Data Link Layer Framing - Complete Study Material

Introduction to Framing

Framing is one of the major responsibilities of the Data Link Layer in the OSI model. It involves converting raw data bits into structured frames for transmission over the network.

What is Framing?

- **Definition**: The process of packing data bits into frames at the Data Link Layer
- **Purpose**: To organize data for transmission and distinguish between different frames
- **Analogy**: Similar to how letters are put into envelopes in the postal system - each envelope has a standard format that helps identify and separate individual letters

Why is Framing Necessary?

The Postal System Analogy

- In postal services, all letters follow the same format when put in envelopes
- Without envelopes, it would be difficult to distinguish one letter from another
- Similarly, in networking, we need a standard way to separate one frame from another
- This separation allows proper delivery and identification of data units

Major Framing Techniques

1. Character Stuffing (Byte Stuffing)

Concept

- Uses **delimiter characters** to mark frame boundaries
- Most commonly used technique in framing
- Special characters act as **flags** to indicate start and end of frames

How it Works

```
[FLAG] [Data Characters] [FLAG] [Data Characters] [FLAG]
```

Basic Example

```
Original data: ABCDEF
Frame format: FLAG + Data + FLAG
Result: [FLAG]ABCDEF[FLAG]
```

Problem Scenario: Flag in Data

What happens if the flag character appears in the actual data?

Example:

- Flag character: `#`
- Data to send: `AB#CD`
- If sent as: `[#]AB#CD[#]`
- **Problem**: Receiver will interpret the `#` in data as a frame delimiter

Solution: Character Stuffing

Rule: If flag character appears in data, add an escape character before it

Example:

- Flag: `#`
- Escape character: `\`
- Original data: `AB#CD`
- After stuffing: `AB\#CD`
- Frame: `[#]AB\#CD[#]`

At Receiver Side:

- Receiver knows the rule: ignore flag if preceded by escape character
- When `\#` is found, receiver removes `\` and treats `#` as data
- Final received data: `AB#CD`

2. Bit Stuffing

Concept

- Instead of character-level stuffing, works at bit level
- Commonly uses the pattern `01111110` as flag (like in HDLC protocol)

How it Works

Flag Pattern: `01111110`

Problem: Flag Pattern in Data

What if the data contains the same bit pattern as the flag?

Solution: Bit Stuffing Rule

Rule: After any sequence of 5 consecutive 1's in data, insert a 0

Example Demonstration

Original Data: `0111111010101`

Step-by-step Process:

1. Scan for five consecutive 1's: `01111`**11010101**
2. After finding `01111`, insert `0`: `01111`**0**`1010101`
3. Result after stuffing: `011110110101`
4. Final frame: `[01111110]011110110101[01111110]`

At Receiver Side:

1. Receiver knows the rule: after five consecutive 1's, remove the next 0
2. Received frame: `[01111110]011110110101[01111110]`
3. Process data: `01111`**0**`110101` → Remove the 0 after five 1's → `0111111010101`
4. Original data recovered: `0111111010101`

Detailed Examples and Scenarios

Scenario 1: Multiple Flags in Data (Character Stuffing)

Given:

- Flag: `$`
- Escape: `*`
- Data: `PAY$100$BILL`

Process:

1. Original: `PAY$100$BILL`
2. Stuff flags: `PAY*$100*$BILL`
3. Final frame: `[$]PAY*$100*$BILL[$]`

Receiver Processing:

- Sees `*$` → interprets as data character `$`
- Final data: `PAY$100$BILL`

Scenario 2: Escape Character in Data

Given:

- Flag: `#`
- Escape: `\`
- Data: `FILE\PATH`

Process:

1. Original: `FILE\PATH`
2. Escape character found in data, so stuff it: `FILE\\PATH`
3. Final frame: `[#]FILE\\PATH[#]`

Scenario 3: Complex Bit Stuffing

Given:

- Flag: `01111110`
- Data: `0011111011111001111`

Process:

1. Find first five 1's: `00`**11111**`10111111001111`
2. Insert 0: `00111110` + remaining data
3. Continue with remaining: `10111111001111`
4. Find next five 1's: `10`**11111**`001111`
5. Insert 0: `10111110` + remaining data
6. Final stuffed data: `0011111010111110001111`
7. Complete frame: `[01111110]0011111010111110001111[01111110]`

Key Points for Examinations

Important Concepts to Remember

1. **Character Stuffing**:
   - Uses delimiter/flag characters
   - Escape character prevents confusion
   - Works at byte/character level
2. **Bit Stuffing**:
   - Uses bit patterns as flags
   - Insertion rule prevents flag patterns in data
   - Works at bit level
   - More efficient than character stuffing

3. **Common Flag Patterns**:
   - HDLC uses: `01111110`
   - PPP uses: `01111110`
   - Custom protocols may use different patterns

Frequently Asked Exam Questions

Type 1: Character Stuffing Problem

Q: If flag is `@` and escape is `%`, frame the data `COST@50%OFF`

Answer:

1. Stuff `@`: `COST%@50%OFF`
2. Stuff `%`: `COST%@50%%OFF`
3. Frame: `[@]COST%@50%%OFF[@]`

Type 2: Bit Stuffing Problem

Q: Frame the data `0111110111110` using flag `01111110`

Answer:

1. After `01111`: insert `0` → `0111100111110`
2. After next `11111`: insert `0` → `01111001111100`
3. Frame: `[01111110]01111001111100[01111110]`

Type 3: Reverse Problem

Q: If received frame is `[F]AB%FCG[F]` with flag `F` and escape `%`, what is original data?

Answer:

1. Remove flags: `AB%FCG`
2. Process escape: `%F` → `F`
3. Original data: `ABFCG`

Advantages and Disadvantages

Character Stuffing

Advantages:

- Simple to implement
- Easy to understand

- Works well with text data

Disadvantages:

- Overhead increases with number of flag characters in data
- Less efficient for binary data

Bit Stuffing

Advantages:

- More efficient
- Works well with any type of data
- Lower overhead on average

Disadvantages:

- Slightly more complex to implement
- Requires bit-level processing

Protocol Examples

HDLC (High-Level Data Link Control)

- Uses bit stuffing
- Flag: `01111110`
- Rule: Insert `0` after five consecutive `1`s

PPP (Point-to-Point Protocol)

- Uses both character and bit stuffing depending on mode
- Async mode: character stuffing
- Sync mode: bit stuffing

Summary

Framing is essential for:

1. **Frame Synchronization**: Identifying frame boundaries
2. **Data Integrity**: Ensuring complete frame reception
3. **Error Detection**: Enabling frame-level error checking
4. **Flow Control**: Managing data transmission rate

Both character stuffing and bit stuffing solve the fundamental problem of distinguishing frame delimiters from actual data, ensuring reliable data transmission in the Data Link Layer.