COMPUTER NETWORKS LAB MANUAL

Arihant Chawla
Punjab Engineering College
17103036

Contents

1 Socket Programming Lab 1 : TCP Echo

TcpEchoServer.c

```c
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define MAXPENDING 5
#define RCVBUFSIZE 32

void DieWithError(char *errorMessage);
void HandleTCPClient(int clntSocket);

int main(int argc, char *argv[])
{
    printf("TCP ECHO SERVER PROGRAM. MADE BY ARIHANT CHAWLA\n");
    int servSock;
    int clntSock;
    struct sockaddr_in echoServAddr;
    struct sockaddr_in echoClntAddr;
    unsigned short echoServPort;
    unsigned int clntLen;
    if (argc != 2)
    {
        fprintf(stderr, "Usage: %s <Server Port>\n", argv[0]) ;
        exit(1);
    }
    echoServPort = atoi(argv[1]);
    if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError( "socket () failed") ;
    memset(&echoServAddr, 0, sizeof(echoServAddr));
    echoServAddr.sin_family = AF_INET;
    echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    echoServAddr.sin_port = htons(echoServPort);
    if (bind(servSock, (struct sockaddr *)&echoServAddr, sizeof(
   echoServAddr)) < 0)
        DieWithError ( "bind () failed");
```

```
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,&
        clntLen)) < 0)
43          DieWithError("accept() failed");
44      printf("Handling client %s\n", inet_ntoa(echoClntAddr.sin_addr));
45      HandleTCPClient (clntSock) ;
46      }
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)) <
        0)
59              DieWithError("recv() failed") ;
60      }
61      close(clntSocket);
62  }
63
64  void DieWithError(char *errorMessage)
65  {
66      perror (errorMessage) ;
67      exit(1);
68  }
```

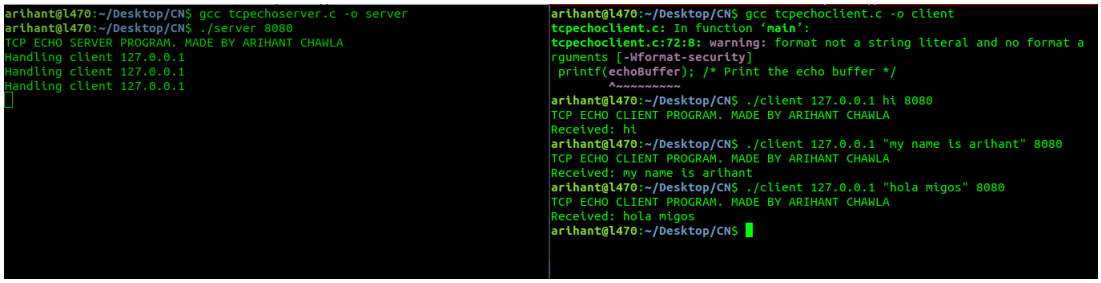
TcpEchoClient.c

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

```
#include <string.h>
#include <unistd.h>
#define RCVBUFSIZE 32
void DieWithError(char *errorMessage);
int main(int argc, char *argv[])
{
    printf("TCP ECHO CLIENT PROGRAM. MADE BY ARIHANT CHAWLA\n");
    int sock;
    struct sockaddr_in echoServAddr;
    unsigned short echoServPort;
    char *servlP;
    char *echoString;
    char echoBuffer[RCVBUFSIZE];
    unsigned int echoStringLen;
    int bytesRcvd, totalBytesRcvd;
    if ((argc< 3) || (argc> 4))
    {
        fprintf(stderr, "Usage: %s <Server IP> <Echo Word> [<Echo Port>]\n"
    ,argv[0]);
        exit(1);
    }
    servlP = argv[1] ;
    echoString = argv[2] ;
    if (argc == 4)
        echoServPort = atoi(argv[3]);
    else
        echoServPort = 7;
    if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError(" socket () failed") ;
    memset(&echoServAddr, 0, sizeof(echoServAddr));
    echoServAddr.sin_family = AF_INET;
    echoServAddr.sin_addr.s_addr = inet_addr(servlP);
    echoServAddr.sin_port = htons(echoServPort);
    if (connect(sock, (struct sockaddr *) & echoServAddr, sizeof(
   echoServAddr)) < 0)
        DieWithError(" connect () failed") ;
    echoStringLen = strlen(echoString) ;
    if (send(sock, echoString, echoStringLen, 0) != echoStringLen)
        DieWithError("send() sent a different number of bytes than expected
   ");
    totalBytesRcvd = 0;
    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:



2 Socket Programming Lab 2 : UDP Echo

UdpEchoServer.c

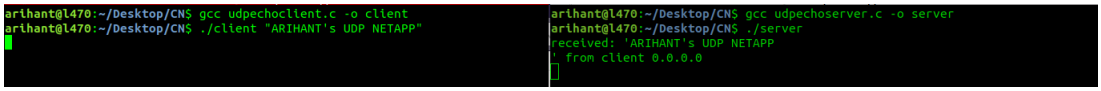
```c
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
  int SERVER_PORT = 8877;
  struct sockaddr_in server_address;
  memset(&server_address, 0, sizeof(server_address));
  server_address.sin_family = AF_INET;
  server_address.sin_port = htons(SERVER_PORT);
  server_address.sin_addr.s_addr = htonl(INADDR_ANY);
  int sock;
  if ((sock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
    printf("could not create socket\n");
    return 1;
  }
  if ((bind(sock, (struct sockaddr *)&server_address,
            sizeof(server_address))) < 0) {
    printf("could not bind socket\n");
    return 1;
  }
  struct sockaddr_in client_address;
  int client_address_len = 0;
  while (true) {
    char buffer[500];
    int len = recvfrom(sock, buffer, sizeof(buffer), 0,
                       (struct sockaddr *)&client_address,
                       &client_address_len);
    buffer[len] = '\0';
    printf("received: '%s' from client %s\n", buffer,
           inet_ntoa(client_address.sin_addr));
    sendto(sock, buffer, len, 0, (struct sockaddr *)&client_address,
           sizeof(client_address));
  }
  return 0;
}
```

UdpEchoClient.c

```c
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

int main() {
  const char* server_name = "localhost";
  const int server_port = 8877;
  struct sockaddr_in server_address;
  memset(&server_address, 0, sizeof(server_address));
  server_address.sin_family = AF_INET;
  inet_pton(AF_INET, server_name, &server_address.sin_addr);
  server_address.sin_port = htons(server_port);
  int sock;
  if ((sock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
    printf("could not create socket\n");
    return 1;
  }
  const char* data_to_send = "ARIHANT's UDP NETAPP\n";
  int len =
      sendto(sock, data_to_send, strlen(data_to_send), 0,
             (struct sockaddr*)&server_address, sizeof(server_address));
  char buffer[100];
  recvfrom(sock, buffer, len, 0, NULL, NULL);
  buffer[len] = '\0';
  printf("recieved: '%s'\n", buffer);
  close(sock);
  return 0;
}
```

TEST RUN:



3 Socket Programming Lab 3 : LAN Chat (Optional)

LANChatServer.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>

void error(const char *msg)
{
    perror(msg);
    exit(1);
}

int main(int argc, char *argv[])
{
    if(argc<2)
    {
        fprintf(stderr,"::usage:: ./a.out [PORT_NO]\nPROGRAM TERMINATED\n");
        exit(1);
    }

    int sockfd,newsockfd,portno,n;
    char buffer[255];
    struct sockaddr_in serv_addr,cli_addr;

    sockfd=socket(AF_INET,SOCK_STREAM,0);
    if(sockfd<0)
    {
        error("ERROR OPENING SOCKET\n");
    }

    portno=atoi(argv[1]);

    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=INADDR_ANY;
```

```
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----------------------------------------------------------------\n\n\
    tWELCOME 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);
```

```
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\
   tWELCOME TO ARIHANT'S CHATTING APP\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

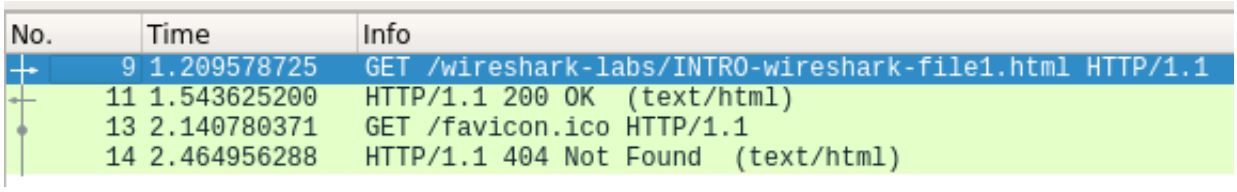
Answer to Q1

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

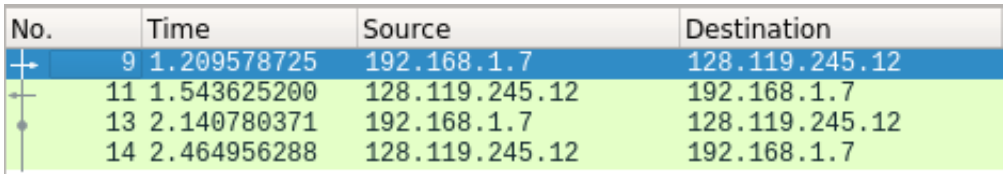
From the time column, the first packet was sent at 1.209578725
The time when the first OK was received was 1.543625200
Therefore time taken from request to OK is 1.543625200 - 1.209578725 = 0.334046475 sec



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
Protocol Length
      9 1.209578725    192.168.1.7           128.119.245.12        GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1
HTTP     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\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/INTRO-wireshark-file1.html]
    [HTTP request 1/2]
    [Response in frame: 11]
    [Next request in frame: 13]
```

