Bit Stuffing - Client-Server Chat

Demonstrating data link layer framing using Bit Stuffing

Presented By

Yash Rai

What is Bit Stuffing?

Definition

A technique used in data link layer framing to distinguish data from control information by inserting extra, non-information bits into the data stream.

Analogy

Like adding an "escape character" in text to use a quote inside a quoted string:

```
"He said \"Hello\" to me."
```

The backslash acts as an escape character

Purpose

Why we need it:

- To prevent the Flag sequence (01111110) from appearing in the actual data
- This ensures the receiver correctly identifies the start and end of a frame

Mechanism

The Rule:

The sender inserts a `0` bit into the data stream after every five consecutive `1` bits.

Example:

Original Data:

1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1

After Bit Stuffing:

1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1

ℹ Red '0' bits are stuffed bits

Client-Side (Sender) — The 3 Steps

1 Data Conversion `text_to_binary`

Input: User text (e.g., "Hello") or raw bit sequence

Action: Convert text to ASCII binary (8 bits per character)

Example:

Input: `"Hello"`

Binary: `01001000 01100101 01101100 01101100 01101111`
`H e l l o`

↓

2 Bit Stuffing `bit_stuff`

Rule: Insert a '0' after every sequence of five consecutive '1's

This prevents accidental flag patterns in the data

Example:

Raw: 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1

Stuffed: 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1

ℹ Red bits are inserted '0's after five consecutive '1's

↓

3 Framing Construction

Final frame structure:

FLAG + Stuffed Data + FLAG

FLAG Sequence (HDLC): `01111110`

Transmission Frame:

0 1 1 1 1 1 1 0
FLAG

Stuffed Data
1 1 ...

0 1 1 1 1 1 1 0
FLAG

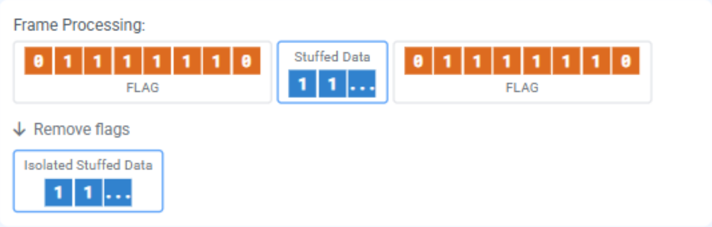
The frame is now ready for transmission via TCP socket

Server-Side (Receiver) — The 4 Steps

1 Frame Synchronization `handle_frame`

Input: Complete transmission frame via TCP socket

Action: Verify start/end flags and remove them to isolate the stuffed data



2 Bit Unstuffing `bit_unstuff`

Rule: Remove a '0' that follows any sequence of five consecutive '1's

This reverses the bit stuffing process

Example:

Stuffed: 1 1 1 1 1 ~~0~~ 0 1 1 1 1 1 ~~0~~ 1 1 1 1

Unstuffed: 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1

ⓘ Red crossed-out bits are removed '0's that followed five consecutive '1's

3 Binary-to-Text Conversion `binary_to_text`

Action: Divide binary data into 8-bit chunks

Each chunk is converted to its corresponding ASCII character

Binary to ASCII conversion:

Binary Chunk	ASCII Value	Character
01001000	72	H
01100101	101	e
01101100	108	l
01101100	108	l
01101111	111	o

Recovered Text: "Hello"

4 Acknowledgment (ACK)

Action: Send confirmation back to client

This completes the communication cycle

✓ **Server response:**

```
ACK: Frame successfully processed...
```

The server confirms successful receipt and decoding of the message

Network Implementation (Code Structure)

Protocol

Uses TCP (Transmission Control Protocol) sockets:

```
import socket

# Address Family
socket.AF_INET → IPv4 addresses

# Socket Type
socket.SOCK_STREAM → TCP (connection-oriented)
```

TCP ensures reliable, ordered data delivery

Configuration

Network settings:

```
   Host: '127.0.0.1' # Localhost (testing on same
         machine)
   Port: 666 # Server listens on this port
 Buffer: 1024 # Maximum data received in bytes
```

Network Architecture:

Client
Server

TCP Connection over localhost

Client (client_gui.py)

Key operations:

```
# Create socket
s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

# Connect to server
s.connect((HOST, PORT))

# Send frame
s.sendall(frame_to_send.encode('utf-8'))
```

⚠ Before sending, client performs bit stuffing and adds flags

Server (server.py)

Key operations:

```
# Create socket
s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

# Bind to address
s.bind((HOST, PORT))
s.listen()

# Accept connection
conn, addr = s.accept()

# Receive data
data = conn.recv(BUFFER_SIZE)
```

After receiving, server performs unstuffing and decodes frame

Conclusion — Key Takeaways

Data Link Layer Mechanism

Bit Stuffing is a critical flow control and framing mechanism at the Data Link Layer

Data Transparency

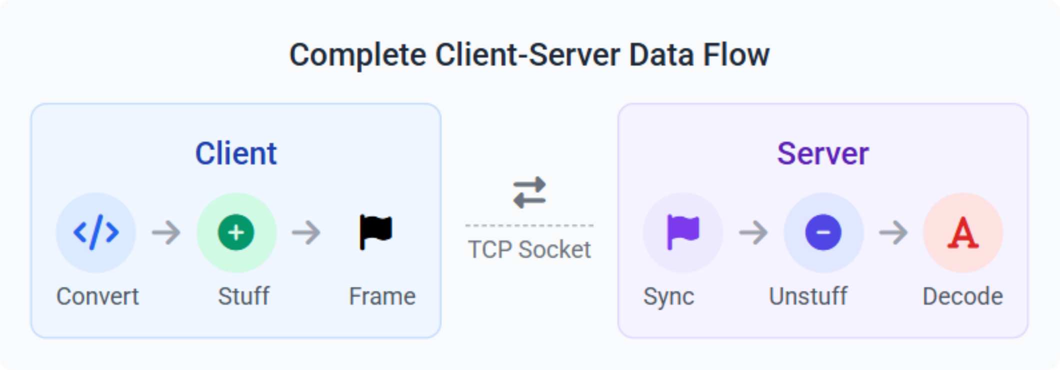
Solves the data transparency problem by preventing flag patterns in data from being mistaken for frame boundaries

Role Separation

Client is responsible for stuffing and framing; Server handles synchronization, unstuffing, and decoding

Implementation Layers

Demonstrates how link-layer principles are implemented before sending data over network sockets


Complete Client-Server Data Flow