LAB 2

Topics: TCP, UDP, IPv4, IPv6

Names and NEPTUN codes:	**name, NEPTUN** **name, NEPTUN**
Room and workstation ID:	**R4D, Workstation ##**
Date and time:	**11.03.2024**

This assignment is a supplement to Computer Networking: A Top-Down Approach, J.F. Kurose and K.W. Ross

PART A

Note: ***Questions to be answered are highlighted with yellow!***

In this part, we'll investigate the behavior of the celebrated TCP protocol in detail. We'll do so by analyzing a trace of the TCP segments sent and received in transferring a 150KB file (containing the text of Lewis Carrol's *Alice's Adventures in Wonderland*) from our computer to a remote server. We'll study TCP's use of sequence and acknowledgement numbers for providing reliable data transfer; we'll see TCP's congestion control algorithm – slow start and congestion avoidance – in action; and we'll look at TCP's receiver-advertised flow control mechanism.

1. A first look at the captured trace

Open the Wireshark captured packet file ***tcp-ethereal-trace-1*** in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip (that is, download the trace and open that trace in Wireshark).

Before analyzing the behavior of the TCP connection in detail, let's take a high-level view of the trace. First, filter the packets displayed in the Wireshark window by entering "tcp" (lowercase, no quotes, and don't forget to press return after entering!) into the display filter specification window towards the top of the Wireshark window.

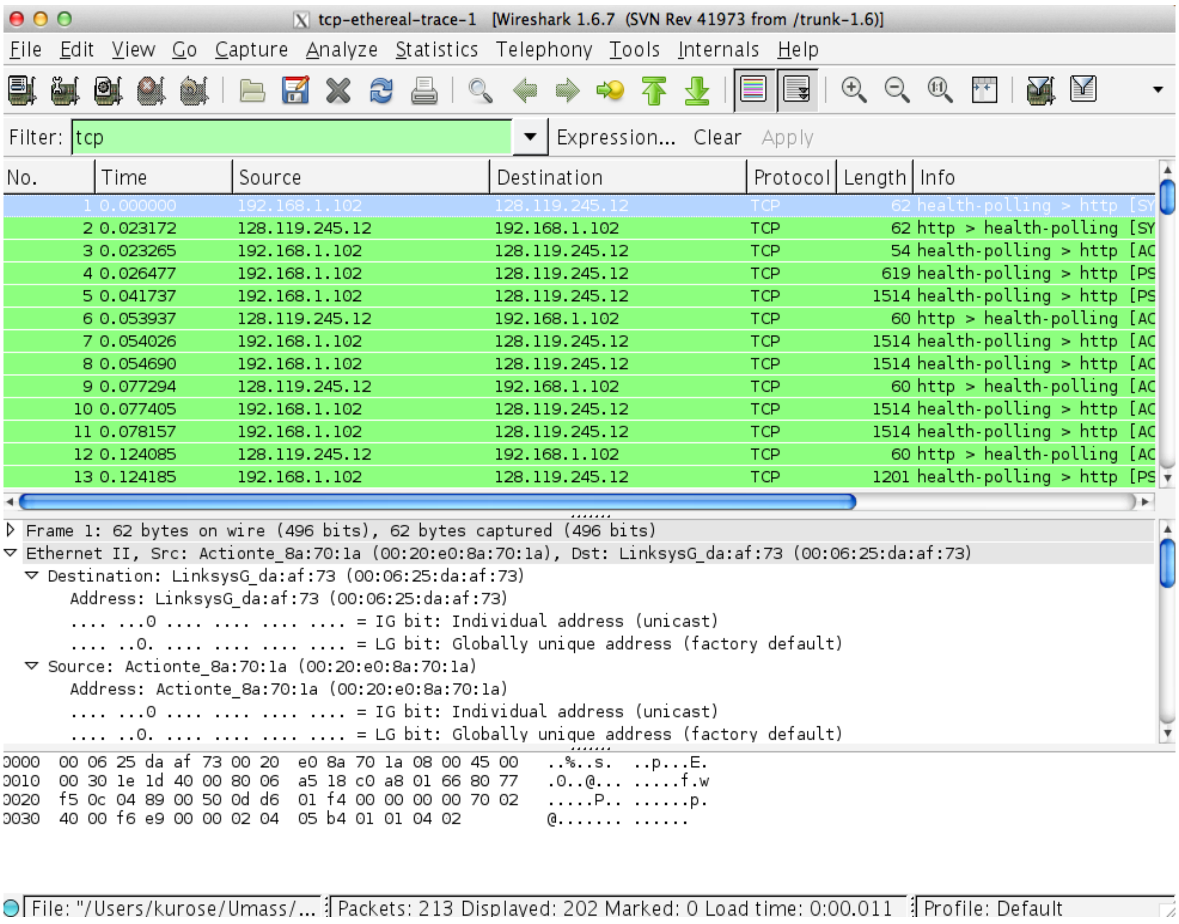
What you should see is series of TCP and HTTP messages between your computer and gaia.cs.umass.edu. You should see the initial three-way handshake containing a SYN message. You should see an HTTP POST message. Depending on the version of Wireshark you are using, you might see a series of "HTTP Continuation" messages being sent from your computer to gaia.cs.umass.edu. Recall from our discussion in the earlier HTTP Wireshark lab, that is no such thing as an HTTP Continuation message – this is Wireshark's way of indicating that there are multiple TCP segments being used to carry a single HTTP message. In more recent versions of Wireshark, you'll see "[TCP segment of a reassembled PDU]" in the Info column of the Wireshark display to indicate that this TCP segment contained data that belonged to an upper layer protocol message (in our case here, HTTP). You should also see TCP ACK segments being returned from gaia.cs.umass.edu to your computer.

