SOFTENG 364: Computer Networks

Assignment 1 (worth 8%)

Due April 23 at 2pm

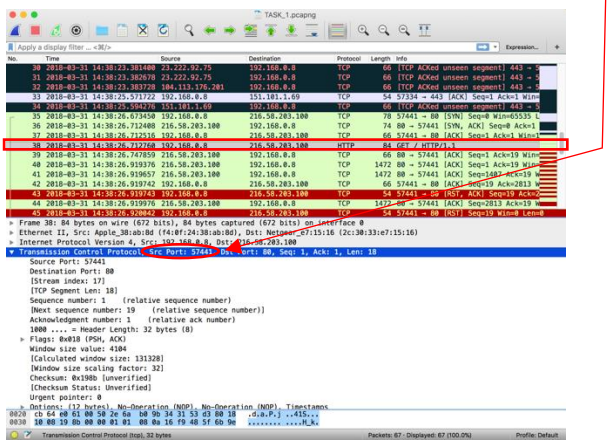
This assignment will focus on protocols in the application, transport and network layers of the Internet Protocol stack, but you will also need to dip into the link layer occasionally. You will be using Wireshark to analyse networks via various saved traces. Also you will be required to write server-client applications using Python Programming language. You'll be submitting a number of files in this assignment, so please zip them up and submit your assignment as a **zip file** via the assignment submission option in Canvas. So there will be a pdf file, your python scripts and finally your captured traffic (where required) that are submitted. Remember to include your UPI as part of the file names e.g. (moni476.py). Marks will be awarded for both technical depth and for readability. Screen shots with pertinent information should be included. Please make sure you correctly list all references used in completing your assignment. All assignments and codes will be checked for plagiarism, and will be awarded zero marks if plagiarism is detected.

Task 1. *(Please use Python 2.7 to do this Task and the following one)*

Refer to the socket program in your TCP lab. Currently, the script does not give any message if any error occurs during the creation of socket. This is not good practice, extend this script and include this error if the socket was not successfully created "Socket creation failed with error ------". Also if the host name of google cannot be resolved an error should be displayed "There was an error resolving the host" and your program should exit. Finally start Wireshark and capture the communication between the server and client. Save the capture, attach it with your submission and answer the following questions **(see attached file, TASK_1/TASK_1.pcapng), and python script crai897.py** .

a. What port number is assigned to your client

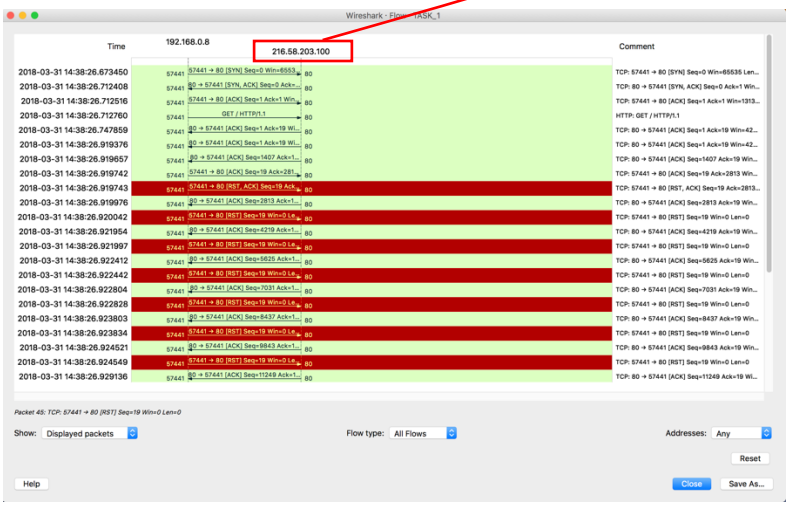
The port number assigned to my client, is 57441. This can be seen in the screenshot attached below, in regard to the HTTP packet sent specifically.



b. Filter based on the IP address of google.com, attach the flow graph of this communication alone.

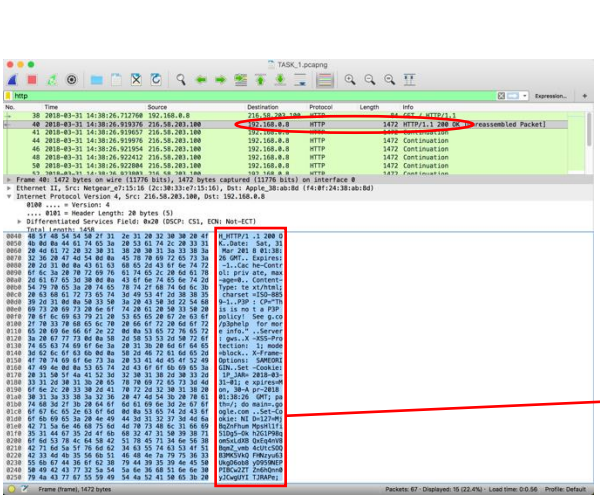
See attached file: TASK_1/TASK_1_B.pdf.

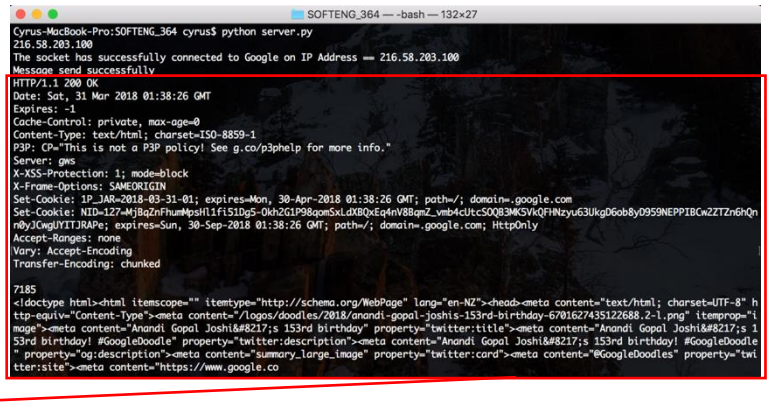
Supplementary to that, is this attached screenshot of proof:



c. Does the first TCP segment reassemble packet message correspond to what is displayed in your command prompt window, attach the two images side by side to justify your answer.

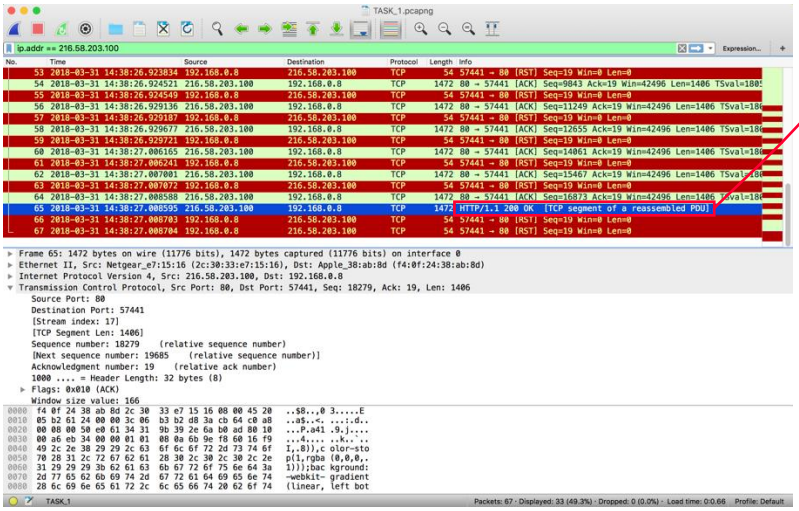
Yes; assuming by '*the first TCP segment reassemble packet message*' you meant essentially the HTTP response sent from server to client, we can clearly see the consistency, in the screenshots attached below. On the left, we can see the GET HTTP request highlighted in blue, and on the right, in the terminal, we can see the response to that same request, in the form of HTML code. On the bottom, we can see the analysis of the packet in WireShark, with the displayed information aligning with everything else.





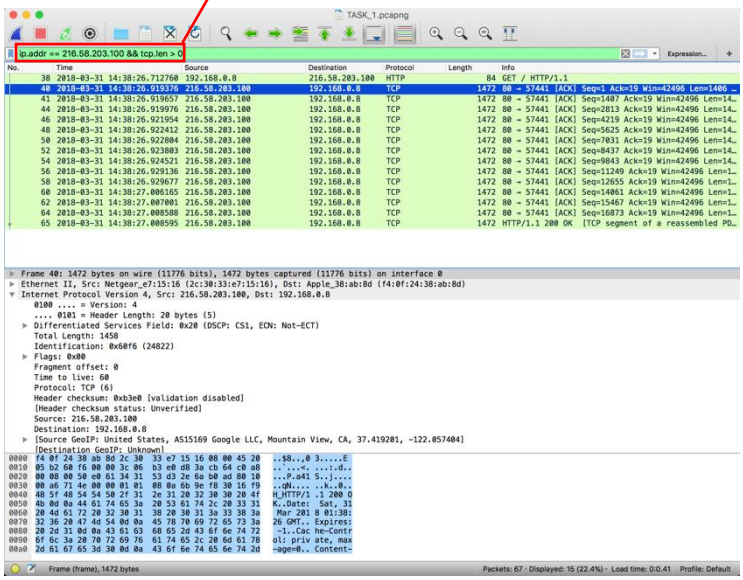
d. What is the status code and phrase in the response?

