ECE F343: Communication Networks
Assignment 1

Pranav Chandra N. V.
2023AAPS0013P

26/06/2026

Contents

1 Quesition 1

1.1 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.

1.2 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]`

1.3 Program Design

1.4 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.

1.5 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

1.6 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

1.7 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)

1.8 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.

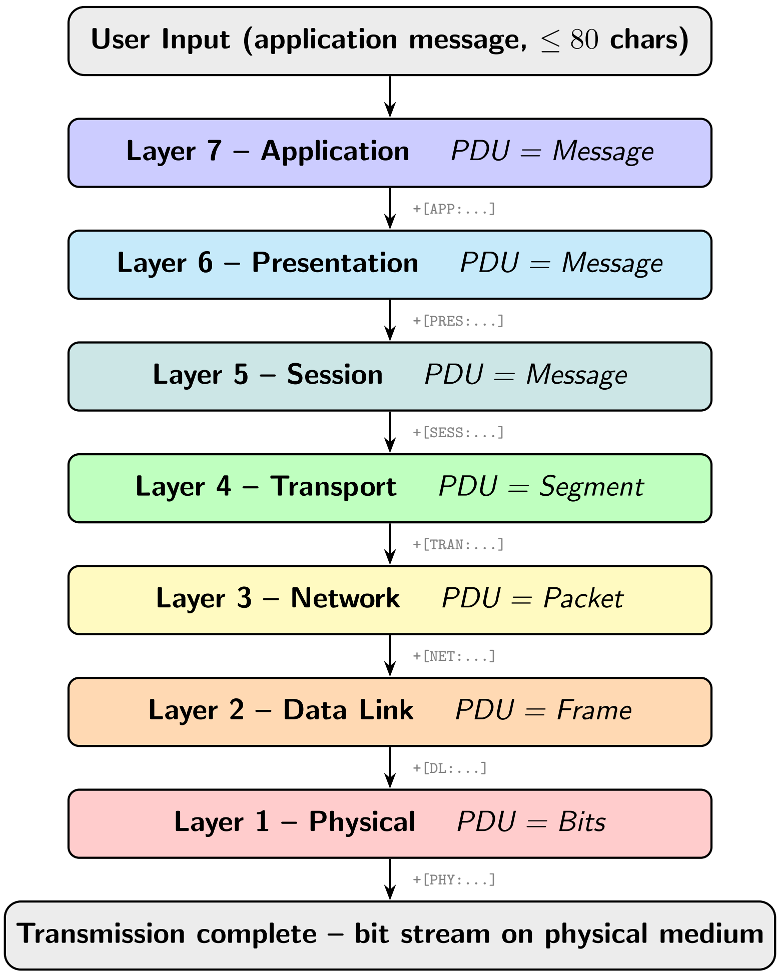

User Input (application message, ≤ 80 chars)
Layer 7 – Application PDU = Message
+[APP:...]
Layer 6 – Presentation PDU = Message
+[PRES:...]
Layer 5 – Session PDU = Message
+[SESS:...]
Layer 4 – Transport PDU = Segment
+[TRAN:...]
Layer 3 – Network PDU = Packet
+[NET:...]
Layer 2 – Data Link PDU = Frame
+[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.