Answer the following questions, by analyzing the same packet file ***tcp-ethereal-trace-1***. Whenever possible, when answering a question, you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. **Annotate** the printout to explain your answer. To print a packet, use *File->Print*, choose *Selected packet only*, choose *Packet summary line,* and select the minimum amount of packet detail that you need to answer the question.

1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it's probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the "details of the selected packet header window" (refer to Figure 2 in the "Getting Started with Wireshark" Lab if you're uncertain about the Wireshark windows.

2. What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?

Since this part is about TCP rather than HTTP, let's change Wireshark's "listing of captured packets" window so that it shows information about the TCP segments containing the HTTP messages, rather than about the HTTP messages. To have Wireshark do this, select *Analyze->Enabled Protocols.* Then uncheck the HTTP box and select *OK.* You should now see a Wireshark window that looks like:


tcp-ethereal-trace-1 [Wireshark 1.6.7 (SVN Rev 41973 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp
Expression... Clear Apply

```
No.  Time      Source          Destination     Protocol Length Info
   1 0.000000  192.168.1.102   128.119.245.12  TCP        62 health-polling > http [SY
   2 0.023172  128.119.245.12  192.168.1.102   TCP        62 http > health-polling [SY
   3 0.023265  192.168.1.102   128.119.245.12  TCP        54 health-polling > http [AC
   4 0.026477  192.168.1.102   128.119.245.12  TCP       619 health-polling > http [PS
   5 0.041737  192.168.1.102   128.119.245.12  TCP      1514 health-polling > http [PS
   6 0.053937  128.119.245.12  192.168.1.102   TCP        60 http > health-polling [AC
   7 0.054026  192.168.1.102   128.119.245.12  TCP      1514 health-polling > http [AC
   8 0.054690  192.168.1.102   128.119.245.12  TCP      1514 health-polling > http [AC
   9 0.077294  128.119.245.12  192.168.1.102   TCP        60 http > health-polling [AC
  10 0.077405  192.168.1.102   128.119.245.12  TCP      1514 health-polling > http [AC
  11 0.078157  192.168.1.102   128.119.245.12  TCP      1514 health-polling > http [AC
  12 0.124085  128.119.245.12  192.168.1.102   TCP        60 http > health-polling [AC
  13 0.124185  192.168.1.102   128.119.245.12  TCP      1201 health-polling > http [PS
```

```
▷ Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 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)
  ▽ Destination: LinksysG_da:af:73 (00:06:25:da:af:73)
      Address: LinksysG_da:af:73 (00:06:25:da:af:73)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
  ▽ Source: Actionte_8a:70:1a (00:20:e0:8a:70:1a)
      Address: Actionte_8a:70:1a (00:20:e0:8a:70:1a)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
```

```
0000  00 06 25 da af 73 00 20  e0 8a 70 1a 08 00 45 00   ..%..s.  ..p...E.
0010  00 30 1e 1d 40 00 80 06  a5 18 c0 a8 01 66 80 77   .0..@... .....f.w
0020  f5 0c 04 89 00 50 0d d6  01 f4 00 00 00 00 70 02   .....P.. ......p.
0030  40 00 f6 e9 00 00 02 04  05 b4 01 01 04 02         @....... ......
```

```
File: "/Users/kurose/Umass/... | Packets: 213 Displayed: 202 Marked: 0 Load time: 0:00.011 | Profile: Default
```

This is what we're looking for - a series of TCP segments sent between your computer and gaia.cs.umass.edu. We will use the packet trace ***tcp-ethereal-trace-1*** in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip to study TCP behavior in the rest of this lab.

2. TCP Basics

Answer the following questions for the TCP segments:

3. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?
4. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?
5. What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you'll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a "POST" within its DATA field.
6. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received? What is the length of each of the first six TCP segments?

Note that the TCP segments in the **tcp-ethereal-trace-1** trace file are all less that 1460 bytes. This is because the computer on which the trace was gathered has an Ethernet card that limits the length of the maximum IP packet to 1500 bytes (40 bytes of TCP/IP header data and 1460 bytes of TCP payload). This 1500-byte value is the standard maximum length allowed by Ethernet.

	Sent time	ACK received time	Segment Length
Segment 1			
Segment 2			
Segment 3			
Segment 4			
Segment 5			
Segment 6			

7. What is the minimum amount of available buffer space advertised at the receiver for the entire trace? Does the lack of receiver buffer space ever throttle the sender?
8. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

   One way to answer this question is to use the *Statistics -> TCP Segment Graphs -> Time Sequence (Stevens)* graph of this trace, all sequence numbers from the source (192.168.1.102) to the destination (128.119.245.12) are increasing monotonically with respect to time (**if you see a flat line, you need to change the direction**). If there is a retransmitted segment, the sequence number of this retransmitted segment should be smaller than those of its neighboring segments.