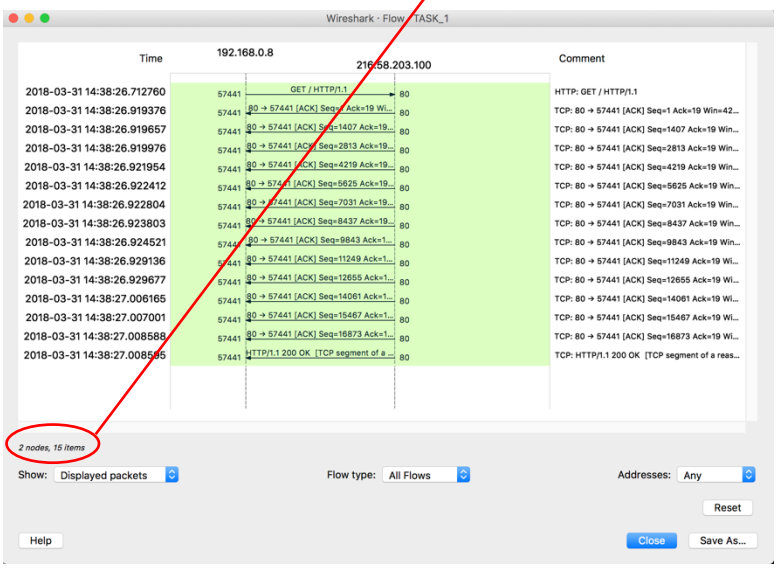
Assuming by response, you are referring to the HTTP response, the status code is '200', and the phrase in the response is 'OK', signifying the success of the request, and following response. This can be seen in the following screenshot, of the (filtered) interactions with the client, based on the IP address of google.com:



e. How many data-containing TCP segments were needed to carry the single HTTP GET and response?

Upon analysing the 'All Flows' graphs, having applied the filter for the interaction with the Google Server, and ensuring all packets with message length zero are ignored, one can analyse both the HTTP GET and response packets, and the corresponding TCP segments needed to communicate both. Here, we can see that between the GET request and the response, 15 TCP segments were required to carry the communication.



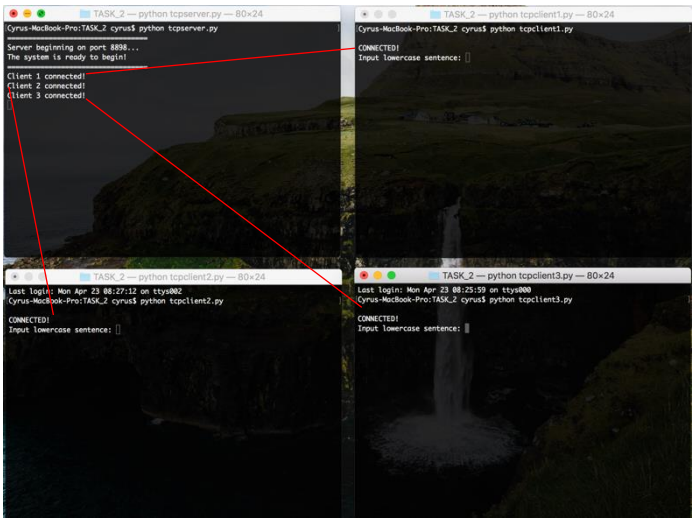


Task 2.

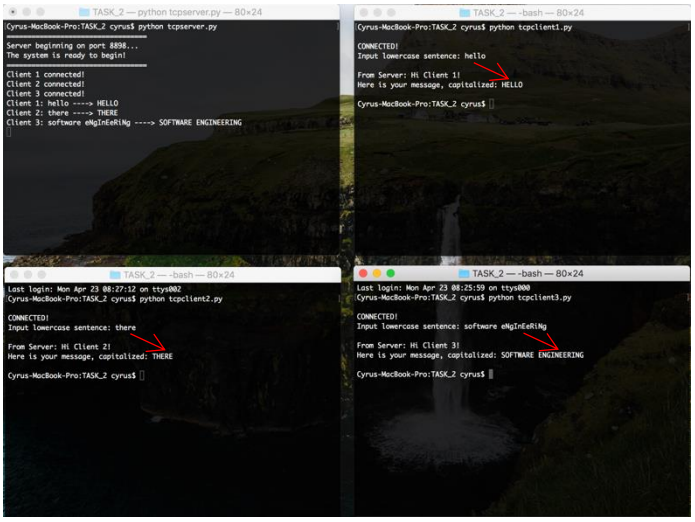
Multi-threading: create a server-side application program that can accept/display information requested by more than one client. For easy implementation, first start with one server to one client communication before upgrading the server to one server to multi-client communication. The communication should be over TCP.

See: *Task_2/tcpclient[1/2/3].py*, and *Task_2/tcpserver.py*.

Here, my server-side application program extends the functionality of an example provided to us in class, of a client being able to send a string to a server, and have it return back the same string, completely capitalized. The server starts off on an arbitrarily numbered port, and creates a new thread for every client that connects to it (via a TCP socket), thus is able to not only handle any number of *n* multiple clients, but handle said multiple clients *simultaneously*. This can be seen in the screenshots below:



Here, the clients first connect to the tcp server, with a message being displayed on both the server and client, upon every new connection.



Here, we can see the individual interactions occurring between the clients and the server, all happening simultaneously. Each client individually sends a different word to the server, which then capitalizes it, displays it in its own terminal, and sends it back.

Note that in relation to the different tcpclient#.py files, there is no difference in the content of the code, I simply created multiple files to properly demonstrate the functionality of handling multiple clients.

Task 3.

(You need to use geolocation with this task, you'll need to install GeoLite on your computer (instructions can be found on line for this, e.g. https://wiki.wireshark.org/HowToUseGeoIP).

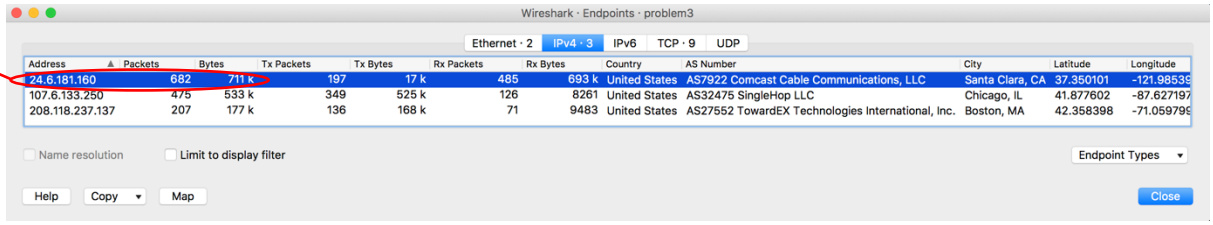
Objective: identify which hosts are the top talkers on the network. Identify which application is using the most bandwidth on the network. You are required to use the Wireshark Statistics: Conversations and Endpoints for this task. As a software engineer, users reported that the internet is always slow late afternoon. During your troubleshooting, you narrow it down that the problem is not from the ISP. Using Wireshark to see what individuals are doing. Open file "problem3.pcapng" and answer the following question.

1. Which host from the IP address has the highest number of packets and bytes (top talker)?

   Of the 3 IP addresses listed under the IPv4.3 column, IP address:

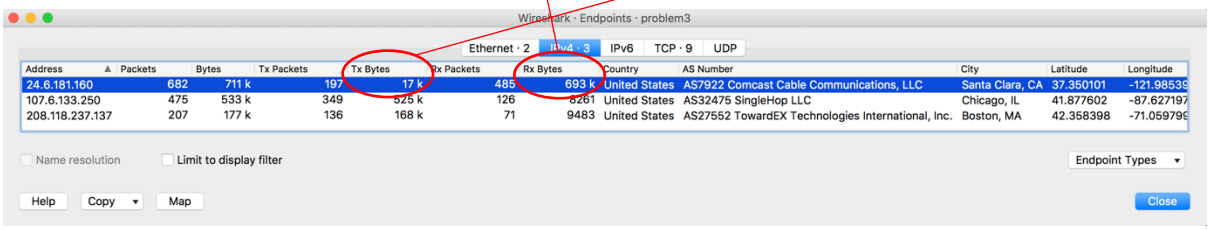
   24.6.181.160

   was the one that had the highest number of listed packets/bytes, under the Statistics > Endpoints window, as attached below.