1.9 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

1.10 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
```

1.11 Summary

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.

2 Question 2

2.1 Wireshark Protocols

Question: What protocols are listed in the Wireshark “protocol” column in your trace file? Make a list of such protocols, identify the layer to which they belong, and briefly explain (in 1-2 lines) the function of each protocol.

Answer: The protocols listed under the ”protocol” column are as follows:

1. **ARP - Layer 2/3 (Network Interface / Network)** - Maps an IPv4 address to a MAC address on a local network.
2. **mDNS - Layer 7 (Application)** - Resolves hostnames locally using multicast without a DNS server.
3. **IGMPv2 - Layer 3 (Network)** - Manages IPv4 multicast group membership.
4. **ICMPv6 - Layer 3 (Network)** - Provides IPv6 error reporting and neighbor discovery.
5. **TCP - Layer 4 (Transport)** - Provides reliable, connection-oriented data transmission.
6. **SSDP - Layer 7 (Application)** - Discovers devices and services on a local network.
7. **LLC - Layer 2 (Data Link)** - Identifies upper-layer protocols within Ethernet frames.
8. **VRRP - Layer 3 (Network)** - Provides gateway redundancy using a virtual IP address.
9. **DHCP - Layer 7 (Application)** - Automatically assigns IP configuration to clients.
10. **TLSv1.2 - Layer 6/7 (Presentation/Application)** - Encrypts and secures application data.
11. **DNS - Layer 7 (Application)** - Translates domain names into IP addresses.
12. **UDP - Layer 4 (Transport)** - Provides fast, connectionless data transmission.
13. **IGMPv3 - Layer 3 (Network)** - Supports source-specific IPv4 multicast membership.
14. **NBNS - Layer 7 (Application)** - Resolves NetBIOS names to IP addresses.