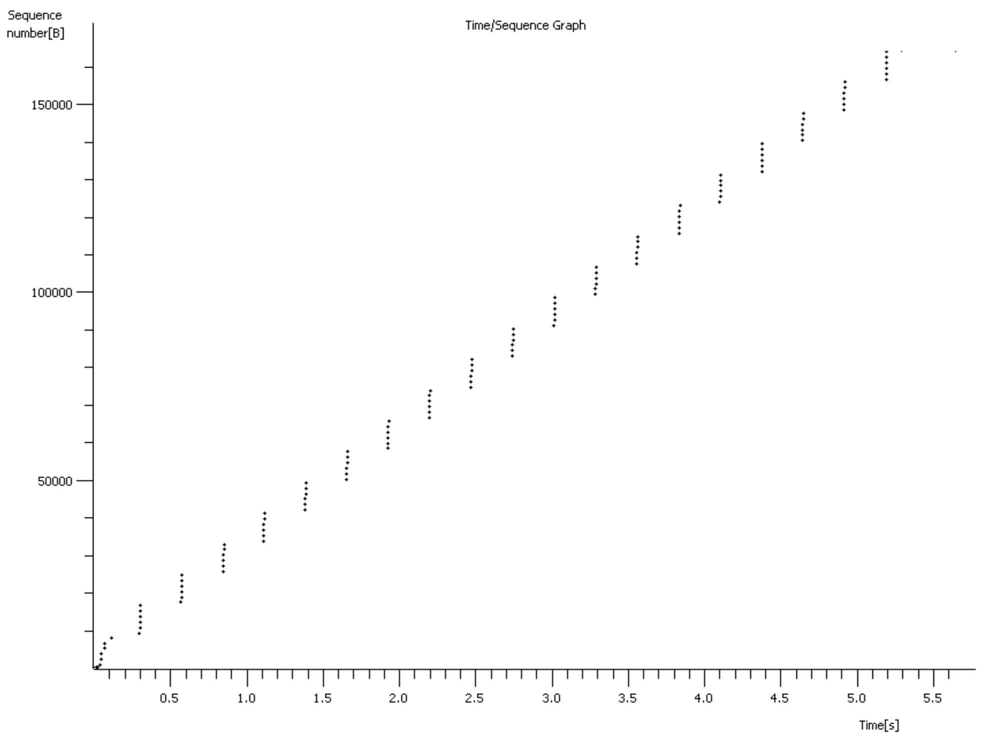
9. Considering the first 6 ACKnowledgements, how much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment.

	acknowledged sequence number	acknowledged data
ACK 1		
ACK 2		
ACK 3		
ACK 4		
ACK 5		
ACK 6		

3. TCP congestion control in action

Let's now examine the amount of data sent per unit time from the client to the server. Rather than (tediously!) calculating this from the raw data in the Wireshark window, we'll use one of Wireshark's TCP graphing utilities - *Time-Sequence-Graph (Stevens*) - to plot out data.

Select a TCP segment in the Wireshark's "listing of captured-packets" window. Then select the menu : *Statistics->TCP Stream Graph-> Time Sequence (Stevens*). You should see a plot that looks similar to the following plot, which was created from the captured packets in the packet trace *tcp-ethereal-trace-1* in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip **(if you see a flat line, you need to change the direction)**:



Here, each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent. Note that a set of dots stacked above each other represents a series of packets that were sent back-to-back by the sender.

Now let's see how can we identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over?

TCP Slow Start begins at the start of the connection, i.e., when the HTTP POST segment is sent out. The identification of the TCP slow start phase and congestion avoidance phase depends on the value of the congestion window size of this TCP sender. However, the value of the congestion window size cannot be obtained directly from the Time-Sequence-Graph (Stevens) graph. Nevertheless, we can estimate the lower bound of the TCP window size by the amount of outstanding data because the outstanding data is the amount of data without acknowledgement. We also know that TCP window is constrained by the receiver window size and the receiver buffer can act as the upper bound of the TCP window size. In this trace, the receiver buffer is not the bottleneck; therefore, this upper bound is not quite useful to infer the TCP window size. Hence, we focus on the lower bound of the TCP window size.

From the following table, we cannot see that the amount outstanding data increases quickly at the start of this TCP flow; however, it never exceeds 8192 Bytes. Therefore, we can ensure that the TCP window size is larger than 8192 Bytes. Nevertheless, we cannot determine the end of the slow start phase and the start of the congestion avoidance phase for this trace. The major reason is that this TCP sender is not sending data aggressively enough to push to the congestion state. By inspecting the amount of outstanding data, we can observe that the application at most sends out a data block of 8192 bytes. Before it receives the acknowledgement for the whole block of these 8192 bytes, the application will not send more data. It indicates before the end of the slow start phase; the application already stops transmission temporally.

Type	No.	Seq.	ACKed seq.	Outstanding data
Data	4	1		565
Data	5	566		2025
ACK	6		566	1460
Data	7	2026		2920
Data	8	3486		4380
ACK	9		2026	2920
Data	10	4946		4380
Data	11	6406		5840
ACK	12		3486	4380
Data	13	7866		5527
ACK	14		4096	4917
ACK	15		6006	3007
ACK	16		7866	1147
ACK	17		9013	0
Data	18	9013		1460
Data	19	10473		2920
Data	20	11933		4380
Data	21	13393		5840
Data	22	14853		7300

Data	23	16313		8192
ACK	24		10473	6732
ACK	25		11933	5272
ACK	26		13393	3812
ACK	27		14853	2352
ACK	28		16313	892
ACK	29		17205	0
Data	30	17205		1460
Data	31	18665		2920
Data	32	20125		4380
Data	33	21585		5840
Data	34	23045		7300
Data	35	24505		8192
ACK	36		18665	6732
ACK	37		20125	5272
ACK	38		21585	3812
ACK	39		23045	2352
ACK	40		24505	892
ACK	41		25397	0
Data	42	25397		1460
Data	43	26857		2920
Data	44	28317		4380
Data	45	29777		5840
Data	46	31237		7300
Data	47	32697		8192
ACK	48		26857	
ACK	49		28317	
ACK	50		29777	
ACK	51		31237	
ACK	52		33589	
Data	53	33589		6732
Data	54	35049		5272
Data	55	36509		3812
Data	56	37969		2352
Data	57	39429		892
Data	58	40889		0
ACK	59		35049	6732
ACK	60		37969	3812
ACK	61		40889	892
ACK	62		41781	0
Data	63	41781		1460
Data	64	43241		2920
Data	65	44701		4380
Data	66	46161		5840
Data	67	47621		7300
Data	68	49081		8192
ACK	69		44701	5272

ACK	70		47621	2352
ACK	71		49973	0
Data	72	49973		1460
Data	73	51433		2920
Data	74	52893		4380
Data	75	54353		5840
Data	76	55813		7300
Data	77	57273		8192
ACK	78		52893	5272
ACK	79		55813	2352
ACK	80		58165	0
Data	81	58165		

Note that the criteria to determine the end of slow start and the beginning of the congestion avoidance is the way how congestion window size reacts to the arrival of ACKs. Upon an ACK arrival, if the congestion window size increases by one MSS, TCP sender still stays in the slow start phase. In the congestion avoidance phase, the congestion window size increases at 1/(current_congestion_window_size). By inspecting the change of the congestion window upon the arrival of ACKs, we can infer the states of the TCP sender.

