Brian Landy

CMPE 670 Exercise #2

Encapsulation and Packets Structure

Due Tuesday 12/11/2019

Assumptions:

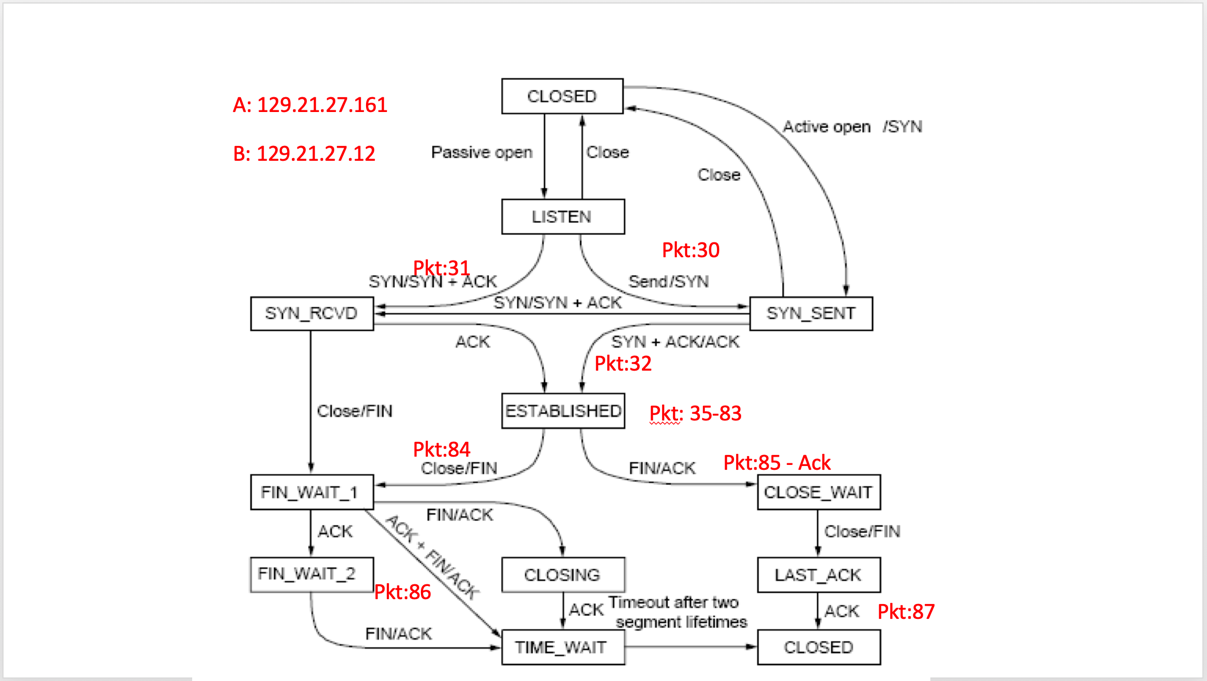
Seq and acks are non-relative in Wireshark (Setting change)

Questions:

1. Q1 PCAP analysis:
   1. (10 pts) By examining the information in the link layer header, indicate whether the network interface from where traffic was captured was wireless (a wifi interface) or wired (an Ethernet interface).
      1. By clicking on any packet and looking at the second layer (link) you can see that the specified method is ethernet II. This is a wired connection.
   2. (10 pts) Create a table where you will list all the different types of packets/protocols you find in the pcap file. For each entry in the table, a second column has to indicate what are the layers that are being encapsulated in each packet type. Hint: the “Protocol” column in the Wireshark window that shows all packets (the top one) does not provide enough information to answer this question (and it may be at times misleading); a better approach is to analyze the information in the middle window (the “Packet Details” window).
      1. Here is my list of packets and protocols:
         1. I used the “frame details->protocols in frame” section to get info

