
COMPUTER NETWORKS LAB MANUAL

Arihant Chawla
Punjab Engineering College
17103036

Contents

1	Socket Programming Lab 1 : TCP Echo	3
2	Socket Programming Lab 2 : UDP Echo	7
3	Socket Programming Lab 3 : LAN Chat (Optional)	9
4	Wireshark Lab 1 : Introduction	14
5	Wireshark Lab 2 : HTTP	18
6	Wireshark Lab 3 : DNS	25
7	Wireshark Lab 4 : TCP	39
8	Wireshark Lab 5 : UDP (Optional)	47
9	Wireshark Lab 6 : IP	49
10	Wireshark Lab 7 : ICMP	54
11	Wireshark Lab 8 : Ethernet and ARP	58
12	Wireshark Lab 9 : DHCP	64

1 SOCKET PROGRAMMING LAB 1 : TCP ECHO

TcpEchoServer.c

```

1 #include <stdio.h>
2 #include <sys/socket.h>
3 #include <arpa/inet.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <unistd.h>
7
8 #define MAXPENDING 5
9 #define RCVBUFSIZE 32
10
11 void DieWithError(char *errorMessage);
12 void HandleTCPClient(int clntSocket);
13
14 int main(int argc, char *argv[])
15 {
16     printf("TCP ECHO SERVER PROGRAM. MADE BY ARIHANT CHAWLA\n");
17     int servSock;
18     int clntSock;
19     struct sockaddr_in echoServAddr;
20     struct sockaddr_in echoClntAddr;
21     unsigned short echoServPort;
22     unsigned int clntLen;
23     if (argc != 2)
24     {
25         fprintf(stderr, "Usage: %s <Server Port>\n", argv[0]);
26         exit(1);
27     }
28     echoServPort = atoi(argv[1]);
29     if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
30         DieWithError("socket () failed");
31     memset(&echoServAddr, 0, sizeof(echoServAddr));
32     echoServAddr.sin_family = AF_INET;
33     echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY);
34     echoServAddr.sin_port = htons(echoServPort);
35     if (bind(servSock, (struct sockaddr *)&echoServAddr, sizeof(
36     echoServAddr)) < 0)
            DieWithError("bind () failed");

```

4

```
37     if (listen(servSock, MAXPENDING) < 0)
38         DieWithError("listen() failed") ;
39     while(1)
40     {
41         clntLen = sizeof(echoClntAddr);
42         if ((clntSock = accept(servSock, (struct sockaddr *) &echoClntAddr,&
43             clntLen)) < 0)
44             DieWithError("accept() failed");
45         printf("Handling client %s\n", inet_ntoa(echoClntAddr.sin_addr));
46         HandleTCPClient (clntSock) ;
47     }
48
49 void HandleTCPClient(int clntSocket)
50 {
51     char echoBuffer[RCVBUFSIZE];
52     int recvMsgSize;
53     if ((recvMsgSize = recv(clntSocket , echoBuffer , RCVBUFSIZE, 0)) < 0)
54         DieWithError("recv() failed") ;
55     while (recvMsgSize > 0) /* zero indicates end of transmission */ {
56         if (send(clntSocket , echoBuffer , recvMsgSize , 0) != recvMsgSize)
57             DieWithError("send() failed");
58         if ((recvMsgSize = recv(clntSocket , echoBuffer , RCVBUFSIZE, 0)) <
59             0)
60             DieWithError("recv() failed") ;
61     }
62     close(clntSocket);
63 }
64 void DieWithError(char *errorMessage)
65 {
66     perror(errorMessage) ;
67     exit(1);
68 }
```

TcpEchoClient.c

```
1 #include <stdio.h>
2 #include <sys/socket.h>
3 #include <arpa/inet.h>
4 #include <stdlib.h>
```

```

5 #include <string.h>
6 #include <unistd.h>
7 #define RCVBUFSIZE 32
8 void DieWithError(char *errorMessage);
9 int main(int argc, char *argv[])
10 {
11     printf("TCP ECHO CLIENT PROGRAM. MADE BY ARIHANT CHAWLA\n");
12     int sock;
13     struct sockaddr_in echoServAddr;
14     unsigned short echoServPort;
15     char *servIP;
16     char *echoString;
17     char echoBuffer[RCVBUFSIZE];
18     unsigned int echoStringLen;
19     int bytesRcvd, totalBytesRcvd;
20     if ((argc< 3) || (argc> 4))
21     {
22         fprintf(stderr, "Usage: %s <Server IP> <Echo Word> [<Echo Port>]\n",
23         , argv[0]);
24         exit(1);
25     }
26     servIP = argv[1];
27     echoString = argv[2];
28     if (argc == 4)
29         echoServPort = atoi(argv[3]);
30     else
31         echoServPort = 7;
32     if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
33         DieWithError(" socket () failed");
34     memset(&echoServAddr, 0, sizeof(echoServAddr));
35     echoServAddr.sin_family = AF_INET;
36     echoServAddr.sin_addr.s_addr = inet_addr(servIP);
37     echoServAddr.sin_port = htons(echoServPort);
38     if (connect(sock, (struct sockaddr *) &echoServAddr, sizeof(
39     echoServAddr)) < 0)
40         DieWithError(" connect () failed");
41     echoStringLen = strlen(echoString);
42     if (send(sock, echoString, echoStringLen, 0) != echoStringLen)
43         DieWithError("send() sent a different number of bytes than expected
44 ");
45     totalBytesRcvd = 0;
46     printf("Received: ");

```

```

44     while ( totalBytesRcvd < echoStringLen )
45     {
46         if (( bytesRcvd = recv(sock , echoBuffer , RCVBUFSIZE - 1, 0)) <= 0)
47             DieWithError("recv() failed or connection closed prematurely");
48         totalBytesRcvd += bytesRcvd;
49         echoBuffer [bytesRcvd] = '\0';
50         printf(echoBuffer);
51     }
52     printf("\n");
53     close(sock);
54     exit(0);
55 }
56
57 void DieWithError(char *errorMessage)
58 {
59     perror(errorMessage) ;
60     exit(1);
61 }
```

TEST RUN:

<pre> arihant@l470:~/Desktop/CNS\$ gcc tcpechoserver.c -o server arihant@l470:~/Desktop/CNS\$./server 8080 TCP ECHO SERVER PROGRAM. MADE BY ARIHANT CHAWLA Handling client 127.0.0.1 Handling client 127.0.0.1 Handling client 127.0.0.1 [...] </pre>	<pre> arihant@l470:~/Desktop/CNS\$ gcc tcpechoclient.c -o client tcpechoclient.c: In function 'main': tcpechoclient.c:7:2: warning: format not a string literal and no format arguments [-Wformat-security] printf(echoBuffer); /* Print the echo buffer */ ^ arihant@l470:~/Desktop/CNS\$./client 127.0.0.1 hi 8080 TCP ECHO CLIENT PROGRAM. MADE BY ARIHANT CHAWLA Received: hi arihant@l470:~/Desktop/CNS\$./client 127.0.0.1 "my name is arihant" 8080 TCP ECHO CLIENT PROGRAM. MADE BY ARIHANT CHAWLA Received: my name is arihant arihant@l470:~/Desktop/CNS\$./client 127.0.0.1 "hola migos" 8080 TCP ECHO CLIENT PROGRAM. MADE BY ARIHANT CHAWLA Received: hola migos arihant@l470:~/Desktop/CNS\$ </pre>
--	--

2 SOCKET PROGRAMMING LAB 2 : UDP ECHO

UdpEchoServer.c

```

1 #include <arpa/inet.h>
2 #include <netinet/in.h>
3 #include <stdbool.h>
4 #include <stdio.h>
5 #include <string.h>
6 int main(int argc, char *argv[]) {
7     int SERVER_PORT = 8877;
8     struct sockaddr_in server_address;
9     memset(&server_address, 0, sizeof(server_address));
10    server_address.sin_family = AF_INET;
11    server_address.sin_port = htons(SERVER_PORT);
12    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
13    int sock;
14    if ((sock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
15        printf("could not create socket\n");
16        return 1;
17    }
18    if ((bind(sock, (struct sockaddr *)&server_address,
19              sizeof(server_address))) < 0) {
20        printf("could not bind socket\n");
21        return 1;
22    }
23    struct sockaddr_in client_address;
24    int client_address_len = 0;
25    while (true) {
26        char buffer[500];
27        int len = recvfrom(sock, buffer, sizeof(buffer), 0,
28                            (struct sockaddr *)&client_address,
29                            &client_address_len);
30        buffer[len] = '\0';
31        printf("received: '%s' from client %s\n", buffer,
32               inet_ntoa(client_address.sin_addr));
33        sendto(sock, buffer, len, 0, (struct sockaddr *)&client_address,
34               sizeof(client_address));
35    }
36    return 0;
37 }
```

UdpEchoClient.c

```

1 #include <arpa/inet.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <sys/socket.h>
5 #include <unistd.h>
6
7 int main() {
8     const char* server_name = "localhost";
9     const int server_port = 8877;
10    struct sockaddr_in server_address;
11    memset(&server_address, 0, sizeof(server_address));
12    server_address.sin_family = AF_INET;
13    inet_pton(AF_INET, server_name, &server_address.sin_addr);
14    server_address.sin_port = htons(server_port);
15    int sock;
16    if ((sock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
17        printf("could not create socket\n");
18        return 1;
19    }
20    const char* data_to_send = "ARIHANT's UDP NETAPP\n";
21    int len =
22        sendto(sock, data_to_send, strlen(data_to_send), 0,
23                (struct sockaddr*)&server_address, sizeof(server_address));
24    char buffer[100];
25    recvfrom(sock, buffer, len, 0, NULL, NULL);
26    buffer[len] = '\0';
27    printf("recieved: %s\n", buffer);
28    close(sock);
29    return 0;
30 }
```

TEST RUN:

<pre>arhant@l470:~/Desktop/CNS\$ gcc udpechoclient.c -o client arhant@l470:~/Desktop/CNS\$./client "ARIHANT's UDP NETAPP" </pre>	<pre>arhant@l470:~/Desktop/CNS\$ gcc udpechoserver.c -o server arhant@l470:~/Desktop/CNS\$./server received: 'ARIHANT's UDP NETAPP ' from client 0.0.0.0 </pre>
---	--

3 SOCKET PROGRAMMING LAB 3 : LAN CHAT (OPTIONAL)

LANChatServer.c

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4 #include<unistd.h>
5 #include<sys/types.h>
6 #include<sys/socket.h>
7 #include<netinet/in.h>
8
9 void error(const char * msg)
10 {
11     perror(msg);
12     exit(1);
13 }
14
15 int main(int argc, char * argv[])
16 {
17     if(argc<2)
18     {
19         fprintf(stderr, "usage :: ./a.out [PORT_NO]\nPROGRAM TERMINATED\n");
20     ;
21     exit(1);
22     }
23
24     sockfd , newsockfd , portno , n;
25     char buffer[255];
26     struct sockaddr_in serv_addr , cli_addr ;
27
28     sockfd=socket (AF_INET,SOCK_STREAM,0) ;
29     if (sockfd<0)
30     {
31         error ("ERROR OPENING SOCKET\n");
32     }
33
34     portno=atoi(argv[1]);
35
36     serv_addr.sin_family=AF_INET;
37     serv_addr.sin_addr.s_addr=INADDR_ANY;

```

10

```
37 serv_addr.sin_port=htons( portno );
38 memset( serv_addr.sin_zero , '\0' , 8 );
39
40 if( bind( sockfd , ( struct sockaddr * )&serv_addr , sizeof( serv_addr ) ) < 0 )
41 {
42     error( "BINDING FAILED\n" );
43 }
44
45 listen( sockfd , 5 );
46 int clilen = sizeof( cli_addr );
47 newsockfd = accept( sockfd , ( struct sockaddr * )&cli_addr , &clilen );
48
49 if( newsockfd < 0 )
50 {
51     error( "ACCEPT FAILED" );
52 }
53 printf( "\n\n-----\nWELCOME TO ARIHANT'S CHATTING APP\n\n-----\n\n" );
54 while( 1 )
55 {
56     bzero( buffer , 255 );
57     n = recv( newsockfd , buffer , 255 , 0 );
58     if( n < 0 )
59     {
60         error( "READ ERROR\n" );
61     }
62     printf( "CLIENT: %s" , buffer );
63     int i = strncmp( "BYE" , buffer , 3 );
64     if( i == 0 )
65         break;
66
67     i = strncmp( "bye" , buffer , 3 );
68     if( i == 0 )
69         break;
70
71     i = strncmp( "Bye" , buffer , 3 );
72     if( i == 0 )
73         break;
74
75     bzero( buffer , 255 );
76     printf( "SERVER: " );
```

```

77     fgets ( buffer ,255 ,stdin ) ;
78
79     n=send ( newsockfd ,buffer ,strlen ( buffer ) ,0 ) ;
80     if ( n<0 )
81     {
82         error ( "ERROR ON WRITING\n" ) ;
83     }
84
85     i=strncmp ( "BYE" ,buffer ,3 ) ;
86     if ( i==0 )
87         break ;
88
89     i=strncmp ( "bye" ,buffer ,3 ) ;
90     if ( i==0 )
91         break ;
92
93     i=strncmp ( "Bye" ,buffer ,3 ) ;
94     if ( i==0 )
95         break ;
96
97 }
98 close ( newsockfd ) ;
99 close ( sockfd ) ;
100 return 0 ;
101 }
```

LANChatClient.c

```

1 #include<stdio .h>
2 #include<stdlib .h>
3 #include<string .h>
4 #include<unistd .h>
5 #include<sys/types .h>
6 #include<sys/socket .h>
7 #include<netinet/in .h>
8 #include <arpa/inet .h>
9
10 void  error ( const  char  * msg)
11 {
12     perror ( msg ) ;
13     exit (1) ;
```

12

```
14 }
15
16 int main( int argc , char* argv [] )
17 {
18     if ( argc < 3 )
19     {
20         fprintf ( stderr , " ::USAGE:: ./ a.out [ SERVER_IP ] [ SERVER_PORTNO ] \n " );
21         exit ( 1 );
22     }
23
24     int sockfd , n ;
25     struct sockaddr_in serv_addr ;
26     char buffer [ 255 ];
27     int portno ;
28
29     portno = atoi ( argv [ 2 ] );
30
31     sockfd = socket ( AF_INET , SOCK_STREAM , 0 ) ;
32     if ( sockfd < 0 )
33     {
34         error ( " ERROR OPENING SOCKET \n " );
35     }
36
37     serv_addr . sin_family = AF_INET ;
38     serv_addr . sin_port = htons ( portno ) ;
39     serv_addr . sin_addr . s_addr = inet_addr ( argv [ 1 ] );
40     memset ( serv_addr . sin_zero , '\0' , 8 );
41
42     if ( connect ( sockfd , ( struct sockaddr * ) & serv_addr , sizeof ( serv_addr ) ) < 0 )
43     {
44         error ( " CONNECTION FAILED \n " );
45     }
46     printf ( "\n\n-----\n\n" );
47     while ( 1 )
48     {
49         bzero ( buffer , 255 );
50         printf ( " CLIENT: " );
51         fgets ( buffer , 255 , stdin );
52
53         n = send ( sockfd , buffer , strlen ( buffer ) , 0 );
```