10. Comment on ways in which the measured data differs from the assumed behavior of TCP discussed in the lectures?

PART B

In this part, we'll take a quick look at the UDP transport protocol. UDP is a streamlined, no-frills protocol. Because UDP is simple and sweet, we'll be able to cover it pretty quickly in this lab.

At this stage, you should be a Wireshark expert. Thus, we are not going to spell out the steps as explicitly as in earlier labs. In particular, we are not going to provide example screenshots for all the steps.

Essentially, it is likely that just by doing nothing (except capturing packets via Wireshark) that some UDP packets sent by others will appear in your trace. In particular, the Simple Network Management Protocol (SNMP) sends SNMP messages inside of UDP, so it's likely that you'll find some SNMP messages (and therefore UDP packets) in your trace.

Download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract the file **http-ethereal-trace-5**, which contains some UDP packets carrying SNMP messages. Then set your packet filter so that Wireshark only displays the UDP packets sent and received at your host. Pick one of these UDP packets, expand the UDP fields in the details window and answer the following questions:

Whenever possible, when answering a question below, you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. **Annotate** the printout to explain your answer. To print a packet, use *File->Print*, choose *Selected packet only*, choose *P acket summary line,* and select the minimum amount of packet detail that you need to answer the question.

11. Select *one* UDP packet from your trace. From this packet, determine how many fields there are in the UDP header. (Answer these questions directly from what you observe in the packet trace.) Name these fields.
12. By consulting the displayed information in Wireshark's packet content field for this packet, determine the length (in bytes) of each of the UDP header fields.
13. The value in the Length field is the length of what? Verify your claim with your captured UDP packet.
14. What is the maximum number of bytes that can be included in a UDP payload? (Hint: the answer to this question can be determined by your answer to 2. above)
15. What is the largest possible source port number? (Hint: see the hint in 4.)
16. What is the protocol number for UDP? Give your answer in both hexadecimal and decimal notation. To answer this question, you'll need to look into the Protocol field of the IP datagram containing this UDP segment.
17. Examine a pair of UDP packets in which your host sends the first UDP packet and the second UDP packet is a reply to this first UDP packet. (Hint: for a second packet to be sent in response to a first packet, the sender of the first packet should be the destination

of the second packet). Describe the relationship between the port numbers in the two packets.

Why can an IP packet be 65535 bytes when Ethernet can only send 1500?

Ethernet is one of several physical layers that can transport IP. The size of the packet a physical layer can transport is specific to this physical layer. Other physical layers have other properties. The 65535 bytes limit in IP is because the length field in the IP header is only 16 bits.

The applications send data to the OS kernel, which then packetizes the data. The data size that can be sent to the kernel at once depends on the size of the socket buffer, which results in the packet size you see. Additionally, the kernel might further split the data to fit best the maximum size of the underlying physical layer (like Ethernet). With TCP, the kernel might also decide to join small data so that they get transmitted together.

PART C

In this part, we'll investigate the IP protocol, focusing on the IP datagram. We'll do so by analyzing a trace of IP datagrams sent and received by an execution of the `traceroute` program.

Before beginning this lab, you'll probably want to review section 3.4 of RFC 2151 [ftp://ftp.rfc-editor.org/in-notes/rfc2151.txt] to update yourself on the operation of the `traceroute` program. You'll also want to read RFC 791 [ftp://ftp.rfc-editor.org/in-notes/rfc791.txt] on hand as well, for a discussion of the IP protocol.

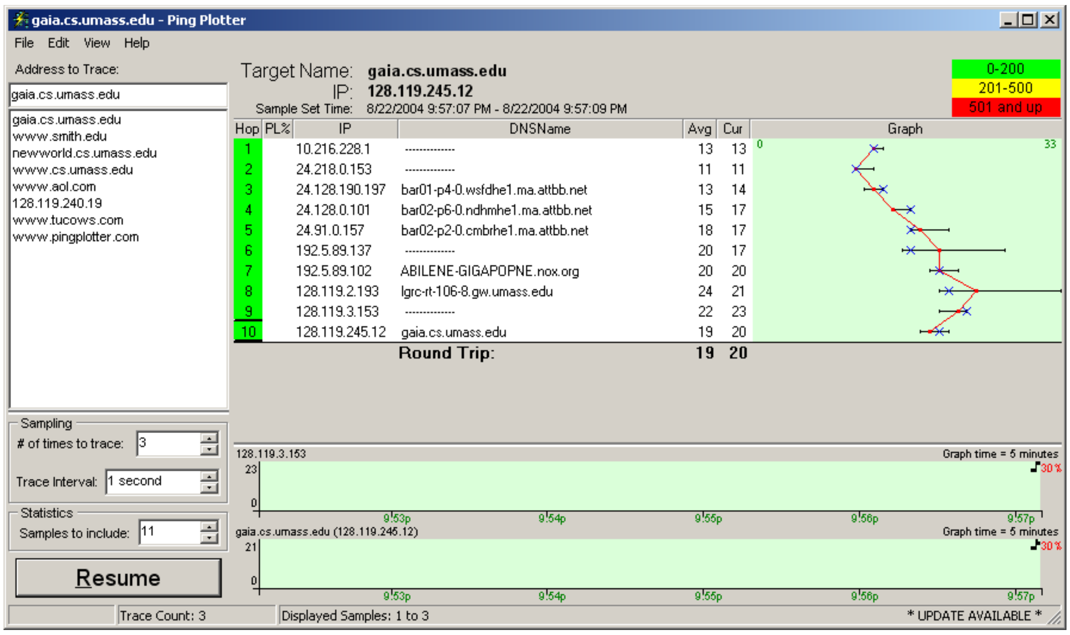
1. Capturing packets from an execution of traceroute

