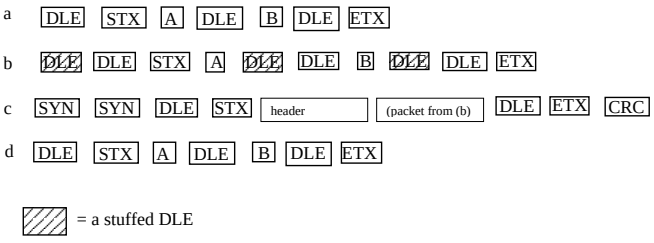
Q: Can packet contain STX or ETX bit code?

A: No problem unless packet contains DLE-STX or DLE-ETX.
- Sending DLC inserts DLE before each accidental DLE.
- Called character stuffing.
- Receiving DLC removes DLE before passing packet to Layer 3.





Example:




= a stuffed DLE

a) Data sent by network layer
b) Data after being stuffed by data link layer
c) Frame (containing data) send over network
d) Data passed to the network layer on receiving side

What if error occurs?

- Corrupted DLE-ETX allows two frames to run together. CRC on successive frame will report error in first frame, but second frame is lost!

- Corrupted packet that inserts a DLE-ETX into packet results in successive bits being used for CRC; receive fails to detect error with probability $1/2^{L}$ for L bit CRC (16 or 32).

- The receiver will wait for DLE-STX before recognizing successive frame, so successive frame is ok.

### Method 2: Bit framing

Today's networks carry a lot more than character data, so this is a popular alternative.

- Each frame begins and ends with 01111110 (denoted $01^60$)

- When sending DLC encounters five consecutive 1's, it stuffs a 0 into outgoing stream -- bit stuffing.

- When receiver sees 111110, it strips 0 bit

- Analogous to character stuffing

*Example:*

Original data: 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0
Sent data: 1 1 1 1 1 *0* 1 0 1 1 1 1 1 1 *0* 1 1 1 1 1 *0* 1 0 1 1 1 1 1 *0* 0

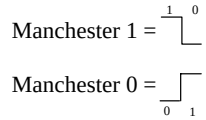
*Overhead:*
Frame length of 1000 bits, identically distributed, random binary variables:
23 overhead bits added to the 1000, or 2.3% overhead.





### Method 3: Let layer 1 frame (used in IEEE 802)

- Certain bit sequences are invalid, due to digital-to-analog encoding:



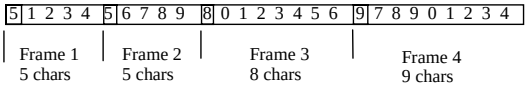

This encoding wastes 1/2 bandwidth but simplifies receiver/sender synchronization.
Low-low-high-high is not a valid encoding for any bit sequence.
So use low-low-high-high for framing.
Advantage - No stuffing required.

But violates layer 1&2 separation. Layer 2 uses knowledge of layer 1.



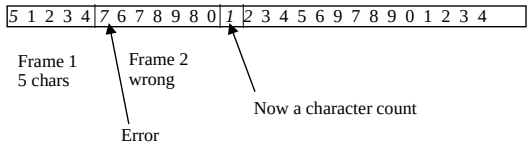


### Method 4: Length field




# = Character count, where each digit is a character

*Suppose an error occurs:*




- If length in header is corrupted, >= 2 frames are misinterpreted & all but first are lost.

- => Rarely used

- Used in DECNET - second CRC for header





### Method 5: Fixed length packets

- No problem! Just fill last packet of message - Layer 3 frames

- Used in recent protocol: ATM

- Problem: voice needs <= 500 bit packets, while much data traffic has less overhead with much longer packets.

### Final Note on all 5 Methods:

Usually DLC's use length counts + char or bit stuffing for redundancy.

**BG 2.5.5 What should maximum frame size be in a DLC protocol?**

*Arguments for long frames:*

Let

M = message length, in bits
V = overhead bits in one frame
$K_{max}$ = maximum packet length

So,

Total bits required to send msg = $M + \lceil M/K_{max} \rceil * V$

overhead

Conclusions on large frames:

a) Long frame increases $K_{max}$, and reduces overhead.

b) For small M, then overhead is even more, due to reduced length of final packet. (If $M = K_{max}+1$, every other frame has almost 100% overhead)

c) In a heavily loaded network, lots of small packets cause lots of queuing delay due too aggregate overhead.

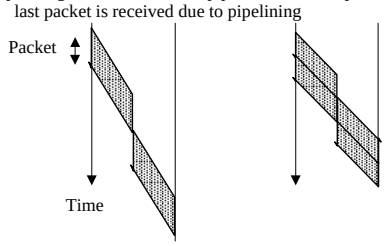
d) Can use lower performance switch because rate at which switch must decode frame headers drops as frame size grows.





*Arguments for short frames*

a) Pipelining effect: for multihop path, ack of first packet can occur before last packet is received due to pipelining




Left: Long packets, right: short packets reduce overall transmission time.

b) Allowing long packets in a heavily loaded network creates a "slow truck" traffic degradation (important for large # hops)

*Expression summarizing relationships:*

Let
- j = # hops
- $K_{max}$ = message length that minimizes E{#bit transmission times to delivery message}

Then

$$K_{max} = [E\{M\}V / (j-1)]^{-1/2}$$

So increase K as overhead (V) increases; decrease K as # hops (j) decreases.

LAN's use large $K_{max}$, because
- #hops is small
- large $K_{max}$ allows most messages to fit in one packet

WAN's use smaller $K_{max}$, because # hops>>1
- ATM -> 53 byte/frame = $K_{max}$