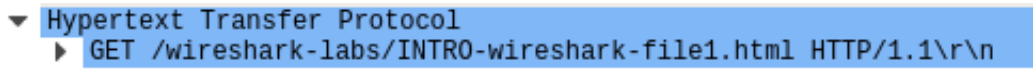
```
No.     Time           Source                Destination           Info
Protocol Length
     11 1.543625200    128.119.245.12        192.168.1.7           HTTP/1.1 200 OK  (text/html)
HTTP     504
Frame 11: 504 bytes on wire (4032 bits), 504 bytes captured (4032 bits) on interface 0
Ethernet II, 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

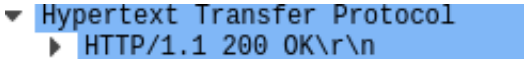
Answer to Q1

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

GET Packet:

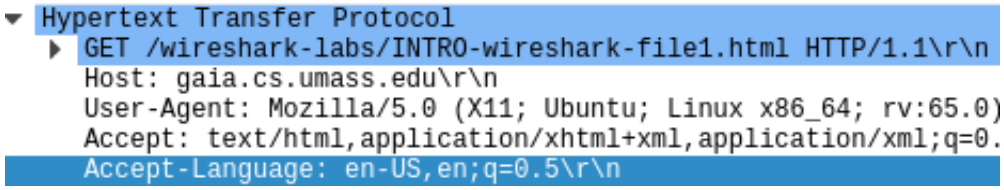


OK Packet:



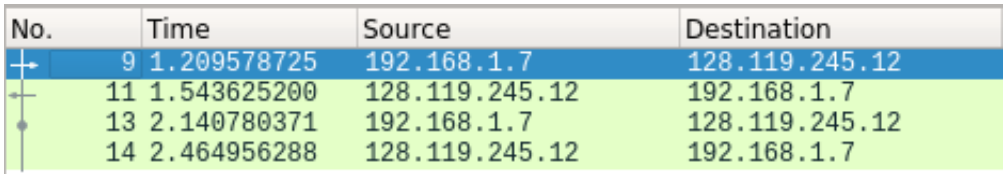
Answer to Q2

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



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



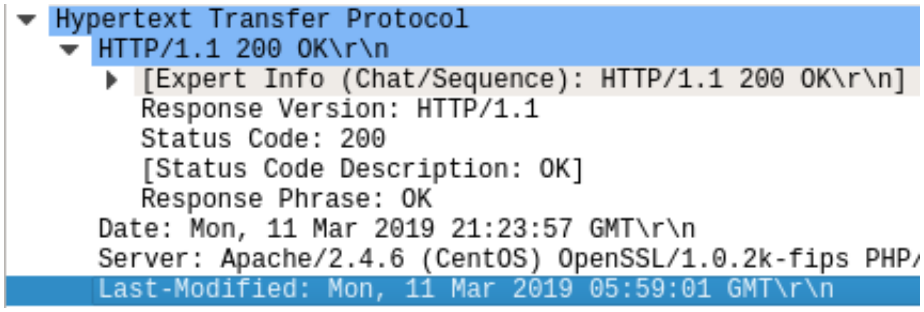
No.	Time	Source	Destination
9	1.209578725	192.168.1.7	128.119.245.12
11	1.543625200	128.119.245.12	192.168.1.7
13	2.140780371	192.168.1.7	128.119.245.12
14	2.464956288	128.119.245.12	192.168.1.7

Answer to Q4

Status Code: 200 (OK)

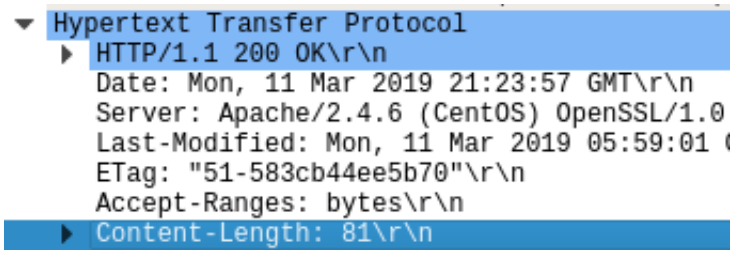
Answer to Q5

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



Answer to Q6

The content length returned is 81 bytes.



Answer to Q7

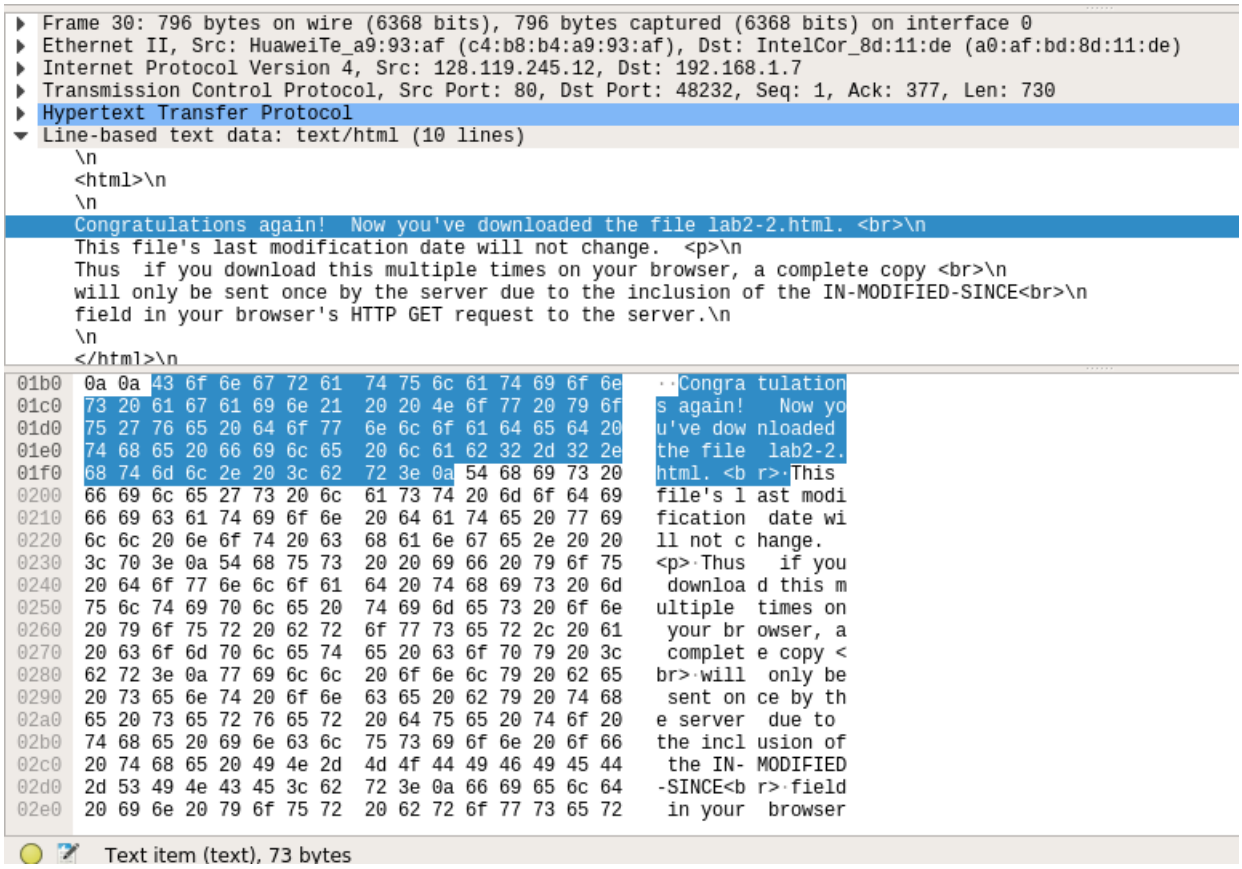
No, all the headers within the raw data are shown by wireshark in the packet listing window.

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.



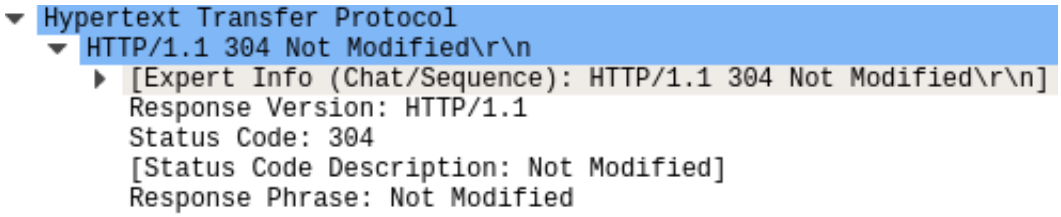
Answer to Q10

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

Answer to Q11

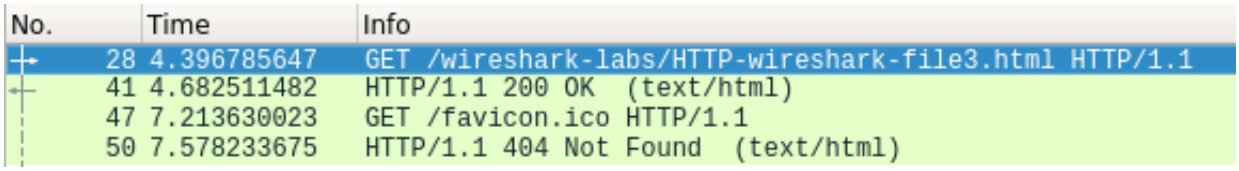
The status code returned is 304: NOT MODIFIED.
This time the server does not explicitly return the data of the webpage. This is because

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.



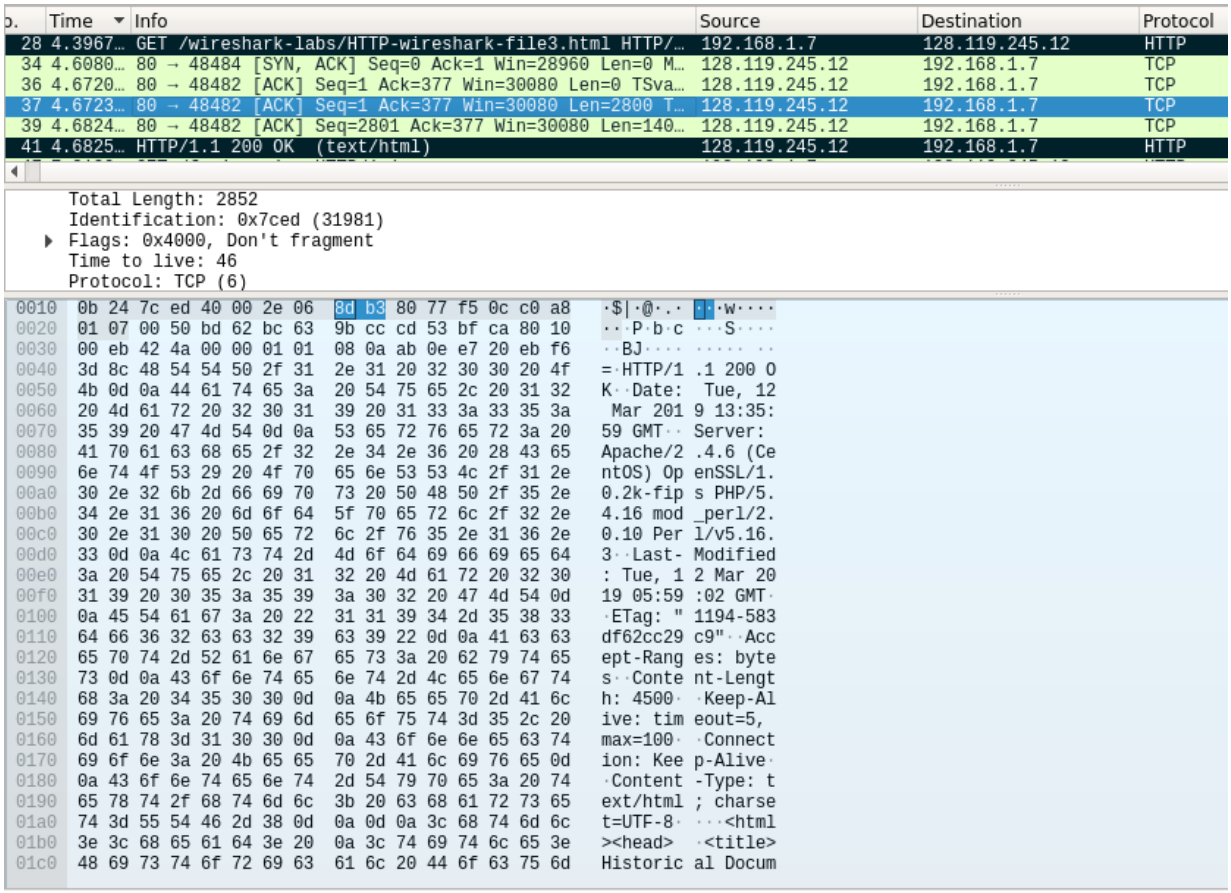
Answer to Q12

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



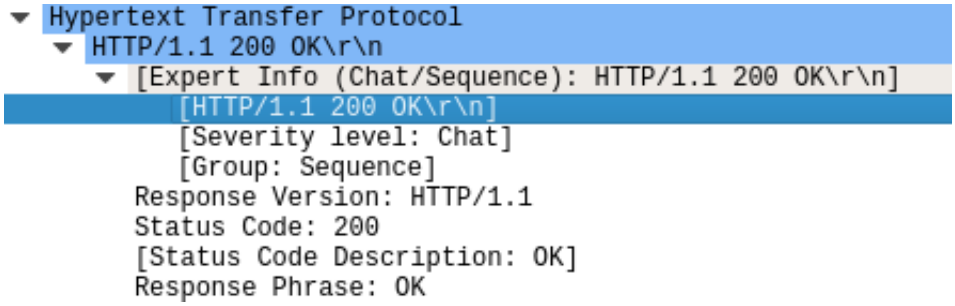
Answer to Q13

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.



Answer to Q14

The status code is 200: OK. The response phrase associated with it is 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 fevicon 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 i spretty 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
87	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 && e
cho
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
```