15. **TLSv1.3 - Layer 6/7 (Presentation/Application)** - Provides faster and more secure encryption than TLS 1.2.
16. **HTTP - Layer 7 (Application)** - Transfers web content using a request-response model.
17. **BROWSER - Layer 7 (Application)** - Maintains shared resource lists in Windows networks.

2.2 HTTP Protocl Stats

Question: Read about HTTTP protocol and its working (Reference: Section 2.1, 8.1 in Garcia). Now in your experiment, determine how long did it take from when the HTTP GET message was sent until the HTTP OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since Wireshark tracing began. (If you want to display the Time field in time-of-day format, select the Wireshark View pull down menu, then select Time Display Format, then select Time-of-day.)

Answer: The HTTPS GET packet was triggered at $t = 19.08404851$ and the OK packet was recieved at $t = 19.830304061$, meaning it took 0.746255551000001 seconds for this communicaiton.

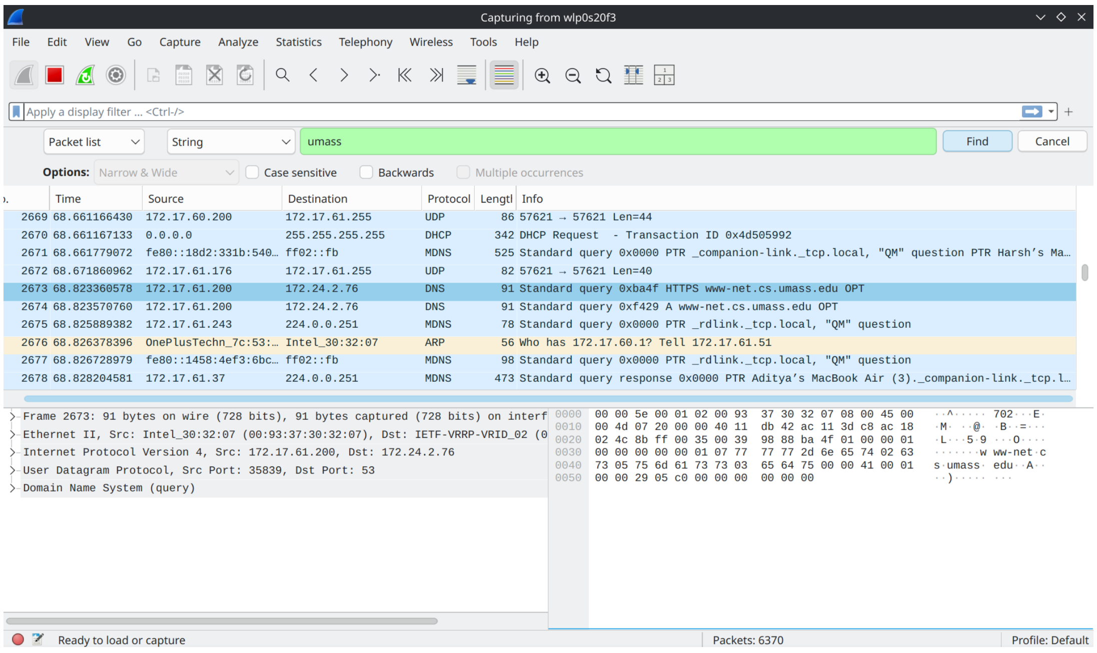
Note: I used another HTTP website (this one) to test HTTP, as I couldn't get the gaia website working even after turning of HTTPS3.

2.3 Internet Address of Gaia

Question What is the Internet address of the gaia.cs.umass.edu (also known as www-net.cs.umass.edu)? What is the Internet address of the computer that sent the HTTP GET message (i.e., your computer)?

Answer: Since the website is a https secured website, we use the dns filter, rather than the http filter on the wireshark protocols list. The source and destination can be found in 2 and are as follows

- **Source:** 172.17.61.200
- **Destination:** 172.24.2.76



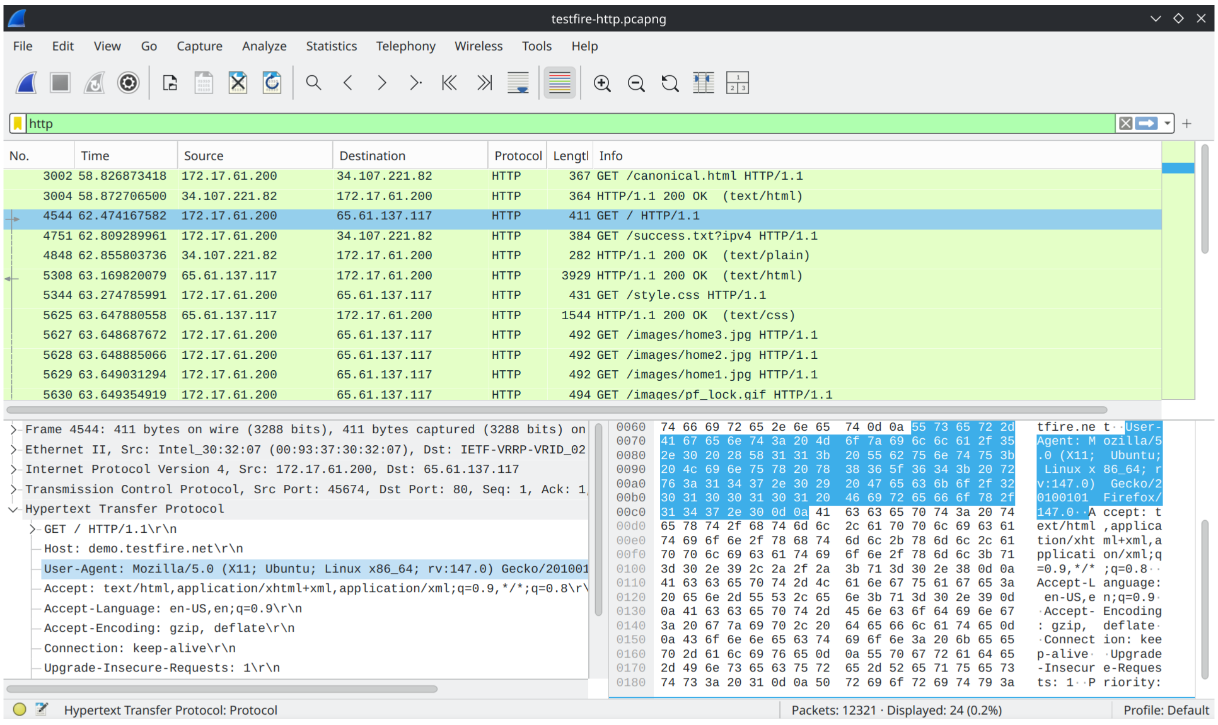
```
Apply a display filter ... <Ctrl-/>
Packet list | String | umass
Options: Narrow & Wide   Case sensitive   Backwards   Multiple occurrences