Trace of IP datagrams for this part was generated using the `traceroute` program that sends datagrams of different sizes towards some destination, *X*. `traceroute` operates by first sending one or more datagrams with the time-to-live (TTL) field in the IP header set to 1; it then sends a series of one or more datagrams towards the same destination with a TTL value of 2; it then sends a series of datagrams towards the same destination with a TTL value of 3; and so on. Recall that a router must decrement the TTL in each received datagram by 1 (actually, RFC 791 says that the router must decrement the TTL by *at least* one). If the TTL reaches 0, the router returns an ICMP message (type 11 – TTL-exceeded) to the sending host. As a result of this behavior, a datagram with a TTL of 1 (sent by the host executing `traceroute`) will cause the router one hop away from the sender to send an ICMP TTL-exceeded message back to the sender; the datagram sent with a TTL of 2 will cause the router two hops away to send an ICMP message back to the sender; the datagram sent with a TTL of 3 will cause the router three hops away to send an ICMP message back to the sender; and so on. In this manner, the host executing `traceroute` can learn the identities of the routers between itself and destination *X* by looking at the source IP addresses in the datagrams containing the ICMP TTL-exceeded messages.

To run `traceroute` and have it send datagrams of various lengths, we would normally do the following (You do not have to perform these steps, we just demonstrate the real world application here. You still need to read through, though):

- **Windows.** The `tracert` program (used for our ICMP Wireshark lab) provided with Windows does not allow one to change the size of the ICMP echo request (ping) message sent by the `tracert` program. A nicer Windows `traceroute` program is *pingplotter*, available both in free version and shareware versions at http://www.pingplotter.com. Download and install *pingplotter*, and test it out by performing a few traceroutes to your favorite sites. The size of the ICMP echo request message can be explicitly set in *pingplotter* by selecting the menu item *Edit-> Options->Packet Options* and then filling in the *Packet Size* field. The default packet size is 56 bytes. Once *pingplotter* has sent a series of packets with the increasing TTL values, it restarts the sending process again with a TTL of 1, after waiting *Trace Interval* amount of time. The value of *Trace Interval* and the number of intervals can be explicitly set in *pingplotter*.

That said, if you are using a Windows platform, you could start up *pingplotter* and enter the name of a target destination in the "Address to Trace Window." Enter 3 in the "# of times to Trace" field, so you don't gather too much data. Select the menu item *Edit->Advanced Options->Packet Options* and enter a value of 56 in the *Packet Size* field and then press OK. Then press the Trace button. You should see a *pingplotter* window that looks something like this:



Target Name: gaia.cs.umass.edu
IP: 128.119.245.12
Sample Set Time: 8/22/2004 9:57:07 PM - 8/22/2004 9:57:09 PM

Hop	PL%	IP	DNSName	Avg	Cur
1		10.216.228.1	-------------	13	13
2		24.218.0.153	-------------	11	11
3		24.128.190.197	bar01-p4-0.wsfdhe1.ma.attbb.net	13	14
4		24.128.0.101	bar02-p6-0.ndhmhe1.ma.attbb.net	15	17
5		24.91.0.157	bar02-p2-0.cmbrhe1.ma.attbb.net	18	17
6		192.5.89.137	-------------	20	17
7		192.5.89.102	ABILENE-GIGAPOPNE.nox.org	20	20
8		128.119.2.193	lgrc-rt-106-8.gw.umass.edu	24	21
9		128.119.3.153	-------------	22	23
10		128.119.245.12	gaia.cs.umass.edu	19	20
			Round Trip:	**19**	**20**

# of times to trace: 3
Trace Interval: 1 second
Samples to include: 11
Trace Count: 3
Displayed Samples: 1 to 3

Next, send a set of datagrams with a longer length, by selecting *Edit->Advanced Options->Packet Options* and enter a value of 2000 in the *Packet Size* field and then press OK. Then press the Resume button.

Finally, send a set of datagrams with a longer length, by selecting *Edit->Advanced Options->Packet Options* and enter a value of 3500 in the *Packet Size* field and then press OK. Then press the Resume button.

- **Linux/Unix/MacOS.** With the Unix/MacOS `traceroute` command, the size of the UDP datagram sent towards the destination can be explicitly set by indicating the number of bytes in the datagram; this value is entered in the `traceroute` command line immediately after the name or address of the destination. For example, to send `traceroute` datagrams of 2000 bytes towards gaia.cs.umass.edu, the command would be:

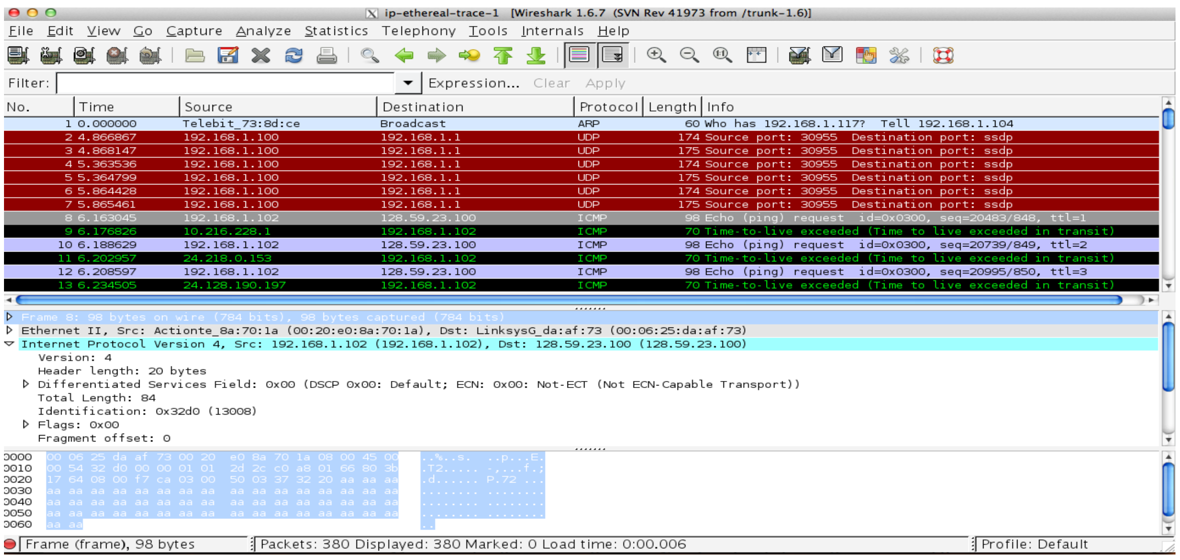
```
%traceroute gaia.cs.umass.edu 2000
```