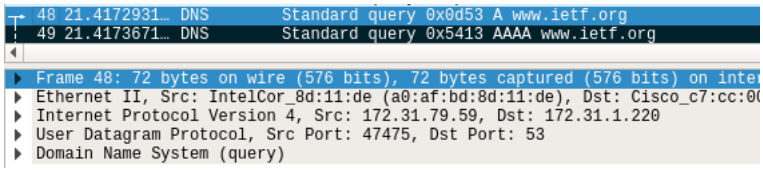
Answer to Q3

```
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

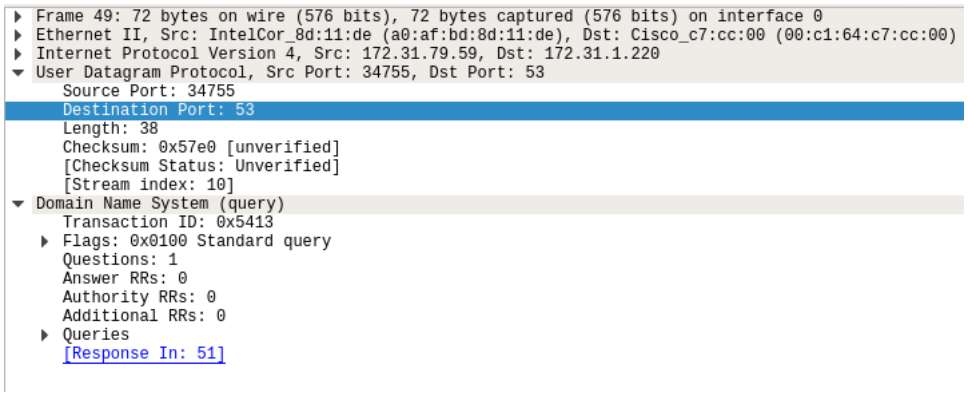
The DNS query and response are sent over UDP



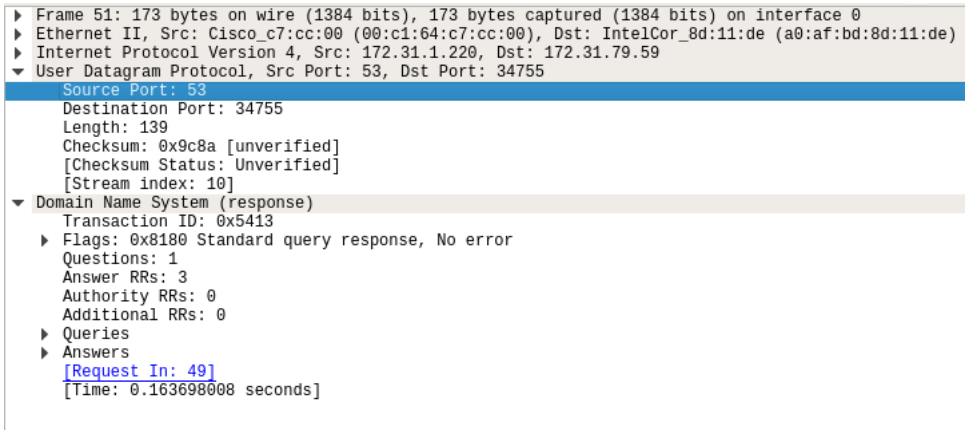
Answer to Q5

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

QUERY



RESPONSE



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, howerver, 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 of 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]
```