 No. Time         Source                 Destination      Protocol Length Info
2669 68.661166430 172.17.60.200          172.17.61.255    UDP       86 57621 → 57621 Len=44
2670 68.661167133 0.0.0.0                255.255.255.255  DHCP     342 DHCP Request  - Transaction ID 0x4d505992
2671 68.661779072 fe80::18d2:331b:540…   ff02::fb         MDNS     525 Standard query 0x0000 PTR _companion-link._tcp.local, "QM" question PTR Harsh’s Ma…
2672 68.671860962 172.17.61.176          172.17.61.255    UDP       82 57621 → 57621 Len=40
2673 68.823360578 172.17.61.200          172.24.2.76      DNS       91 Standard query 0xba4f HTTPS www-net.cs.umass.edu OPT
2674 68.823570760 172.17.61.200          172.24.2.76      DNS       91 Standard query 0xf429 A www-net.cs.umass.edu OPT
2675 68.825889382 172.17.61.243          224.0.0.251      MDNS      78 Standard query 0x0000 PTR _rdlink._tcp.local, "QM" question
2676 68.826378396 OnePlusTechn_7c:53:…   Intel_30:32:07   ARP       56 Who has 172.17.60.1? Tell 172.17.61.51
2677 68.826728979 fe80::1458:4ef3:6bc…   ff02::fb         MDNS      98 Standard query 0x0000 PTR _rdlink._tcp.local, "QM" question
2678 68.828204581 172.17.61.37           224.0.0.251      MDNS     473 Standard query response 0x0000 PTR Aditya’s MacBook Air (3)._companion-link._tcp.l…