```
54     if (n<0)
55     {
56         error( "ERROR ON SEND" );
57     }
58     int i=strncmp( "BYE" ,buffer ,3 );
59     if ( i==0)
60         break;
61
62     i=strncmp( "Bye" ,buffer ,3 );
63     if ( i==0)
64         break;
65
66     i=strncmp( "bye" ,buffer ,3 );
67     if ( i==0)
68         break;
69     bzero( buffer ,255 );
70     n=recv( sockfd ,buffer ,255 ,0 );
71     if (n<0)
72     {
73         error( "ERROR ON READING" );
74     }
75
76     printf( "SERVER: %s" ,buffer );
77
78     i=strncmp( "BYE" ,buffer ,3 );
79     if ( i==0)
80         break;
81
82     i=strncmp( "Bye" ,buffer ,3 );
83     if ( i==0)
84         break;
85
86     i=strncmp( "bye" ,buffer ,3 );
87     if ( i==0)
88         break;
89
90 }
91 close( sockfd );
92 return 0;
93 }
```

4 WIRESHARK LAB 1 : INTRODUCTION

Answer to Q1

The different protocols that appear in the protocol column in the unfiltered packet-listing window (obviously may/not be linked to the packet we want to analyze) are:

1. DNS
2. TCP
3. HTTP
4. UDP
5. TLSv1.2
6. ICMPv6

Answer to Q2

From the time column, the first packet was sent at 1.209578725

The time when the first OK was received was 1.543625200

Therefore time taken from request to OK is $1.543625200 - 1.209578725 = 0.334046475$ sec

No.	Time	Info
9	1.209578725	GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1
11	1.543625200	HTTP/1.1 200 OK (text/html)
13	2.140780371	GET /favicon.ico HTTP/1.1
14	2.464956288	HTTP/1.1 404 Not Found (text/html)

Answer to Q3

The IP Address of gaia.cs.umass.edu (Destination) is 128.119.245.12

The IP Address of my laptop (source of HTTP request) is 192.168.1.7

No.	Time	Source	Destination
9	1.209578725	192.168.1.7	128.119.245.12
11	1.543625200	128.119.245.12	192.168.1.7
13	2.140780371	192.168.1.7	128.119.245.12
14	2.464956288	128.119.245.12	192.168.1.7

Answer to Q4

The following two pages contain the printed pdfs of GET and OK packets respectively.

No.	Time	Source	Destination	Info
9	1.209578725	192.168.1.7	128.119.245.12	GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1 HTTP/1.1 443 Frame 9: 443 bytes on wire (3544 bits), 443 bytes captured (3544 bits) on interface 0 Ethernet II, Src: IntelCor_8d:11:de (a0:af:bd:8d:11:de), Dst: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af) Internet Protocol Version 4, Src: 192.168.1.7, Dst: 128.119.245.12 Transmission Control Protocol, Src Port: 43762, Dst Port: 80, Seq: 1, Ack: 1, Len: 377 Source Port: 43762 Destination Port: 80 [Stream index: 0] [TCP Segment Len: 377] Sequence number: 1 (relative sequence number) [Next sequence number: 378 (relative sequence number)] Acknowledgment number: 1 (relative ack number) 1000 = Header Length: 32 bytes (8) Flags: 0x018 (PSH, ACK) Window size value: 229 [Calculated window size: 29312] [Window size scaling factor: 128] Checksum: 0x537e [unverified] [Checksum Status: Unverified] Urgent pointer: 0 Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps [SEQ/ACK analysis] [Timestamps] TCP payload (377 bytes) Hypertext Transfer Protocol GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1\r\nHost: gaia.cs.umass.edu\r\nUser-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\nAccept-Language: en-US,en;q=0.5\r\nAccept-Encoding: gzip, deflate\r\nConnection: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\n\r\n[Full request URI: http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html] [HTTP request 1/2] [Response in frame: 11] [Next request in frame: 13]

No.	Time	Source	Destination	Info
Protocol Length				
11	1.543625200	128.119.245.12	192.168.1.7	HTTP/1.1 200 OK (text/html)
HTTP	504			
Frame 11: 504 bytes on wire (4032 bits), 504 bytes captured (4032 bits) on interface 0				
Ethernet II, Src: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af), Dst: IntelCor_8d:11:de (a0:af:bd:8d:11:de)				
Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.1.7				
Transmission Control Protocol, Src Port: 80, Dst Port: 43762, Seq: 1, Ack: 378, Len: 438				
Source Port: 80				
Destination Port: 43762				
[Stream index: 0]				
[TCP Segment Len: 438]				
Sequence number: 1 (relative sequence number)				
[Next sequence number: 439 (relative sequence number)]				
Acknowledgment number: 378 (relative ack number)				
1000 = Header Length: 32 bytes (8)				
Flags: 0x018 (PSH, ACK)				
Window size value: 235				
[Calculated window size: 30080]				
[Window size scaling factor: 128]				
Checksum: 0x222a [unverified]				
[Checksum Status: Unverified]				
Urgent pointer: 0				
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps				
[SEQ/ACK analysis]				
[Timestamps]				
TCP payload (438 bytes)				
Hypertext Transfer Protocol				
HTTP/1.1 200 OK\r\n				
Date: Mon, 11 Mar 2019 21:23:57 GMT\r\n				
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/5.4.16 mod_perl/2.0.10 Perl/v5.16.3\r\n				
Last-Modified: Mon, 11 Mar 2019 05:59:01 GMT\r\n				
ETag: "51-583cb44ee5b70"\r\n				
Accept-Ranges: bytes\r\n				
Content-Length: 81\r\n				
Keep-Alive: timeout=5, max=100\r\n				
Connection: Keep-Alive\r\n				
Content-Type: text/html; charset=UTF-8\r\n				
\r\n				
[HTTP response 1/2]				
[Time since request: 0.334046475 seconds]				
[Request in frame: 9]				
[Next request in frame: 13]				
[Next response in frame: 14]				
File Data: 81 bytes				
Line-based text data: text/html (3 lines)				

5 WIRESHARK LAB 2 : HTTP

Answer to Q1

Both my browser and the server is running HTTP v1.1

GET Packet:

```
▼ Hypertext Transfer Protocol
  ▶ GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1\r\n
```

OK Packet:

```
▼ Hypertext Transfer Protocol
  ▶ HTTP/1.1 200 OK\r\n
```

Answer to Q2

The browser tells the server that it accepts US English as the language.

```
▼ Hypertext Transfer Protocol
  ▶ GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1\r\n
    Host: gaia.cs.umass.edu\r\n
    User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0)
    Accept: text/html,application/xhtml+xml,application/xml;q=0.
    Accept-Language: en-US,en;q=0.5\r\n
```

Answer to Q3

The IP Address of gaia.cs.umass.edu (Destination) is 128.119.245.12

The IP Address of my laptop (source of HTTP request) is 192.168.1.7

No.	Time	Source	Destination
+	9 1.209578725	192.168.1.7	128.119.245.12
-	11 1.543625200	128.119.245.12	192.168.1.7
.	13 2.140780371	192.168.1.7	128.119.245.12
	14 2.464956288	128.119.245.12	192.168.1.7

Answer to Q4

Status Code: 200 (OK)

```

▼ Hypertext Transfer Protocol
  ▶ HTTP/1.1 200 OK\r\n

```

Answer to Q5

Last modified date of the webpage is in the 'Last-Modified' attribute of the OK packet.

```

▼ Hypertext Transfer Protocol
  ▼ HTTP/1.1 200 OK\r\n
    ▶ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
      Date: Mon, 11 Mar 2019 21:23:57 GMT\r\n
      Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/
      Last-Modified: Mon, 11 Mar 2019 05:59:01 GMT\r\n

```

Answer to Q6

The content length returned is 81 bytes.

```

▼ Hypertext Transfer Protocol
  ▶ HTTP/1.1 200 OK\r\n
    Date: Mon, 11 Mar 2019 21:23:57 GMT\r\n
    Server: Apache/2.4.6 (CentOS) OpenSSL/1.0
    Last-Modified: Mon, 11 Mar 2019 05:59:01
    ETag: "51-583cb44ee5b70"\r\n
    Accept-Ranges: bytes\r\n
  ▶ Content-Length: 81\r\n

```

Answer to Q7

No, all the headers within the raw data are shown by wireshark in the packet listing window.

Answer to Q8

No, there is no 'if-modified-since' line in the first HTTP get packet.

Answer to Q9

Yes, the server explicitly returns the content of the file. It is present in the data portion of the raw data of the packet encapsulated by the headers.

```

▶ Frame 30: 796 bytes on wire (6368 bits), 796 bytes captured (6368 bits) on interface 0
▶ Ethernet II, Src: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af), Dst: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
▶ Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.1.7
▶ Transmission Control Protocol, Src Port: 80, Dst Port: 48232, Seq: 1, Ack: 377, Len: 730
▶ Hypertext Transfer Protocol
▼ Line-based text data: text/html (10 lines)
  \n
  <html>\n
  \n
  Congratulations again! Now you've downloaded the file lab2-2.html. <br>\n
  This file's last modification date will not change. <p>\n
  Thus if you download this multiple times on your browser, a complete copy <br>\n
  will only be sent once by the server due to the inclusion of the IN-MODIFIED-SINCE<br>\n
  field in your browser's HTTP GET request to the server.\n
  \n
  </html>\n

```

01b0	0a 0a 43 6f 6e 67 72 61 74 75 6c 61 74 69 6f 6e	..Congra tulation
01c0	73 20 61 67 61 69 6e 21 20 20 4e 6f 77 20 79 6f	s again! Now yo
01d0	75 27 76 65 20 64 6f 77 6e 6c 6f 61 64 65 64 20	u've dow nloaded
01e0	74 68 65 20 66 69 6c 65 20 6c 61 62 32 2d 32 2e	the file lab2-2.
01f0	68 74 6d 6c 2e 20 3c 62 72 3e 0a 54 68 69 73 20	html. <b r>.This
0200	66 69 6c 65 27 73 20 6c 61 73 74 20 6d 6f 64 69	file's l ast modi
0210	66 69 63 61 74 69 6f 6e 20 64 61 74 65 20 77 69	fication date wi
0220	6c 6c 20 6e 6f 74 20 63 68 61 6e 67 65 2e 20 20	ll not c hange.
0230	3c 70 3e 0a 54 68 75 73 20 20 69 66 20 79 6f 75	<p>.Thus if you
0240	20 64 6f 77 6e 6c 6f 61 64 20 74 68 69 73 20 6d	downloa d this m
0250	75 6c 74 69 70 6c 65 20 74 69 6d 65 73 20 6f 6e	ultiple times on
0260	20 79 6f 75 72 20 62 72 6f 77 73 65 72 2c 20 61	your br owser, a
0270	20 63 6f 6d 70 6c 65 74 65 20 63 6f 70 79 20 3c	complet e copy <
0280	62 72 3e 0a 77 69 6c 6c 20 6f 6e 6c 79 20 62 65	br>.will only be
0290	20 73 65 6e 74 20 6f 6e 63 65 20 62 79 20 74 68	sent on ce by th
02a0	65 20 73 65 72 76 65 72 20 64 75 65 20 74 6f 20	e server due to
02b0	74 68 65 20 69 6e 63 6c 75 73 69 6f 6e 20 6f 66	the incl usion of
02c0	20 74 68 65 20 49 4e 2d 4d 4f 44 49 46 49 45 44	the IN- MODIFIED
02d0	2d 53 49 4e 43 45 3c 62 72 3e 0a 66 69 65 6c 64	-SINCE<b r>.field
02e0	20 69 6e 20 79 6f 75 72 20 62 72 6f 77 73 65 72	in your browser

Text item (text), 73 bytes

Answer to Q10

Yes, there is an 'IF-MODIFIED-SINCE' line in the second GET packet. Just after this is 'IF-NONE-MATCH'.

```

If-Modified-Since: Tue, 12 Mar 2019 05:59:02 GMT\r\n
If-None-Match: "173-583df62cc77ea"\r\n

```

Answer to Q11

The status code returned is 304: NOT MODIFIED.

This time the server does not explicitly return the data of the webpage. This is because

the web page has not been modified. It has been brought into the browser's cache and is now retrieved from here until it gets modified.

```
▼ Hypertext Transfer Protocol
  ▼ HTTP/1.1 304 Not Modified\r\n
    ► [Expert Info (Chat/Sequence): HTTP/1.1 304 Not Modified\r\n]
      Response Version: HTTP/1.1
      Status Code: 304
      [Status Code Description: Not Modified]
      Response Phrase: Not Modified
```

Answer to Q12

Only one GET request was sent by my browser (ignoring the favicon request which the manual tells to ignore).

No.	Time	Info
+	28 4.396785647	GET /wireshark-labs/HTTP-wireshark-file3.html HTTP/1.1
+	41 4.682511482	HTTP/1.1 200 OK (text/html)
+	47 7.213630023	GET /favicon.ico HTTP/1.1
+	50 7.578233675	HTTP/1.1 404 Not Found (text/html)

Answer to Q13

There are total 4 TCP packets between the GET and OK packets but out of them **only 2** contain data. This is analysed by looking at the TCP payload. The length of the first two packets are 0 bytes, so they have no data payload and the next two have length 2800 and 1400 bytes.

S.	Time	Info	Source	Destination	Protocol
28	4.3967...	GET /wireshark-labs/HTTP-wireshark-file3.html HTTP/...	192.168.1.7	128.119.245.12	HTTP
34	4.6080...	80 → 48484 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 M..	128.119.245.12	192.168.1.7	TCP
36	4.6720...	80 → 48482 [ACK] Seq=1 Ack=377 Win=30080 Len=0 TSva...	128.119.245.12	192.168.1.7	TCP
37	4.6723...	80 → 48482 [ACK] Seq=1 Ack=377 Win=30080 Len=2800 T..	128.119.245.12	192.168.1.7	TCP
39	4.6824...	80 → 48482 [ACK] Seq=2801 Ack=377 Win=30080 Len=140..	128.119.245.12	192.168.1.7	TCP
41	4.6825...	HTTP/1.1 200 OK (text/html)	128.119.245.12	192.168.1.7	HTTP

Total Length: 2852
Identification: 0x7ced (31981)
Flags: 0x4000, Don't fragment
Time to live: 46
Protocol: TCP (6)

```

0010 0b 24 7c ed 40 00 2e 06 8d b3 80 77 f5 0c c0 a8 ·$|·@... ···w···
0020 01 07 00 50 bd 62 bc 63 9b cc cd 53 bf ca 80 10 ···P·b·c ···S···
0030 00 eb 42 4a 00 00 01 01 08 0a ab 0e e7 20 eb f6 ··BJ·····
0040 3d 8c 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f =-HTTP/1.1 200 0
0050 4b 0d 0a 44 61 74 65 3a 20 54 75 65 2c 20 31 32 K·Date: Tue, 12
0060 20 4d 61 72 20 32 30 31 39 20 31 33 3a 33 35 3a Mar 2019 13:35:
0070 35 39 20 47 4d 54 0d 0a 53 65 72 76 65 72 3a 20 59 GMT· Server:
0080 41 70 61 63 68 65 2f 32 2e 34 2e 36 20 28 43 65 Apache/2.4.6 (Ce
0090 6e 74 4f 53 29 20 4f 70 65 6e 53 53 4c 2f 31 2e ntOS) OpenSSL/1.
00a0 30 2e 32 6b 2d 66 69 70 73 20 50 48 50 2f 35 2e 0.2k-fip s PHP/5.
00b0 34 2e 31 36 20 6d 6f 64 5f 70 65 72 6c 2f 32 2e 4.16 mod_perl/2.
00c0 30 2e 31 30 20 50 65 72 6c 2f 76 35 2e 31 36 2e 0.10 Per 1/v5.16.
00d0 33 0d 0a 4c 61 73 74 2d 4d 6f 64 69 66 69 65 64 3>Last-Modified
00e0 3a 20 54 75 65 2c 20 31 32 20 4d 61 72 20 32 30 :Tue, 1 2 Mar 20
00f0 31 39 20 30 35 3a 35 39 3a 30 32 20 47 4d 54 0d 19 05:59 :02 GMT·
0100 0a 45 54 61 67 3a 20 22 31 31 39 34 2d 35 38 33 ·ETag: "1194-583
0110 64 66 36 32 63 63 32 39 63 39 22 0d 0a 41 63 63 df62cc29 c9"·Acc
0120 65 70 74 2d 52 61 6e 67 65 73 3a 20 62 79 74 65 ept-Rang es: byte
0130 73 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 s·Conte nt-Lengt
0140 68 3a 20 34 35 30 30 0d 0a 4b 65 65 70 2d 41 6c h: 4500· ·Keep-Al
0150 69 76 65 3a 20 74 69 6d 65 6f 75 74 3d 35 2c 20 ive: tim eout=5,
0160 6d 61 78 3d 31 30 30 0d 0a 43 6f 6e 6e 65 63 74 max=100· ·Connect
0170 69 6f 6e 3a 20 4b 65 65 70 2d 41 6c 69 76 65 0d ion: Kee p-Alive·
0180 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74 ·Content-Type: t
0190 65 78 74 2f 68 74 6d 6c 3b 20 63 68 61 72 73 65 ext/html ; charse
01a0 74 3d 55 54 46 2d 38 0d 0a 0d 0a 3c 68 74 6d 6c t=UTF-8· ·<html
01b0 3e 3c 68 65 61 64 3e 20 0a 3c 74 69 74 6c 65 3e ><head> ·<title>
01c0 48 69 73 74 6f 72 69 63 61 6c 20 44 6f 63 75 6d Historic al Docum

```

Answer to Q14

The status code is 200: OK. The response phrase associated with it is OK.

```

▼ Hypertext Transfer Protocol
  ▼ HTTP/1.1 200 OK\r\n
    ▼ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      [HTTP/1.1 200 OK\r\n]
      [Severity level: Chat]
      [Group: Sequence]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK

```

Answer to Q15

There is one HTTP continuation packet.

10 HTTP	GET /wireshark-labs/HTTP-wireshark-file3.html	HTTP/1.1
11 TCP	80 → 43620 [ACK] Seq=1 Ack=366 Win=229 Len=0	
12 TCP	80 → 43620 [PSH, ACK] Seq=1 Ack=366 Win=229 Len=2920 [TCP segment of a reassembled PDU]	
13 TCP	43620 → 80 [ACK] Seq=366 Ack=2921 Win=274 Len=0	
14 HTTP	HTTP/1.1 200 OK (text/html)	

Answer to Q16

The browser sends 3 GET requests (ignoring the favicon GET request). All of them are to 128.119.245.12

http contains "GET"					
No.	Time	Info	Source	Destination	
4	0.4415...	GET /wireshark-labs/HTTP-wireshark-file4...	192.168.1.7	128.119.245.12	
12	1.4793...	GET /pearson.png HTTP/1.1	192.168.1.7	128.119.245.12	
28	1.8688...	GET /~kurose/cover_5th_ed.jpg HTTP/1.1	192.168.1.7	128.119.245.12	

Answer to Q17

Comparing the timestamps of the OK of the first image and the GET of the second image, it is pretty clear that the download of the two images occurs serially as the second image is requested only after the first has been received by the browser.

No.	Time	Info
4	0.441548835	GET /wireshark-labs/HTTP-wireshark-file4...
7	0.775852131	HTTP/1.1 200 OK (text/html)
12	1.479326081	GET /pearson.png HTTP/1.1
23	1.781681436	HTTP/1.1 200 OK (PNG)
...	1.868858722	GET /~kurose/cover_5th_ed.jpg HTTP/1.1
+	3.076931120	HTTP/1.1 200 OK (JPEG JFIF image)

Answer to Q18

The status code is 401: Unauthorized. The response phrase is 'Unauthorized'

```
▼ Hypertext Transfer Protocol
  ▼ HTTP/1.1 401 Unauthorized\r\n
    ▼ [Expert Info (Chat/Sequence): HTTP/1.1 401 Unauthorized\r\n]
      [HTTP/1.1 401 Unauthorized\r\n]
      [Severity level: Chat]
      [Group: Sequence]
      Response Version: HTTP/1.1
      Status Code: 401
      [Status Code Description: Unauthorized]
      Response Phrase: Unauthorized
```

Answer to Q19

The difference between the two GET requests is that the new one has an authorization field which wasn't there in the initial GET request. It says 'Authorization: Basic' and then some unencrypted base64 message, which I assume are the login credentials.

INITIAL GET

```
▼ Hypertext Transfer Protocol
  ▶ GET /wireshark-labs/protected_pages/HTTP-wireshark-file5.html HTTP/1.1\r\n
  Host: gaia.cs.umass.edu\r\n
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
\r\n
[Full request URI: http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html]
[HTTP request 1/1]
[Response in frame: 21]
```

GET AFTER LOGIN

```
▼ Hypertext Transfer Protocol
  ▶ GET /wireshark-labs/protected_pages/HTTP-wireshark-file5.html HTTP/1.1\r\n
  Host: gaia.cs.umass.edu\r\n
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Authorization: Basic d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms=\r\n
\r\n
[Full request URI: http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html]
[HTTP request 1/1]
[Response in frame: 168]
```

On decoding the base64 text, it is easy to see both the username and the password. This login is not really immune to packet analyzing or man in the middle attacks.