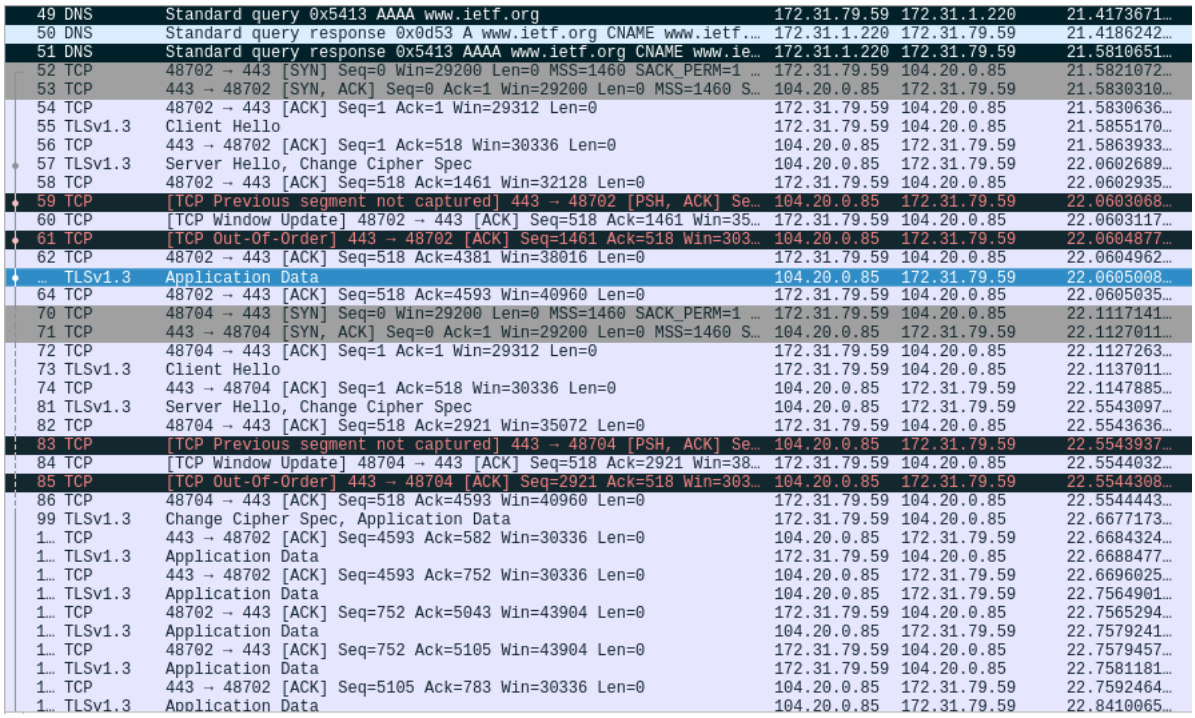
Answer to Q9

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 webpage 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.ie…	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	22.0603068…
60	TCP	[TCP Window Update] 48702 → 443 [ACK] Seq=518 Ack=1461 Win=35…	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	22.0604877…
62	TCP	48702 → 443 [ACK] Seq=518 Ack=4381 Win=38016 Len=0	172.31.79.59	104.20.0.85	22.0604962…
…	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	22.1147885…
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.dscb.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.dscb.akamaiedge.net
  ▾ e9566.dscb.akamaiedge.net: type A, class IN, addr 23.43.29.195
      Name: e9566.dscb.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

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.ne…    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

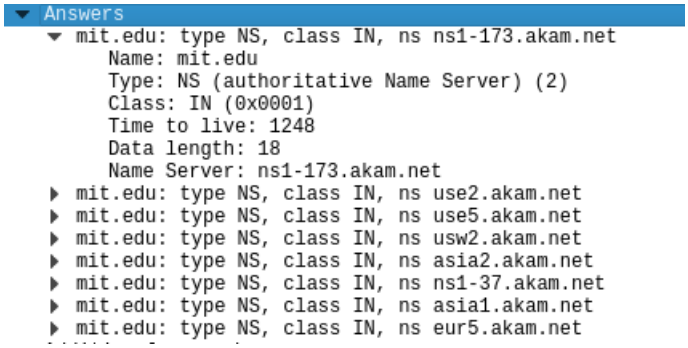
5. asia2.akam.net
6. ns1-37.akam.net
7. asia1.akam.net
8. eur5.akam.net

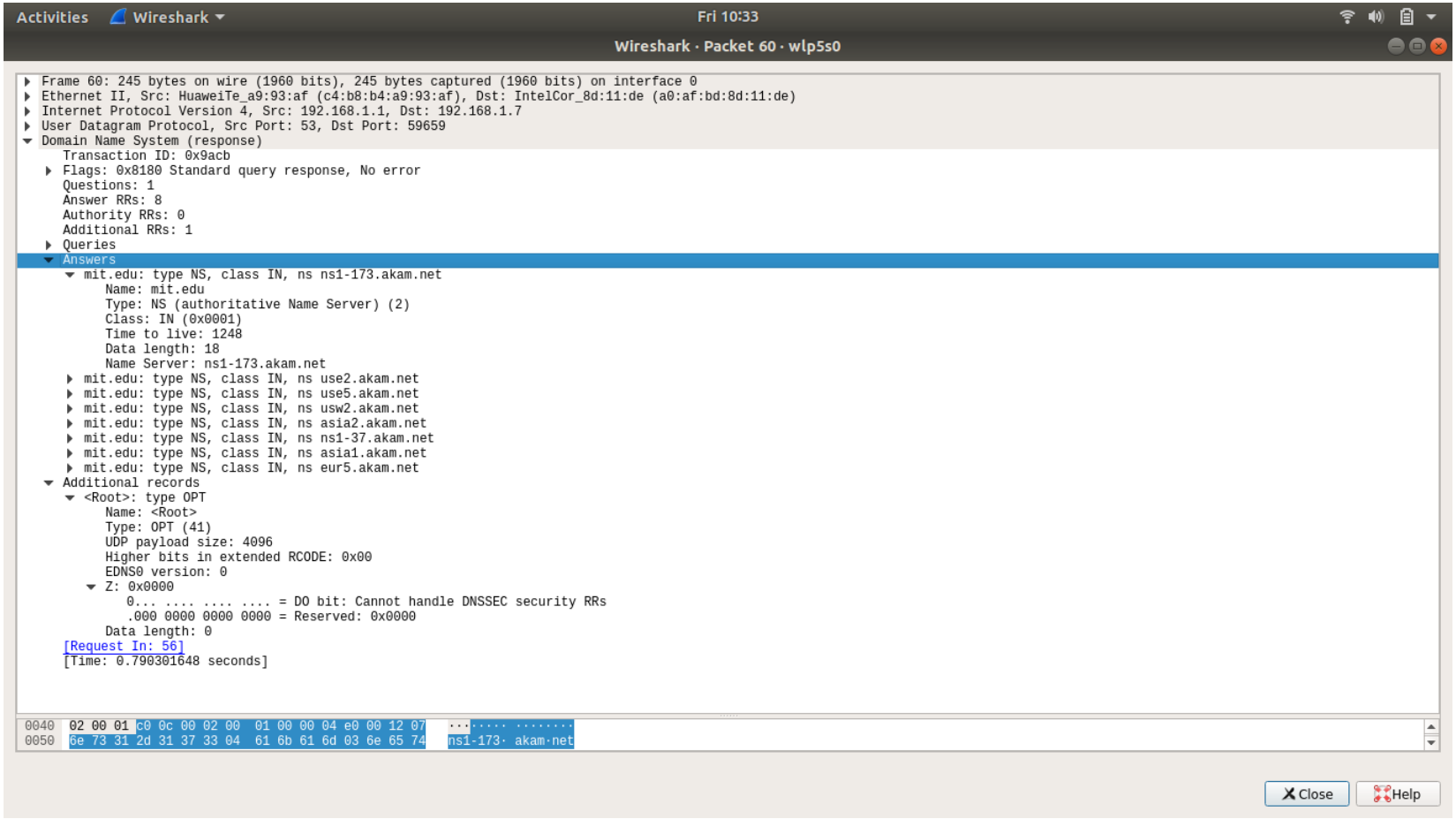
The response message does not contain IP addresses.

```
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:
```





Answer to Q19

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 ww
 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.