> Frame 2673: 91 bytes on wire (728 bits), 91 bytes captured (728 bits) on interf
> Ethernet II, Src: Intel_30:32:07 (00:93:37:30:32:07), Dst: IETF-VRRP-VRID_02 (0
> Internet Protocol Version 4, Src: 172.17.61.200, Dst: 172.24.2.76
> User Datagram Protocol, Src Port: 35839, Dst Port: 53
> Domain Name System (query)

0000  00 00 5e 00 01 02 00 93  37 30 32 07 08 00 45 00   ··^····· 702···E·
0010  00 4d 07 20 00 00 40 11  db 42 ac 11 3d c8 ac 18   ·M· ··@· ·B··=···
0020  02 4c 8b ff 00 35 00 39  98 88 ba 4f 01 00 00 01   ·L···5·9 ···O····
0030  00 00 00 00 00 01 07 77  77 77 2d 6e 65 74 02 63   ·······w ww-net·c
0040  73 05 75 6d 61 73 73 03  65 64 75 00 00 41 00 01   s·umass· edu··A··
0050  00 00 29 05 c0 00 00 00  00 00 00                  ··)····· ···

Ready to load or capture    Packets: 6370    Profile: Default
```

Figure 2: Screen Capture from Wireshark for the Website DNS Request

2.4 HTTP Details

Question: Expand the information on the HTTP message in the Wireshark "Details of selected packet" window (see Figure 2 of the Tutorial sheet) so you can see the fields in the HTTP GET request message. What type of Web browser issued the HTTP request? The answer is shown at the right end of the information following the "User-Agent:" field in the expanded HTTP message display. [This field value in the HTTP message is how a web server learns what type of browser you are using.]

Answer: You can see inFigure 3that the user agent field is populated with the browser I used, which is Mozilla Firefox. My operating system and system type is also sent, with `Ubuntu, Linux_x86_64` being seen.


testfire-http.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
No.   Time          Source          Destination     Protocol Lengtl Info
 3002 58.826873418  172.17.61.200   34.107.221.82   HTTP     367 GET /canonical.html HTTP/1.1
 3004 58.872706500  34.107.221.82   172.17.61.200   HTTP     364 HTTP/1.1 200 OK  (text/html)
 4544 62.474167582  172.17.61.200   65.61.137.117   HTTP     411 GET / HTTP/1.1
 4751 62.809289961  172.17.61.200   34.107.221.82   HTTP     384 GET /success.txt?ipv4 HTTP/1.1
 4848 62.855803736  34.107.221.82   172.17.61.200   HTTP     282 HTTP/1.1 200 OK  (text/plain)
 5308 63.169820079  65.61.137.117   172.17.61.200   HTTP    3929 HTTP/1.1 200 OK  (text/html)
 5344 63.274785991  172.17.61.200   65.61.137.117   HTTP     431 GET /style.css HTTP/1.1
 5625 63.647880558  65.61.137.117   172.17.61.200   HTTP    1544 HTTP/1.1 200 OK  (text/css)
 5627 63.648687672  172.17.61.200   65.61.137.117   HTTP     492 GET /images/home3.jpg HTTP/1.1
 5628 63.648885066  172.17.61.200   65.61.137.117   HTTP     492 GET /images/home2.jpg HTTP/1.1
 5629 63.649031294  172.17.61.200   65.61.137.117   HTTP     492 GET /images/home1.jpg HTTP/1.1
 5630 63.649354919  172.17.61.200   65.61.137.117   HTTP     494 GET /images/pf_lock.gif HTTP/1.1
```

```
> Frame 4544: 411 bytes on wire (3288 bits), 411 bytes captured (3288 bits) on
> Ethernet II, Src: Intel_30:32:07 (00:93:37:30:32:07), Dst: IETF-VRRP-VRID_02
> Internet Protocol Version 4, Src: 172.17.61.200, Dst: 65.61.137.117
> Transmission Control Protocol, Src Port: 45674, Dst Port: 80, Seq: 1, Ack: 1,
v Hypertext Transfer Protocol
  > GET / HTTP/1.1\r\n
    Host: demo.testfire.net\r\n
    User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:147.0) Gecko/201001
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\
    Accept-Language: en-US,en;q=0.9\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
```

```
0060  74 66 69 72 65 2e 6e 65  74 0d 0a 55 73 65 72 2d   tfire.ne t··User-
0070  41 67 65 6e 74 3a 20 4d  6f 7a 69 6c 6c 61 2f 35   Agent: M ozilla/5
0080  2e 30 20 28 58 31 31 3b  20 55 62 75 6e 74 75 3b   .0 (X11;  Ubuntu;
0090  20 4c 69 6e 75 78 20 78  38 36 5f 36 34 3b 20 72    Linux x 86_64; r
00a0  76 3a 31 34 37 2e 30 29  20 47 65 63 6b 6f 2f 32   v:147.0)  Gecko/2
00b0  30 31 30 30 31 30 31 20  46 69 72 65 66 6f 78 2f   0100101  Firefox/
00c0  31 34 37 2e 30 0d 0a 41  63 63 65 70 74 3a 20 74   147.0··A ccept: t
00d0  65 78 74 2f 68 74 6d 6c  2c 61 70 70 6c 69 63 61   ext/html ,applica
00e0  74 69 6f 6e 2f 78 68 74  6d 6c 2b 78 6d 6c 2c 61   tion/xht ml+xml,a
00f0  70 70 6c 69 63 61 74 69  6f 6e 2f 78 6d 6c 3b 71   pplicati on/xml;q
0100  3d 30 2e 39 2c 2a 2f 2a  3b 71 3d 30 2e 38 0d 0a   =0.9,*/* ;q=0.8··
0110  41 63 63 65 70 74 2d 4c  61 6e 67 75 61 67 65 3a   Accept-L anguage:
0120  20 65 6e 2d 55 53 2c 65  6e 3b 71 3d 30 2e 39 0d    en-US,e n;q=0.9·
0130  0a 41 63 63 65 70 74 2d  45 6e 63 6f 64 69 6e 67   ·Accept- Encoding
0140  3a 20 67 7a 69 70 2c 20  64 65 66 6c 61 74 65 0d   : gzip,  deflate·
0150  0a 43 6f 6e 6e 65 63 74  69 6f 6e 3a 20 6b 65 65   ·Connect ion: kee
0160  70 2d 61 6c 69 76 65 0d  0a 55 70 67 72 61 64 65   p-alive· ·Upgrade
0170  2d 49 6e 73 65 63 75 72  65 2d 52 65 71 75 65 73   -Insecur e-Reques
0180  74 73 3a 20 31 0d 0a 50  72 69 6f 72 69 74 79 3a   ts: 1··P riority:
```