|  |  |
| --- | --- |
| Protocol/Packet Type | Layers being encapsulated |
| UDP | UDP is a protocol that is part of the transport layer |
| IPV6 | IPv6 is a protocol that belongs to the network layer like ipv4 |
| Ethernet 2 | This encapsulates the physical layer and datalink layer |
| Dhcpv6 | Dynamic host config protocol for ipv6 is part of the |
| Arp | Address resolution protocol is part of the network layer |
| Spanning Tree | This is a protocol that is part of the Data link layer |
| Logical Link Control | This is a protocol that is part of the Data link layer |
| IP | IPv4 protocol -> Part of the Network Layer |
| TCP | TCP is a protocol that part of the transport layer |
| Data | This data packet is a udp/tcp data packet meaning it is part of the transport layer. |

* 1. (10 pts) Examine the ARP (Address Resolution Protocol) exchange in packets 28 and 29 and explain what is happening with this exchange. Also, indicate whether the source and destination addresses are MAC or IP addresses and explain why.
     1. An ARP exchange is one where a node will broadcast that it is trying to find which piece of hardware has a certain IP. Here, the source node (Intel\_ea) knows the IP but not the MAC of the destination. The owner of this IP (Hewlett\_P) will send its MAC directly back to the source node who broadcasted the request.
     2. The source and destination addresses are MAC addresses. This protocol is used specifically to identify which MAC addresses correspond to certain IP addresses. IPs are used for identification but the MAC addresses at the source and destination are what are used to make the connection, so MAC is the address type.
  2. (20 pts) Most of the packets in the pcap file correspond to a TCP connection where a file is sent from one host to another (the file exchange is NOT using FTP, SFTP or any other standardized file exchange protocol). Identify the packets from this connection (by their packet number) and indicate the IPv4 addresses for the two hosts involved in the exchange. Use the TCP state transition diagram to record the setup and ending of the connection. For this, indicate on a state transition diagram the packet number that corresponds to each transition (use the provided powerpoint file for this). Use this work to identify next the IPv4 address for the host sending the file (the one starting the connection) and the host receiving the file.
     1. At packet 30, the TCP state diagram can finally be filled out. The prior packets being sent were unrelated to the connection between the 2 hosts. The packets that correspond to this messaging are from packet #30 to 87 with the exclusion of 33 and 34. The two hosts involved in the exchange are 129.21.27.161 and 129.21.27.12. The setup and ending is outlined in the TCP state diagram below.



1. The second part of this exercise involves creating your own code to parse through the packets (doing “de-encapsulating”) and extract information. Note: you are free to choose the system where to work (MS Windows, Mac, Ubuntu, etc.) but keep in mind that in some cases you may need to swap the endianness of data you are reading. The provided code includes two functions to do this if needed.
   1. Your first task is to compile and run the skeleton code as provided. To compile the code in a MS Windows environment, I strongly recommend the use of MinGW. Type “gcc skeletoncode.c -o skeletoncode” and run the code doing “skeletoncode exercise1.pcap”. You should find a new file “outdata.txt”, with content like the following:

packet: 24, type: 8

packet: 30, type: 8

packet: 31, type: 8

packet: 32, type: 8

packet: 35, type: 8

...... same pattern follows for packets 36 through 84 ......

packet: 85, type: 8

packet: 86, type: 8

packet: 87, type: 8

Done. Adjustments made for working on a 64bit system. Outdata matches expected result.