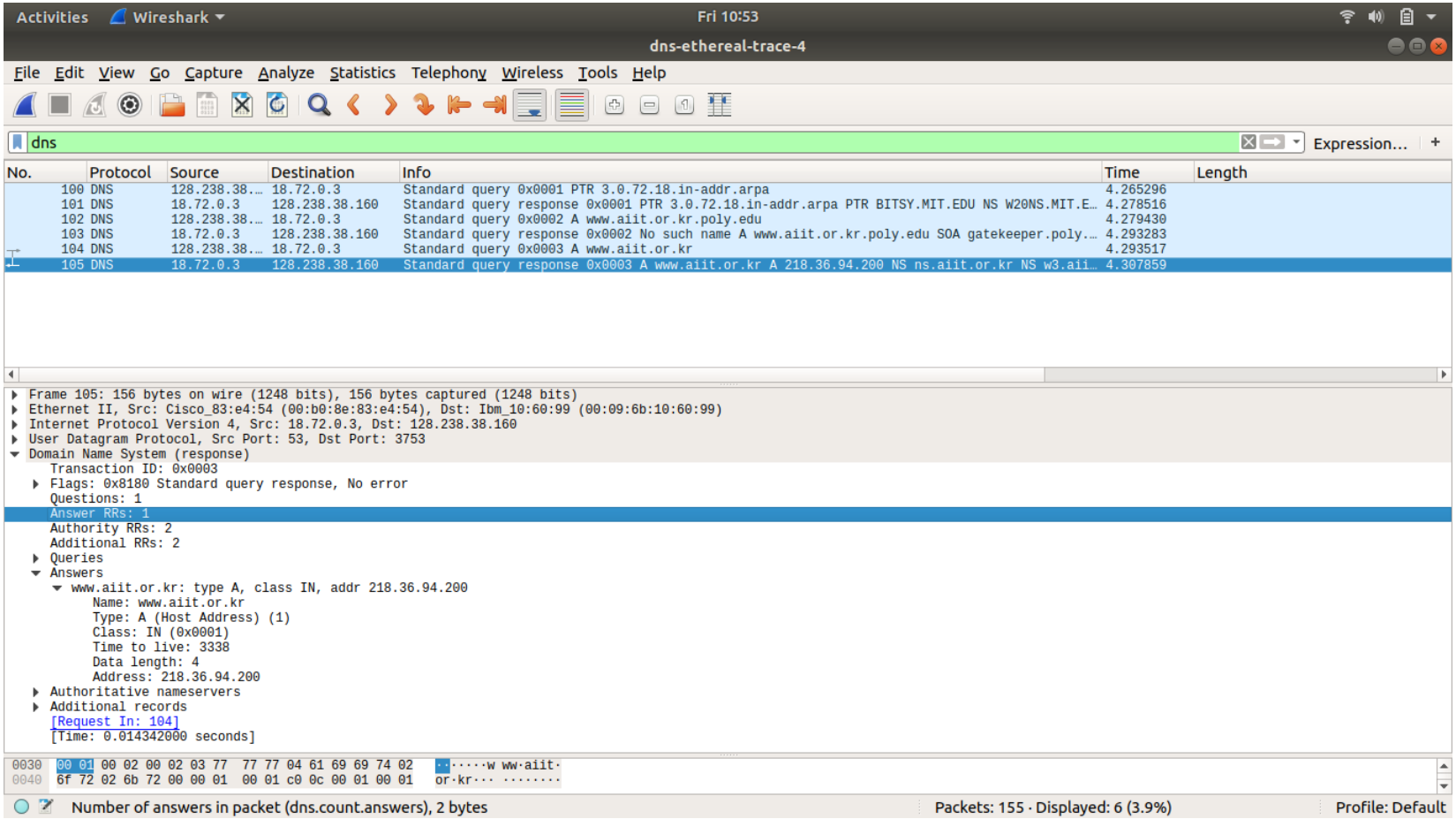
```
▼ Domain Name System (query)
    Transaction ID: 0x0003
  ▶ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▼ Queries
    ▶ www.aiit.or.kr: type A, class IN
    [Response In: 105]
```

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
```

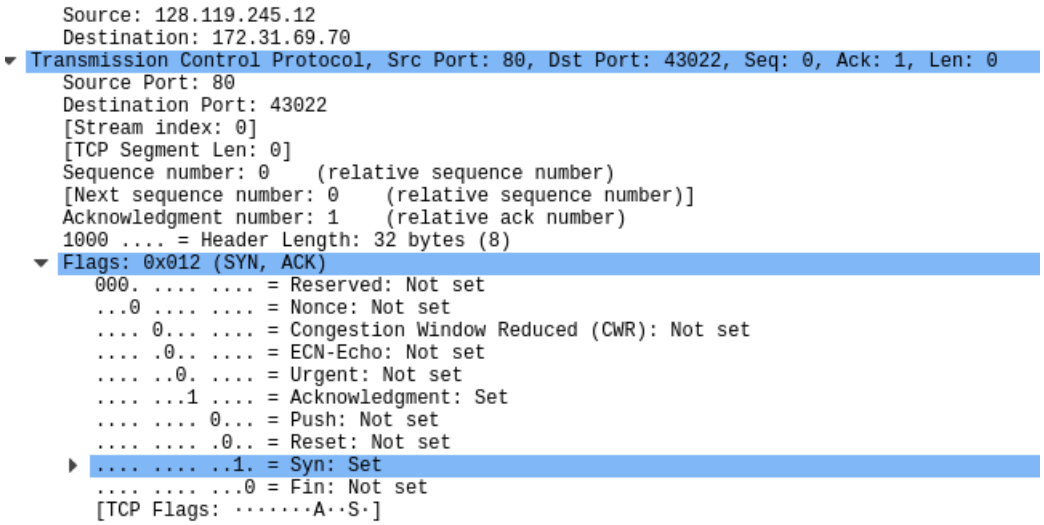
Answer to Q4

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: ··········S·]
```

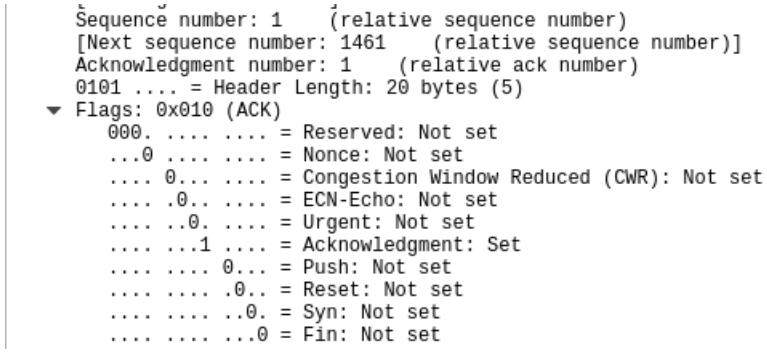
Answer to Q5

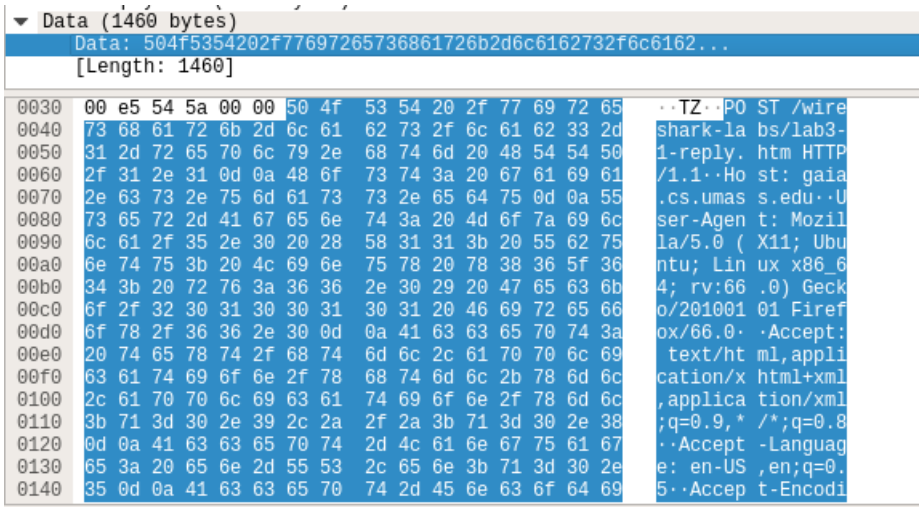
The sequence number of the SYNACK packet sent by the webserver is 0 (relative sequence number). The Acknowledgment field is 1 (relative acknowledgement number). This is the value of the sequence number of the next byte the receiver expects to receive and so it is 1 as the Sequence number is 0. What makes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



Answer to Q6

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





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:
EstimatedRTT = 0.875 * EstimatedRTT + 0.125 * 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.