```
arihant@l470:~$ echo d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms= | base64 --decode && echo
echo the newline character
wireshark-students:network
```

6 WIRESHARK LAB 3 : DNS

Answer to Q1

For a server in Asia, I looked up the DNS of Flipkart

```
arihant@l470:~$ nslookup www.flipkart.com/
Server:          127.0.0.53
Address:         127.0.0.53#53

Non-authoritative answer:
Name:   www.flipkart.com/
Address: 104.239.213.7
Name:   www.flipkart.com/
Address: 198.105.244.11
```

Answer to Q2

For a university in Europe, I looked up the DNS of University of Barcelona

```
arihant@l470:~$ nslookup www.ub.edu
Server:          127.0.0.53
Address:         127.0.0.53#53

Non-authoritative answer:
Name:   www.ub.edu
Address: 161.116.100.2
```

Answer to Q3

```
arihant@l470:~$ nslookup
> mta6.am0.yahoodns.net www.ub.edu
Server:          127.0.0.53
Address:         127.0.0.53#53

Non-authoritative answer:
Name:   mta6.am0.yahoodns.net
Address: 98.136.102.54
Name:   mta6.am0.yahoodns.net
Address: 66.218.85.52
Name:   mta6.am0.yahoodns.net
Address: 74.6.137.64
Name:   mta6.am0.yahoodns.net
Address: 66.218.85.139
Name:   mta6.am0.yahoodns.net
Address: 98.136.101.117
Name:   mta6.am0.yahoodns.net
Address: 67.195.229.58
Name:   mta6.am0.yahoodns.net
Address: 98.137.159.25
Name:   mta6.am0.yahoodns.net
Address: 98.137.159.28
> exit
```

Answer to Q4

The DNS query and response are sent over UDP

```

T 48 21.4172931... DNS      Standard query 0x0d53 A www.ietf.org
! 49 21.4173671... DNS      Standard query 0x5413 AAAA www.ietf.org
|-
► Frame 48: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface 0
► Ethernet II, Src: IntelCor_8d:11:de (a0:af:bd:8d:11:de), Dst: Cisco_c7:cc:00 (00:c1:64:c7:cc:00)
► Internet Protocol Version 4, Src: 172.31.79.59, Dst: 172.31.1.220
► User Datagram Protocol, Src Port: 47475, Dst Port: 53
► Domain Name System (query)

```

Answer to Q5

The destination port of DNS query is 53 and the source port of the response is also 53

QUERY

```

► Frame 49: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface 0
► Ethernet II, Src: IntelCor_8d:11:de (a0:af:bd:8d:11:de), Dst: Cisco_c7:cc:00 (00:c1:64:c7:cc:00)
► Internet Protocol Version 4, Src: 172.31.79.59, Dst: 172.31.1.220
▼ User Datagram Protocol, Src Port: 34755, Dst Port: 53
  Source Port: 34755
  Destination Port: 53
  Length: 38
  Checksum: 0x57e0 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 10]
▼ Domain Name System (query)
  Transaction ID: 0x5413
  ► Flags: 0x0100 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  ► Queries
    [Response In: 51]

```

RESPONSE

```

► Frame 51: 173 bytes on wire (1384 bits), 173 bytes captured (1384 bits) on interface 0
► Ethernet II, Src: Cisco_c7:cc:00 (00:c1:64:c7:cc:00), Dst: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
► Internet Protocol Version 4, Src: 172.31.1.220, Dst: 172.31.79.59
▼ User Datagram Protocol, Src Port: 53, Dst Port: 34755
  Source Port: 53
  Destination Port: 34755
  Length: 139
  Checksum: 0x9c8a [unverified]
  [Checksum Status: Unverified]
  [Stream index: 10]
▼ Domain Name System (response)
  Transaction ID: 0x5413
  ► Flags: 0x8100 Standard query response, No error
  Questions: 1
  Answer RRs: 3
  Authority RRs: 0
  Additional RRs: 0
  ► Queries
  ► Answers
    [Request In: 49]
    [Time: 0.163698008 seconds]

```

Answer to Q6

The DNS query is sent to the IP 172.31.1.220. Upon looking at the local DNS provided by DHCP, it turns out that both are same.

→ 49 21.4173671... DNS	Standard query 0x5413 AAAA www.ietf.org	172.31.79.59 172.31.1.220
← ... 21.5810651... DNS	Standard query response 0x5413 AAAA www.ietf.o...	172.31.1.220 172.31.79.59

```
arihant@l470:/etc/init.d$ nmcli device show wlp5s0 | grep IP4
IP4.ADDRESS[1]:                         172.31.79.59/20
IP4.GATEWAY:                            172.31.64.1
IP4.ROUTE[1]:                           dst = 0.0.0.0/0, nh = 172.31.64.1, mt = 600
IP4.ROUTE[2]:                           dst = 172.31.64.0/20, nh = 0.0.0.0, mt = 600
IP4.ROUTE[3]:                           dst = 169.254.0.0/16, nh = 0.0.0.0, mt = 1000
IP4.DNS[1]:                             172.31.1.220
```

Answer to Q7

There are two sets of queries and responses. In one instance, the DNS query is of the type AAAA and in the other of type A. These are corresponding to IPv6 and IPv4 DNS addresses. Both, however, do not contain any answers

```
▼ Domain Name System (query)
  Transaction ID: 0x0d53
  ▶ Flags: 0x0100 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  ▼ Queries
    ▼ www.ietf.org: type A, class IN
      Name: www.ietf.org
      [Name Length: 12]
      [Label Count: 3]
      Type: A (Host Address) (1)
      Class: IN (0x0001)
      [Response In: 50]
```

```

▶ Frame 49: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface 0
▶ Ethernet II, Src: IntelCor_8d:11:de (a0:af:bd:8d:11:de), Dst: Cisco_c7:cc:00 (00:c1:64:c7:cc:00)
▶ Internet Protocol Version 4, Src: 172.31.79.59, Dst: 172.31.1.220
▶ User Datagram Protocol, Src Port: 34755, Dst Port: 53
▼ Domain Name System (query)
    Transaction ID: 0x5413
    ▶ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
    ▼ Queries
        ▶ www.ietf.org: type AAAA, class IN
            Name: www.ietf.org
            [Name Length: 12]
            [Label Count: 3]
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
\[Response In: 51\]

```

Answer to Q8

The two DNS responses corresponding to IPv6 and IPv4 are of the type AAAA and A respectively. Both contained three answers each, two of them of the type AAAA/A and one of them of the type CNAME. They contained information about the type, class, time to live and data length as well as the address or canonical name depending on the type of the DNS server

```

▼ Queries
    ▶ www.ietf.org: type A, class IN
        Name: www.ietf.org
        [Name Length: 12]
        [Label Count: 3]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
    ▼ Answers
        ▶ www.ietf.org: type CNAME, class IN, cname www.ietf.org.cdn.cloudflare.net
            Name: www.ietf.org
            Type: CNAME (Canonical NAME for an alias) (5)
            Class: IN (0x0001)
            Time to live: 256
            Data length: 33
            CNAME: www.ietf.org.cdn.cloudflare.net
        ▶ www.ietf.org.cdn.cloudflare.net: type A, class IN, addr 104.20.1.85
            Name: www.ietf.org.cdn.cloudflare.net
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 260
            Data length: 4
            Address: 104.20.1.85
        ▶ www.ietf.org.cdn.cloudflare.net: type A, class IN, addr 104.20.0.85
            Name: www.ietf.org.cdn.cloudflare.net
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 260
            Data length: 4
            Address: 104.20.0.85
\[Request In: 48\]
\[Time: 0.001331137 seconds\]

```

```

    ▼ Queries
      ▼ www.ietf.org: type AAAA, class IN
        Name: www.ietf.org
        [Name Length: 12]
        [Label Count: 3]
        Type: AAAA (IPv6 Address) (28)
        Class: IN (0x0001)

    ▼ Answers
      ▼ www.ietf.org: type CNAME, class IN, cname www.ietf.org.cdn.cloudflare.net
        Name: www.ietf.org
        Type: CNAME (Canonical NAME for an alias) (5)
        Class: IN (0x0001)
        Time to live: 256
        Data length: 33
        CNAME: www.ietf.org.cdn.cloudflare.net
      ▼ www.ietf.org.cdn.cloudflare.net: type AAAA, class IN, addr 2606:4700:10::6814:55
        Name: www.ietf.org.cdn.cloudflare.net
        Type: AAAA (IPv6 Address) (28)
        Class: IN (0x0001)
        Time to live: 300
        Data length: 16
        AAAA Address: 2606:4700:10::6814:55
      ▼ www.ietf.org.cdn.cloudflare.net: type AAAA, class IN, addr 2606:4700:10::6814:155
        Name: www.ietf.org.cdn.cloudflare.net
        Type: AAAA (IPv6 Address) (28)
        Class: IN (0x0001)
        Time to live: 300
        Data length: 16
        AAAA Address: 2606:4700:10::6814:155
[Request In: 49]
[Time: 0.163698008 seconds]

```

Answer to Q9

The destination IP address is 104.20.0.85 which does not match to either of the previous IP addresses of the DNS.

49 DNS	Standard query 0x5413 AAAA www.ietf.org	172.31.79.59	172.31.1.220
51 DNS	Standard query response 0x5413 AAAA www.ietf.o...	172.31.1.220	172.31.79.59
... TCP	48702 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=14..	172.31.79.59	104.20.0.85

Answer to Q10

There does not seem to be any DNS query/reponse corresponding to the IETF web-page when downloading the images.

49 DNS	Standard query 0x5413 AAAA www.ietf.org	172.31.79.59	172.31.1.220	21.4173671...
50 DNS	Standard query response 0x0d53 A www.ietf.org CNAME www.ietf... 172.31.1.220 172.31.79.59	21.4186242...		
51 DNS	Standard query response 0x5413 AAAA www.ietf.org CNAME www.ietf... 172.31.1.220 172.31.79.59	21.5810651...		
52 TCP	48702 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 ... 172.31.79.59 104.20.0.85	21.5821072...		
53 TCP	443 → 48702 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 S... 104.20.0.85 172.31.79.59	21.5830310...		
54 TCP	48702 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0	172.31.79.59	104.20.0.85	21.5830636...
55 TLSv1.3	Client Hello	172.31.79.59	104.20.0.85	21.5855170...
56 TCP	443 → 48702 [ACK] Seq=1 Ack=518 Win=30336 Len=0	104.20.0.85	172.31.79.59	21.5863933...
57 TLSv1.3	Server Hello, Change Cipher Spec	104.20.0.85	172.31.79.59	22.0602689...
58 TCP	48702 → 443 [ACK] Seq=518 Ack=1461 Win=32128 Len=0	172.31.79.59	104.20.0.85	22.0602935...
59 TCP	[TCP Previous segment not captured] 443 → 48702 [PSH, ACK] Se... 104.20.0.85 172.31.79.59	22.0663068...		
60 TCP	[TCP Window Update] 48702 → 443 [ACK] Seq=518 Ack=1461 Win=303...	172.31.79.59	104.20.0.85	22.0603117...
61 TCP	[TCP Out-Of-Order] 443 → 48702 [ACK] Seq=1461 Ack=518 Win=303...	104.20.0.85	172.31.79.59	22.0664877...
62 TCP	48702 → 443 [ACK] Seq=518 Ack=4381 Win=38016 Len=0	172.31.79.59	104.20.0.85	22.0684962...
... TLSv1.3	Application Data	104.20.0.85	172.31.79.59	22.0605008...
64 TCP	48702 → 443 [ACK] Seq=518 Ack=4593 Win=40960 Len=0	172.31.79.59	104.20.0.85	22.0605035...
70 TCP	48704 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 ... 172.31.79.59 104.20.0.85	22.1117141...		
71 TCP	443 → 48704 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 S... 104.20.0.85 172.31.79.59	22.1127011...		
72 TCP	48704 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0	172.31.79.59	104.20.0.85	22.1127263...
73 TLSv1.3	Client Hello	172.31.79.59	104.20.0.85	22.1137011...
74 TCP	443 → 48704 [ACK] Seq=1 Ack=518 Win=30336 Len=0	104.20.0.85	172.31.79.59	22.1147895...
81 TLSv1.3	Server Hello, Change Cipher Spec	104.20.0.85	172.31.79.59	22.5543097...
82 TCP	48704 → 443 [ACK] Seq=518 Ack=2921 Win=35072 Len=0	172.31.79.59	104.20.0.85	22.5543636...
83 TCP	[TCP Previous segment not captured] 443 → 48704 [PSH, ACK] Se... 104.20.0.85 172.31.79.59	22.5543937...		
84 TCP	[TCP Window Update] 48704 → 443 [ACK] Seq=518 Ack=2921 Win=38...	172.31.79.59	104.20.0.85	22.5544032...
85 TCP	[TCP Out-Of-Order] 443 → 48704 [ACK] Seq=2921 Ack=518 Win=303...	104.20.0.85	172.31.79.59	22.5544308...
86 TCP	48704 → 443 [ACK] Seq=518 Ack=4593 Win=40960 Len=0	172.31.79.59	104.20.0.85	22.5544443...
99 TLSv1.3	Change Cipher Spec, Application Data	172.31.79.59	104.20.0.85	22.6677173...
1.. TCP	443 → 48702 [ACK] Seq=4593 Ack=582 Win=30336 Len=0	104.20.0.85	172.31.79.59	22.6684324...
1.. TLSv1.3	Application Data	172.31.79.59	104.20.0.85	22.6688477...
1.. TCP	443 → 48702 [ACK] Seq=4593 Ack=752 Win=30336 Len=0	104.20.0.85	172.31.79.59	22.6696025...
1.. TLSv1.3	Application Data	104.20.0.85	172.31.79.59	22.7564901...
1.. TCP	48702 → 443 [ACK] Seq=752 Ack=5043 Win=43904 Len=0	172.31.79.59	104.20.0.85	22.7565294...
1.. TLSv1.3	Application Data	104.20.0.85	172.31.79.59	22.7579241...
1.. TCP	48702 → 443 [ACK] Seq=752 Ack=5105 Win=43904 Len=0	172.31.79.59	104.20.0.85	22.7579457...
1.. TLSv1.3	Application Data	172.31.79.59	104.20.0.85	22.7581181...
1.. TCP	443 → 48702 [ACK] Seq=5105 Ack=783 Win=30336 Len=0	104.20.0.85	172.31.79.59	22.7592464...
1.. TLSv1.3	Application Data	104.20.0.85	172.31.79.59	22.8410065...

Answer to Q11

The destination port of DNS query is 53 and the source port of the response is also 53

QUERY

- ▼ User Datagram Protocol, Src Port: 42532, Dst Port: 53
 - Source Port: 42532
 - Destination Port: 53
 - Length: 49
 - Checksum: 0x815f [unverified]
 - [Checksum Status: Unverified]
 - [Stream index: 0]

RESPONSE

- ▼ User Datagram Protocol, Src Port: 53, Dst Port: 42532
 - Source Port: 53
 - Destination Port: 42532
 - Length: 101
 - Checksum: 0x9fbf [unverified]
 - [Checksum Status: Unverified]
 - [Stream index: 0]

Answer to Q12

The DNS query is sent to the IP 172.31.1.220. Upon looking at the local DNS provided by DHCP, it turns out that both are same.