Now download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract the ***ip-ethereal-trace-1*** packet trace file that was captured while following the steps above.

2. A look at the provided captured trace

In your trace, you should be able to see the series of ICMP Echo Request (in the case of Windows machine) or the UDP segment (in the case of Unix) sent by our computer and the ICMP TTL-exceeded messages returned to our computer by the intermediate routers. In the questions below, we'll use a trace collected on a Windows machine. Whenever possible, when answering a question below you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. When you hand in your assignment, annotate the output so that it's clear where in the output you're getting the information for your answer (e.g., for our classes, we ask that students markup paper copies with a pen, or annotate electronic copies with text in a colored font).To print a packet, use *File->Print*, choose *Selected packet only*, choose *Packet summary line,* and select the minimum amount of packet detail that you need to answer the question.

18. Select the first ICMP Echo Request message sent by our computer, and expand the Internet Protocol part of the packet in the packet details window.


ip-ethereal-trace-1 [Wireshark 1.6.7 (SVN Rev 41973 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply

```
No.  Time      Source            Destination     Protocol Length Info
  1 0.000000   Telebit_73:8d:ce  Broadcast       ARP       60 Who has 192.168.1.117?  Tell 192.168.1.104
  2 4.866867   192.168.1.100     192.168.1.1     UDP      174 Source port: 30955  Destination port: ssdp
  3 4.868147   192.168.1.100     192.168.1.1     UDP      175 Source port: 30955  Destination port: ssdp
  4 5.363536   192.168.1.100     192.168.1.1     UDP      174 Source port: 30955  Destination port: ssdp
  5 5.364799   192.168.1.100     192.168.1.1     UDP      175 Source port: 30955  Destination port: ssdp
  6 5.864428   192.168.1.100     192.168.1.1     UDP      174 Source port: 30955  Destination port: ssdp
  7 5.865461   192.168.1.100     192.168.1.1     UDP      175 Source port: 30955  Destination port: ssdp
  8 6.163045   192.168.1.102     128.59.23.100   ICMP      98 Echo (ping) request  id=0x0300, seq=20483/848, ttl=1
  9 6.176826   10.216.228.1      192.168.1.102   ICMP      70 Time-to-live exceeded (Time to live exceeded in transit)
 10 6.188629   192.168.1.102     128.59.23.100   ICMP      98 Echo (ping) request  id=0x0300, seq=20739/849, ttl=2
 11 6.202957   24.218.0.153      192.168.1.102   ICMP      70 Time-to-live exceeded (Time to live exceeded in transit)
 12 6.208597   192.168.1.102     128.59.23.100   ICMP      98 Echo (ping) request  id=0x0300, seq=20995/850, ttl=3
 13 6.234505   24.128.190.197    192.168.1.102   ICMP      70 Time-to-live exceeded (Time to live exceeded in transit)

▷ Frame 8: 98 bytes on wire (784 bits), 98 bytes captured (784 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 (192.168.1.102), Dst: 128.59.23.100 (128.59.23.100)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0x32d0 (13008)
  ▷ Flags: 0x00
    Fragment offset: 0

0000  00 06 25 da af 73 00 20  e0 8a 70 1a 08 00 45 00   ..%..s. ..p...E.
0010  00 54 32 d0 00 00 01 01  2d 2c c0 a8 01 66 80 3b   .T2..... -,...f.;
0020  17 64 08 00 f7 ca 03 00  50 03 37 32 20 aa aa aa   .d...... P.72 ...
0030  aa aa aa aa aa aa aa aa  aa aa aa aa aa aa aa aa   ........ ........
0040  aa aa aa aa aa aa aa aa  aa aa aa aa aa aa aa aa   ........ ........
0050  aa aa aa aa aa aa aa aa  aa aa aa aa aa aa aa aa   ........ ........
0060  aa aa                                              ..

Frame (frame), 98 bytes | Packets: 380 Displayed: 380 Marked: 0 Load time: 0:00.006 | Profile: Default
```

What is the IP address of the computer?

19. Within the IP packet header, what is the value in the upper layer protocol field?
20. How many bytes are in the IP header? How many bytes are in the payload *of the IP datagram*? Explain how you determined the number of payload bytes.