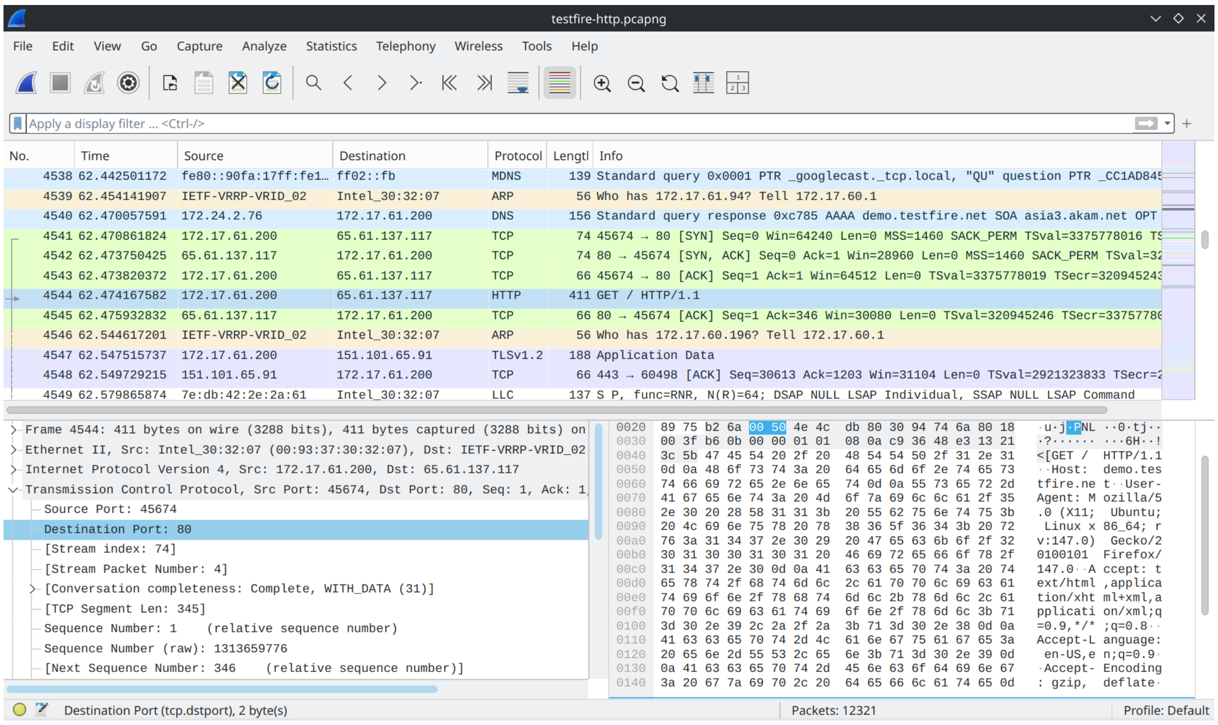
Hypertext Transfer Protocol: Protocol   Packets: 12321 · Displayed: 24 (0.2%)   Profile: Default

Figure 3: HTTP Browser Request Details

2.5 TCP Stats

Question: Expand the information on the Transmission Control Protocol (TCP is a transport layer protocol, reference: Section 8.5 in Garcia) for this packet in the Wireshark "Details of selected packet" window so you can see the fields in the TCP segment carrying the HTTP message. What is the destination port number (the number following "Dest Port:" for the TCP segment containing the HTTP request) to which this HTTP request is being sent?