```
8 DNS Standard query 0x3c3e A www.mit.edu.edgekey.net 172.31.79.59 172.31.1.220
15 DNS Standard query response 0x3c3e A www.mit.edu.edgekey.net ... 172.31.1.220 172.31.79.59
```

```
arihant@l470:/etc/init.d$ nmcli device show wlp5s0 | grep IP4
IP4.ADDRESS[1]: 172.31.79.59/20
IP4.GATEWAY: 172.31.64.1
IP4.ROUTE[1]: dst = 0.0.0.0/0, nh = 172.31.64.1, mt = 600
IP4.ROUTE[2]: dst = 172.31.64.0/20, nh = 0.0.0.0, mt = 600
IP4.ROUTE[3]: dst = 169.254.0.0/16, nh = 0.0.0.0, mt = 1000
IP4.DNS[1]: 172.31.1.220
```

Answer to Q13

There DNS query is of the type A. It does not, however, contain any answers

```
▼ Queries
  ▼ www.mit.edu.edgekey.net: type A, class IN
    Name: www.mit.edu.edgekey.net
    [Name Length: 23]
    [Label Count: 5]
    Type: A (Host Address) (1)
    Class: IN (0x0001)
    [Response In: 15]
```

Answer to Q14

The DNS responses is of the type A. It contained two answers, one of the type A and the other of type CNAME. They contained information about the type, class, time to live and data length as well as the address or canonical name depending on the type of the DNS server

```
▼ Answers
  ▼ www.mit.edu.edgekey.net: type CNAME, class IN, cname e9566.dsrb.akamaiedge.net
    Name: www.mit.edu.edgekey.net
    Type: CNAME (Canonical NAME for an alias) (5)
    Class: IN (0x0001)
    Time to live: 46
    Data length: 24
    CNAME: e9566.dsrb.akamaiedge.net
  ▼ e9566.dsrb.akamaiedge.net: type A, class IN, addr 23.43.29.195
    Name: e9566.dsrb.akamaiedge.net
    Type: A (Host Address) (1)
    Class: IN (0x0001)
    Time to live: 15
    Data length: 4
    Address: 23.43.29.195
    [Request In: 8]
    [Time: 0.388468526 seconds]
```

Answer to Q15

File Edit View Go Capture Analyze Statistics Telephone Wireless Tools Help						
No.	Proto	Info	Source	Destination	Time	Length
8	DNS	Standard query 0x3c3e A www.mit.edu.edgekey.net	172.31.79.59	172.31.1.220	1.113622460	1.502100986
I 15 DNS Standard query response 0x3c3e A www.mit.edu.edgekey.net ... 172.31.1.220 172.31.79.59						
Domain Name System (response)						
Transaction ID: 0xc3c3e						
Flags: 0x8180 Standard query response, No error						
Questions: 1						
Answer RRs: 2						
Authority RRs: 0						
Additional RRs: 0						
Queries						
▼ www.mit.edu.edgekey.net: type A, class IN						
Name: www.mit.edu.edgekey.net						
[Name Length: 23]						
[Label Count: 5]						
Type: A (Host Address) (1)						
Class: IN (0x0001)						
Answers						
▼ www.mit.edu.edgekey.net: type CNAME, class IN, cname e9566.dscb.akamaiedge.net						
Name: www.mit.edu.edgekey.net						
Type: CNAME (Canonical NAME for an alias) (5)						
Class: IN (0x0001)						
Time to live: 46						
.....						
0030	00	02 00 00 00 00 03 77 77 77 03 6d 69 74 03 65w w.w.mit.e			
0040	64	75 07 65 64 67 65 6b 65 79 03 6e 65 74 00 00	du-edgekey.net. .			
0050	01	00 01 c0 0c 05 00 01 00 00 00 2e 00 18 05			
0060	65	39 35 36 04 64 73 63 62 0a 61 6b 61 6d 61	e9566.ds cb.akama			
0070	69	65 64 67 65 c0 29 c0 35 00 01 00 00 01 00 00	edge. . 5.....			
0080	0f	00 04 17 2b 1d c3+..			

Answer to Q16

The DNS query is sent to the IP 192.168.1.1. Upon looking at the local DNS provided by DHCP, it turns out that both are same.

```
37 DNS Standard query 0x3de3 NS mit.edu OPT           192.168.1.7 192.168.1.1
38 DNS Standard query response 0x3de3 NS mit.edu NS use5.akam.net 192.168.1.1 192.168.1.7
```

```
arihant@l470:/etc/init.d$ nmcli device show wlp5s0 | grep IP4
IP4.ADDRESS[1]:                         192.168.1.7/24
IP4.GATEWAY:                            192.168.1.1
IP4.ROUTE[1]:                           dst = 0.0.0.0/0, nh = 192.168.1.1, mt = 600
IP4.ROUTE[2]:                           dst = 192.168.1.0/24, nh = 0.0.0.0, mt = 600
IP4.ROUTE[3]:                           dst = 169.254.0.0/16, nh = 0.0.0.0, mt = 1000
IP4.DNS[1]:                             192.168.1.1
```

Answer to Q17

The query is of the type NS (authoritative name server) and contains no answers.

```
▼ Domain Name System (query)
  Transaction ID: 0x3de3
  ▶ Flags: 0x0100 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 1
  ▼ Queries
    ▼ mit.edu: type NS, class IN
      Name: mit.edu
      [Name Length: 7]
      [Label Count: 2]
      Type: NS (authoritative Name Server) (2)
      Class: IN (0x0001)
  ▶ Additional records
  [Response In: 38]
```

Answer to Q18

The following nameservers are provided in the DNS response message:

1. ns1-173.akam.net
2. use2.akam.net
3. use5.akam.net
4. usw2.akam.net

5. asia2.akam.net
6. ns1-37.akam.net
7. asia1.akam.net
8. eur5.akam.net

The response message does not contain IP addresses.

```
arihant@l470:~$ nslookup -type=NS mit.edu
Server:      127.0.0.53
Address:     127.0.0.53#53

Non-authoritative answer:
mit.edu nameserver = ns1-173.akam.net.
mit.edu nameserver = use2.akam.net.
mit.edu nameserver = use5.akam.net.
mit.edu nameserver = usw2.akam.net.
mit.edu nameserver = asia2.akam.net.
mit.edu nameserver = ns1-37.akam.net.
mit.edu nameserver = asia1.akam.net.
mit.edu nameserver = eur5.akam.net.

Authoritative answers can be found from:
```

▼ Answers

- mit.edu: type NS, class IN, ns ns1-173.akam.net
 - Name: mit.edu
 - Type: NS (authoritative Name Server) (2)
 - Class: IN (0x0001)
 - Time to live: 1248
 - Data length: 18
 - Name Server: ns1-173.akam.net
- mit.edu: type NS, class IN, ns use2.akam.net
- mit.edu: type NS, class IN, ns use5.akam.net
- mit.edu: type NS, class IN, ns usw2.akam.net
- mit.edu: type NS, class IN, ns asia2.akam.net
- mit.edu: type NS, class IN, ns ns1-37.akam.net
- mit.edu: type NS, class IN, ns asia1.akam.net
- mit.edu: type NS, class IN, ns eur5.akam.net

Answer to Q19

Wireshark - Packet 60 - wlq550 Fri 10:33

Frame 60: 245 bytes on wire (1960 bits), 245 bytes captured (1960 bits) on interface 0
 Ethernet II, Src: HuaweiTE-a9:93:af (14:08:b4:a9:93:af), Dst: IntelCor-8d:11:de (ae:af:bd:8d:11:de)
 Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.7
 User Datagram protocol, Src Port: 53, Dst Port: 59659
 Domain Name System (response)
 Transaction ID: 0xach
 Flags: 0x08180 Standard query response, No error
 Questions: 1
 Answer RRs: 8
 Authority RRs: 0
 Additional RRs: 1

Queries

Answers

mit.edu: type NS, class IN, ns ns1-173.akam.net
 Name: mit.edu
 Type: NS (authoritative Name Server) (2)
 Class: IN (0x0001)
 Time to live: 1248
 Data length: 18
 Name Server: ns1-173.akam.net
 mit.edu: type NS, class IN, ns user2.akam.net
 mit.edu: type NS, class IN, ns user5.akam.net
 mit.edu: type NS, class IN, ns user2.akam.net
 mit.edu: type NS, class IN, ns asia2.akam.net
 mit.edu: type NS, class IN, ns ns1-37.akam.net
 mit.edu: type NS, class IN, ns asia1.akam.net
 mit.edu: type NS, class IN, ns eur5.akam.net

Additional records

<Root>: type OPT
 Name: <Root>
 Type: OPT (41)
 UDP payload size: 4996
 Higher bits in extended RCODE: 0x00
 EDNS0 version: 0

Z 0x0000
 0... = DO bit: Cannot handle DNSSEC security RRs
 Data length: 0
 000 0000 0000 0000 = Reserved: 0x0000

[Request In: 56]
[Time: 0.799301648 seconds]

0040	02 00 01 00 0c 00 00 02 00 01 00 00 04 00 00 12 07
0050	6e 73 31 2d 31 37 33 04 61 6b 61 6d 03 6e 65 74 ns1-173.akam.net

X Close Help

For the fourth trace connection was getting timed out. So after repeated failed attempts, I started working on the trace file provided for the question.

Answer to Q20

The DNS queries are sent to 18.72.0.3. It did not seem like the IP of the Kurose and Rose webpage, from where the traces were originally run. So, I performed a whois query and it revealed that the IP did actually belong to MIT.

100 DNS	128.238.38....	18.72.0.3	Standard query 0x0001 PTR 3.0.72.18.in-addr.arpa
101 DNS	18.72.0.3	128.238.38.160	Standard query response 0x0001 PTR 3.0.72.18.in-
102 DNS	128.238.38....	18.72.0.3	Standard query 0x0002 A www.aiit.or.kr.poly.edu
103 DNS	18.72.0.3	128.238.38.160	Standard query response 0x0002 No such name A www.aiit.or.kr
104 DNS	128.238.38....	18.72.0.3	Standard query 0x0003 A www.aiit.or.kr
105 DNS	18.72.0.3	128.238.38.160	Standard query response 0x0003 A www.aiit.or.kr

```
arihant@l470:~$ whois 18.72.0.3

#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/resources/registry/whois/tou/
#
# If you see inaccuracies in the results, please report at
# https://www.arin.net/resources/registry/whois/inaccuracy_reporting/
#
# Copyright 1997-2019, American Registry for Internet Numbers, Ltd.
#


NetRange:      18.0.0.0 - 18.127.255.255
CIDR:         18.0.0.0/9
NetName:       MIT
NetHandle:     NET-18-0-0-0-1
Parent:        NET18 (NET-18-0-0-0-0)
NetType:       Direct Assignment
OriginAS:      AS3
Organization:  Massachusetts Institute of Technology (MIT-2)
RegDate:       1994-01-01
Updated:       2018-06-29
Ref:          https://rdap.arin.net/registry/ip/18.0.0.0
```

Answer to Q21

This is a type A query with no answers.

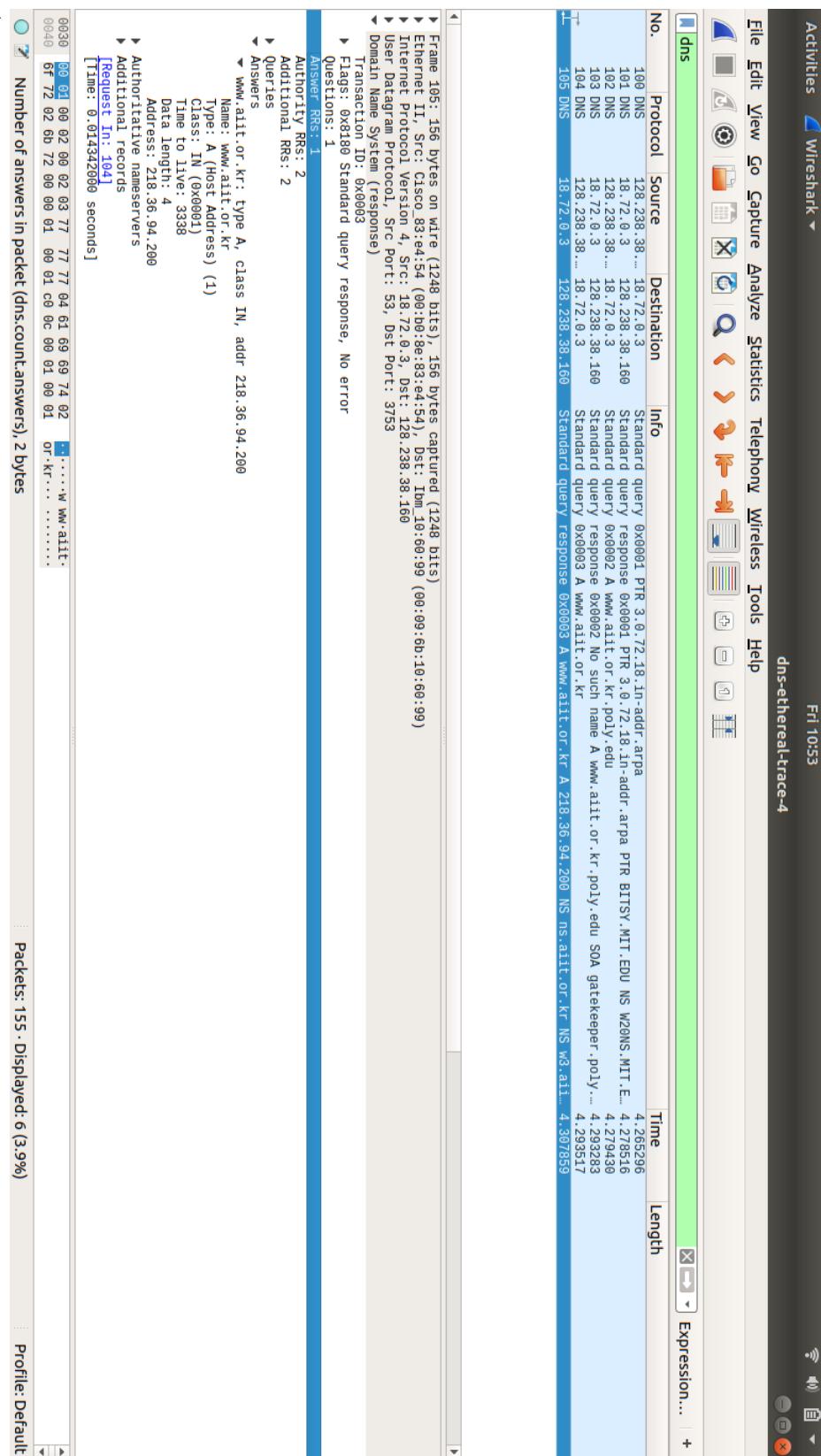
```
▼ Domain Name System (query)
  Transaction ID: 0x0003
  ▶ Flags: 0x0100 Standard query
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  ▼ Queries
    ▶ www.aiit.or.kr: type A, class IN
      [Response In: 105]
```

Answer to Q22

The DNS response message contains one answer. The answer contains NAME, TYPE, CLASS, TTL, Data Length and Address.

```
Answer RRs: 1
Authority RRs: 2
Additional RRs: 2
▶ Queries
▼ Answers
  ▶ www.aiit.or.kr: type A, class IN, addr 218.36.94.200
    Name: www.aiit.or.kr
    Type: A (Host Address) (1)
    Class: IN (0x0001)
    Time to live: 3338
    Data length: 4
    Address: 218.36.94.200
  ▶ Authoritative nameservers
```

Answer to Q23



7 WIRESHARK LAB 4 : TCP

Answer to Q1 and Q3

The source (client) IP is 172.31.69.70 and the source port number is 43022

```
Source: 172.31.69.70
Destination: 128.119.245.12
Transmission Control Protocol, Src Port: 43022, Dst Port: 80,
Source Port: 43022
Destination Port: 80
```

Answer to Q2

The destination (gaia.cs.umass.edu) IP is 128.119.245.12 and the port number it sends and receives TCP segments is 80.

```
Source: 172.31.69.70
Destination: 128.119.245.12
Transmission Control Protocol, Src Port: 43022, Dst Port: 80,
Source Port: 43022
Destination Port: 80
```

Answer to Q4

The sequence number of the SYN packet used to establish the connection is 0 (relative sequence number). What makes it a SYN segment is the fact that the FLAG field of the TCP header has SYN flag enabled and no else.

```
Source: 172.31.69.70
Destination: 128.119.245.12
▼ Transmission Control Protocol, Src Port: 43022, Dst Port: 80, Seq: 0, Len: 0
  Source Port: 43022
  Destination Port: 80
  [Stream index: 0]
  [TCP Segment Len: 0]
  Sequence number: 0      (relative sequence number)
  [Next sequence number: 0      (relative sequence number)]
  Acknowledgment number: 0
  1010 .... = Header Length: 40 bytes (10)
  ▼ Flags: 0x002 (SYN)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0.... .... = Congestion Window Reduced (CWR): Not set
    .... .0... .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...0 .... = Acknowledgment: Not set
    .... ...0.... = Push: Not set
    .... .... .0.. = Reset: Not set
    ▶ .... .... .1. = Syn: Set
    .... .... 0 = Fin: Not set
    ... [TCP Flags: .....$..]
```

Answer to Q5

The sequence number of the SYNACK packet sent by the webserver is 0 (relative sequence number). The Acknowledgment field is 1 (relative acknowledgement number). This is the value of the sequence number of the next byte the receiver expects to receive and so it is 1 as the Sequence number is 0. What makes it a SYNACK segment is the fact that the FLAG field of the TCP header has SYN and ACK flag enabled and no other.