* 1. (10 pts) Study the function “int isgetTCPIP(BYTE \*pcktbuf, u\_int \*size\_ip, u\_int \*size\_tcp)”
* What is the information coded in the struct sniff\_ethernet? What is the function and how does it work the line “ethernet = (struct sniff\_ethernet\*)(pcktbuf);”?
  + The sniff\_ethernet struct contains ethernet header information. It holds source and destination addresses (MAC) as well as the type of data being transmitted in this ethernet packet. The line ethernet = … sets the address of ethernet to the address of pcktbuf and casts it to the sniff\_ethenet struct. Now the information in the pcktbuf can be accessed by accessing struct parameters from sniff\_ethernet.
* Do you see within the same function other examples of the same type of programming (type casting a pointer)?
  + Yes, the same steps are followed for ip, tcp and data information in the packet.
* Explain how this function modifies pointers that effectively allow for parsing through the packet, undoing the encapsulation process done during the packet transmission.
  + To access further layers of the pcktbuf, you must cast the data again to different structures at deeper points in the buffer. To get ip header information you use sniff\_ip and have cast pcktbuf at an offset of SIZE\_ETHERNET. This is because everything from 0 to SIZE\_ETH was for the Ethernet header. The ip header is what follows. This same casting technique now makes the ip information available via sniff\_ip. It happens again for tcp specific header data and then the actual data of the packet. You would keep adding increments to get the memory you want. TCP is after SIZE\_ETH+SIZE\_IP and the data is after SIZE\_ETH+SIZE\_IP+SIZE\_TCP. Different protocols will require different offset locations for headers in the packet. This is how you de-encapsulate the information.
* Enumerate from which protocols is this function separating the headers and from which layers these protocols are.
  + This function separates out 4 layers of the packet. They are the Ethernet header (from the physical layer and data link layer), the ip header (from the network layer), the tcp header (from the transport layer) and data section of the packet (from the transport layer).

* 1. (10 pts) Create a new version of the skeleton code which outputs to outdata.txt the following information for each TCP packet:
* -  Packet number
* -  Source IPv4 address and port number.
* -  Destination IPv4 address and port number.
* -  Sequence number.
* -  Acknowledgement number.

Show the content of outdata.txt (Note: IPv4 addresses shall be in the standard format of four 8-bit decimal numbers separated by dots).

Here is the output of my program. In Wireshark I needed to turn off TCP relative seq and ack nums

packet: 24, Source IPv4: 182.69.137.186 and port: 44478, Dest IPv4: 129.21.27.181 and Port: 23, Seq Num: 2165644213, Ack Num: 0

packet: 30, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959304545, Ack Num: 0

packet: 31, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951210, Ack Num: 959304546

packet: 32, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959304546, Ack Num: 1347951211

packet: 35, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959304546, Ack Num: 1347951211

packet: 36, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959304808

packet: 37, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959304808, Ack Num: 1347951211

packet: 38, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959306256

packet: 39, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959306256, Ack Num: 1347951211

packet: 40, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959307704

packet: 41, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959307704, Ack Num: 1347951211

packet: 42, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959313496

packet: 43, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959313496, Ack Num: 1347951211

packet: 44, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959319288

packet: 45, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959319288, Ack Num: 1347951211

packet: 46, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959321052

packet: 47, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959321052, Ack Num: 1347951211

packet: 48, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959326844

packet: 49, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959326844, Ack Num: 1347951211

packet: 50, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959331188

packet: 51, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959331188, Ack Num: 1347951211

packet: 52, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959335532

packet: 53, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959335532, Ack Num: 1347951211

packet: 54, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959340440

packet: 55, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959340440, Ack Num: 1347951211

packet: 56, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959343336

packet: 57, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959343336, Ack Num: 1347951211

packet: 58, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959347680

packet: 59, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959347680, Ack Num: 1347951211

packet: 60, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959350576

packet: 61, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959350576, Ack Num: 1347951211

packet: 62, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959354920

packet: 63, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959354920, Ack Num: 1347951211

packet: 64, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959362160

packet: 65, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959362160, Ack Num: 1347951211

packet: 66, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959367952

packet: 67, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959367952, Ack Num: 1347951211

packet: 68, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959373744

packet: 69, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959373744, Ack Num: 1347951211

packet: 70, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959379536

packet: 71, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959379536, Ack Num: 1347951211

packet: 72, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959385328

packet: 73, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959385328, Ack Num: 1347951211

packet: 74, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959388386

packet: 75, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959388386, Ack Num: 1347951211

packet: 76, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959392730

packet: 77, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959392730, Ack Num: 1347951211

packet: 78, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959397074

packet: 79, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959397074, Ack Num: 1347951211