Answer: The destination port, as can be seen in Figure 4 is 80, and it's source port is 45675.



```
No.  Time         Source              Destination     Protocol Lengtl Info
4538 62.442501172 fe80::90fa:17ff:fe1… ff02::fb        MDNS     139 Standard query 0x0001 PTR _googlecast._tcp.local, "QU" question PTR _CC1AD845
4539 62.454141907 IETF-VRRP-VRID_02   Intel_30:32:07  ARP       56 Who has 172.17.61.94? Tell 172.17.60.1
4540 62.470057591 172.24.2.76         172.17.61.200   DNS      156 Standard query response 0xc785 AAAA demo.testfire.net SOA asia3.akam.net OPT
4541 62.470861824 172.17.61.200       65.61.137.117   TCP       74 45674 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3375778016 TS
4542 62.473750425 65.61.137.117       172.17.61.200   TCP       74 80 → 45674 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM TSval=32
4543 62.473820372 172.17.61.200       65.61.137.117   TCP       66 45674 → 80 [ACK] Seq=1 Ack=1 Win=64512 Len=0 TSval=3375778019 TSecr=32094524
4544 62.474167582 172.17.61.200       65.61.137.117   HTTP     411 GET / HTTP/1.1
4545 62.475932832 65.61.137.117       172.17.61.200   TCP       66 80 → 45674 [ACK] Seq=1 Ack=346 Win=30080 Len=0 TSval=320945246 TSecr=3375778
4546 62.544617201 IETF-VRRP-VRID_02   Intel_30:32:07  ARP       56 Who has 172.17.60.196? Tell 172.17.60.1
4547 62.547515737 172.17.61.200       151.101.65.91   TLSv1.2  188 Application Data
4548 62.549729215 151.101.65.91       172.17.61.200   TCP       66 443 → 60498 [ACK] Seq=30613 Ack=1203 Win=31104 Len=0 TSval=2921323833 TSecr=2
4549 62.579865874 7e:db:42:2e:2a:61   Intel_30:32:07  LLC      137 S P, func=RNR, N(R)=64; DSAP NULL LSAP Individual, SSAP NULL LSAP Command
```

```
> Frame 4544: 411 bytes on wire (3288 bits), 411 bytes captured (3288 bits) on
> Ethernet II, Src: Intel_30:32:07 (00:93:37:30:32:07), Dst: IETF-VRRP-VRID_02
> Internet Protocol Version 4, Src: 172.17.61.200, Dst: 65.61.137.117
v Transmission Control Protocol, Src Port: 45674, Dst Port: 80, Seq: 1, Ack: 1,
    Source Port: 45674
    Destination Port: 80
    [Stream index: 74]
    [Stream Packet Number: 4]
  > [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 345]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1313659776
    [Next Sequence Number: 346    (relative sequence number)]
```

```
0020  89 75 b2 6a 00 50 4e 4c  db 80 30 94 74 6a 80 18   ·u·j·PNL ··0·tj··
0030  00 3f b6 0b 00 00 01 01  08 0a c9 36 48 e3 13 21   ·?······ ···6H··!
0040  3c 5b 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 31   <[GET /  HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20  64 65 6d 6f 2e 74 65 73   ··Host:  demo.tes
0060  74 66 69 72 65 2e 6e 65  74 0d 0a 55 73 65 72 2d   tfire.ne t··User-
0070  41 67 65 6e 74 3a 20 4d  6f 7a 69 6c 6c 61 2f 35   Agent: M ozilla/5
0080  2e 30 20 28 58 31 31 3b  20 55 62 75 6e 74 75 3b   .0 (X11;  Ubuntu;
0090  20 4c 69 6e 75 78 20 78  38 36 5f 36 34 3b 20 72    Linux x 86_64; r
00a0  76 3a 31 34 37 2e 30 29  20 47 65 63 6b 6f 2f 32   v:147.0)  Gecko/2
00b0  30 31 30 30 31 30 31 20  46 69 72 65 66 6f 78 2f   0100101  Firefox/
00c0  31 34 37 2e 30 0d 0a 41  63 63 65 70 74 3a 20 74   147.0··A ccept: t
00d0  65 78 74 2f 68 74 6d 6c  2c 61 70 70 6c 69 63 61   ext/html ,applica
00e0  74 69 6f 6e 2f 78 68 74  6d 6c 2b 78 6d 6c 2c 61   tion/xht ml+xml,a
00f0  70 70 6c 69 63 61 74 69  6f 6e 2f 78 6d 6c 3b 71   pplicati on/xml;q
0100  3d 30 2e 39 2c 2a 2f 2a  3b 71 3d 30 2e 38 0d 0a   =0.9,*/* ;q=0.8··
0110  41 63 63 65 70 74 2d 4c  61 6e 67 75 61 67 65 3a   Accept-L anguage:
0120  20 65 6e 2d 55 53 2c 65  6e 3b 71 3d 30 2e 39 0d    en-US,e n;q=0.9·
0130  0a 41 63 63 65 70 74 2d  45 6e 63 6f 64 69 6e 67   ·Accept- Encoding
0140  3a 20 67 7a 69 70 2c 20  64 65 66 6c 61 74 65 0d   : gzip,  deflate·
```