```

Source: 128.119.245.12
Destination: 172.31.69.70
▼ Transmission Control Protocol, Src Port: 80, Dst Port: 43022, Seq: 0, Ack: 1, Len: 0
  Source Port: 80
  Destination Port: 43022
  [Stream index: 0]
  [TCP Segment Len: 0]
  Sequence number: 0      (relative sequence number)
  [Next sequence number: 0      (relative sequence number)]
  Acknowledgment number: 1    (relative ack number)
  1000 .... = Header Length: 32 bytes (8)
  ▼ Flags: 0x012 (SYN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0.... .... = Congestion Window Reduced (CWR): Not set
    .... .0... .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0.... = Push: Not set
    .... .... .0... = Reset: Not set
    ► .... .... .1. = Syn: Set
    .... .... ...0 = Fin: Not set
    [TCP Flags: .....A..S..]

```

Answer to Q6

The sequence number of the TCP segment containing the HTTP POST command is 1.

```

Sequence number: 1      (relative sequence number)
[Next sequence number: 1461      (relative sequence number)]
Acknowledgment number: 1    (relative ack number)
0101 .... = Header Length: 20 bytes (5)
▼ Flags: 0x010 (ACK)
  000. .... .... = Reserved: Not set
  ...0 .... .... = Nonce: Not set
  .... 0.... .... = Congestion Window Reduced (CWR): Not set
  .... .0... .... = ECN-Echo: Not set
  .... ..0. .... = Urgent: Not set
  .... ...1 .... = Acknowledgment: Set
  .... .... 0.... = Push: Not set
  .... .... .0... = Reset: Not set
  .... .... ..0. = Syn: Not set
  .... .... ...0 = Fin: Not set

```

▼ Data (1460 bytes)	
Data: 504f5354202f77697265736861726b2d6c6162732f6c6162... [Length: 1460]	
0030	00 e5 54 5a 00 00 50 4f 53 54 20 2f 77 69 72 65
0040	73 68 61 72 6b 2d 6c 61 62 73 2f 6c 61 62 33 2d
0050	31 2d 72 65 70 6c 79 2e 68 74 6d 20 48 54 54 50
0060	2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 67 61 69 61
0070	2e 63 73 2e 75 6d 61 73 73 2e 65 64 75 0d 0a 55
0080	73 65 72 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c
0090	6c 61 2f 35 2e 30 20 28 58 31 31 3b 20 55 62 75
00a0	6e 74 75 3b 20 4c 69 6e 75 78 20 78 38 36 5f 36
00b0	34 3b 20 72 76 3a 36 36 2e 30 29 29 47 65 63 6b
00c0	6f 2f 32 30 31 30 30 31 30 31 20 46 69 72 65 66
00d0	6f 78 2f 36 36 2e 30 0d 0a 41 63 63 65 70 74 3a
00e0	20 74 65 78 74 2f 68 74 6d 6c 2c 61 70 70 6c 69
00f0	63 61 74 69 6f 6e 2f 78 68 74 6d 6c 2b 78 6d 6c
0100	2c 61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 6d 6c
0110	3b 71 3d 30 2e 39 2c 2a 2f 2a 3b 71 3d 30 2e 38
0120	0d 0a 41 63 63 65 70 74 2d 4c 61 6e 67 75 61 67
0130	65 3a 20 65 6e 2d 55 53 2c 65 6e 3b 71 3d 30 2e
0140	35 0d 0a 41 63 63 65 70 74 2d 45 6e 63 6f 64 69

At this point my laptop shut down accidentally and the .pcapng file was not saved so started using the trace file provided.

Answer to Q7

Segment #	Sent Time	ACK Receive Time	RTT
566	0.026477	0.053937	0.02746
2026	0.041737	0.077294	0.035557
3486	0.054026	0.124085	0.070059
4946	0.054690	0.169118	0.11443
6406	0.077405	0.217299	0.13989
7866	0.078157	0.267802	0.18964

The formula for EstimatedRTT:

$$\text{EstimatedRTT} = 0.875 * \text{EstimatedRTT} + 0.125 * \text{SampleRTT}$$

For segment# 566: EstimatedRTT = RTT (1st segment) = 0.0276 sec.

For segment# 2026: EstimatedRTT = $0.875 * 0.02746 + 0.125 * 0.035557 = 0.0285$ sec.

For segment# 3486: EstimatedRTT = $0.875 * 0.0285 + 0.125 * 0.070059 = 0.0337$ sec.

For segment# 4946: EstimatedRTT = $0.875 * 0.0337 + 0.125 * 0.11443 = 0.0438$ sec.

42

For segment# 6406: EstimatedRTT = $0.875 * 0.0438 + 0.125 * 0.13989 = 0.0558$ sec.

For segment# 7866: EstimatedRTT = $0.875 * 0.0558 + 0.125 * 0.18964 = 0.0725$ sec.

5 TCP	1161 → 80 [PSH, ACK]	Seq=566 Ack=1 Win=17520 Len=1460	[TCP segment of a re...	192.168.1.1..	128.119.245.12	0.041737
7 TCP	1161 → 80 [ACK]	Seq=2026 Ack=1 Win=17520 Len=1460	[TCP segment of a rea...	192.168.1.1..	128.119.245.12	0.054026
8 TCP	1161 → 80 [ACK]	Seq=3486 Ack=1 Win=17520 Len=1460	[TCP segment of a rea...	192.168.1.1..	128.119.245.12	0.054690
10 TCP	1161 → 80 [ACK]	Seq=4946 Ack=1 Win=17520 Len=1460	[TCP segment of a rea...	192.168.1.1..	128.119.245.12	0.077405
11 TCP	1161 → 80 [ACK]	Seq=6406 Ack=1 Win=17520 Len=1460	[TCP segment of a rea...	192.168.1.1..	128.119.245.12	0.078157
13 TCP	1161 → 80 [PSH, ACK]	Seq=7866 Ack=1 Win=17520 Len=1147	[TCP segment of a rea...	192.168.1.1..	128.119.245.12	0.124185
18 TCP	1161 → 80 [ACK]	Seq=9013 Ack=1 Win=17520 Len=1460	[TCP segment of a rea...	192.168.1.1..	128.119.245.12	0.305040

Answer to Q8

The length of the first segment is 565 and the rest are of length 1460 each.

4 TCP	1161 → 80 [PSH, ACK]	Seq=1 Ack=1 Win=17520 Len=565	[TCP segment of a re...
5 TCP	1161 → 80 [PSH, ACK]	Seq=566 Ack=1 Win=17520 Len=1460	[TCP segment of a re...
7 TCP	1161 → 80 [ACK]	Seq=2026 Ack=1 Win=17520 Len=1460	[TCP segment of a rea...
8 TCP	1161 → 80 [ACK]	Seq=3486 Ack=1 Win=17520 Len=1460	[TCP segment of a rea...
10 TCP	1161 → 80 [ACK]	Seq=4946 Ack=1 Win=17520 Len=1460	[TCP segment of a rea...
11 TCP	1161 → 80 [ACK]	Seq=6406 Ack=1 Win=17520 Len=1460	[TCP segment of a rea...

Answer to Q9

The minimum amount of buffer space (rwnd) advertised by the server in the first ACK it sends (while establishing connection) is 5840 bytes.

2 TCP 80 → 1161 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 ..

The calculated size of the receiving window grows upto 62780 bytes and then plateaus out. It doesn't seem like the sender is ever throttled by the lack of buffer space as there is no re-transmission.

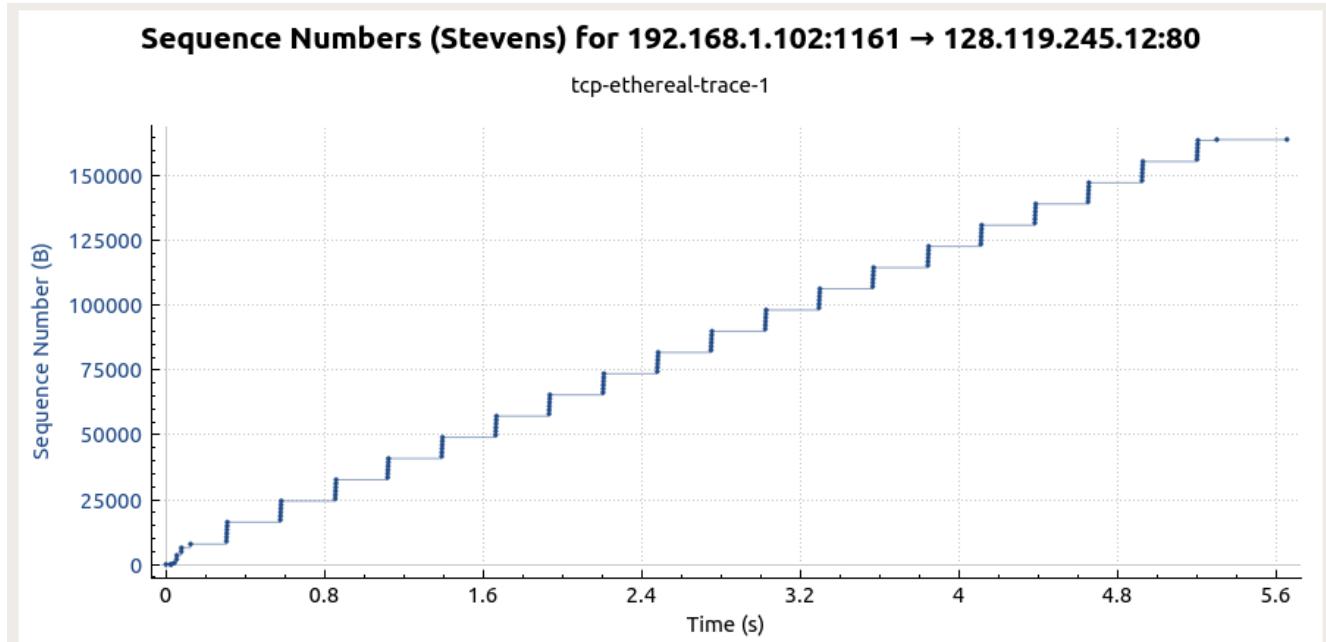
2	TCP	80 -> 1161 [SYN]	Seq=0	Ack=1	Win=5840	Len=0	MSS=1460
6	TCP	80 -> 1161 [ACK]	Seq=1	Ack=566	Win=6780	Len=0	
9	TCP	80 -> 1161 [ACK]	Seq=1	Ack=2026	Win=8760	Len=0	
12	TCP	80 -> 1161 [ACK]	Seq=1	Ack=3486	Win=11680	Len=0	
14	TCP	80 -> 1161 [ACK]	Seq=1	Ack=4946	Win=14600	Len=0	
15	TCP	80 -> 1161 [ACK]	Seq=1	Ack=6466	Win=17520	Len=0	
16	TCP	80 -> 1161 [ACK]	Seq=1	Ack=7866	Win=20440	Len=0	
17	TCP	80 -> 1161 [ACK]	Seq=1	Ack=9013	Win=23360	Len=0	
24	TCP	80 -> 1161 [ACK]	Seq=1	Ack=10473	Win=26280	Len=0	
25	TCP	80 -> 1161 [ACK]	Seq=1	Ack=11933	Win=29200	Len=0	
26	TCP	80 -> 1161 [ACK]	Seq=1	Ack=13393	Win=32120	Len=0	
27	TCP	80 -> 1161 [ACK]	Seq=1	Ack=14853	Win=35040	Len=0	
28	TCP	80 -> 1161 [ACK]	Seq=1	Ack=16313	Win=37960	Len=0	
29	TCP	80 -> 1161 [ACK]	Seq=1	Ack=17205	Win=37960	Len=0	
36	TCP	80 -> 1161 [ACK]	Seq=1	Ack=18665	Win=40880	Len=0	
37	TCP	80 -> 1161 [ACK]	Seq=1	Ack=20125	Win=43800	Len=0	
38	TCP	80 -> 1161 [ACK]	Seq=1	Ack=21585	Win=46720	Len=0	
39	TCP	80 -> 1161 [ACK]	Seq=1	Ack=23045	Win=49640	Len=0	
40	TCP	80 -> 1161 [ACK]	Seq=1	Ack=24505	Win=52560	Len=0	
41	TCP	80 -> 1161 [ACK]	Seq=1	Ack=25397	Win=52560	Len=0	
48	TCP	80 -> 1161 [ACK]	Seq=1	Ack=26857	Win=55480	Len=0	
49	TCP	80 -> 1161 [ACK]	Seq=1	Ack=28317	Win=58400	Len=0	
50	TCP	80 -> 1161 [ACK]	Seq=1	Ack=29777	Win=61320	Len=0	
51	TCP	80 -> 1161 [ACK]	Seq=1	Ack=31237	Win=62780	Len=0	
52	TCP	80 -> 1161 [ACK]	Seq=1	Ack=33589	Win=62780	Len=0	
59	TCP	80 -> 1161 [ACK]	Seq=1	Ack=35049	Win=62780	Len=0	
60	TCP	80 -> 1161 [ACK]	Seq=1	Ack=37969	Win=62780	Len=0	
61	TCP	80 -> 1161 [ACK]	Seq=1	Ack=40889	Win=62780	Len=0	
62	TCP	80 -> 1161 [ACK]	Seq=1	Ack=41781	Win=62780	Len=0	
69	TCP	80 -> 1161 [ACK]	Seq=1	Ack=44701	Win=62780	Len=0	
70	TCP	80 -> 1161 [ACK]	Seq=1	Ack=47621	Win=62780	Len=0	
71	TCP	80 -> 1161 [ACK]	Seq=1	Ack=49973	Win=62780	Len=0	
78	TCP	80 -> 1161 [ACK]	Seq=1	Ack=52893	Win=62780	Len=0	
79	TCP	80 -> 1161 [ACK]	Seq=1	Ack=55813	Win=62780	Len=0	
80	TCP	80 -> 1161 [ACK]	Seq=1	Ack=58165	Win=62780	Len=0	
87	TCP	80 -> 1161 [ACK]	Seq=1	Ack=61085	Win=62780	Len=0	
88	TCP	80 -> 1161 [ACK]	Seq=1	Ack=64005	Win=62780	Len=0	
89	TCP	80 -> 1161 [ACK]	Seq=1	Ack=66357	Win=62780	Len=0	
96	TCP	80 -> 1161 [ACK]	Seq=1	Ack=69277	Win=62780	Len=0	
97	TCP	80 -> 1161 [ACK]	Seq=1	Ack=72197	Win=62780	Len=0	
98	TCP	80 -> 1161 [ACK]	Seq=1	Ack=74549	Win=62780	Len=0	
105	TCP	80 -> 1161 [ACK]	Seq=1	Ack=77469	Win=62780	Len=0	
106	TCP	80 -> 1161 [ACK]	Seq=1	Ack=80389	Win=62780	Len=0	
107	TCP	80 -> 1161 [ACK]	Seq=1	Ack=82741	Win=62780	Len=0	
114	TCP	80 -> 1161 [ACK]	Seq=1	Ack=85561	Win=62780	Len=0	
115	TCP	80 -> 1161 [ACK]	Seq=1	Ack=88581	Win=62780	Len=0	
116	TCP	80 -> 1161 [ACK]	Seq=1	Ack=90933	Win=62780	Len=0	
123	TCP	80 -> 1161 [ACK]	Seq=1	Ack=93853	Win=62780	Len=0	
124	TCP	80 -> 1161 [ACK]	Seq=1	Ack=96773	Win=62780	Len=0	
125	TCP	80 -> 1161 [ACK]	Seq=1	Ack=99125	Win=62780	Len=0	
132	TCP	80 -> 1161 [ACK]	Seq=1	Ack=102045	Win=62780	Len=0	
133	TCP	80 -> 1161 [ACK]	Seq=1	Ack=104965	Win=62780	Len=0	
134	TCP	80 -> 1161 [ACK]	Seq=1	Ack=107317	Win=62780	Len=0	
141	TCP	80 -> 1161 [ACK]	Seq=1	Ack=110237	Win=62780	Len=0	
142	TCP	80 -> 1161 [ACK]	Seq=1	Ack=113157	Win=62780	Len=0	
143	TCP	80 -> 1161 [ACK]	Seq=1	Ack=115509	Win=62780	Len=0	
150	TCP	80 -> 1161 [ACK]	Seq=1	Ack=118429	Win=62780	Len=0	
151	TCP	80 -> 1161 [ACK]	Seq=1	Ack=121349	Win=62780	Len=0	
152	TCP	80 -> 1161 [ACK]	Seq=1	Ack=123701	Win=62780	Len=0	
159	TCP	80 -> 1161 [ACK]	Seq=1	Ack=126621	Win=62780	Len=0	
160	TCP	80 -> 1161 [ACK]	Seq=1	Ack=129541	Win=62780	Len=0	
161	TCP	80 -> 1161 [ACK]	Seq=1	Ack=131893	Win=62780	Len=0	
168	TCP	80 -> 1161 [ACK]	Seq=1	Ack=134813	Win=62780	Len=0	
169	TCP	80 -> 1161 [ACK]	Seq=1	Ack=137733	Win=62780	Len=0	
170	TCP	80 -> 1161 [ACK]	Seq=1	Ack=140085	Win=62780	Len=0	
177	TCP	80 -> 1161 [ACK]	Seq=1	Ack=143005	Win=62780	Len=0	
178	TCP	80 -> 1161 [ACK]	Seq=1	Ack=145925	Win=62780	Len=0	
179	TCP	80 -> 1161 [ACK]	Seq=1	Ack=148277	Win=62780	Len=0	
186	TCP	80 -> 1161 [ACK]	Seq=1	Ack=151197	Win=62780	Len=0	
190	TCP	80 -> 1161 [ACK]	Seq=1	Ack=154117	Win=62780	Len=0	
191	TCP	80 -> 1161 [ACK]	Seq=1	Ack=156469	Win=62780	Len=0	
198	TCP	80 -> 1161 [ACK]	Seq=1	Ack=159389	Win=62780	Len=0	
200	TCP	80 -> 1161 [ACK]	Seq=1	Ack=162309	Win=62780	Len=0	
201	TCP	80 -> 1161 [ACK]	Seq=1	Ack=164041	Win=62780	Len=0	
202	TCP	80 -> 1161 [ACK]	Seq=1	Ack=164091	Win=62780	Len=0	
203	HTTP	HTTP/1.1 200 OK		(text/html)			

Answer to Q10

No, there are no re-transmissions. I checked this using a couple of built in tools to wire-shark. First, I checked the flow graph which basically shows the flow of packets between all the sources and destinations involved in the packet flow. It can show re-transmissions in the packet flow, which I did not see any trace of (pun intended)

Then I looked at the TCP time sequence stream graph (stevens) which showed that the sequence numbers from the source 192.168.1.102 to destination 128.119.245.12 increased monotonically, which is proof that all packets were, temporally, sent in order and so no re-transmission could have occurred.

0.078157	1161	1161 → 80 [ACK] Seq=6406 Ack=1 Win...	80
0.124085	1161	80 → 1161 [ACK] Seq=1 Ack=3486 Win...	80
0.124185	1161	1161 → 80 [PSH, ACK] Seq=7866 Ack=1...	80
0.169118	1161	80 → 1161 [ACK] Seq=1 Ack=4946 Win...	80
0.217299	1161	80 → 1161 [ACK] Seq=1 Ack=6406 Win...	80
0.267802	1161	80 → 1161 [ACK] Seq=1 Ack=7866 Win...	80
0.304807	1161	80 → 1161 [ACK] Seq=1 Ack=9013 Win...	80
0.305040	1161	1161 → 80 [ACK] Seq=9013 Ack=1 Win...	80
0.305813	1161	1161 → 80 [ACK] Seq=10473 Ack=1 Wi...	80
0.306692	1161	1161 → 80 [ACK] Seq=11933 Ack=1 Wi...	80
0.307571	1161	1161 → 80 [ACK] Seq=13393 Ack=1 Wi...	80
0.308699	1161	1161 → 80 [ACK] Seq=14853 Ack=1 Wi...	80
0.309553	1161	1161 → 80 [PSH, ACK] Seq=16313 Ack=...	80
0.356437	1161	80 → 1161 [ACK] Seq=1 Ack=10473 Wi...	80
0.400164	1161	80 → 1161 [ACK] Seq=1 Ack=11933 Wi...	80
0.448613	1161	80 → 1161 [ACK] Seq=1 Ack=13393 Wi...	80
0.500029	1161	80 → 1161 [ACK] Seq=1 Ack=14853 Wi...	80
0.545052	1161	80 → 1161 [ACK] Seq=1 Ack=16313 Wi...	80
...	...	80 → 1161 [ACK] Seq=1 Ack=17205 Wi...	...



Answer to Q11

To find out the data that the server is ACKing, we need to look at the ACK number of two consecutive ACKs. The difference between the two will give the amount of data acknowledged.

Looking at the packets, we find that typically the difference between the two packets is 1460 bytes. But at the start it is 565 bytes. In one of the packets it is 1147. Sometimes the ACK happens once for every two packets as there is an occasional data acknowledgement of 2920 bytes

46

6 TCP	80 → 1161 [ACK] Seq=1 Ack=566 Win=6780 Len=0
9 TCP	80 → 1161 [ACK] Seq=1 Ack=2026 Win=8760 Len=0
12 TCP	80 → 1161 [ACK] Seq=1 Ack=3486 Win=11680 Len=0
14 TCP	80 → 1161 [ACK] Seq=1 Ack=4946 Win=14600 Len=0
15 TCP	80 → 1161 [ACK] Seq=1 Ack=6406 Win=17520 Len=0
16 TCP	80 → 1161 [ACK] Seq=1 Ack=7866 Win=20440 Len=0
17 TCP	80 → 1161 [ACK] Seq=1 Ack=9013 Win=23360 Len=0
24 TCP	80 → 1161 [ACK] Seq=1 Ack=10473 Win=26280 Len=0
25 TCP	80 → 1161 [ACK] Seq=1 Ack=11933 Win=29200 Len=0
26 TCP	80 → 1161 [ACK] Seq=1 Ack=13393 Win=32120 Len=0
27 TCP	80 → 1161 [ACK] Seq=1 Ack=14853 Win=35040 Len=0
28 TCP	80 → 1161 [ACK] Seq=1 Ack=16313 Win=37960 Len=0
29 TCP	80 → 1161 [ACK] Seq=1 Ack=17205 Win=37960 Len=0
36 TCP	80 → 1161 [ACK] Seq=1 Ack=18665 Win=40880 Len=0
37 TCP	80 → 1161 [ACK] Seq=1 Ack=20125 Win=43800 Len=0
38 TCP	80 → 1161 [ACK] Seq=1 Ack=21585 Win=46720 Len=0
39 TCP	80 → 1161 [ACK] Seq=1 Ack=23045 Win=49640 Len=0
40 TCP	80 → 1161 [ACK] Seq=1 Ack=24505 Win=52560 Len=0
41 TCP	80 → 1161 [ACK] Seq=1 Ack=25397 Win=52560 Len=0
48 TCP	80 → 1161 [ACK] Seq=1 Ack=26857 Win=55480 Len=0
49 TCP	80 → 1161 [ACK] Seq=1 Ack=28317 Win=58400 Len=0
50 TCP	80 → 1161 [ACK] Seq=1 Ack=29777 Win=61320 Len=0
51 TCP	80 → 1161 [ACK] Seq=1 Ack=31237 Win=62780 Len=0
52 TCP	80 → 1161 [ACK] Seq=1 Ack=33589 Win=62780 Len=0
59 TCP	80 → 1161 [ACK] Seq=1 Ack=35049 Win=62780 Len=0
60 TCP	80 → 1161 [ACK] Seq=1 Ack=37969 Win=62780 Len=0
61 TCP	80 → 1161 [ACK] Seq=1 Ack=40889 Win=62780 Len=0
62 TCP	80 → 1161 [ACK] Seq=1 Ack=41781 Win=62780 Len=0
69 TCP	80 → 1161 [ACK] Seq=1 Ack=44701 Win=62780 Len=0
70 TCP	80 → 1161 [ACK] Seq=1 Ack=47621 Win=62780 Len=0

Answer to Q12

The throughput is calculated as the amount of data transmitted per unit time. To calculate throughput I calculated the amount of bytes sent by looking at the difference between the ACK of the last packet acknowledged and ACK of the first packet acknowledged and the time difference between the two.

No of bytes sent is $164091 - 1 = 164090$ bytes

Time taken = $5.455830 - 0.023172 = 5.432658$ sec

Throughput = $164090 / 5.432658 = 30204.37$ bytes/sec = 241634.94 bits/sec

6 TCP	80 → 1161 [ACK] Seq=1 Ack=566 Win=6780 Len=0	0.053937
202 TCP	80 → 1161 [ACK] Seq=1 Ack=164091 Win=62780 Len=0	5.455830

8 WIRESHARK LAB 5 : UDP (OPTIONAL)

I used one of the DNS packets as DNS works on UDP. Here are my findings for the same.

Answer to Q1

There are 4 fields in the UDP header. They are Source Port, Destination Port, Length and Checksum