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… 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 … 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…
 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 …
```

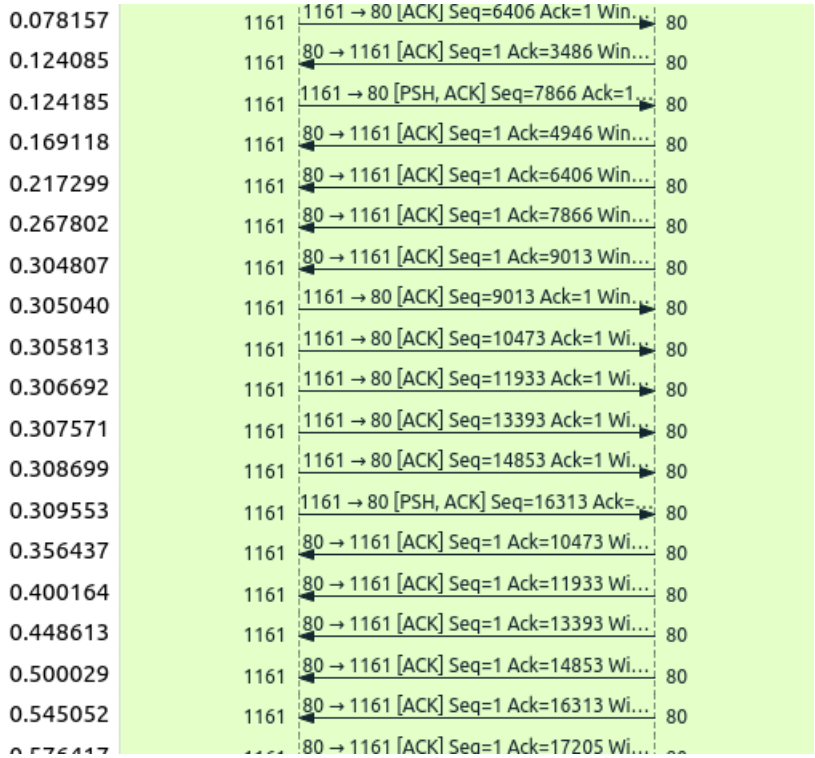
The calculated size of the receiving window grows upto 62780 bytes and then plateaus out. It doesn't seem like the sender is ever throttled by the lack of buffer space as there is no re-transmission.

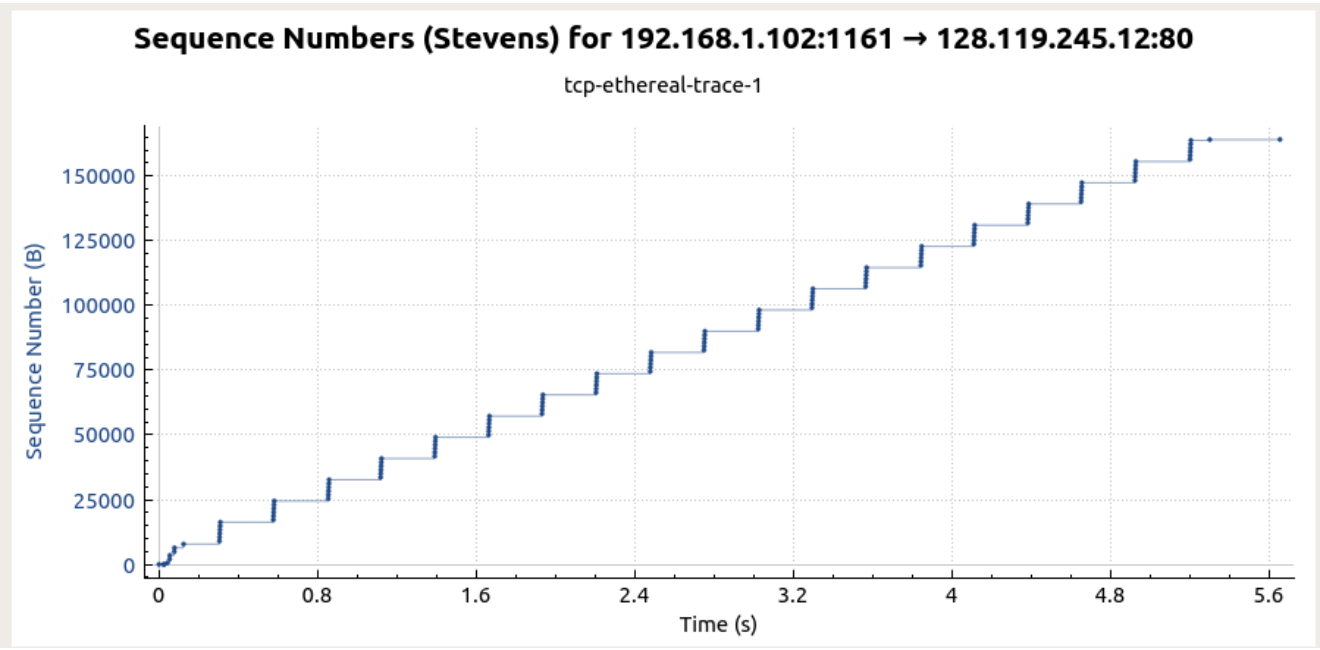
```
  2 TCP          80 → 1161 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1
  6 TCP          80 → 1161 [ACK] Seq=1 Ack=566 Win=6780 Len=0
  9 TCP          80 → 1161 [ACK] Seq=1 Ack=2026 Win=8760 Len=0
 12 TCP          80 → 1161 [ACK] Seq=1 Ack=3486 Win=11680 Len=0
 14 TCP          80 → 1161 [ACK] Seq=1 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
 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
```

```
 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=85661 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 wireshark. 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 occured.