Destination Port (tcp.dstport), 2 byte(s) Packets: 12321 Profile: Default

Figure 4: TCP Packet Showing Destination Port

Note that, in the representation of the packet, you see the port number represented as `00 50`, but that's because this value is in hex. In decimal the packet number becomes 80.

2.6 Printing the HTTP Requests

Question: Print the two HTTP messages (GET and OK) referred to in question 2 above. To do so, select Print from the Wireshark File command menu, and select the "Selected Packet Only" and "Print as displayed" radial buttons, and then click OK.

Answer: The details of the HTTP packets have been printed, and the pdf is appended to the next page.

```
No.     Time           Source                Destination           Protocol Length Info
   4751 62.809289961   172.17.61.200         34.107.221.82         HTTP     384    GET /success.txt?ipv4 HTTP/1.1 
Frame 4751: 384 bytes on wire (3072 bits), 384 bytes captured (3072 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: Intel_30:32:07 (00:93:37:30:32:07), Dst: IETF-VRRP-VRID_02 (00:00:5e:00:01:02)
Internet Protocol Version 4, Src: 172.17.61.200, Dst: 34.107.221.82
Transmission Control Protocol, Src Port: 36004, Dst Port: 80, Seq: 1, Ack: 1, Len: 318
    Source Port: 36004
    Destination Port: 80
    [Stream index: 77]
    [Stream Packet Number: 4]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 318]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3279544848
    [Next Sequence Number: 319    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 1653599650
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x018 (PSH, ACK)
    Window: 63
    [Calculated window size: 64512]
    [Window size scaling factor: 1024]
    Checksum: 0xeafb [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    [Timestamps]
    [SEQ/ACK analysis]
    TCP payload (318 bytes)
Hypertext Transfer Protocol
No.     Time           Source                Destination           Protocol Length Info
   4848 62.855803736   34.107.221.82         172.17.61.200         HTTP     282    HTTP/1.1 200 OK  (text/plain)
Frame 4848: 282 bytes on wire (2256 bits), 282 bytes captured (2256 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: JuniperNetwo_b6:d7:f0 (7c:e2:ca:b6:d7:f0), Dst: Intel_30:32:07 (00:93:37:30:32:07)
Internet Protocol Version 4, Src: 34.107.221.82, Dst: 172.17.61.200
Transmission Control Protocol, Src Port: 80, Dst Port: 36004, Seq: 1, Ack: 319, Len: 216
    Source Port: 80
    Destination Port: 36004
    [Stream index: 77]
    [Stream Packet Number: 6]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 216]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1653599650
    [Next Sequence Number: 217    (relative sequence number)]
    Acknowledgment Number: 319    (relative ack number)
    Acknowledgment number (raw): 3279545166
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x018 (PSH, ACK)
    Window: 235
    [Calculated window size: 30080]
    [Window size scaling factor: 128]
    Checksum: 0x281a [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    [Timestamps]
    [SEQ/ACK analysis]
    TCP payload (216 bytes)
Hypertext Transfer Protocol
Line-based text data: text/plain (1 lines)
```