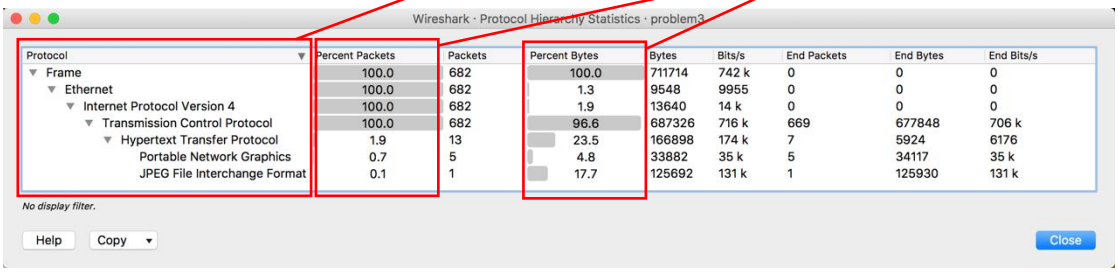


2. How many bytes of data have the IP address in question 1 transmitted and received

   The IP address in question, had a Tx Byte (Bytes Transmitted) value of 17kB, and an Rx (Bytes Received) value of 693kB. These are highlighted in the screenshot below.



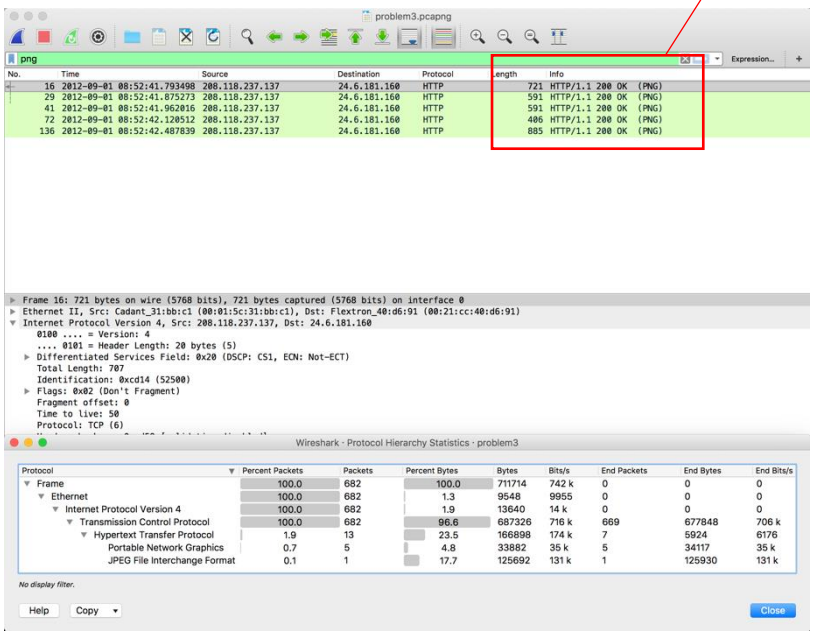
3. Identify the active applications and protocols with their respective percentages in the given file capture.



From the above screenshot, we can see noteworthy applications/protocols, e.g:
HTTP -1.9% of all packets, 23.5% of all Bytes
TCP - 100% of all packets, 96.6% of all Bytes
PNG - 0.7% of all packets, 4.8% of all Bytes etc, etc.
Just as an aside, the difference between any given protocol's percentage of packets/percentage of bytes, is due to the fact that packets usually contain multiple protocols, and can in fact the same protocol more than once.

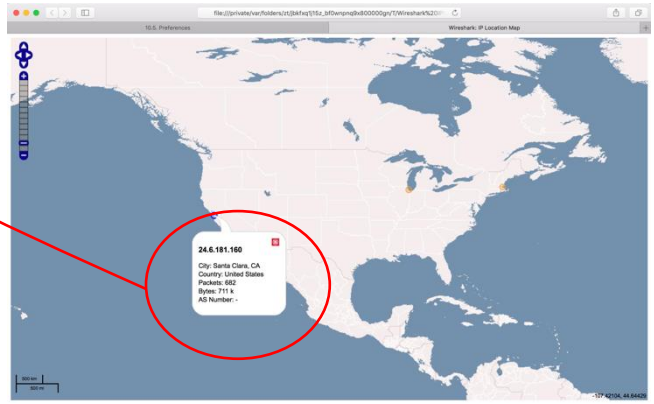
4. Apply the Portable Network Graphics from question 3 as filter (right click on it and apply as filter), how many PNG OK messages were in the communication, attach a screen shot of this.

Having applied the filter, one can see there were 5 PNG OK messages, as seen in the screenshot attached below.



5. Using Wireshark GeoIP, find the destination of the server being visited by this person

Upon utilising Wireshark GeoIP's tool 'IP location Map', I was able to see that



the destination of the server (with IP address 24.6.181.160), was Santa Clara, California, in the US. This is shown in the screenshot below, which was generated after clicking the 'Map' button, within the Endpoints window:

Task 4.

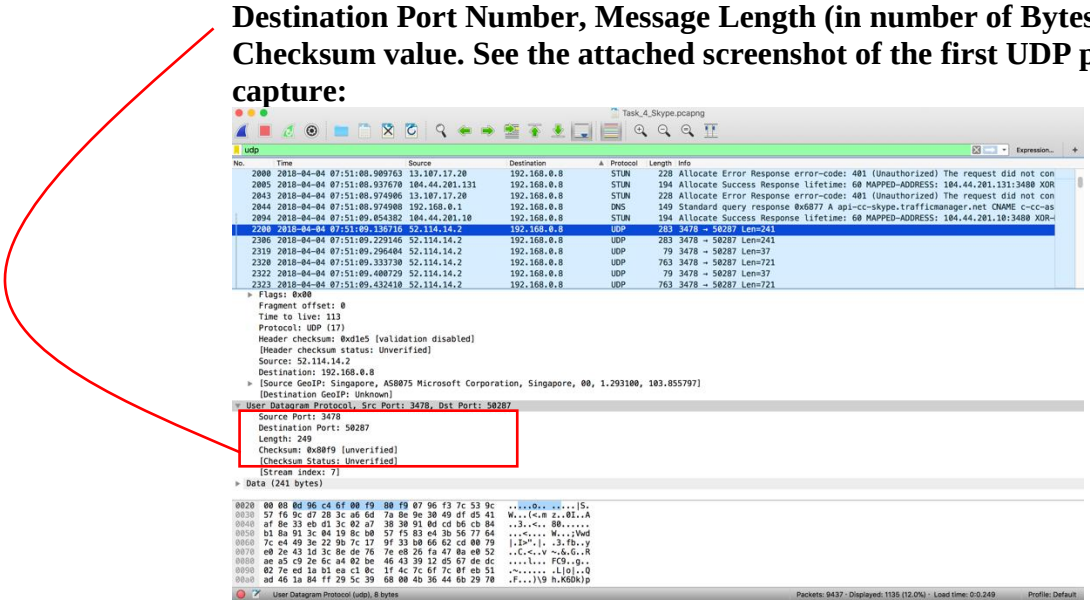
For the purpose of this exercise, we have created a Skype username and password (this is the same as the email address and password created for the labs). You are welcome to use your personal skype information as well. Make a video call to a friend or use the skype call test. Before you start the call, start Wireshark. Stop Wireshark after the call comes to an end, save your capture and attach it to your submission. Set your filter to UDP and answer the following: **(For the capture, see TASK_4/Task_4_Skype.pcapng).**

a. Why is Skype using UDP instead of TCP

Skype facilitates VoIP, and video-calling. When TCP is used, packets are retransmitted by the sender, in the case that they get lost. When it comes to the nature of data being sent for a real-time call, this is not desired; live audio/video has a high loss-tolerance, and values speed and low transmission delay, over reliability and getting every single packet. Thus, UDP is the clear choice, over TCP, for skype video/audio calls.

b. Select one packet preferably the first UDP packet, from this packet determine how many fields are in the UDP header. Name these fields.

