Principles of The Data Link Layer: The Framing Problem

We'll discuss some key principles of the data link layer in this lesson.

We'll cover the following

Limitations Imposed Upon The Data Link Layer

Limitations Imposed by The Data Link Layer

The data link layer uses the service provided by the physical layer. Although there are many different implementations of the physical layer from a technological perspective, they all provide a service that enables the data link layer to send and receive bits between directly connected devices.

Most data link layer technologies **impose limitations on the size of the frames**:

1. Some technologies only impose a maximum frame size.
2. Others enforce both minimum and maximum frame sizes.
3. Finally, some technologies only support a single frame size. In this case, the data link layer will usually include an adaptation sub-layer to allow the network layer to send and receive variable-length packets. This adaptation layer may include fragmentation and reassembly mechanisms.

Limitations Imposed By The Physical Layer

The physical layer service facilitates the sending and receiving of bits, but it's usually far from perfect:

- The physical layer **may change the value of a bit** being transmitted due to any reason, e.g., electromagnetic interferences.
- The Physical layer **may deliver more bits** to the receiver than the bits sent by the sender.
- The Physical layer **may deliver fewer bits** to the receiver than the bits sent by the sender.

The Framing Problem

The data link layer must allow end systems to exchange frames containing packets despite all of these limitations.

On point-to-point links and Local Area Networks, the first problem to be solved is **how to encode a frame as a sequence of bits** so that the receiver can easily recover the received frame **despite the limitations of the physical layer**. This is the **framing problem**. It can be defined as: "*How does a sender encode frames so that the receiver can efficiently extract them from the stream of bits that it receives from the physical layer?*" Several solutions have been proposed and are used in practice in different data link layer technologies.

Solution #1: Idle Physical Layer

A first solution to solve the framing problem is to **require the physical layer to remain idle for some time after the transmission of each frame**. These idle periods can be detected by the receiver and serve as a marker **to indicate frame boundaries**.

Unfortunately, this solution is **not sufficient for two reasons**:

1. First, some physical layer implementations **can't remain idle** and always need to transmit bits.
2. Second, inserting an idle period between frames **decreases the maximum bandwidth that can be achieved** by the data link layer.

Solution #2: Multi-symbol Encodings

Some physical layer implementations provide an alternative to this idle period. **All physical layer types are able to send and receive physical symbols that represent values** 0 **and** 1. Also, **several physical layer types are able to exchange other physical symbols as well**. Some technologies use these other special symbols as markers for the beginning or end of frames. **For example, the Manchester encoding used in several physical layers can send four different symbols**. Apart from the encodings for 0 and 1, the Manchester encoding also **supports two additional symbols: InvH and InvB**.

Solution #3: Stuffing

Unfortunately, multi-symbol encodings cannot be used by all physical layer implementations and a generic solution with which any physical layer that is able to transmit and receive only 0s and 1s works is required. This **generic solution is called stuffing** and two variants exist:

1. Bit stuffing
2. Character stuffing.

To enable a receiver to easily delineate the frame boundaries, these two techniques **reserve special bit strings as frame boundary markers** and encode the frames such that these special bit strings do not appear inside the frames.

Bit Stuffing

Bit stuffing **reserves a special bit pattern**, for example, the 01111110 bit string as the frame boundary marker. However, if the same bit pattern occurs in the data link layer payload, it must be modified before being sent, otherwise, the receiving data link layer entity will detect it as a start or end of frame.

Assuming that the 01111110 pattern is used as the frame delimiter, a frame is sent as follows:

1. First, the sender transmits the marker, i.e. 01111110.
2. Then, it sends all the bits of the frame and inserts an additional bit set to 0 after each sequence of five consecutive 1 bits. This ensures that the sent frame never contains a sequence of six consecutive bits set to 1. As a consequence, the marker pattern cannot appear inside the frame sent.
3. The marker is also sent to mark the end of the frame.
4. The receiver performs the opposite to decode a received frame.
    - It first detects the beginning of the frame thanks to the 01111110 marker.
    - Then, it processes the received bits and counts the number of consecutive bits set to 1.
    - If a 0 follows five consecutive bits set to 1, this bit is removed since it was inserted by the sender.
    - If a 1 follows five consecutive bits set to 1, it indicates a marker if it is followed by a bit set to 0.