```
▼ User Datagram Protocol, Src Port: 53, Dst Port: 51804
  Source Port: 53
  Destination Port: 51804
  Length: 69
  Checksum: 0xf578 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 54]
```

Answer to Q2

By seeing the hex dump of the UDP packet, we find that all of these four fields is 16 bits each.

```
▼ User Datagram Protocol, Src Port: 53, Dst Port: 51804
  Source Port: 53
  Destination Port: 51804
  Length: 69
  Checksum: 0xf578 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 54]
▶ Domain Name System (response)

0000 a0 af bd 8d 11 de c4 b8 b4 a9 93 af 08 00 45 00
0010 00 59 91 21 40 00 40 11 26 1a c0 a8 01 01 c0 a8
0020 01 07 00 35 ca 5c 00 45 f5 78 94 b1 81 80 00 01
0030 00 00 00 00 01 05 74 69 6c 65 73 05 72 35 33
0040 2d 32 08 73 65 72 76 69 63 65 73 07 6d 6f 7a 69
0050 6c 6c 61 03 63 6f 6d 00 00 1c 00 01 00 00 29 10
0060 00 00 00 00 00 00 00
```

Answer to Q3

Length field of the UDP header gives the length of the UDP datagram (header+data). This is verified by the hex dump of the data, which is exactly 69 bytes long and so the same length as told by the length field.

Answer to Q4

As the length field is 16 bit long, there are theoretically $2^{16}-1 = 65535$ bytes long data allowed. Out of this 8 bytes will be for the header. So 65527 bytes are allowed for the data. However, this theoretical limit is not right as the IP also has a header that has to be incorporated in the 65535 bytes. As header of IP is 20 bytes long, therefore, the total length allowed for the data is 65507 bytes.

Answer to Q5

Again the source port number is specified by a 16 bit field in the UDP header. So this can go from 0 to 65535. Therefore, the largest port number allowed is 65535.

Answer to Q6

The protocol number for UDP is 17 in decimal and 11 in hexadecimal.

```
Protocol: UDP (17)
Header checksum: 0x261a |
[Header checksum status:
Source: 192.168.1.1
10 a0 af bd 8d 11 de c4 b8
0 00 59 91 21 40 00 40 11]
```

Answer to Q7

The source port number of the DNS query is the destination port number in the query response and vice versa.

```
▼ User Datagram Protocol, Src Port: 51804, Dst Port: 53
  Source Port: 51804
  Destination Port: 53
  Length: 69
  Checksum: 0x7607 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 54]
▶ Domain Name System (query)

▼ User Datagram Protocol, Src Port: 53, Dst Port: 51804
  Source Port: 53
  Destination Port: 51804
  Length: 69
  Checksum: 0xf578 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 54]
▶ Domain Name System (response)
```

9 WIRESHARK LAB 6 : IP

Answer to Q1

The source IP address is 192.168.1.7

```
▼ Internet Control Message Protocol
  Type: 11 (Time-to-live exceeded)
  Code: 0 (Time to live exceeded in transit)
  Checksum: 0x2c69 [correct]
  [Checksum Status: Good]
  ▼ Internet Protocol Version 4, Src: 192.168.1.7, Dst: 128.119.245.12
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x5bb3 (23475)
    ▶ Flags: 0x0000
    ▶ Time to live: 1
    Protocol: UDP (17)
    Header checksum: 0x26cf [validation disabled]
```

Answer to Q2

The protocol field in IP header is ICMP (1)

```
▼ Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.7
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
  Total Length: 84
  Identification: 0xa078 (41080)
  ▶ Flags: 0x0000
  Time to live: 64
  Protocol: ICMP (1)
  Header checksum: 0x5618 [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.1.1
  Destination: 192.168.1.7
```

Answer to Q3

Total length of the IP datagram is 84 bytes. Header length is 20 bytes. Therefore, payload length is $84 - 20 = 64$ bytes.

```
▼ Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.7
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
  Total Length: 84
```

Answer to Q4

No the packet is not fragmented as the MF (More Fragments) Flag is off. And as it is the first packet, it can't be part of any previous fragment.

```
▼ Flags: 0x0000
  0... .... .... = Reserved bit: Not set
  .0... .... .... = Don't fragment: Not set
  ..0. .... .... = More fragments: Not set
  ...0 0000 0000 0000 = Fragment offset: 0
Time to live: 64
Protocol: ICMP (1)
```

At this point, as there were no echo request messages (using traceroute on ubuntu), I switched to working on the trace provided.

Answer to Q5

Identification, Time to live and Header Checksum fields always change.

8	ICMP	Echo (ping) request	id=0x0300, seq=20483/848, ttl=1 (no response found!)	192.168.1.102	128.59.23.100	6.163045
10	ICMP	Echo (ping) request	id=0x0300, seq=20739/849, ttl=2 (no response found!)	192.168.1.102	128.59.23.100	6.188629
12	ICMP	Echo (ping) request	id=0x0300, seq=20995/850, ttl=3 (no response found!)	192.168.1.102	128.59.23.100	6.208597
14	ICMP	Echo (ping) request	id=0x0300, seq=21251/851, ttl=4 (no response found!)	192.168.1.102	128.59.23.100	6.238695
16	ICMP	Echo (ping) request	id=0x0300, seq=21507/852, ttl=5 (no response found!)	192.168.1.102	128.59.23.100	6.258750
18	ICMP	Echo (ping) request	id=0x0300, seq=21763/853, ttl=6 (no response found!)	192.168.1.102	128.59.23.100	6.288750
20	ICMP	Echo (ping) request	id=0x0300, seq=22019/854, ttl=7 (no response found!)	192.168.1.102	128.59.23.100	6.308748
22	ICMP	Echo (ping) request	id=0x0300, seq=22275/855, ttl=8 (no response found!)	192.168.1.102	128.59.23.100	6.338804
23	ICMP	Echo (ping) request	id=0x0300, seq=22531/856, ttl=9 (no response found!)	192.168.1.102	128.59.23.100	6.358888
28	ICMP	Echo (ping) request	id=0x0300, seq=22787/857, ttl=10 (no response found!)	192.168.1.102	128.59.23.100	6.388921
29	ICMP	Echo (ping) request	id=0x0300, seq=23043/858, ttl=11 (no response found!)	192.168.1.102	128.59.23.100	6.408895
32	ICMP	Echo (ping) request	id=0x0300, seq=23299/859, ttl=12 (no response found!)	192.168.1.102	128.59.23.100	6.439037
33	ICMP	Echo (ping) request	id=0x0300, seq=23555/860, ttl=13 (reply in 35)	192.168.1.102	128.59.23.100	6.465882
39	ICMP	Echo (ping) request	id=0x0300, seq=23811/861, ttl=1 (no response found!)	192.168.1.102	128.59.23.100	11.159759
41	ICMP	Echo (ping) request	id=0x0300, seq=24067/862, ttl=2 (no response found!)	192.168.1.102	128.59.23.100	11.185772
43	ICMP	Echo (ping) request	id=0x0300, seq=24323/863, ttl=3 (no response found!)	192.168.1.102	128.59.23.100	11.205764
45	ICMP	Echo (ping) request	id=0x0300, seq=24579/864, ttl=4 (no response found!)	192.168.1.102	128.59.23.100	11.235816
46	ICMP	Echo (ping) request	id=0x0300, seq=24835/865, ttl=5 (no response found!)	192.168.1.102	128.59.23.100	11.255838
49	TCPMD	Echo (ping) request	id=0x0300, seq=25001/866, ttl=6 (no response found!)	192.168.1.102	128.59.23.100	11.285045

Answer to Q6

The fields that stay constant across the IP datagrams are:

1. Version (since we are using IPv4 for all packets)
2. Header length (since these are ICMP packets)
3. Source IP (since we are sending from the same source)
4. Destination IP (since we are sending to the same dest)
5. Differentiated Services (since all packets are ICMP they use the same type of Service class)

6. Upper Layer Protocol (since these are ICMP packets)

All of these are supposed to remain constant because of the respective reasons mentioned above.

On the other hand, Identification, Time to live and Header Checksum fields must change. This is because the packet numbers for identification must be unique, TTL is incremented in subsequent transmissions by the traceroute program and as header changes, so must the checksum.

Answer to Q7

They are incremented by one every time. They start with 0x32d0 in this trace and go on increasing by 1 thereafter.

Answer to Q8

The TTL is 255 and Identification is 0x9d7c.

```
▼ Internet Protocol Version 4, Src: 10.216.228.1, Dst: 192.168.1.102
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▼ Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    1100 00.. = Differentiated Services Codepoint: Class Selector 6 (48)
    .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 56
  Identification: 0x9d7c (40316)
  ▼ Flags: 0x0000
    0... .... .... = Reserved bit: Not set
    .0... .... .... = Don't fragment: Not set
    ..0. .... .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0
  Time to live: 255
```

Answer to Q9

The Identification field keeps on changing as it has to be unique to identify a packet. The TTL remains constant for the first hop router (and has to as each packet from the router will take equal time to traverse back to the source and so each will have same TTL).

Answer to Q10

Yes, this packet is fragmented

Answer to Q11

We know this packet is fragmented as the MF is 1 and Fragment Offset is 0. So it must be the first packet that's fragmented. The packet length is 1500 (including the headers, data -> 1460)

```
| 92 IPv4  Fragmented IP protocol (proto=ICMP 1, off=0, ID=32f9) [Reassembled in #93]
| 
▶ Frame 92: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
▶ Ethernet II, Src: Actionte_8a:70:1a (00:20:e0:8a:70:1a), Dst: LinksysG_da:af:73 (00:06:25:da:af:73)
▼ Internet Protocol Version 4, Src: 192.168.1.102, Dst: 128.59.23.100
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1500
  Identification: 0x32f9 (13049)
  ▼ Flags: 0x2000, More Fragments
    0.... .... .... = Reserved bit: Not set
    .0.... .... .... = Don't fragment: Not set
    ..1.... .... .... = More fragments: Set
    ...0 0000 0000 0000 = Fragment offset: 0
  ▶ Time to live: 1
  Protocol: ICMP (1)
  Header checksum: 0x077b [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.1.102
  Destination: 128.59.23.100
  Reassembled IPv4 in frame: 93
```

Answer to Q12

No, there are no further fragments. This is ascertained as the MF field is 0. This is an intermediate fragment and not the first one as the fragment offset is non zero.