packet: 80, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959401418

packet: 81, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959401418, Ack Num: 1347951211

packet: 82, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959401748

packet: 83, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951211, Ack Num: 959401748

packet: 84, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951229, Ack Num: 959401748

packet: 85, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959401748, Ack Num: 1347951229

packet: 86, Source IPv4: 129.21.27.161 and port: 32856, Dest IPv4: 129.21.27.12 and Port: 4444, Seq Num: 959401748, Ack Num: 1347951230

packet: 87, Source IPv4: 129.21.27.12 and port: 4444, Dest IPv4: 129.21.27.161 and Port: 32856, Seq Num: 1347951230, Ack Num: 959401749

* 1. (20 pts) Consider next only the packets involved in the main TCP connection mentioned in question 1c) and of these packets only those that contain fragments of the file that is being transferred. Create a new version of the skeleton code which outputs the payload size in bytes and the sequence number. Do some further calculations with the sequence numbers to calculate the payload size of each packet and verify that the results equal the payload sizes obtained from your code that parses through exercise1.pcap. Explain the procedure that you followed and show your results. Also, use your results to calculate the size (in bytes) of the file that is transferred in the TCP connection. Hint: there are some fields in the packet headers that you can use to derive the payload size after compensating for some overhead lengths.

Here is the output of the program that records sizes:

packet: 35, Seq Num: 959304546, PLoad Size: 262

packet: 37, Seq Num: 959304808, PLoad Size: 1448

packet: 39, Seq Num: 959306256, PLoad Size: 1448

packet: 41, Seq Num: 959307704, PLoad Size: 5792

packet: 43, Seq Num: 959313496, PLoad Size: 5792

packet: 45, Seq Num: 959319288, PLoad Size: 1764

packet: 47, Seq Num: 959321052, PLoad Size: 5792

packet: 49, Seq Num: 959326844, PLoad Size: 4344

packet: 51, Seq Num: 959331188, PLoad Size: 4344

packet: 53, Seq Num: 959335532, PLoad Size: 4908

packet: 55, Seq Num: 959340440, PLoad Size: 2896

packet: 57, Seq Num: 959343336, PLoad Size: 4344

packet: 59, Seq Num: 959347680, PLoad Size: 2896

packet: 61, Seq Num: 959350576, PLoad Size: 4344

packet: 63, Seq Num: 959354920, PLoad Size: 7240

packet: 65, Seq Num: 959362160, PLoad Size: 5792

packet: 67, Seq Num: 959367952, PLoad Size: 5792

packet: 69, Seq Num: 959373744, PLoad Size: 5792

packet: 71, Seq Num: 959379536, PLoad Size: 5792

packet: 73, Seq Num: 959385328, PLoad Size: 3058

packet: 75, Seq Num: 959388386, PLoad Size: 4344

packet: 77, Seq Num: 959392730, PLoad Size: 4344

packet: 79, Seq Num: 959397074, PLoad Size: 4344

packet: 81, Seq Num: 959401418, PLoad Size: 330

packet: 83, Seq Num: 1347951211, PLoad Size: 18

Total file transferred: 97220

Hand calculations:

In order to verify pcap, I had to do hand calculations. What I did was I looked for all of the messages containing data and that were coming from the source: 129.21.27.161. Then I looked at the sequence numbers and took a difference of the second from the first of those messages. The acks could be ignored, as the messages with payloads were what was important. From here I was able to gather the byte length of each payload sent. From what I calculated, there were no discrepancies in the programs output. This is also further verified with Wireshark. All three methods have the same results.

* 1. (10 pts) Same as in part d), consider next only the packets involved in the main TCP connection mentioned in question 1c) and of these packets only those that contain fragments of the file that is being transferred. Create a new version of the skeleton code that outputs to outdata.txt the file that is transferred. The exchanged file is a plain text file, so you should be able to open outdata.txt and read the file (it is OK if you see some very few odd characters). What is the subject of the file?

The file is RFC 791which talks about the internet protocol and is made by darpa.

RFC: 791