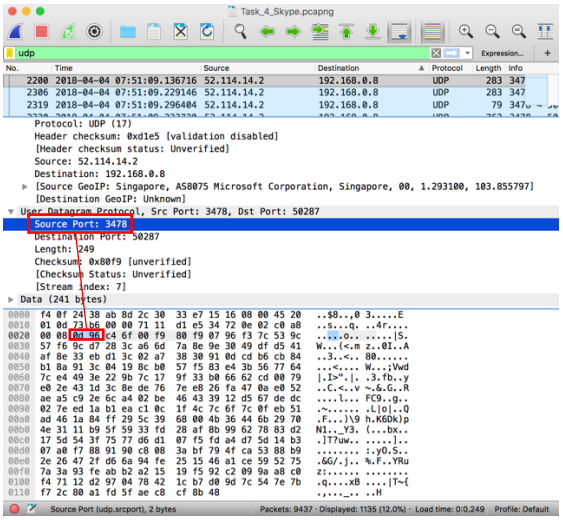
From analysis, there are 4 fields in the UDP header; Source Port Number, Destination Port Number, Message Length (in number of Bytes), and a Checksum value. See the attached screenshot of the first UDP packet in my capture:

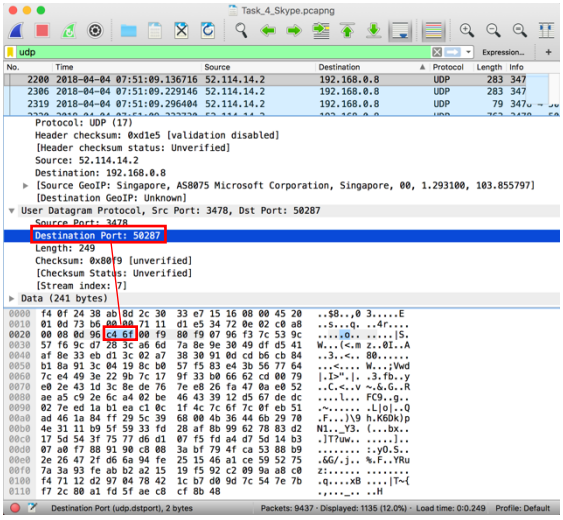


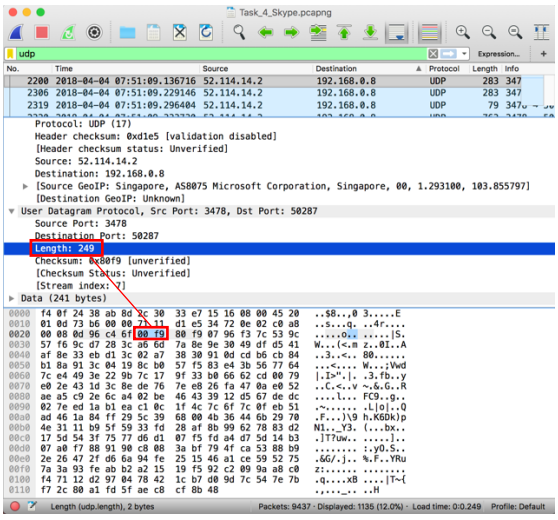
c. From the packet content field, determine the length (in bytes) of each of the UDP header fields.

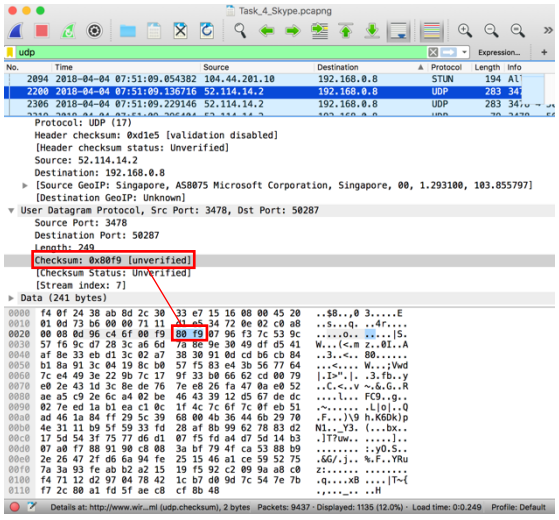
After analyzing the packet content field, I can safely deduce the length (in bytes), of each of the UDP header fields as follows:
Source Port field: 2 bytes
Destination Port field: 2 bytes

