ECE F343: Communication Networks

Assignment 1 - Question 1

Pranav Chandra N. V.
2023AAPS0013P

26/06/2026

Contents

1 Introduction

This program was written for Assignment 1, Question 1. The goal is to show how message encapsulation works in the OSI model by simulating the downward flow of data from Layer 7 (Application) to Layer 1 (Physical).

The user enters a message of up to 80 characters. The program passes it through seven functions, one per layer. Each function sticks its own header onto the front of whatever it received, prints the result, and then calls the function for the layer below it.

2 OSI Reference Model Overview

The OSI model splits network communication into seven layers, each with a specific job. Table 1 lists each layer, the name for its unit of data (PDU), and the header string used in this program.

Table 1: OSI Layers, PDU names, and simulated headers

Layer	Name	PDU	Simulated Header
7	Application	Message	`[APP: HTTP GET /index.html]`
6	Presentation	Message	`[PRES: ENC=UTF-8 FORMAT=ASCII COMPRESS=NONE]`
5	Session	Message	`[SESS: ID=A1B2C3 SEQ=001 TYPE=DATA]`
4	Transport	Segment	`[TRAN: TCP SRC=1234 DST=80 SEQ=100 ACK=0]`
3	Network	Packet	`[NET: SRC=192.168.1.1 DST=10.0.0.1 TTL=64]`
2	Data Link	Frame	`[DL: SRC=AA:BB:CC DST=11:22:33 TYPE=IPv4]`
1	Physical	Bits	`[PHY: ENC=NRZ SIGNAL=DIGITAL]`

3 Program Design

3.1 Design Decisions

- A helper function `prepend_header()` takes care of copying the header and message into a new buffer, so the layer functions themselves stay short and readable.
- Buffers are fixed-size arrays on the stack (`MAX_BUF = 1024` bytes). This is more than enough for seven headers plus an 80-char message, and the helper exits if that ever overflows.
- All headers are kept under 64 characters as required by the spec.
- Each layer function just calls the one below it directly, keeping the call chain simple and easy to follow.

3.2 Constants and Macros

```c
#define MAX_APP_MSG   80
#define MAX_HEADER    64
#define MAX_BUF       1024

#define HDR_APP    "[APP: HTTP GET /index.html]"
#define HDR_PRES   "[PRES: ENC=UTF-8 FORMAT=ASCII COMPRESS=NONE]"
#define HDR_SESS   "[SESS: ID=A1B2C3 SEQ=001 TYPE=DATA]"
#define HDR_TRAN   "[TRAN: TCP SRC=1234 DST=80 SEQ=100 ACK=0]"
#define HDR_NET    "[NET: SRC=192.168.1.1 DST=10.0.0.1 TTL=64]"
#define HDR_DL     "[DL: SRC=AA:BB:CC DST=11:22:33 TYPE=IPv4]"
#define HDR_PHY    "[PHY: ENC=NRZ SIGNAL=DIGITAL]"
```

Listing 1: Buffer and header constants

3.3 Helper: `prepend_header()`

```c
static int prepend_header(const char *header, const char *msg,
   int msg_size, char *out_buf, int out_buf_size){
    int hdr_len  = (int)strlen(header);
    int new_size = hdr_len + msg_size;

    if (new_size >= out_buf_size) {
        fprintf(stderr, "Buffer overflow prevented.\n");
        exit(1);
    }

    memcpy(out_buf,            header, hdr_len);
    memcpy(out_buf + hdr_len, msg,     msg_size);
    out_buf[new_size] = '\0';

    return new_size;
}
```

Listing 2: Header prepending helper

3.4 Layer Functions

All seven layer functions work the same way:

1. Allocate a local buffer (`MAX_BUF` bytes).
2. Call `prepend_header()` to build the new PDU.
3. Print it.
4. Call the next layer down (the Physical layer skips this last step).

```c
void application_layer(char *msg, int size)
{
    char buf[MAX_BUF];
    int  new_size = prepend_header(HDR_APP, msg, size, buf,
       MAX_BUF);
    printf("[Layer 7 - Application]  PDU: %s\n\n", buf);
    presentation_layer(buf, new_size);
}
```

Listing 3: Application Layer (representative example)

4 Message Flow Flowchart

Figure 1 shows the message moving down through the layers. The PDU gets longer at each step as another header is added to the front.