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

```
  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 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

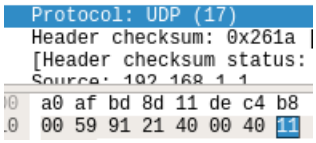
As the length field is 16 bit long, there are theoretically 2$\hat{1}$6-1 = 65535 bytes long data allowed. Out of this 8 bytes will be for the header. So 65527 bytes are allowed for the data. However, this theoretical limit is not right as the IP also has a header that has to be incorporated in the 65535 bytes. As header of IP is 20 bytes long, therefore, the total length allowed fot 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.

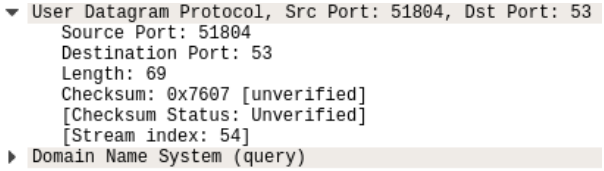
Answer to Q6

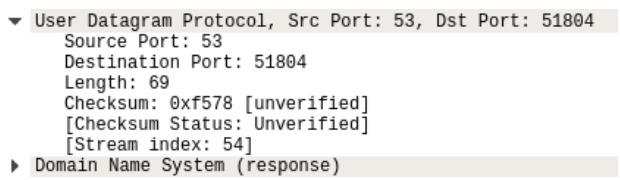
The protocol number for UDP is 17 in decimal and 11 in hexadecimal.



Answer to Q7

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

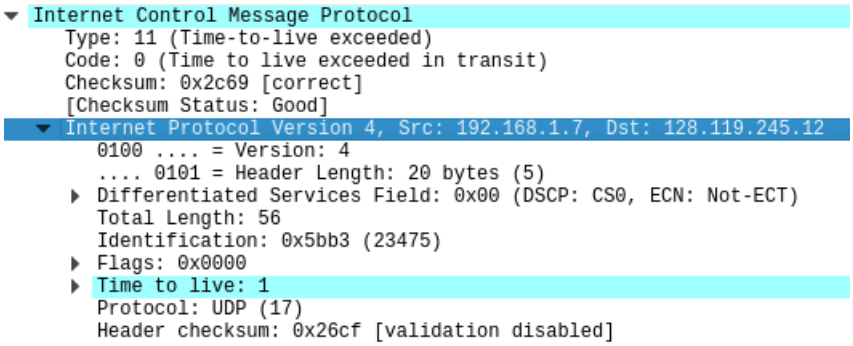




9 Wireshark Lab 6 : IP

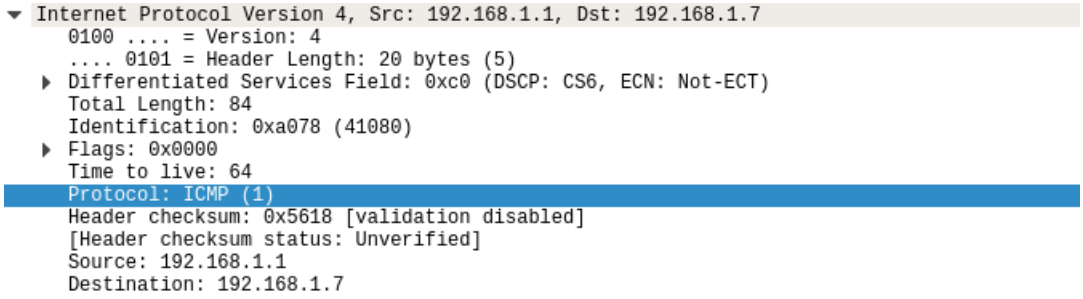
Answer to Q1

The source IP address is 192.168.1.7



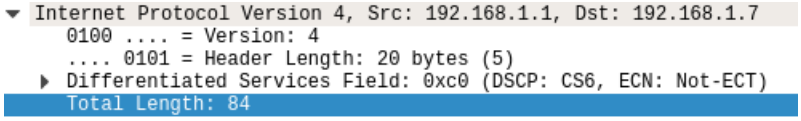
Answer to Q2

The protocol field in IP header is ICMP (1)



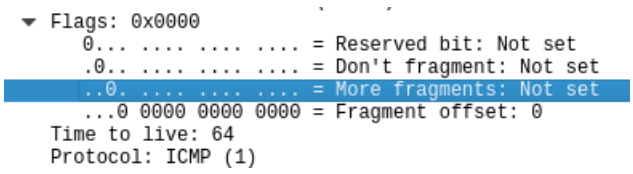
Answer to Q3

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



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.



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.

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)

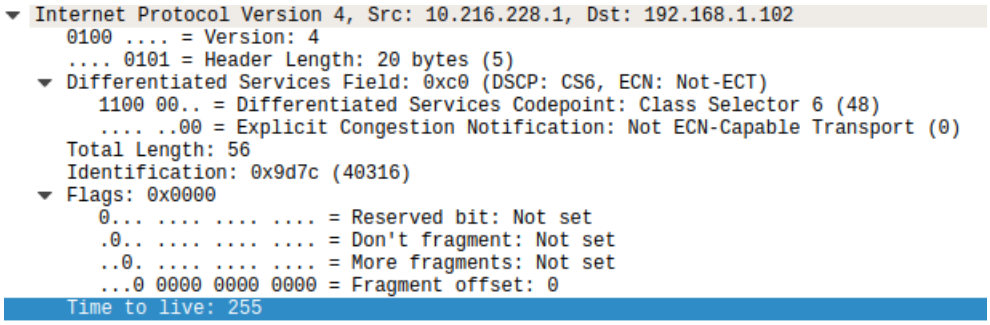
All of these are supposed to remain constant because of the respective reasons mentioned above.
On the other hand, Identification, Time to live and Header Checksum fields must change. This is because the packet numbers for identification must be unique, TTL is incremented in subsequent transmissions 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.



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
▾ 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

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 firsttwo packets and the last packet, we see a change in total length, and also in the flags. The first two packets havea total length of 1500, with the more fragments bit set to 1, and the last packet has a total length of 540, with themore 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.

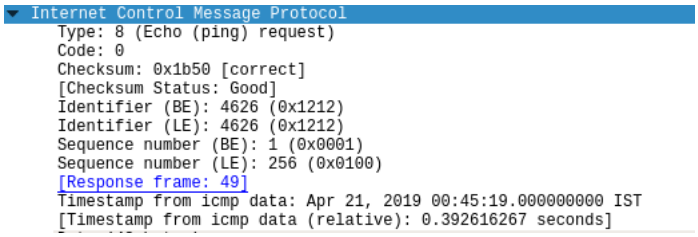
	Protocol	Info	Source	Destination
48	ICMP	Echo (ping) request id=0x1212, seq=1/256, ttl=64 (reply in 49)	192.168.1.7	184.26.196.231
49	ICMP	Echo (ping) reply id=0x1212, seq=1/256, ttl=59 (request in 48)	184.26.196.231	192.168.1.7
59	ICMP	Echo (ping) request id=0x1212, seq=2/512, ttl=64 (reply in 60)	192.168.1.7	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.

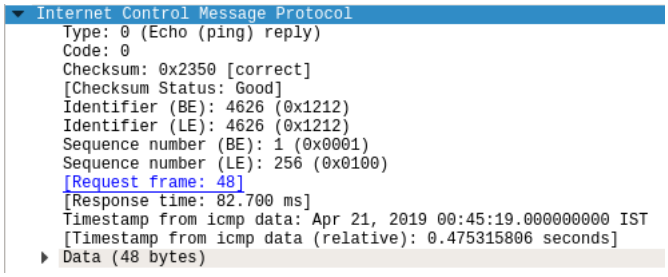
Answer to Q3

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



Answer to Q4

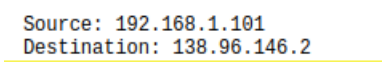
The ICMP type is 0, and the code number is 0. The ICMP packet also has checksum, identifier, sequence number, and data fields. The checksum, sequence number and identifier fields are two bytes each.



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.

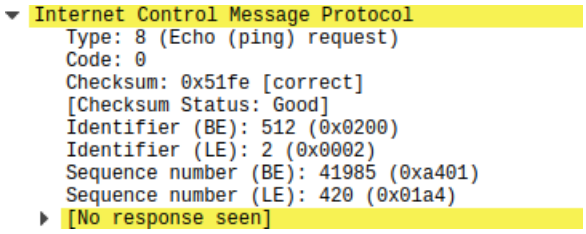


Answer to Q6

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

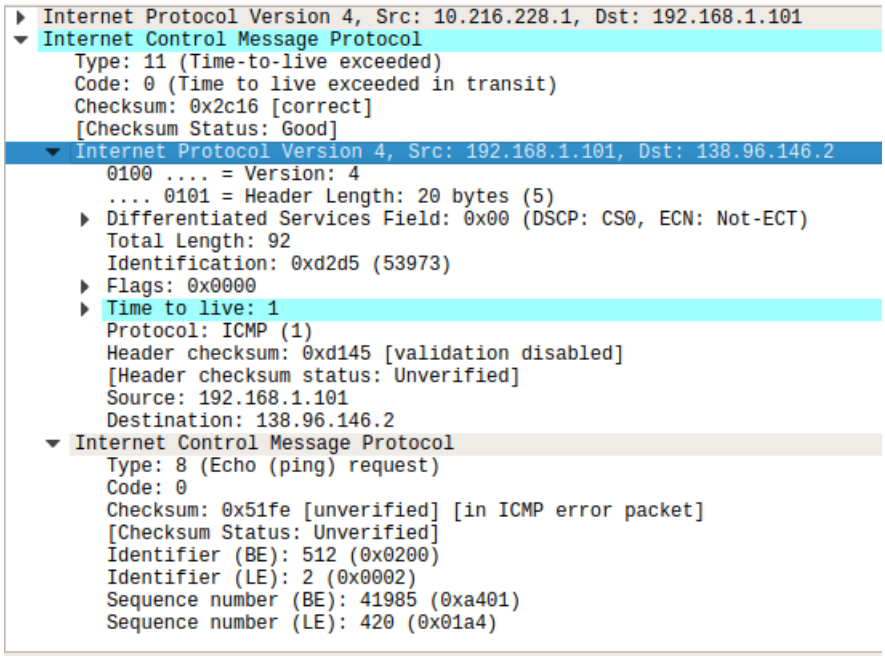
Answer to Q7

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.



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.



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.

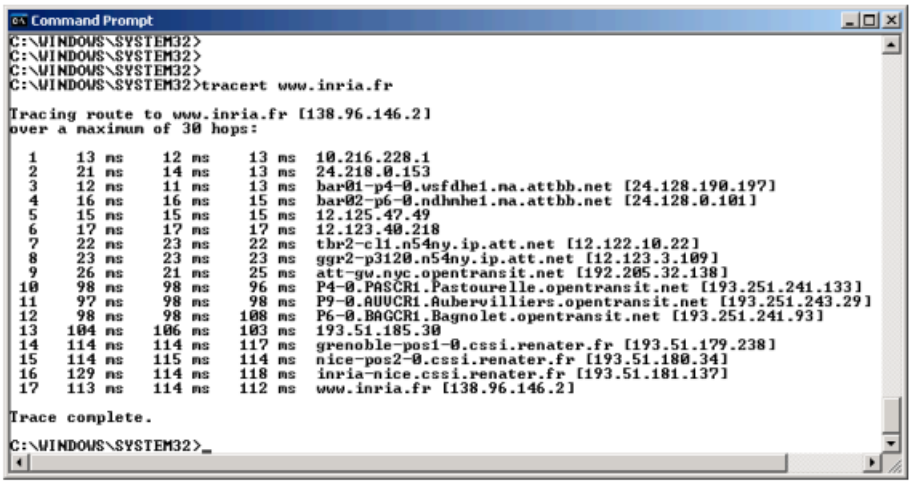
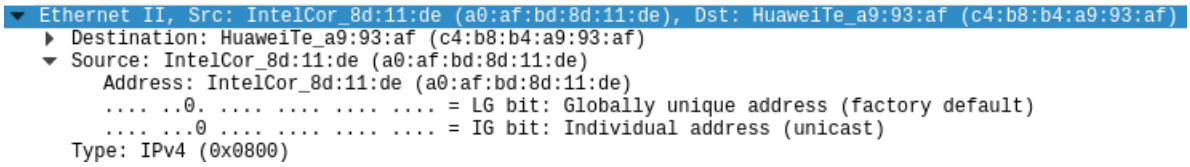


Figure 4 Command Prompt window displays the results of the Traceroute program.

11 Wireshark Lab 8 : Ethernet and ARP

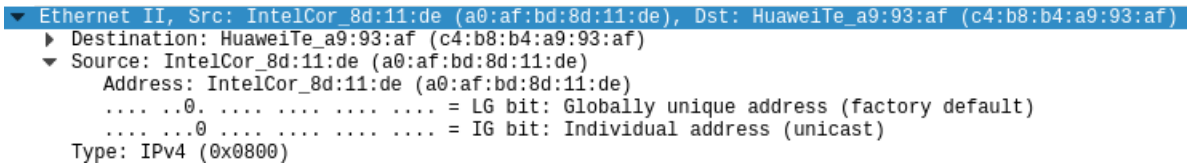
Answer to Q1

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



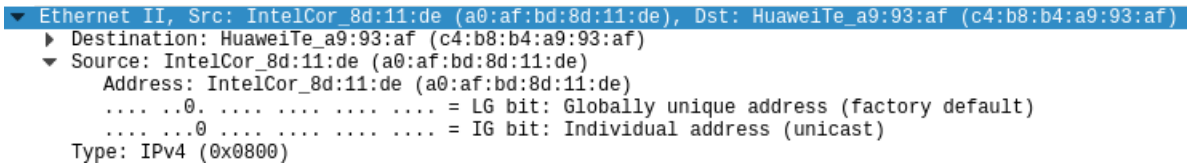
Answer to Q2

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.



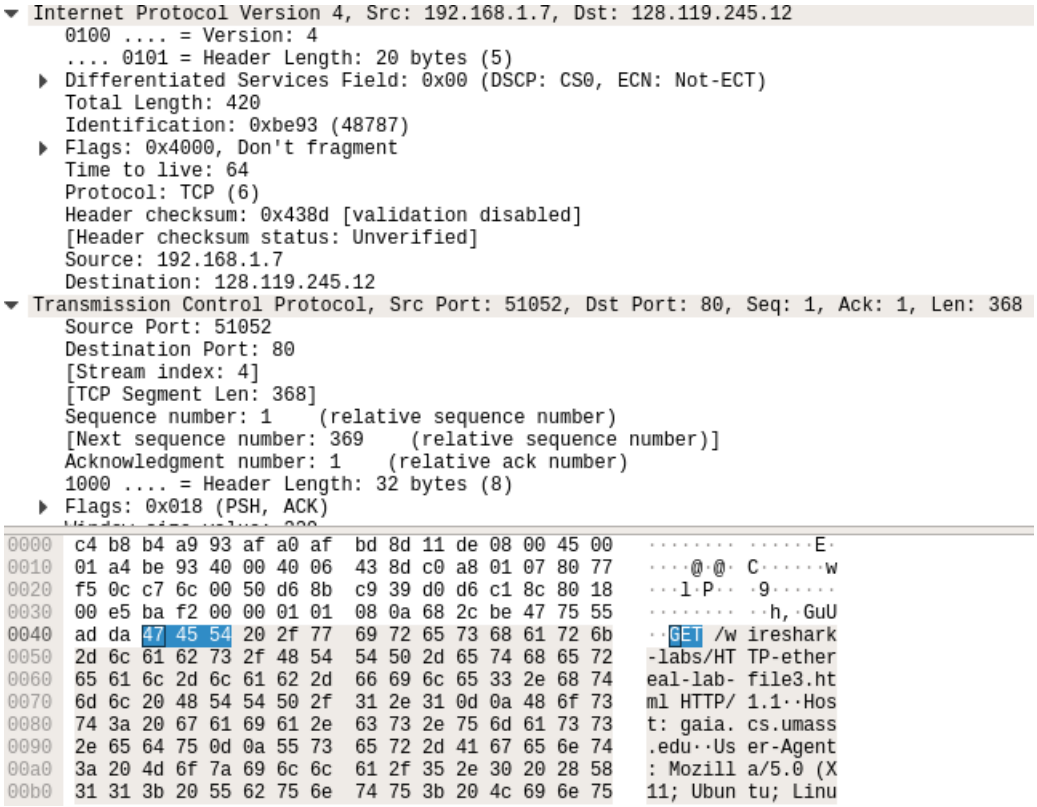
Answer to Q3

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



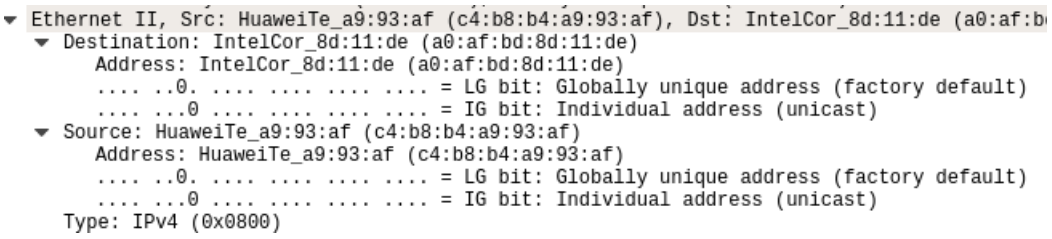
Answer to Q4

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.



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.



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: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 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: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 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: S at, 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   IGHTS</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 âĂIJTarget MAC addressâĂİ 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.

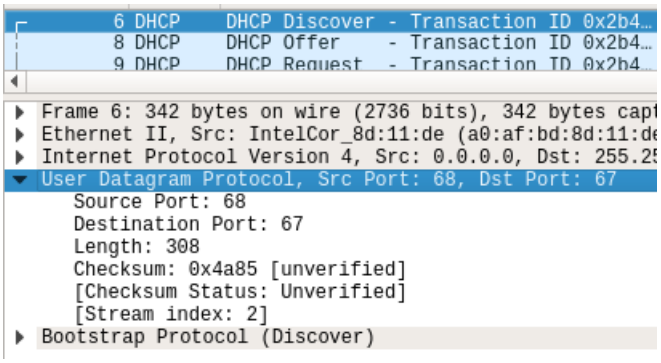
Answer to Q15

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

12 Wireshark Lab 9 : DHCP

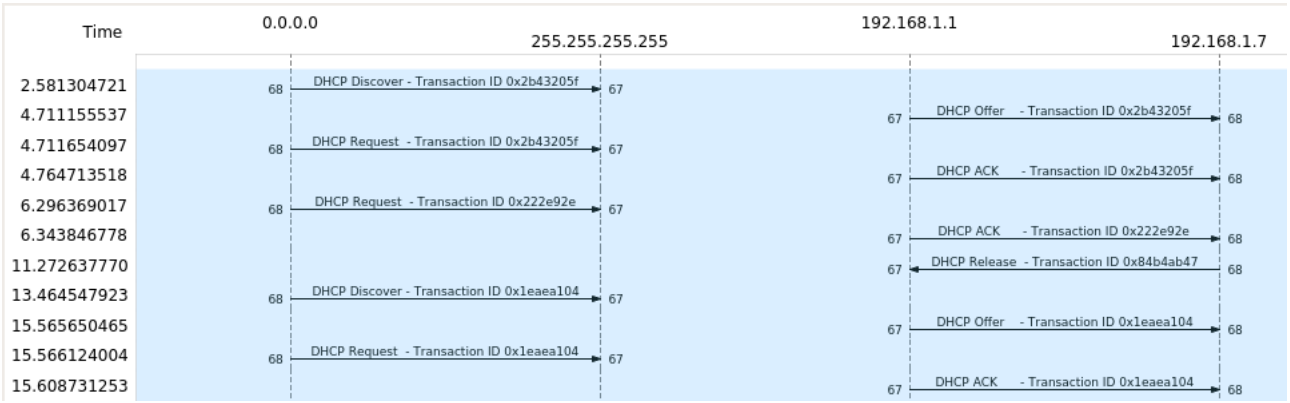
Answer to Q1

The DHCP messages are sent over UDP.



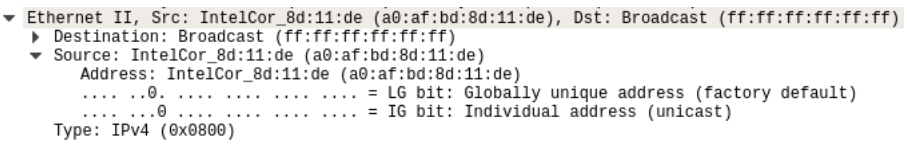
Answer to Q2

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.



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.