Length: **2 Bytes**
Checksum: **2 Bytes**





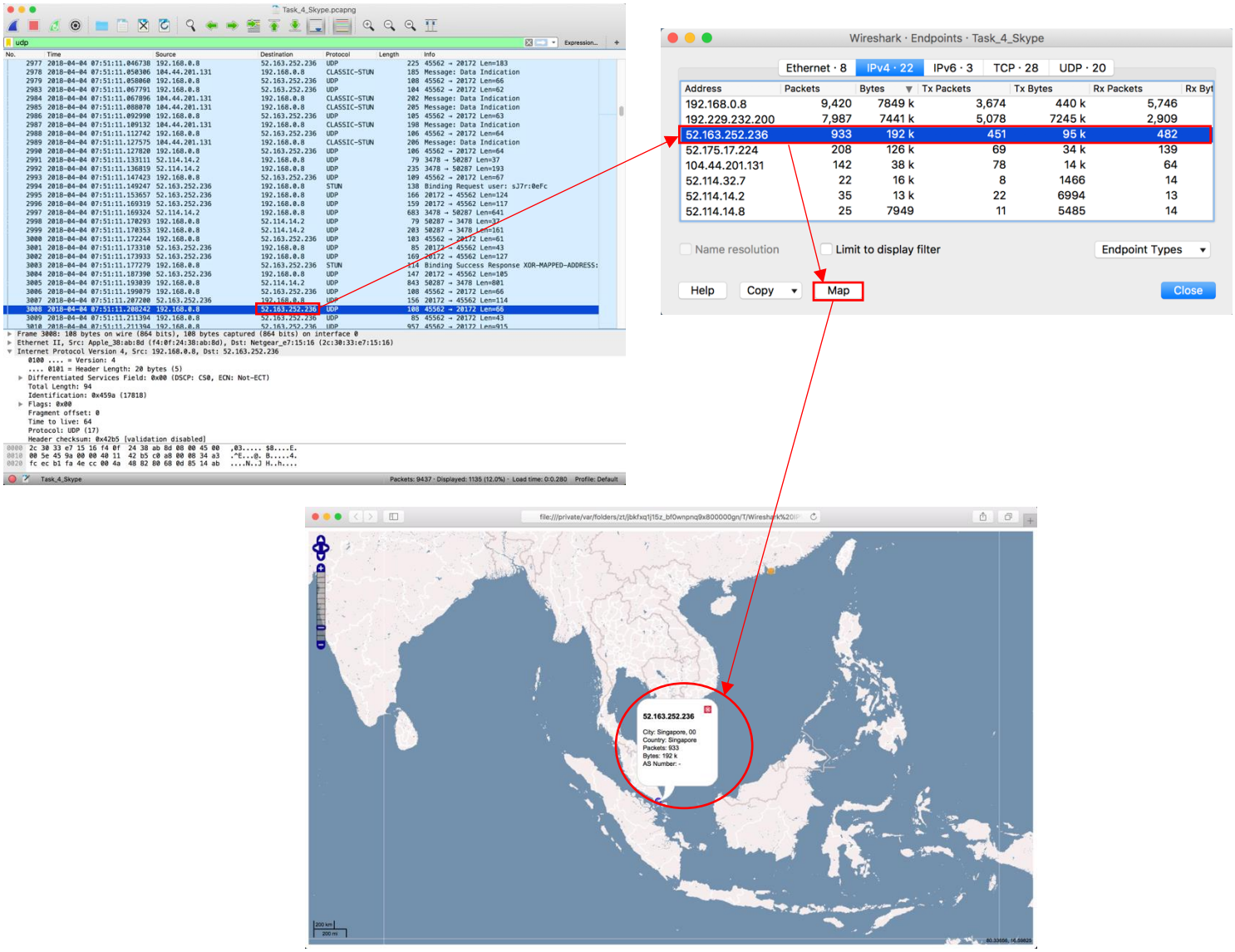




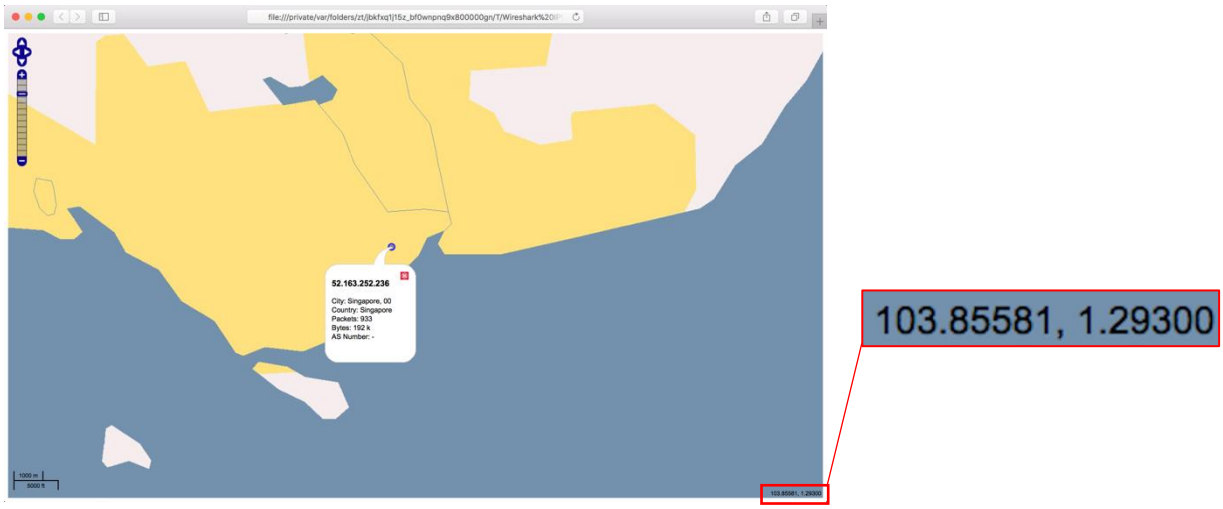
This is characterized by the two two-digit hexadecimal numbers, with each two-digit hexadecimal number representing a byte, thus 2 bytes per field, for the 4 header fields in a UDP message.

d. What is the location of the skype server on which your call is being routed (use the endpoint map to find this location and attach the image)

After analyzing the majority of traffic running over UDP during my skype test call, I identified the assumed IP address (52.163.252.236) and location of the skype server on which my call would have been routed (I deduced this via also knowing what the IP address of my own personal computer was; 192.168.0.8). The identification of the IP address can be seen below, alongside the endpoint map function being used to identify the physical address corresponding to the IP address.

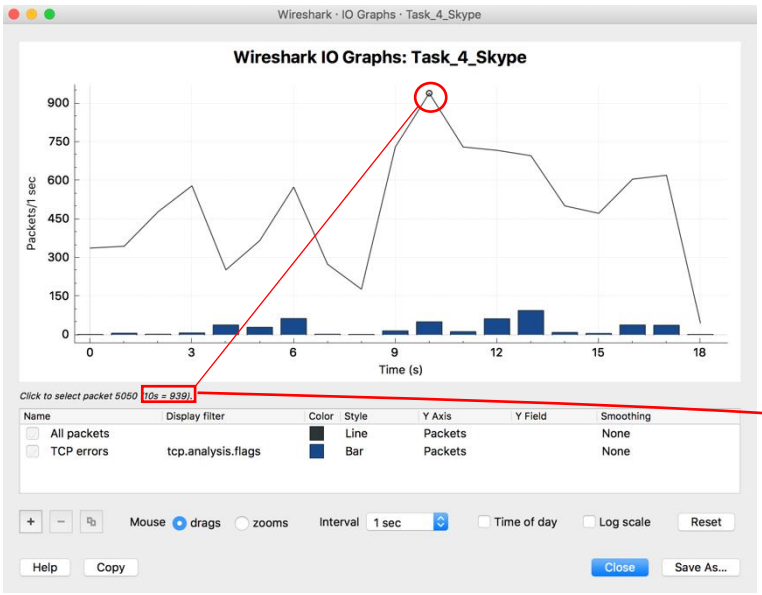


e. Using the map from question d, identify where your host computer is located and attach the screen shot.



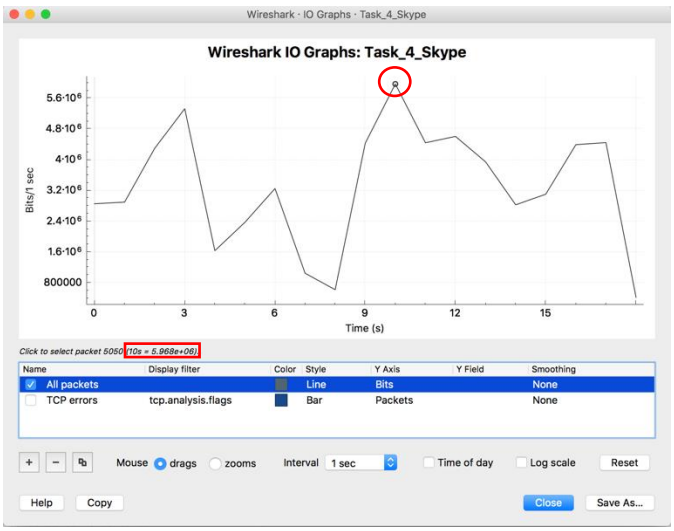
After having zoomed in on the location of the host computer, I have been able to identify it as being in Singapore. This is paired with latitude and longitudinal co-ordinates (1.29300, 103.85581), as shown at the bottom of the screenshot. I took the meaning of 'host computer' to be that which is defined in the website: *http://www.businessdictionary.com/definition/host-computer.html* (in this example, the Skype server, not my own computer).

f. Create an IO Graph for your skype call trace file and establish what is the approximate highest packets-per-second value seen in your trace file, and at what time did it occur? (please attach your annotated screen shot)



After analyzing the IO graph created from my skype call trace file, I have been able to identify the highest packets-per-second value as being 939, at 10 seconds.

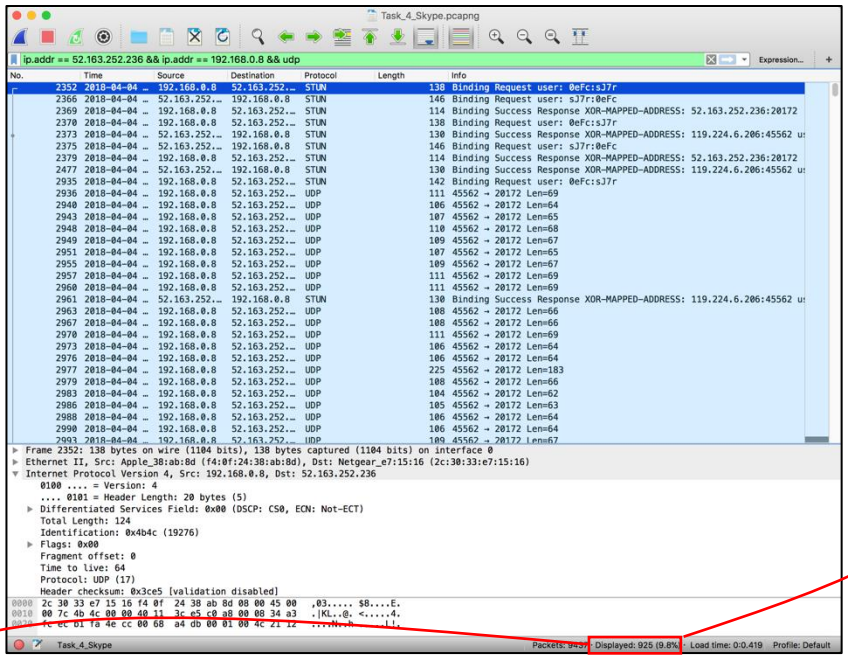
g. Create an IO Graph for your skype call trace file and establish what is the approximate highest bits-per-second value seen in this trace file, and at what time did it occur? (please attach your annotated screen shot).

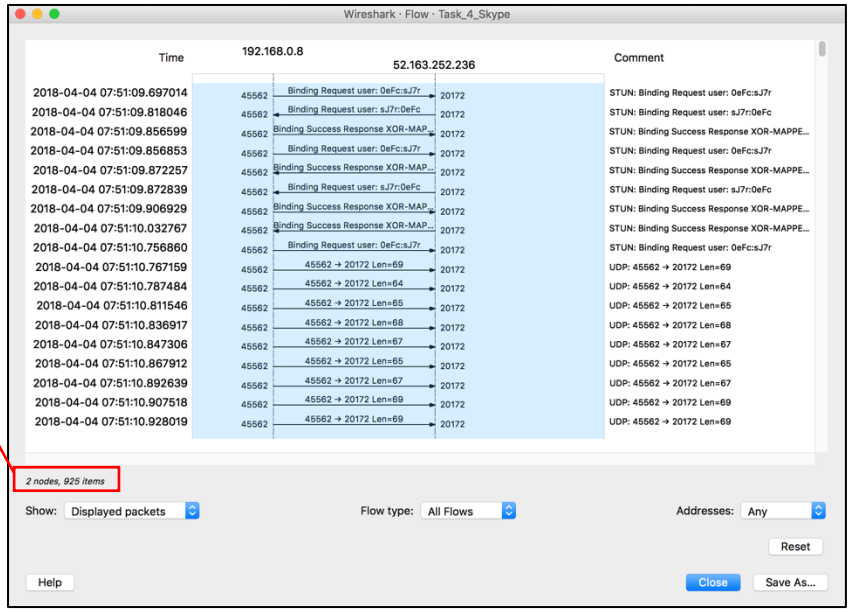


After analyzing the IO graph created from my skype call trace file, I have been able to identify the highest bits-per-second value as being 5.968Mbits/s, at 10 seconds.

h. Plot a flow graph of only the conversations between your host computer IP address and Skype IP address, how many UDP packets were used for your skype call.