The table below illustrates the application of bit stuffing to some frames.

Original frame	Transmitted frame
000100100100100100100011	011111100001001001001001001000 01101111110
01111110	011111100111110100111110

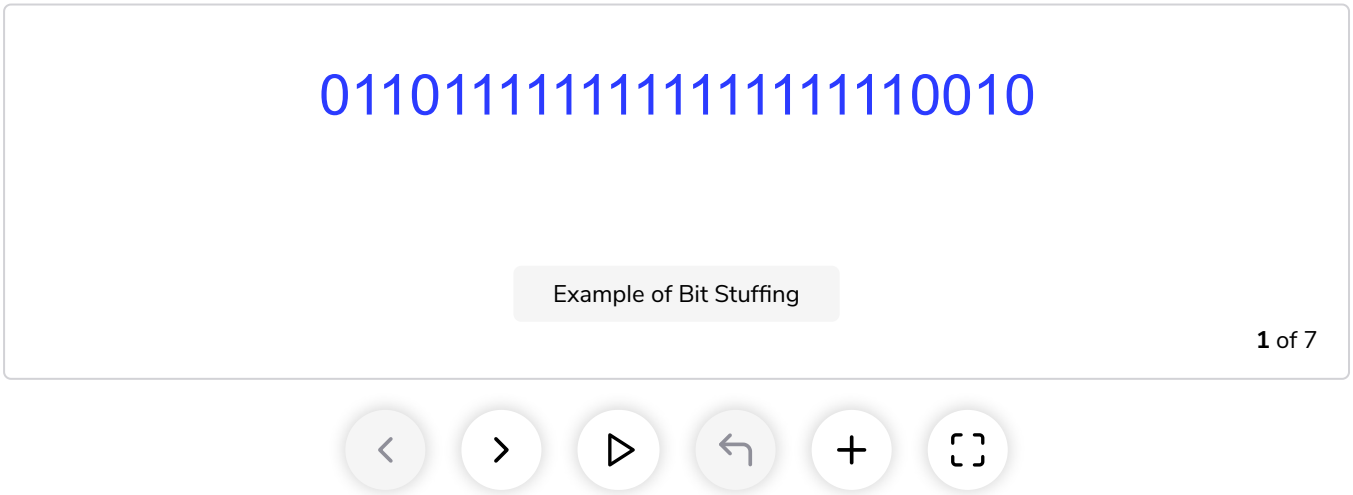
Example #

For example, consider the transmission of 011011111111111111110010.

1. The sender will first send the 01111110 marker followed by 011011111.

2. After these five consecutive bits set to 1, it inserts a bit set to 0 followed by 11111.
3. A new 0 is inserted, followed by 11111.
4. A new 0 is inserted followed by the end of the frame 110010 and the 01111110 marker.

Have a look at the slides for a visual demonstration of bit stuffing.



You might be wondering **what happens if the sequence** 011111010 **appears in the data**? Bit stuffing inserts a 0 bit after every consecutive five 1s in the payload. What does that give us? Well, the payload could be one of the following:

- 0111 11 00
- 0111 11 01
- 0111 11 10
- 0111 11 11

These will be encoded to, respectively:

- 0111 11 0 00
- 0111 11 0 01
- 0111 11 0 10
- 0111 11 0 11 So, in any case, **the receiver can only expect** 01111110 **at the beginning and end of frame**. If it receives five consecutive 1s, followed by a 0, it removes the 0 as redundancy. If it receives six

consecutive 1s, there must've been an error.

Character Stuffing

This technique operates on frames that contain an integer number of characters of a fixed size, such as 8-bit characters. Some characters are used as markers to delineate the frame boundaries. Many character stuffing techniques use the **DLE**, **STX** and **ETX** characters of the ASCII character set. **DLE STX** is **used to mark the beginning of a frame**, and **DLE ETX** is used to mark the **end of a frame**.