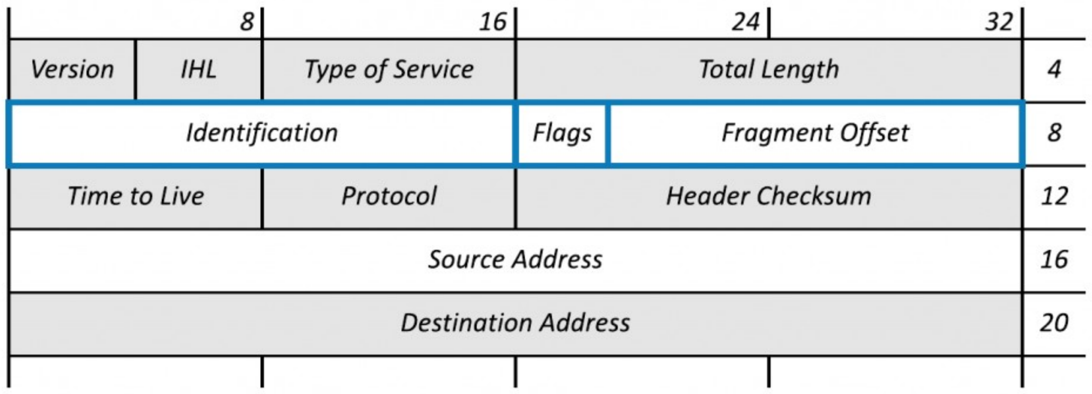
3. Fragmentation

Let's quickly overview how IP fragmentation works (src: https://packetpushers.net/ip-fragmentation-in-detail/). When a host sends an IP packet onto the network it cannot be larger than the maximum size supported by that local network. This size is determined by the network's data link and IP Maximum Transmission Units (MTUs) which are usually the same.

A typical contemporary office, campus or data centre network provided over Ethernet will have 1500 byte MTUs.

However, packets that are initially transmitted over a network supporting one MTU may need be routed across networks (such as a WAN or VPN tunnel) with a smaller MTU. In these cases, if the packet size exceeds the lower MTU the data in the packet must be fragmented (if possible). This means it is broken into pieces carried within new packets (fragments) that are equal to or smaller than the lower MTU. This is called Fragmentation and the data in these fragments is then typically reassembled when they reach their destination.

The processes of fragmentation and reassembly involve a number of IP header fields being set in the fragments. Here's a reminder of all the fields and their order, with fragmentation headers highlighted:

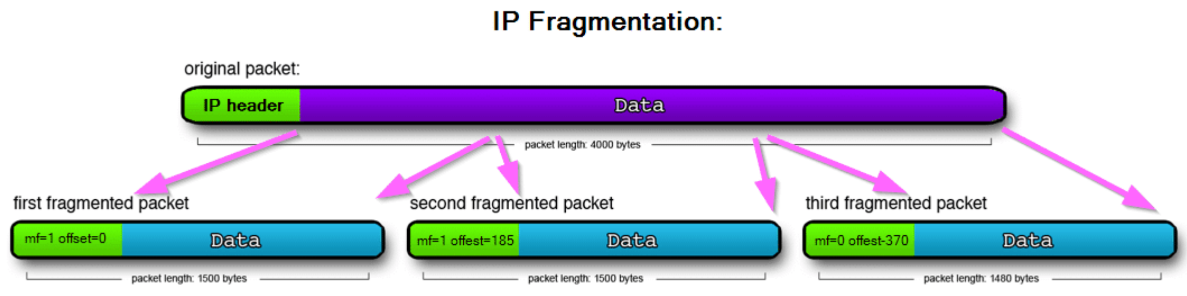


Fragmentation's operation relies upon three IP header fields (32 bits in total), all of which will have very different values in the fragments compared to the original packet:

- The **Identification** field (16 bits) is populated with an ID number unique for the combination of source & destination addresses and **Protocol** field value of the original packet, allowing the destination to distinguish between the fragments of different packets (from the same source). This does not mean the same ID should be used when fragmenting packets where the source, destination and protocol are the same but that the same ID could be used when they are not.
- Like the original packet, the first, reserved bit of the **Flags** field (3 bits) will be 0 (unset) and the second bit, **Don't Fragment (DF)**, will also be unset. Unlike the original packet, all but the last fragment will have the third bit of the field, **More Fragments (MF)**, set to 1. The last packet will have all bits in this field set to 0 just like the original packet (unless it was a fragment itself). If the Don't Fragment flag *was* set in the original packet, this prevents fragmentation and results in packets that require it being discarded. An ICMP error of type 3: 'Destination Unreachable', code 4: 'Fragmentation required, and DF set' should be sent to the sender. See the following PMTUD section for more on this.
- The **Fragment Offset** field (13 bits) is used to indicate the starting position of the data in the fragment in relation to the start of the data in the original packet. This information is used to reassemble the data from all the fragments (whether they arrive in order or not). In the first fragment the offset is 0 as the data in this packet starts in the same place

as the data in the original packet (at the beginning). In subsequent fragments, the value is the offset of the data the fragment contains from the beginning of the data in the first fragment (offset 0), in 8 byte 'blocks' (aka octawords). If a packet containing 800 bytes of data is split into two equal fragments carrying 400 bytes of data, the fragment offset of the first fragment is 0, of the second fragment 50 (400/8). The offset value must be the number of 8 byte blocks of data, which means the data in the prior fragment must be a multiple of 8 bytes. The last fragment can carry data that isn't a multiple of 8 bytes as there won't be a further fragment with an offset that must meet the 8 byte 'rule'.