After creating and applying a filter to my capture, to find the UDP packets used for my Skype call, in between my host computer IP address and Skype IP Address, I have come to the conclusion that 925 UDP Packets were used for my skype call, as evidenced by the screenshot and flow graph below.





Task 5.

Objective: Investigating the type of protocols and applications present on your network using the Protocol Hierarchy window. You can right-click on any of the protocol and application in this window and apply as filter to give further information in the Wireshark Packet Pane Window for further research. Suspicious protocols and applications can impede the performance of your network. In particular you can identify when a host on your network may have been compromised. Look out for unusual network applications such as Internet Relay Chat (IRC) traffic, Distributed Computing Environment/Remote Procedure Call (DCE/RPC) and sometimes Trivial File Transfer Protocol (TFTP). Wireshark particularly tags packets with un-recognised port number and dissector with the name "Data". So anytime you see Data listed directly under TCP or UDP it is worth investigating further.

In 3-4 paragraphs discuss IRC, DCE/RPC and TFTP. In your discussion include what the protocols do, why they are needed, but also why the presence of packets with these protocols could be an indication that a network was been compromised. Please include references at the end of this answer.

IRC (Internet Relay Chat), is a protocol that runs on the application-layer and facilitates communication via text. IRC is well-liked, for the principle fact that it provides a 'no-frills' approach to facilitating real-time messaging. It can also run on many machines, in a 'distributed fashion'. In regard to a compromised network, hackers may sometimes install backdoor programs to your computer, allowing them to control your host remotely. Such programs may listen on TCP/UDP connections, and can connect to IRC channels, where they can then act as a backdoor of information, for said hacker.

DCE/RPC (Distributed Computing Environment/ Remote Procedure Calls) ***provides a specification for a remote call procedure mechanism, that defines an over-the-network protocol and APIs (https://wiki.wireshark.org/DCE/RPC).*** **It essentially facilitates the masking of calls on a remote host, as if they were local procedures, to that host. RPC in itself is very important, as it's used as protocol to request services from programs located in other computers in a network, foregoing the need to understand details of the network. RPC in itself has a vulnerability when used to communicate over TCP/IP, that could potentially allow an attacker to run code with Local System Privileges (pretending to be the legitimate user), on an affected system.**

TFTP (Trivial File Transfer Protocol), is a protocol that is used to transfer files with a simplistic approach. Due to its small size and straight-forward implementation, it is frequently used in embedded systems, to grab files from servers upon booting up. TFTP does not implement authentication. The presence of TFTP packets could signal an attempt at a DDoS (Distributed Denial-of-Service Attack) on a server, with reflection/amplification of said attack, leveraged via TFTP.

References (APA):

(n.d.). Retrieved from http://www.doublersolutions.com/docs/dce/OSFdocs/htmls/overview/Dceint63.htm

DDoS Attacks Abuse TFTP for Reflection and Amplification. (n.d.). Retrieved from https://www.securityweek.com/ddos-attacks-abuse-tftp-reflection-and-amplification

Internet Relay Chat. (2018, April 16). Retrieved from https://en.wikipedia.org/wiki/Internet_Relay_Chat

Buffer overrun in rpc may allow code execution. (n.d.). Retrieved from https://support.microsoft.com/en-us/help/823980/ms03-026-buffer-overrun-in-rpc-may-allow-code-execution

TFTP. (n.d.). Retrieved from https://wiki.wireshark.org/TFTP

Trivial File Transfer Protocol Used in New DDoS Attack. (n.d.). Retrieved from https://securityintelligence.com/news/trivial-file-transfer-protocol-used-in-new-ddos-attack

What is RPC and why is it so important? (n.d.). Retrieved from https://superuser.com/questions/616098/what-is-rpc-and-why-is-it-so-important

Windows Forensics: Have I been Hacked? (n.d.). Retrieved from https://www.bleepingcomputer.com/tutorials/have-i-been-hacked

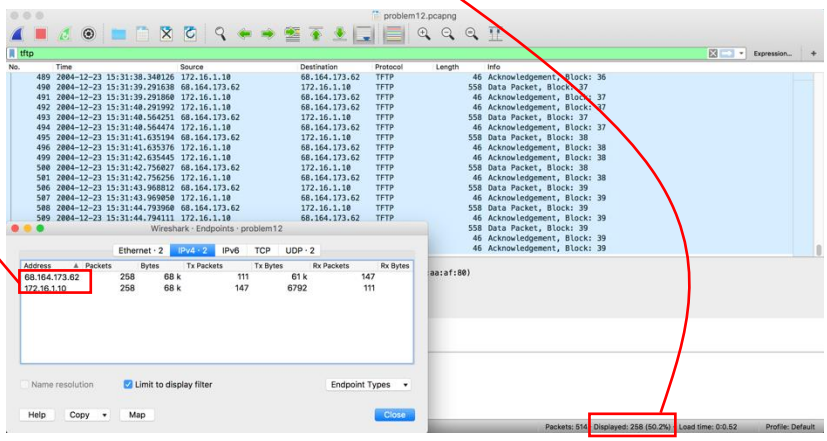
What is IRC Chat, and why do you use it? (n.d.). Retrieved from https://www.quora.com/What-is-IRC-chat-and-why-do-you-use-it

Open problem12.pcapng and answer the following questions.

1. Which source and destination IP addresses on your network are sending/using the Trivial File Transfer Protocol (TFTP)? How many packets are being transmitted using TFTP?

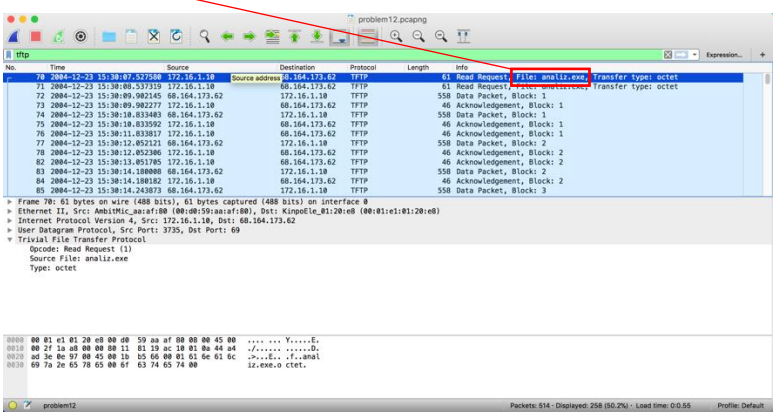
   The source and destination IP addresses on the network using the TFTP, are: *68.164.173.62* and *172.16.1.10*. This information was gathered from opening the endpoints window in WireShark, after applying the 'tftp' filter on the packet capture.

   There are exactly 258 packets transmitted via TFTP, as shown by the statistic at the bottom of the window, in the screenshot below, upon applying the 'tftp' filter on the packet capture.



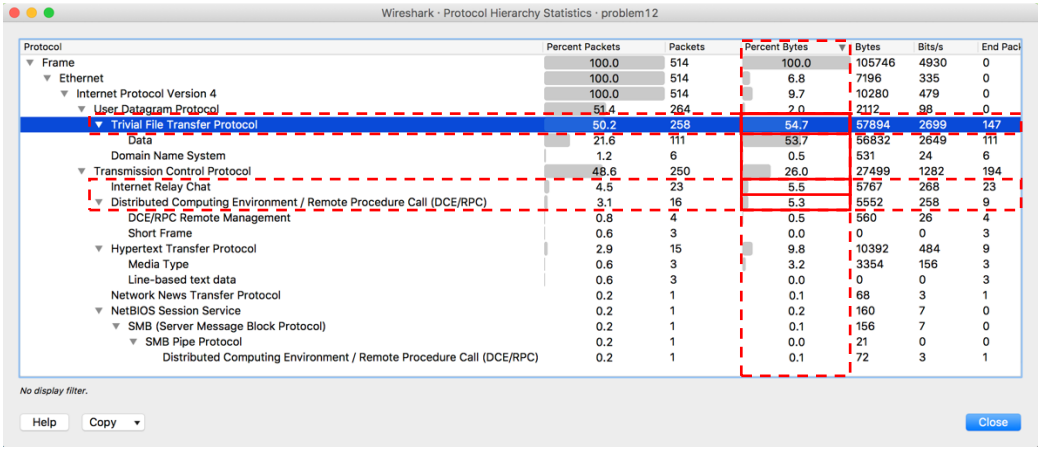
2. What is the name of the file being transferred using TFTP?