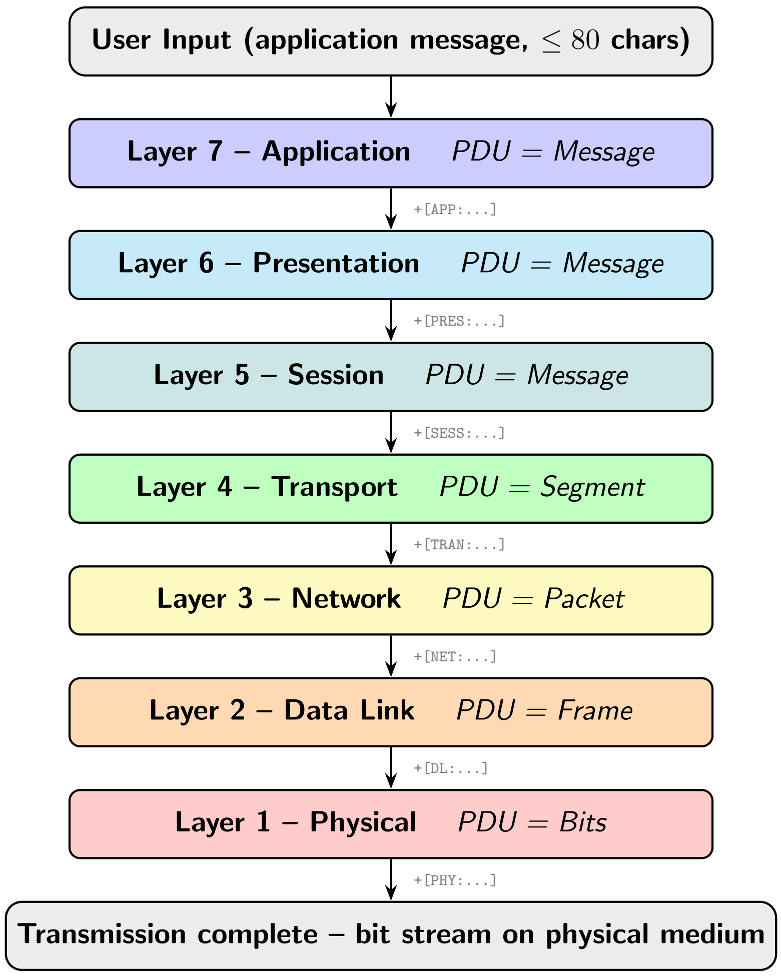
User Input (application message, ≤ 80 chars)
Layer 7 – Application PDU = Message
+[APP:...]
Layer 6 – Presentation PDU = Message
+[PRES:...]
Layer 5 – Session PDU = Message
+[SESS:...]
Layer 4 – Transport PDU = Segment
+[TRAN:...]
Layer 3 – Network PDU = Packet
+[NET:...]
Layer 2 – Data Link PDU = Frame
+[DL:...]
Layer 1 – Physical PDU = Bits
+[PHY:...]
Transmission complete – bit stream on physical medium

Figure 1: Message flow and encapsulation across the OSI 7 layers.

5 Sample Output

Running the program with the input `"Hello, Network!"` gives:

```
Enter application message (up to 80 characters):
> Hello, Network!

=== OSI Encapsulation ===

[Layer 7 - Application]  PDU: [APP: HTTP GET /index.html]Hello, Network!

[Layer 6 - Presentation] PDU: [PRES: ENC=UTF-8 FORMAT=ASCII COMPRESS=NONE][APP: HTTP GET /index.html]Hello, Network!

[Layer 5 - Session]      PDU: [SESS: ID=A1B2C3 SEQ=001 TYPE=DATA][PRES: ...]...[APP: ...]Hello, Network!

[Layer 4 - Transport]    Segment: [TRAN: TCP SRC=1234 DST=80 SEQ=100 ACK=0][SESS: ...]...[APP: ...]Hello, Network!

[Layer 3 - Network]      Packet: [NET: SRC=192.168.1.1 DST=10.0.0.1 TTL=64][TRAN: ...]...Hello, Network!

[Layer 2 - Data Link]    Frame: [DL: SRC=AA:BB:CC DST=11:22:33 TYPE=IPv4][NET: ...]...Hello, Network!

[Layer 1 - Physical]     Bits: [PHY: ENC=NRZ SIGNAL=DIGITAL][DL: ...]...Hello, Network!

=== Transmission complete: 271 bytes on the wire ===
```

Listing 4: Sample program output

6 Build and Run

The project uses a simple `Makefile`:

```
CC     = gcc
CFLAGS = -Wall -Wextra -std=c11
TARGET = osi_model
SRC    = osi_model.c

all: $(TARGET)

$(TARGET): $(SRC)
    $(CC) $(CFLAGS) -o $(TARGET) $(SRC)

clean:
    rm -f $(TARGET)
```

Listing 5: Makefile

Compile and run with:

```
make
./osi_model
```

7 Conclusion

This program gives a rough idea of how encapsulation works going down the OSI stack. Each layer adds its own header before passing the data on, so by the time it reaches Layer 1 the original message is buried inside several headers. The headers here are simplified — they do not carry real protocol data — but the structure matches what each layer is actually responsible for: application requests at Layer 7, encoding info at Layer 6, session tracking at Layer 5, port and sequence numbers at Layer 4, IP addresses at Layer 3, MAC addresses at Layer 2, and signalling info at Layer 1.