```
▼ Bootstrap Protocol (Discover)
    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: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  ▶ Option: (53) DHCP Message Type (Discover)
  ▶ Option: (50) Requested IP Address
  ▶ Option: (12) Host Name
  ▶ Option: (55) Parameter Request List
  ▶ Option: (255) End
    Padding: 000000000000000000000000000000000000000000000000...
```

```
▼ 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: 00000000000000000000
    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
  ▶ Option: (12) Host Name
  ▶ Option: (55) Parameter Request List
  ▶ Option: (255) End
    Padding: 000000000000000000000000000000000000000000000000
```

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
```

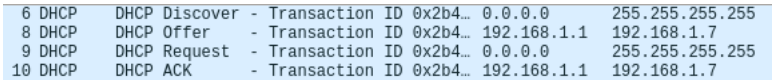
Answer to Q6

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 alloted 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 192.168.1.1

```
▾ 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
```

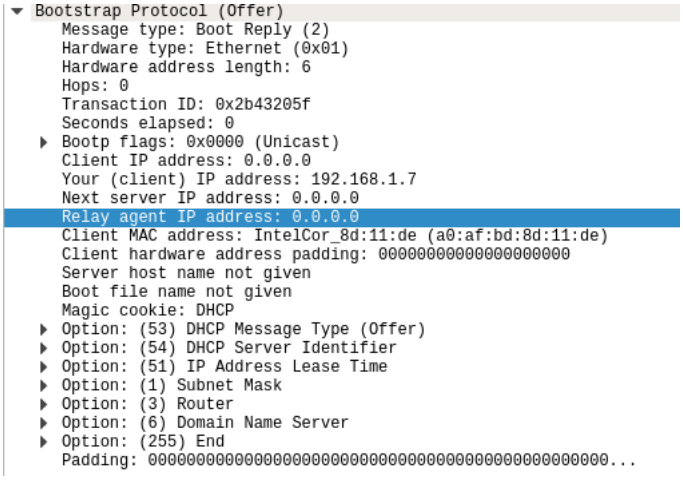
Answer to Q8

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

```
▾ 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: 00000000000000000000
    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: 000000000000000000000000000000000000000000000000...
```

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



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: 00000000000000000000
    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: 000000000000000000000000000000000000000000000000
```

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: 00000000000000000000
    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: 000000000000000000000000000000000000000000000000...
```

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.