   File: analiz.exe (as seen in the screenshot below)



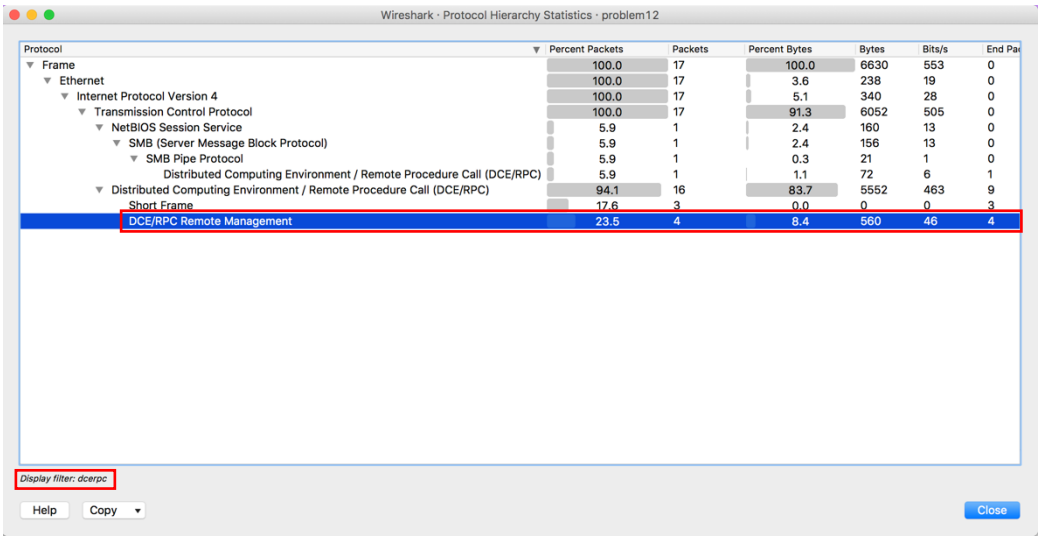
3. Out of all the unusual network applications (Note that there is a difference between protocol and application) present on this capture file, which one is responsible for the highest bytes, and what percentage of the total bytes is this (show your calculations).

In the brief of task 5, unusual network applications are referred to as: "*unusual network applications such as Internet Relay Chat (IRC) traffic, Distributed Computing Environment/Remote Procedure Call (DCE/RPC) and sometimes Trivial File Transfer Protocol (TFTP)*". Going off of this, we can analyse the contribution of each in terms of the percentage of total bytes sent/receive that used each one, using the protocol hierarchy window:



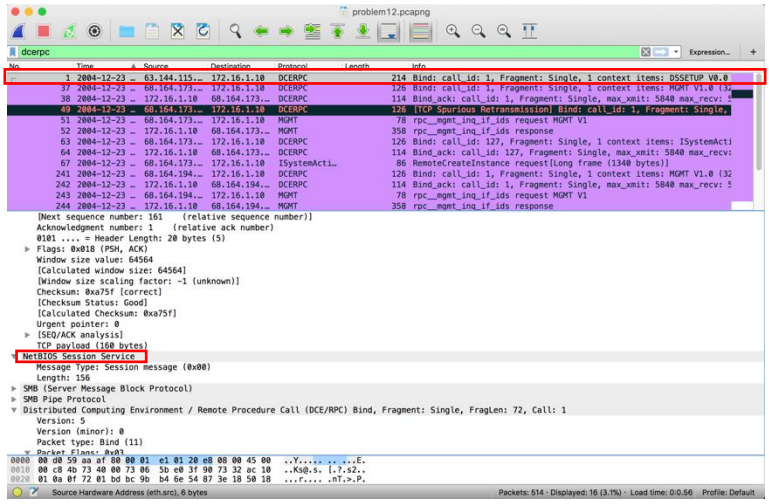
Analysing this, we can note that of the identified unusual network applications, TFTP (Trivial File Transfer Protocol) had the highest bytes (at 57894 bytes, out of a total of 105746), which – as also shown in the protocol hierarchy window – works out to around 54.7% of the total bytes.

4. What activity is DCE/RPC being used for in this capture?



After analysing the Protocol Hierarchy window, alongside the actual capture, we can see that DCE/RPC itself is mainly being used for DCE/RPC Remote

Management; that is, a user is a part NetBIOS session (purposed for connecting two computers, to transfer heavy amounts of data). This can be seen in the first screenshot below:

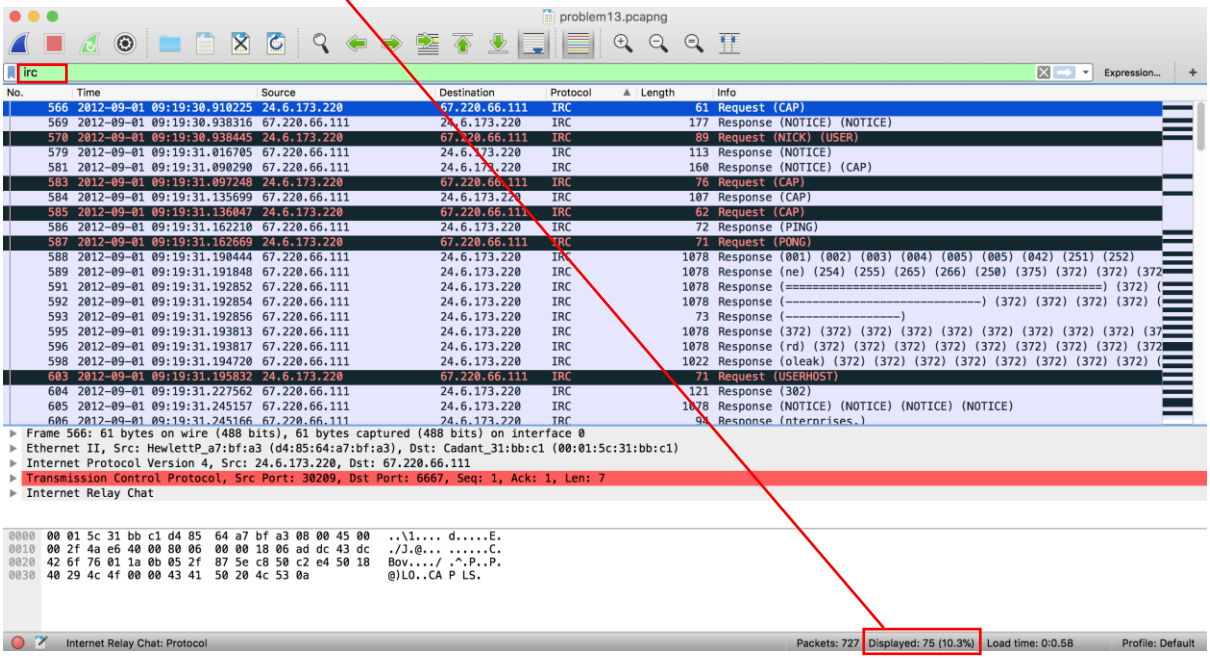


Using this information – alongside the fact that in the capture, the IP address 63.144.115 is mentioned once (in the first DCE/RPC packet, and the IP addresses 172.16.1.10 and 68.164.173 are mentioned several times – it is potentially fair to assume that malicious actions are going on, in the form of one host using DCE/RPC to masquerade as the host of another IP address, leveraging a NetBIOS session to transfer a lot of data.