In the following figure, we provide an example of how IP fragmentation works in practice:



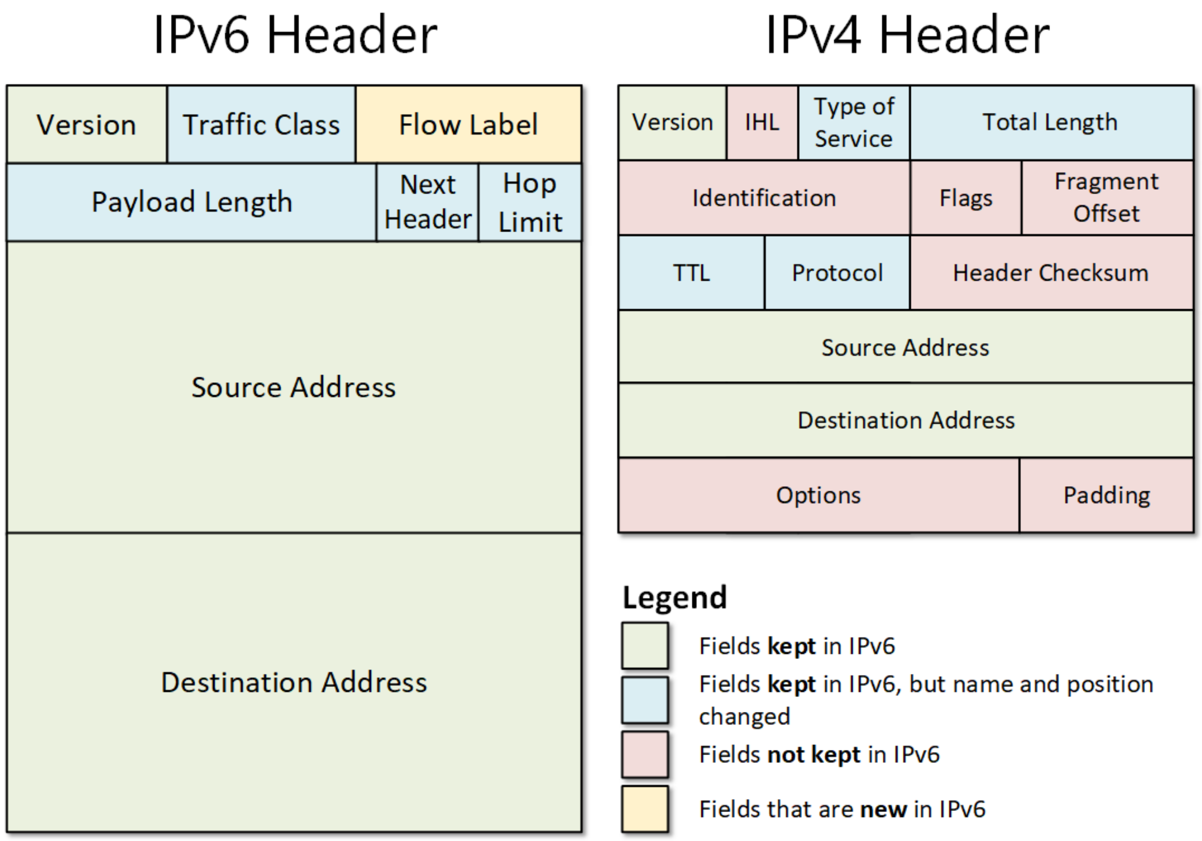
21. Considering the above mentioned, has the IP datagram been fragmented? Explain how you determined whether or not the datagram has been fragmented.

Sort the packet listing according to time again by clicking on the *Time* column.

22. Find the first ICMP Echo Request message that was sent by our computer after we changed the *Packet Size* in *pingplotter* to be 2000. Has that message been fragmented across more than one IP datagram? [Note: if your computer has an Ethernet interface, a packet size of 2000 *should* cause fragmentation.]
23. Print out the first fragment of the fragmented IP datagram. What information in the IP header indicates that the datagram been fragmented? What information in the IP header indicates whether this is the first fragment versus a latter fragment? How long is this IP datagram?
24. Print out the second fragment of the fragmented IP datagram. What information in the IP header indicates that this is not the first datagram fragment? Are there more fragments? How can you tell?
25. What fields change in the IP header between the first and second fragment?

PART D

IPv6 is using two main types of headers: Main IPv6 Header and the new IPv6 Extension Headers. The main IPv6 header is equivalent to the IPv4 one with some field differences introduced for better efficiency. The figure below compares both headers.



(https://www.networkacademy.io/sites/default/files/inline-images/comparing-ipv4-and-ipv6-headers.png)

Note that the IPv6 header has fewer fields which makes it more efficient and faster to process. Another big advantage is that the header length is fixed size 40 bytes, comparing to the variable length size of the IPv4 header.

1. Basic IPv6 Exercise

In this exercise, we are going to use cloudshark. Cloudshark enables you to share PCAPs with a larger audience or access them from anywhere in the world. Using CloudShark means you don't need any software other than a web browser to do packet analysis! Access the pcap file containing ICMPv6 echos across an IPv6-in-IP tunnel via https://www.cloudshark.org/captures/2eb8c48dcfd6

Tunneling provides a way to use an existing IPv4 routing infrastructure to carry IPv6 traffic. The key to a successful IPv6 transition is compatibility with the existing installed base of IPv4 hosts and routers. Maintaining compatibility with IPv4 while deploying IPv6 streamlines the task of transitioning the Internet to IPv6. While the IPv6 infrastructure is being deployed, the existing IPv4routing infrastructure can remain functional, and can be used to carry IPv6 traffic. IPv6 or IPv4 hosts and routers can tunnel IPv6 datagrams over regions of IPv4 routing topology by encapsulating them within IPv4 packets.

The entry node of the tunnel (the encapsulating node) creates an encapsulating IPv4 header and transmits the encapsulated packet. The exit node of the tunnel (the decapsulating node) receives the encapsulated packet, removes the IPv4 header, updates the IPv6 header, and processes the received IPv6 packet. However, the encapsulating node needs to maintain soft state information for each tunnel, such as the maximum transmission unit (MTU) of the tunnel, to process IPv6 packets forwarded into the tunnel.

Using the traffic trace, answer the following questions:

26. What are the IPv4 source and destination IP addresses?
27. What is the size of the IPv4 header? What is the size of the payload?
28. At which layer in the trace file is the IPv5 address located?
29. What are the IPv6 source and destination IP addresses?
30. What is the size of the IPv6 header? What is the size of the payload?
31. What is the destination port number?
32. What message type is being carried (encapsulated) in the IPv6 message? What is the code type of this message?
33. Is the traffic trace bidirectional?