```
| 93 ICMP  Echo (ping) request  id=0x0300, seq=30467/887, ttl=1 (no response found!)  192.168.1.102
| 
Frame 93: 562 bytes on wire (4496 bits), 562 bytes captured (4496 bits)
Ethernet II, Src: Actionte_8a:70:1a (00:20:e0:8a:70:1a), Dst: LinksysG_da:af:73 (00:06:25:da:af:73)
Internet Protocol Version 4, Src: 192.168.1.102, Dst: 128.59.23.100
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 548
  Identification: 0x32f9 (13049)
  ▼ Flags: 0x00b9
    0.... .... .... = Reserved bit: Not set
    .0.... .... .... = Don't fragment: Not set
    ..0.... .... .... = More fragments: Not set
    ...0 0000 1011 1001 = Fragment offset: 185
  ▶ Time to live: 1
  Protocol: ICMP (1)
  Header checksum: 0x2a7a [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.1.102
  Destination: 128.59.23.100
  ▶ [2 IPv4 Fragments (2008 bytes): #92(1480), #93(528)]
```

Answer to Q13

The IP header fields that changed between the fragments are: total length, flags, fragment offset, and checksum

Answer to Q14

When we talk of the packet with 3500 byte data, it takes three fragments to send the complete data.

Answer to Q15

The IP header fields that changed between all of the packets are: fragment offset, and checksum. Between the first two packets and the last packet, we see a change in total length, and also in the flags. The first two packets have a total length of 1500, with the more fragments bit set to 1, and the last packet has a total length of 540, with the more fragments bit set to 0

10 WIRESHARK LAB 7 : ICMP

Since I am running Ubuntu, I ran the ping command with c flag as 10 as the equivalent of the command given and pinged the server at mit.edu.

Answer to Q1

The IP address of my host is 192.168.1.7 and the IP address of destination is 184.26.196.231.

Protocol	Info	Source	Destination
48 ICMP	Echo (ping) request id=0x1212, seq=1/256, ttl=64 (reply in 49)	192.168.1.7	184.26.196.231
49 ICMP	Echo (ping) reply id=0x1212, seq=1/256, ttl=59 (request in 48)	184.26.196.231	192.168.1.7
59 ICMP	Echo (ping) request id=0x1212, seq=2/512, ttl=64 (reply in 60)	192.168.1.7	184.26.196.231
60 ICMP	Echo (ping) reply id=0x1212, seq=2/512, ttl=59 (request in 59)	184.26.196.231	192.168.1.7
64 ICMP	Echo (ping) request id=0x1212, seq=3/768, ttl=64 (reply in 65)	192.168.1.7	184.26.196.231
65 ICMP	Echo (ping) reply id=0x1212, seq=3/768, ttl=59 (request in 64)	184.26.196.231	192.168.1.7
68 ICMP	Echo (ping) request id=0x1212, seq=4/1024, ttl=64 (reply in 71)	192.168.1.7	184.26.196.231
71 ICMP	Echo (ping) reply id=0x1212, seq=4/1024, ttl=59 (request in 68)	184.26.196.231	192.168.1.7
72 ICMP	Echo (ping) request id=0x1212, seq=5/1280, ttl=64 (reply in 73)	192.168.1.7	184.26.196.231
73 ICMP	Echo (ping) reply id=0x1212, seq=5/1280, ttl=59 (request in 72)	184.26.196.231	192.168.1.7
74 ICMP	Echo (ping) request id=0x1212, seq=6/1536, ttl=64 (reply in 75)	192.168.1.7	184.26.196.231
75 ICMP	Echo (ping) reply id=0x1212, seq=6/1536, ttl=59 (request in 74)	184.26.196.231	192.168.1.7
78 ICMP	Echo (ping) request id=0x1212, seq=7/1792, ttl=64 (reply in 79)	192.168.1.7	184.26.196.231
79 ICMP	Echo (ping) reply id=0x1212, seq=7/1792, ttl=59 (request in 78)	184.26.196.231	192.168.1.7
84 ICMP	Echo (ping) request id=0x1212, seq=8/2048, ttl=64 (reply in 85)	192.168.1.7	184.26.196.231
85 ICMP	Echo (ping) reply id=0x1212, seq=8/2048, ttl=59 (request in 84)	184.26.196.231	192.168.1.7
86 ICMP	Echo (ping) request id=0x1212, seq=9/2304, ttl=64 (reply in 87)	192.168.1.7	184.26.196.231
87 ICMP	Echo (ping) reply id=0x1212, seq=9/2304, ttl=59 (request in 86)	184.26.196.231	192.168.1.7
88 ICMP	Echo (ping) request id=0x1212, seq=10/2560, ttl=64 (reply in 89)	192.168.1.7	184.26.196.231
89 ICMP	Echo (ping) reply id=0x1212, seq=10/2560, ttl=59 (request in 88)	184.26.196.231	192.168.1.7

Answer to Q2

Even though ICMP is carried as payload in a datagram, it is very much meant to be a network level protocol. It was designed to communicate network-layer information between hosts and routers, not between application layer processes. And since it does not deal with the application layer, it does not have any port numbers that are required in transport layer protocols like TCP and UDP. The ICMP messages are implemented and handled at the software level and it is able to do so only using the type/code pair in ICMP, it does not require any port numbers.

Answer to Q3

The ICMP type is 8, and the code number is 0. The ICMP packet also has checksum, identifier, sequence number, and data fields. The checksum, sequence number and identifier fields are two bytes each

```
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x1b50 [correct]
  [Checksum Status: Good]
  Identifier (BE): 4626 (0x1212)
  Identifier (LE): 4626 (0x1212)
  Sequence number (BE): 1 (0x0001)
  Sequence number (LE): 256 (0x0100)
  [Response frame: 49]
  Timestamp from icmp data: Apr 21, 2019 00:45:19.000000000 IST
  [Timestamp from icmp data (relative): 0.392616267 seconds]
```

Answer to Q4

The ICMP type is 0, and the code number is 0. The ICMP packet also has checksum, identifier, sequence number, and data fields. The checksum, sequence number and identifier fields are two bytes each.

```
▼ Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0x2350 [correct]
  [Checksum Status: Good]
  Identifier (BE): 4626 (0x1212)
  Identifier (LE): 4626 (0x1212)
  Sequence number (BE): 1 (0x0001)
  Sequence number (LE): 256 (0x0100)
  [Request frame: 48]
  [Response time: 82.700 ms]
  Timestamp from icmp data: Apr 21, 2019 00:45:19.000000000 IST
  [Timestamp from icmp data (relative): 0.475315806 seconds]
  ▶ Data (48 bytes)
```

The implementation of Traceroute program is very different in Linux than in Windows so I have used the trace provided for the remainder of the practical.

Answer to Q5

The IP address of my host is 192.168.1.101. The IP address of the destination host is 138.96.146.2.

```
Source: 192.168.1.101
Destination: 138.96.146.2
```

Answer to Q6

No. If ICMP sent UDP packets instead, the IP protocol number should be 0x11 (because UDP has protocol number as 17).

Answer to Q7

The only difference between the two is that in this header no response is seen while in the ping packet in the first half of the experiment, response is received typically in the very next datagram.

```
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x51fe [correct]
  [Checksum Status: Good]
  Identifier (BE): 512 (0x0200)
  Identifier (LE): 2 (0x0002)
  Sequence number (BE): 41985 (0xa401)
  Sequence number (LE): 420 (0x01a4)
▶ [No response seen]
```

Answer to Q8

The ICMP error message has more fields than the original. It has a copy of the datagram of the original message as well, so it's IP header as well as the echo's ICMP header.

```
▶ Internet Protocol Version 4, Src: 10.216.228.1, Dst: 192.168.1.101
▼ Internet Control Message Protocol
  Type: 11 (Time-to-live exceeded)
  Code: 0 (Time to live exceeded in transit)
  Checksum: 0x2c16 [correct]
  [Checksum Status: Good]
  ▶ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 138.96.146.2
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 92
    Identification: 0xd2d5 (53973)
    Flags: 0x0000
    ▶ Time to live: 1
      Protocol: ICMP (1)
      Header checksum: 0xd145 [validation disabled]
      [Header checksum status: Unverified]
      Source: 192.168.1.101
      Destination: 138.96.146.2
    ▼ Internet Control Message Protocol
      Type: 8 (Echo (ping) request)
      Code: 0
      Checksum: 0x51fe [unverified] [in ICMP error packet]
      [Checksum Status: Unverified]
      Identifier (BE): 512 (0x0200)
      Identifier (LE): 2 (0x0002)
      Sequence number (BE): 41985 (0xa401)
      Sequence number (LE): 420 (0x01a4)
```

Answer to Q9

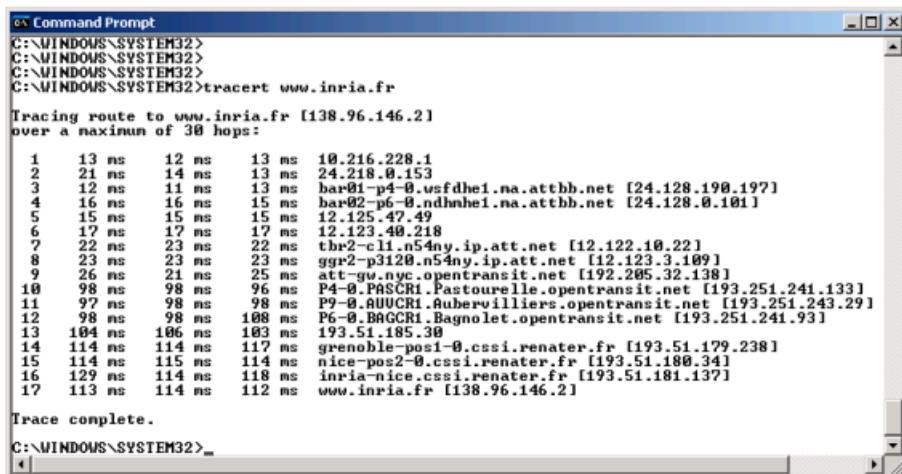
The last three ICMP packets received by the host are different as their code is 0x00

(echo reply). This is so because the last three requests reach the destination host before TTL happens.

```
▼ Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0x29fe [correct]
  [Checksum Status: Good]
  Identifier (BE): 512 (0x0200)
  Identifier (LE): 2 (0x0002)
  Sequence number (BE): 54273 (0xd401)
  Sequence number (LE): 468 (0x01d4)
  [Request frame: 97]
  [Response time: 113.456 ms]
```

Answer to Q10

Because I have used the trace provided, I am using the accompanying Picture 4 provided in the lab files. According to this the link between steps 9 and 10 has a significantly longer delay. Based on the screenshot, the link is from New York to Pastourelle, France.



```
C:\> Command Prompt
C:\> tracert www.inria.fr
Tracing route to www.inria.fr [138.96.146.2]
over a maximum of 30 hops:
 1  13 ms   12 ms   13 ms  10.216.228.1
 2  21 ms   14 ms   13 ms  24.218.0.153
 3  12 ms   11 ms   13 ms  bar01-p4-0.wsf.dhei.na.attbb.net [24.128.190.197]
 4  16 ms   16 ms   15 ms  bar02-p6-0.ndhuhei.na.attbb.net [24.128.0.101]
 5  15 ms   15 ms   15 ms  12.125.47.49
 6  17 ms   17 ms   17 ms  12.123.40.218
 7  22 ms   23 ms   22 ms  tbr2-c11.n54ny.ip.att.net [12.122.10.22]
 8  23 ms   23 ms   23 ms  ggr2-p3120.n54ny.ip.att.net [12.123.3.109]
 9  26 ms   21 ms   25 ms  att-gw.nyc.opentransit.net [192.205.32.138]
10  98 ms   98 ms   96 ms  P4-0.PASCR1.Pastourelle.opentransit.net [193.251.241.133]
11  97 ms   98 ms   98 ms  P9-0.AUVCRI.Aubervilliers.opentransit.net [193.251.243.29]
12  98 ms   98 ms  108 ms  P6-0.BAGCR1.Bagnolet.opentransit.net [193.251.241.93]
13  104 ms  106 ms  103 ms  193.51.185.30
14  114 ms  114 ms  117 ms  grenoble-pos1-0.cssi.renater.fr [193.51.179.238]
15  114 ms  115 ms  114 ms  nice-pos2-0.cssi.renater.fr [193.51.180.34]
16  129 ms  114 ms  118 ms  inria-nice.cssi.renater.fr [193.51.181.137]
17  113 ms  114 ms  112 ms  www.inria.fr [138.96.146.2]

Trace complete.
```

Figure 4 Command Prompt window displays the results of the Traceroute program.

11 WIRESHARK LAB 8 : ETHERNET AND ARP

Answer to Q1

The 48 bit physical address of my machine is a0:af:bd:8d:11:de.

```
▼ Ethernet II, Src: IntelCor_8d:11:de (a0:af:bd:8d:11:de), Dst: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
  ▶ Destination: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
  ▶ Source: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    Address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
```

Answer to Q2

The 48 bit physical address of destination is c4:b8:b4:a9:93:af. No, it is not the physical address of the gaia server. Rather, it is the physical address of my router which is the NAT enabled link used to get off the subnet.

```
▼ Ethernet II, Src: IntelCor_8d:11:de (a0:af:bd:8d:11:de), Dst: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
  ▶ Destination: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
  ▶ Source: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    Address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
```

Answer to Q3

The two byte frame type value is 0x0800, which corresponds to IPv4.

```
▼ Ethernet II, Src: IntelCor_8d:11:de (a0:af:bd:8d:11:de), Dst: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
  ▶ Destination: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
  ▶ Source: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    Address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
```

Answer to Q4

The ASCII "G" appears 66 bytes from the start of the Ethernet frame. There are 14 bytes of Ethernet frame, and then 20 bytes of IP header followed by 32 bytes of TCP header before the HTTP data is encountered.

```

▼ Internet Protocol Version 4, Src: 192.168.1.7, Dst: 128.119.245.12
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 420
  Identification: 0xbe93 (48787)
▶ Flags: 0x4000, Don't fragment
  Time to live: 64
  Protocol: TCP (6)
  Header checksum: 0x438d [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.1.7
  Destination: 128.119.245.12
▼ Transmission Control Protocol, Src Port: 51052, Dst Port: 80, Seq: 1, Ack: 1, Len: 368
  Source Port: 51052
  Destination Port: 80
  [Stream index: 4]
  [TCP Segment Len: 368]
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 369 (relative sequence number)]
  Acknowledgment number: 1 (relative ack number)
  1000 .... = Header Length: 32 bytes (8)
  ▶ Flags: 0x018 (PSH, ACK)
  Window size: 32000
0000 c4 b8 b4 a9 93 af a0 af bd 8d 11 de 08 00 45 00 ..... .E.
0010 01 a4 be 93 40 00 40 06 43 8d c0 a8 01 07 80 77 ..... @ @ C.....w
0020 f5 0c c7 6c 00 50 d6 8b c9 39 d0 d6 c1 8c 80 18 ...1.P...9.....
0030 00 e5 ba f2 00 00 01 01 08 0a 68 2c be 47 75 55 ..... h,GuU
0040 ad da 47 45 54 20 2f 77 69 72 65 73 68 61 72 6b ..GET /w ireshark
0050 2d 6c 61 62 73 2f 48 54 54 50 2d 65 74 68 65 72 -labs/HT TP-ether
0060 65 61 6c 2d 6c 61 62 2d 66 69 6c 65 33 2e 68 74 eal-lab- file3.ht
0070 6d 6c 20 48 54 54 50 2f 31 2e 31 0d 0a 48 6f 73 m1 HTTP/ 1.1..Hos
0080 74 3a 20 67 61 69 61 2e 63 73 2e 75 6d 61 73 73 t: gaia. cs.umass
0090 2e 65 64 75 0d 0a 55 73 65 72 2d 41 67 65 6e 74 .edu..Us er-Agent
00a0 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 58 : Mozilla/5.0 (X
00b0 31 31 3b 20 55 62 75 6e 74 75 3b 20 4c 69 6e 75 11; Ubuntu; Linu

```

Answer to Q5

The 48 bit physical address of source is c4:b8:b4:a9:93:af. No, it is not the physical address of the gaia server. Rather, it is the physical address of my router which is the NAT enabled link used to get off the subnet.

```

▼ Ethernet II, Src: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af), Dst: IntelCor_8d:11:de (a0:af:b
  ▶ Destination: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    Address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  ▶ Source: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    Address: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)

```

Answer to Q6

The 48 bit physical address of destination is a0:af:bd:8d:11:de. Yes, it is the MAC address of my machine.

```

▼ Ethernet II, Src: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af), Dst: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
  ▼ Destination: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    Address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  ▼ Source: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    Address: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)

```

Answer to Q7

The two byte frame type value is 0x0800, which corresponds to IPv4.