Open problem13.pcapng and answer the following questions.
(Hints: Using the protocol hierarchical statistics to investigate unusual IRC applications, apply this as a filter and expand the IRC section in the Packet Details pane for further information)

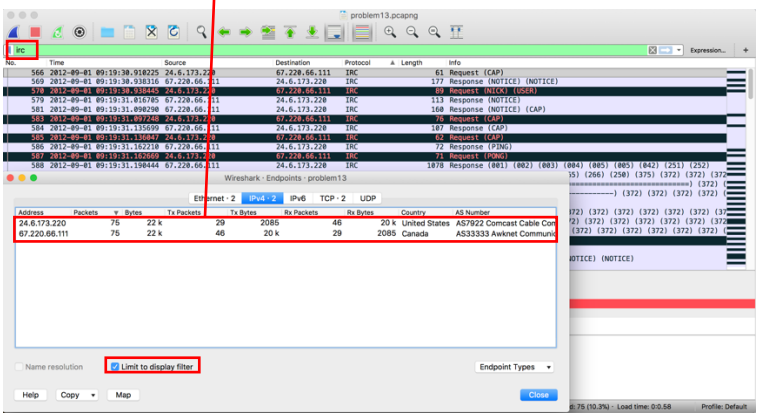
5. How many IRC packets are in this capture

   There are exactly 75 IRC packets in this capture, as shown by the statistics at the bottom of the window shown:



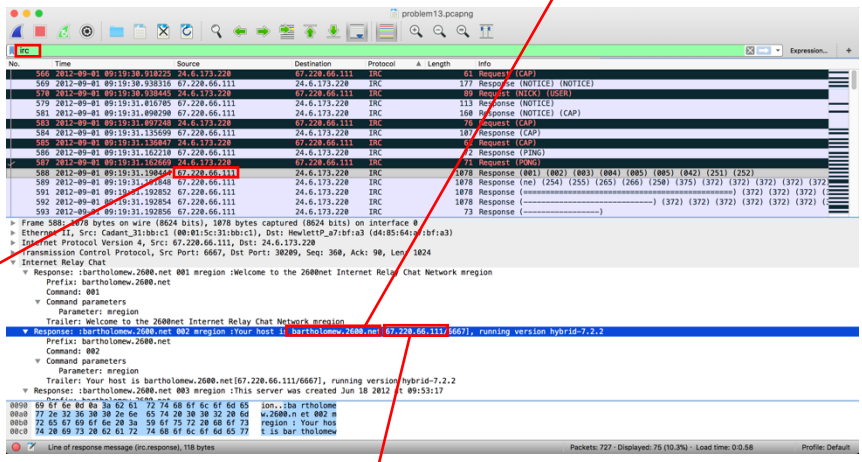
6. How many hosts are using IRC?

   As we can count via the Endpoints window in Wireshark, there are approximately 2 hosts that have used IRC in the packet capture:



7. What is the host name of the IRC server

The hostname of the IRC server is bartholomew.2600.net, as shown by the IRC server response details below:

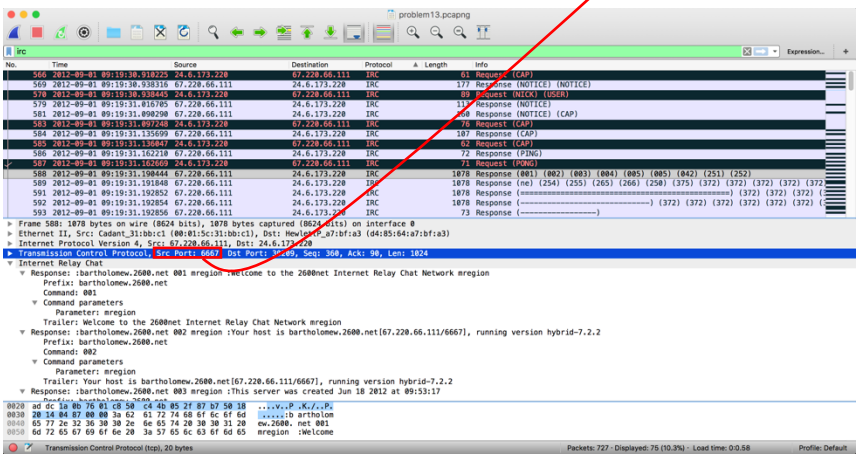


8. What is its IP address?

It's IP address is 67.220.66.111, as also shown in the 'Destination' column of the above screenshot of the Packet Capture.

9. On which port is the server communicating?

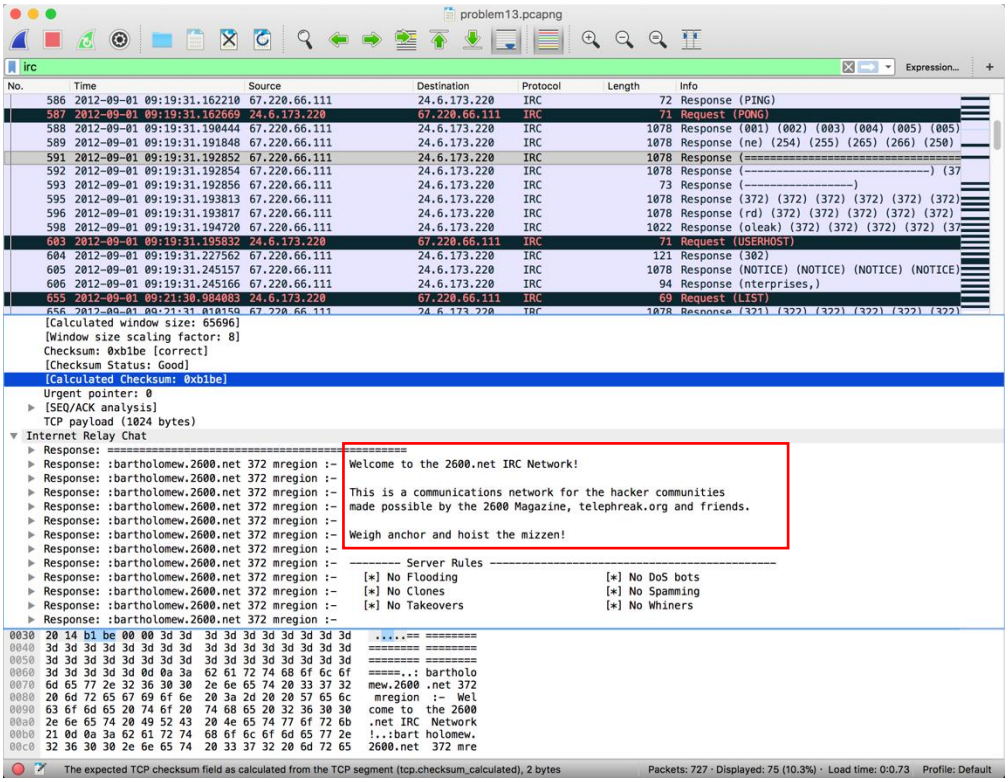
The server is communicating via it's port number 6667. This can be seen as the src port field within the header, of an IRC response packet sent by the server, back to the client. This is shown below:



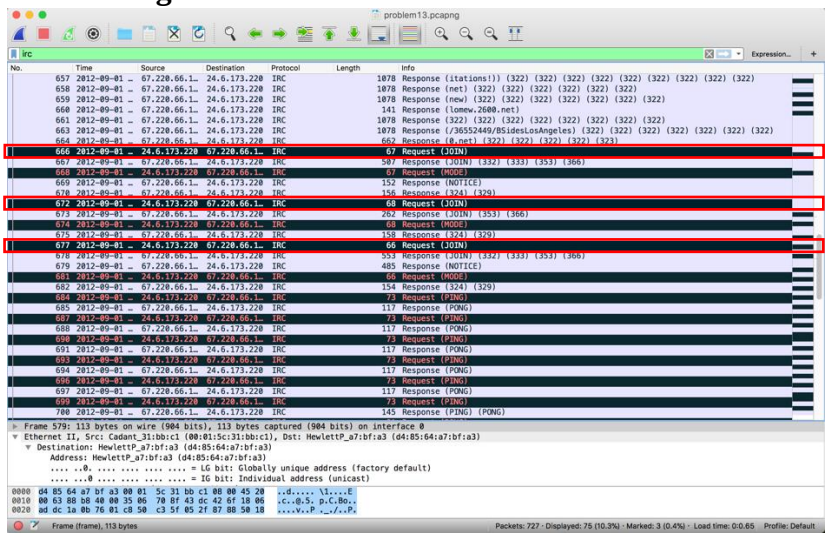
10. What activity is IRC being used for in this capture?

The IRC protocol, is here being used to facilitate a communication between an end user, and a network (named 2600.net), with the network itself being

purposed for hacker communities. This is shown in the first screenshot below:



Judging from the contents of the various messages sent/received via IRC, the capture depicts a user connecting to the hacker community network, and specifically to the #oc2600 channel (specified as *':Defcon 20 / BSides LA #2*'), followed by the #hackbbs channel, and the #warez channel. These are depicted by the selected packets within the following capture, that have white text on a black background.



Following these, in the capture itself, one can note that there are several PING and PONG packets sent. In relation to the IRC protocol, these packets are continually sent to keep the connection open and valid; if the frequency of the PING packets being sent to the IRC server goes below a certain threshold, the server will drop the connection it has with the client.