If the character DLE appears in the payload, the data link layer entity prepends DLE as an escape character before the transmitted DLE character from the payload. This ensures that none of the markers can appear inside the transmitted frame. The receiver detects the frame boundaries and removes the second DLE when it receives two consecutive DLE characters.

>  **Note: Software implementations prefer to process characters than bits**, hence software-based data link layers usually use **character stuffing**.

Examples

For example, to transmit frame **1 2 3 DLE STX 4**:

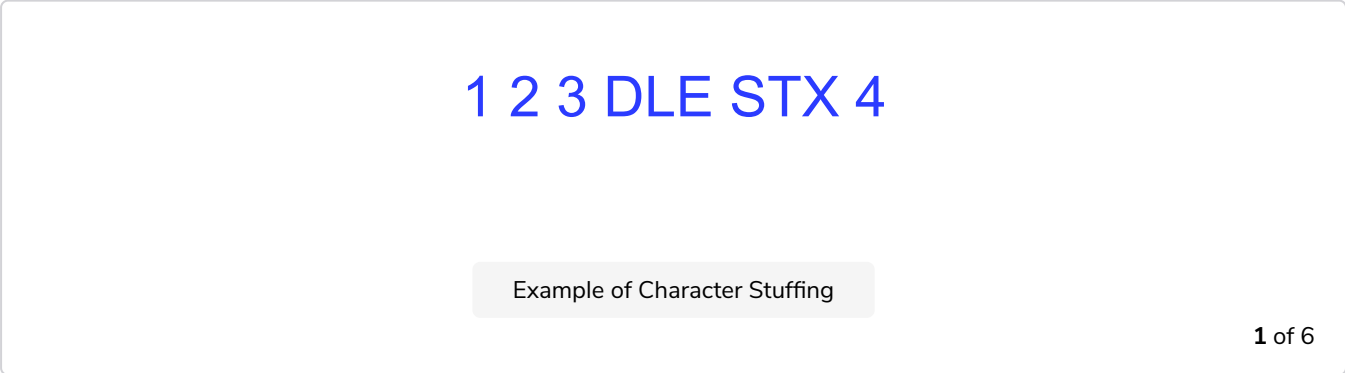
1. A sender will first send **DLE STX** as a marker
2. Followed by **1 2 3 DLE**
3. Then, the sender transmits an additional **DLE** character
4. Followed by **STX 4** and the **DLE ETX** marker
5. The final string is: **DLE STX 1 2 3 DLE DLE STX 4 DLE ETX**

Have a look at the following table for more details:

Original frame	Transmitted frame
1 2 3 4	DLE STX 1 2 3 4 DLE ETX

1 2 3 DLE STX 4 DLE STX DLE ETX	DLE STX 1 2 3 DLE DLE STX 4 DLE ETX DLE STX DLE DLE STX DLE DLE ETX DLE ETX

> 🔎 **Did You Know? DLE** is the bit pattern 00010000, **STX** is 00000010 and **ETX** is 00000011.



Disadvantages of Stuffing

1. In character stuffing and in bit stuffing, the length of the transmitted frames is increased. The worst case redundant frame in case of bit stuffing is one that has a long sequence of all 1s, whereas in the case of character stuffing, it's a frame consisting entirely of DLE characters.
2. When transmission errors occur, the receiver may incorrectly decode one or two frames (e.g., if the errors occur in the markers). However, it'll be able to resynchronize itself with the next correctly received markers.
3. Bit stuffing can be easily implemented in hardware. However, implementing it in software is difficult given the higher overhead of bit manipulations in software.

Quick Quiz!

1

Consider that **bit stuffing** is used and the delimiting sequence is 01111110. The following bit pattern is received. Where does the frame start and where does it end?

11100111111001000101010001111111010

A) The entire sequence is one frame.

B) There is an error. A sequence like this cannot be received.

C) The frame is the bold part here 111001111**1100100010101000111**11010.

D) The frame is the bold part here: 1110**0111111001000101010001111111**010.

COMPLETED 0%

1 of 4

In the next lesson, we'll study error detection in the data link layer.