```

▼ Ethernet II, Src: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af), Dst: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
  ▼ Destination: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    Address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  ▼ Source: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    Address: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)

```

Answer to Q8

The ASCII of "O" from OK starts at byte number 14.

0000	48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 4b 0d	HTTP/1.1 200 OK
0010	0a 44 61 74 65 3a 20 53 61 74 2c 20 32 30 20 41	Date: Sat, 20 A
0020	70 72 20 32 30 31 39 20 32 30 3a 31 36 3a 33 35	pr 2019 20:16:35
0030	20 47 4d 54 0d 0a 53 65 72 76 65 72 3a 20 41 70	GMT..Se rver: Ap
0040	61 63 68 65 2f 32 2e 34 2e 36 20 28 43 65 6e 74	ache/2.4 .6 (Cent
0050	4f 53 29 20 4f 70 65 6e 53 53 4c 2f 31 2e 30 2e	OS) Open SSL/1.0.
0060	32 6b 2d 66 69 70 73 20 50 48 50 2f 35 2e 34 2e	2k-fips PHP/5.4.
0070	31 36 20 6d 6f 64 5f 70 65 72 6c 2f 32 2e 30 2e	16 mod_p erl/2.0.
0080	31 30 20 50 65 72 6c 2f 76 35 2e 31 36 2e 33 0d	10 Perl/ v5.16.3.
0090	0a 4c 61 73 74 2d 4d 6f 64 69 66 69 65 64 3a 20	Last-Mo dified:
00a0	53 61 74 2c 20 32 30 20 41 70 72 20 32 30 31 39	Sat, 20 Apr 2019
00b0	20 30 35 3a 35 39 3a 30 32 20 47 4d 54 0d 0a 45	05:59:0 2 GMT..E
00c0	54 61 67 3a 20 22 31 31 39 34 2d 35 38 36 65 66	Tag: "11 94-586ef
00d0	65 65 39 63 32 36 31 34 22 0d 0a 41 63 63 65 70	ee9c2614 "...Accep
00e0	74 2d 52 61 6e 67 65 73 3a 20 62 79 74 65 73 0d	t-Ranges : bytes.
00f0	0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 68 3a	Content -Length:
0100	20 34 35 30 30 0d 0a 4b 65 65 70 2d 41 6c 69 76	4500..K eep-Aliv
0110	65 3a 20 74 69 6d 65 6f 75 74 3d 35 2c 20 6d 61	e: timeo ut=5, ma
0120	78 3d 31 30 30 0d 0a 43 6f 6e 6e 65 63 74 69 6f	x=100..C onnectio
0130	6e 3a 20 4b 65 65 70 2d 41 6c 69 76 65 0d 0a 43	n: Keep- Alive..C
0140	6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74 65 78	ontent-T ype: tex
0150	74 2f 68 74 6d 6c 3b 20 63 68 61 72 73 65 74 3d	t/html; charset=
0160	55 54 46 2d 38 0d 0a 0d 0a 3c 68 74 6d 6c 3e 3c	UTF-8... <html><
0170	68 65 61 64 3e 20 0a 3c 74 69 74 6c 65 3e 48 69	head> < title>Hi
0180	73 74 6f 72 69 63 61 6c 20 44 6f 63 75 6d 65 6e	storical Documen
0190	74 73 3a 54 48 45 20 42 49 4c 4c 20 4f 46 20 52	ts:THE B ILL OF R
01a0	49 47 48 54 53 3c 2f 74 69 74 6c 65 3e 3c 2f 68	IIGHTS</t itle></h
01b0	65 61 64 3e 0a 0a 0a 3c 62 6f 64 79 20 62 67 63	ead>...< body bgc

Answer to Q9

The fields in the ARP query are:

1. Address (IP or hostname, depending on flags)
2. HWType (Underlying Hardware)
3. HWAddress (MAC Address)
4. Flags Mask (According to ARP man page, each complete entry in the ARP cache will be marked with the C flag. Permanent entries are marked with M and published entries have the P flag.)
5. Iface (Interface on which the machine is communicating)

```
arihant@l470:~$ arp
Address          HWtype  HWaddress           Flags Mask      Iface
_gateway         ether    c4:b8:b4:a9:93:af  C          wlp5s0
192.168.1.15    ether    bc:d1:1f:7b:0b:61  C          wlp5s0
192.168.1.6     ether    64:a2:f9:07:db:fd  C          wlp5s0
192.168.1.14    ether    64:a2:f9:7d:eb:93  C          wlp5s0
arihant@l470:~$ arp -n
Address          HWtype  HWaddress           Flags Mask      Iface
192.168.1.1     ether    c4:b8:b4:a9:93:af  C          wlp5s0
192.168.1.15    ether    bc:d1:1f:7b:0b:61  C          wlp5s0
192.168.1.6     ether    64:a2:f9:07:db:fd  C          wlp5s0
192.168.1.14    ether    64:a2:f9:7d:eb:93  C          wlp5s0
```

Answer to Q10

The 48 bit physical address of source is c4:b8:b4:a9:93:af and the 48 bit physical address of destination is a0:af:bd:8d:11:de.

```
▼ Ethernet II, Src: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af), Dst: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
  ▼ Destination: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    Address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  ▼ Source: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    Address: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
Type: ARP (0x0806)
```

Answer to Q11

The two byte frame type value is 0x0806, which corresponds to ARP.

```
▼ Ethernet II, Src: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af), Dst: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
  ▼ Destination: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    Address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  ▼ Source: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    Address: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  Type: ARP (0x0806)
```

Answer to Q12

Opcode: request (1)	
Sender MAC address: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)	
Sender IP address: 192.168.1.1	
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)	
Target IP address: 192.168.1.7	
0000	a0 af bd 8d 11 de c4 b8 b4 a9 93 af 08 06 00 01
0010	08 00 06 04 00 01 c4 b8 b4 a9 93 af c0 a8 01 01
0020	00 00 00 00 00 00 c0 a8 01 07

1. The ARP opcode field begins 20 bytes from the very beginning of the Ethernet frame.
2. Opcode value is 0x0001, which corresponds to request.
3. Yes, the ARP message contains the IP address of the sender. It is 192.168.1.1.
4. The field "Target MAC address" is set to 00:00:00:00:00:00 to question the machine whose corresponding IP address (192.168.1.7) is being queried.

Answer to Q13

```
▼ Address Resolution Protocol (reply)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (2)
  Sender MAC address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
  Sender IP address: 192.168.1.7
  Target MAC address: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
  Target IP address: 192.168.1.1
```

1. The ARP opcode field begins 20 bytes from the very beginning of the Ethernet frame.
2. Opcode value is 0x0002, which corresponds to reply.
3. The field "Target MAC address" is set to a0:af:bd:8d:11:de corresponding to the IP address (192.168.1.7) as the "answer" to the question in the ARP request.

Answer to Q14

The 48 bit physical address of destination is c4:b8:b4:a9:93:af and the 48 bit physical address of source is a0:af:bd:8d:11:de.

```
▼ Ethernet II, Src: IntelCor_8d:11:de (a0:af:bd:8d:11:de), Dst: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
  ▼ Destination: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    Address: HuaweiTe_a9:93:af (c4:b8:b4:a9:93:af)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  ▼ Source: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    Address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    .... ..0. .... .... .... = LG bit: Globally unique address (factory default)
    .... ..0. .... .... .... = IG bit: Individual address (unicast)
  Type: ARP (0x0806)
```

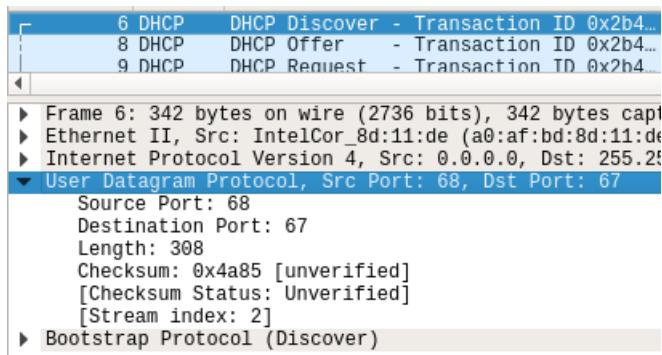
Answer to Q15

There is no reply in this trace, because we are not at the machine that sent the request. The ARP request is broadcast, but the ARP reply is sent back directly to the sender's Ethernet address.

12 WIRESHARK LAB 9 : DHCP

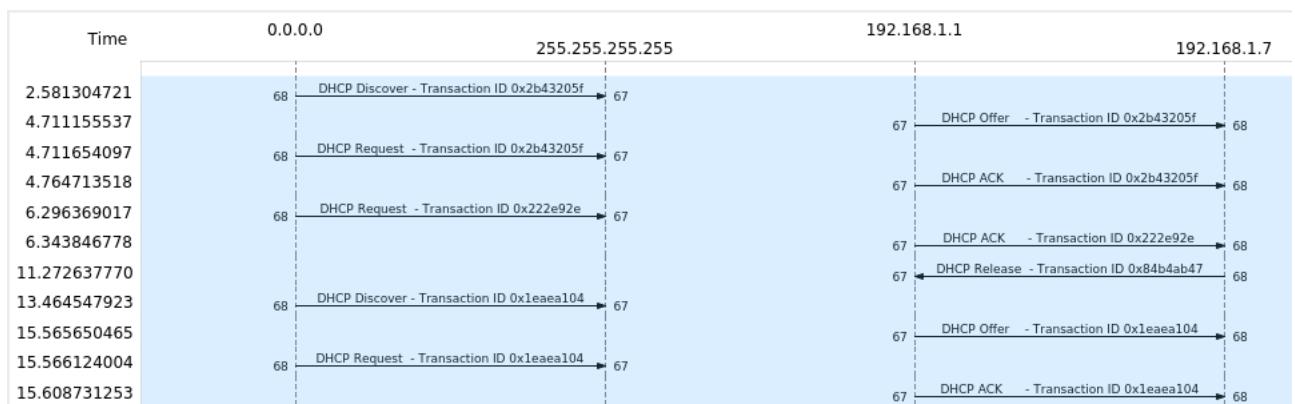
Answer to Q1

The DHCP messages are sent over UDP.



Answer to Q2

I used the flow graph feature of wireshark to draw the timing diagram. Yes, port numbers 67 and 68 and



Answer to Q3

The 48 bit physical address of my machine is a0:af:bd:8d:11:de.

```
▼ Ethernet II, Src: IntelCor_8d:11:de (a0:af:bd:8d:11:de), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▼ Source: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    Address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
    .... .0. .... .... .... = LG bit: Globally unique address (factory default)
    .... .0. .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
```

Answer to Q4

DHCP message type is different namely Discover and Request, and the Request message has a server identifier option which wasn't present in the Discover message.

Answer to Q5

The transaction ID of the first set of messages is 0x2b43205f and for the second set is 0x84b4ab47. The purpose of the transaction ID is so that the host can differentiate between different sets of DHCP requests.

Transaction ID: 0x2b43205f

Transaction ID: 0x84b4ab47

Answer to Q6

According to the theory of DHCP, the answer to this question should have been.

1. Discover: 0.0.0.0/255.255.255.255
2. Offer: 192.168.1.1/255.255.255.255
3. Request: 0.0.0.0/255.255.255.255
4. ACK: 192.168.1.1/255.255.255.255

However, in my trace I found very different and frankly, disturbing results.

1. Discover: 0.0.0.0/255.255.255.255
2. Offer: 192.168.1.1/192.168.1.7
3. Request: 0.0.0.0/255.255.255.255
4. ACK: 192.168.1.1/192.168.1.7

6	DHCP	DHCP Discover	- Transaction ID 0x2b4...	0.0.0.0	255.255.255.255
8	DHCP	DHCP Offer	- Transaction ID 0x2b4...	192.168.1.1	192.168.1.7
9	DHCP	DHCP Request	- Transaction ID 0x2b4...	0.0.0.0	255.255.255.255
10	DHCP	DHCP ACK	- Transaction ID 0x2b4...	192.168.1.1	192.168.1.7

This somehow meant that my DHCP was addressing my machine using the IP it has to allot it before it was allotted to me. This remains a mystery to me, so I looked at the trace provided and it gave the results that correspond to the original conjecture.

1. Discover: 0.0.0.0/255.255.255.255
2. Offer: 192.168.1.1/255.255.255.255
3. Request: 0.0.0.0/255.255.255.255
4. ACK: 192.168.1.1/255.255.255.255

DHCP Discover	- Transaction ID 0x3...	0.0.0.0	255.255.255.255
DHCP Offer	- Transaction ID 0x3...	192.168.1.1	255.255.255.255
DHCP Request	- Transaction ID 0x3...	0.0.0.0	255.255.255.255
DHCP ACK	- Transaction ID 0x3...	192.168.1.1	255.255.255.255

Answer to Q7

The IP address of my DHCP server is 192.168.1.11

```
    ▼ Internet Protocol Version 4, Src: 192.168.1.1, Dst: 255.255.255.255
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 576
      Identification: 0x0108 (264)
    ▶ Flags: 0x0000
      Time to live: 150
      Protocol: UDP (17)
      Header checksum: 0x5ffc [validation disabled]
      [Header checksum status: Unverified]
      Source: 192.168.1.1
      Destination: 255.255.255.255
```

Answer to Q8

The IP is offered in the DHCP offer message. The IP offered is in the Your IP field which shows 192.168.1.1

Answer to Q9

The field in the packet that tells about the relay agent and the non existence thereof is the relay IP. In the given screenshot, just like my trace has relay IP of 0.0.0.0

```
▼ Bootstrap Protocol (Offer)
  Message type: Boot Reply (2)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x2b43205f
  Seconds elapsed: 0
  ▶ Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0
  Your (client) IP address: 192.168.1.7
  Next server IP address: 0.0.0.0
  Relay agent IP address: 0.0.0.0
  Client MAC address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
  Client hardware address padding: 00000000000000000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
  ▶ Option: (53) DHCP Message Type (Offer)
  ▶ Option: (54) DHCP Server Identifier
  ▶ Option: (51) IP Address Lease Time
  ▶ Option: (1) Subnet Mask
  ▶ Option: (3) Router
  ▶ Option: (6) Domain Name Server
  ▶ Option: (255) End
  Padding: 00000000000000000000000000000000000000000000000000000000000000...
```

Answer to Q10

The subnet mask line tells the client which subnet mask to use.

The router line indicates where the client should send messages by default.

Answer to Q11

The client accepts the IP address given in the offer message within the request message. After being offered the IP address 192.168.1.7 in the offer message, my client sent back the Request message further requesting that specific IP address.

```

▼ Bootstrap Protocol (Request)
  Message type: Boot Request (1)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x2b43205f
  Seconds elapsed: 0
  ▶ Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0
  Your (client) IP address: 0.0.0.0
  Next server IP address: 0.0.0.0
  Relay agent IP address: 0.0.0.0
  Client MAC address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
  Client hardware address padding: 00000000000000000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
  ▶ Option: (53) DHCP Message Type (Request)
  ▶ Option: (54) DHCP Server Identifier
  ▶ Option: (50) Requested IP Address
    Length: 4
    Requested IP Address: 192.168.1.7
  ▶ Option: (12) Host Name
  ▶ Option: (55) Parameter Request List
  ▶ Option: (255) End
  Padding: 000000000000000000000000000000000000000000000000000000000000000

```

Answer to Q12

The purpose of lease time is to tell the client how long they can use the specific IP address assigned by the server before they will have to be assigned a new one.

The lease time in my experiment is 1 day.

```

▼ Bootstrap Protocol (ACK)
  Message type: Boot Reply (2)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x2b43205f
  Seconds elapsed: 0
  ▶ Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0
  Your (client) IP address: 192.168.1.7
  Next server IP address: 0.0.0.0
  Relay agent IP address: 0.0.0.0
  Client MAC address: IntelCor_8d:11:de (a0:af:bd:8d:11:de)
  Client hardware address padding: 00000000000000000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
  ▶ Option: (53) DHCP Message Type (ACK)
  ▶ Option: (54) DHCP Server Identifier
  ▶ Option: (51) IP Address Lease Time
    Length: 4
    IP Address Lease Time: (86400s) 1 day
  ▶ Option: (1) Subnet Mask
  ▶ Option: (3) Router
  ▶ Option: (6) Domain Name Server
  ▶ Option: (255) End
  Padding: 000000000000000000000000000000000000000000000000000000000000000...

```

Answer to Q13

The purpose of the release message is to release the IP address back to the server.

There is no verification that the release message has been received by the server.

If the message is lost, the client releases the IP address, but the server will not reassign that address until the clients lease on the address expires. (in this experiment, for exam-

ple, 1 day).

Answer to Q14

Yes, since the machine has basically rejoined the network new, these ARP messages appear to be broadcasts sent out by the network to build up the known IP addresses.

No.	Protocol	Info	Source	Destination	Time	Length
7	ARP	Who has 192.168.1.7? Tell 192.168.1.1	HuaweiTe_a9...	Broadcast	2.660490043	
11	ARP	Who has 192.168.1.1? Tell 192.168.1.7	IntelCor_8d...	Broadcast	4.784387650	
12	ARP	192.168.1.1 is at c4:b8:b4:a9:93:af	HuaweiTe_a9...	IntelCor_8d:11:...	4.786721999	
38	ARP	Who has 192.168.1.7? Tell 192.168.1.1	HuaweiTe_a9...	IntelCor_8d:11:...	5.853619526	
39	ARP	192.168.1.7 is at a0:af:bd:8d:11:de	IntelCor_8d...	HuaweiTe_a9:93:...	5.853653012	
97	ARP	Who has 192.168.1.7? Tell 192.168.1.1	HuaweiTe_a9...	Broadcast	13.5143736...	
101	ARP	Who has 192.168.1.1? Tell 192.168.1.7	IntelCor_8d...	Broadcast	15.6242338...	
102	ARP	192.168.1.1 is at c4:b8:b4:a9:93:af	HuaweiTe_a9...	IntelCor_8d:11:...	15.6252137...	