Wireshark Lab: TCP

Submit online as a pdf in Canvas

**General Instructions:** **What to hand in: Please answer the questions posed in this lab, please make it clear what questions you are answering, and please use screenshots to support your answers. 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. Marks will be awarded for correctness, completeness, and professionalism. You’ll also be using this document to study for your tests and exams.**

**Learning Outcome:** At the end of this lab you should:

* be familiar with the different TCP message types.
* understand the role of the TCP in any network.
* write a socket program over TCP.
* finally, be able to explain the operation of TCP.

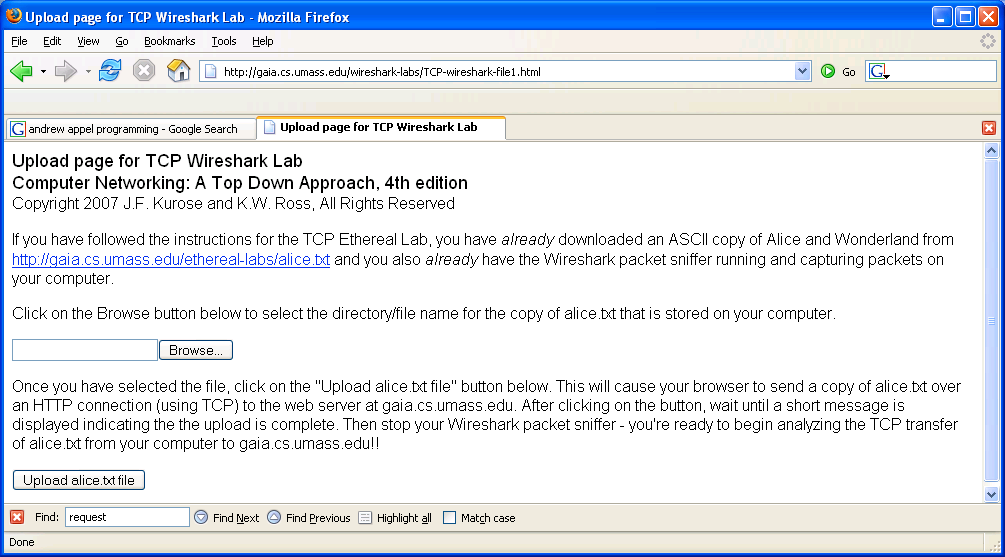
## Exercise 1

**Objectives for Exercise 1**: To see how file can be uploaded to a remote server from your computer using the TCP protocol. To investigate the operation of TCP.

Before beginning our exploration of TCP, we’ll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You’ll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text of *Alice in Wonderland*), and then transfer the file to a Web server using the HTTP POST method (see section 2.2.3 in the text). We’re using the POST method rather than the GET method as we’d like to transfer a large amount of data *from* your computer to another computer. Of course, we’ll be running Wireshark during this time to obtain the trace of the TCP segments sent and received from your computer. (Make sure TCP relative sequence numbers box is checked in preference)

Do the following:

* + Start up your web browser. Go the <http://gaia.cs.umass.edu/wireshark->labs/alice.txt and retrieve an ASCII copy of *Alice in Wonderland.* Store this file somewhere on your computer.
  + Next, go to [http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html](http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html.)
  + You should see a screen that looks like:



* Use the *Browse* button in this form to enter the name of the file (full path name) on your computer containing *Alice in Wonderland* (or do so manually). Don’t yet press the “*Upload alice.txt file*” button.
* Now start up Wireshark and begin packet capture *(Capture->Start)* and then press *OK* on the Wireshark Packet Capture Options screen (we’ll not need to select any options here).
* Returning to your browser, press the “*Upload alice.txt file*” button to upload the file to the gaia.cs.umass.edu server. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.
* Stop Wireshark packet capture.
  1. Filter the capture to only TCP SYN packets and TCP ACK packets you should see this. (Hint: tcp.flags.syn==1 && tcp.flags.ack==1). This finds the middle step in the three-way handshake. Find the middle step in the three-way handshake between your machine, and the machine that hosts [http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html](http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html.)

It will look something like the screenshot in figure 1.

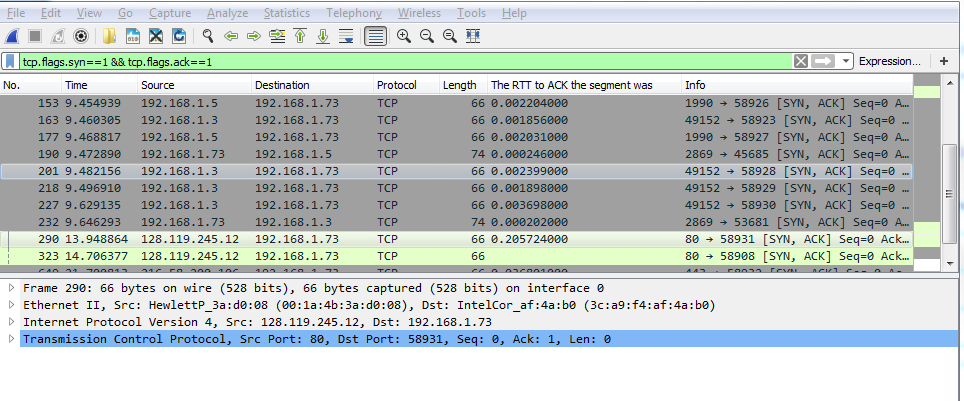


Figure 1.

1.1.1 Explore the TCP Header of this TCP packets in your capture and fill in the picture below.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Source Port # 80 | | | | | | | | Destination Port # 57262 |
| Sequence # 0 | | | | | | | | |
| Acknowledgement # 1 | | | | | | | | |
| Header  Len 32 bytes | Not  Used | U 0 | A 1 | P 0 | R 0 | S 1 | F 0 | Receive Window 29200 |
| Checksum  0x3776 | | | | | | | | Urg Pointer 0 |
| Options: Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted, No-Operation (NOP), Window scale | | | | | | | | |
| Data | | | | | | | | |

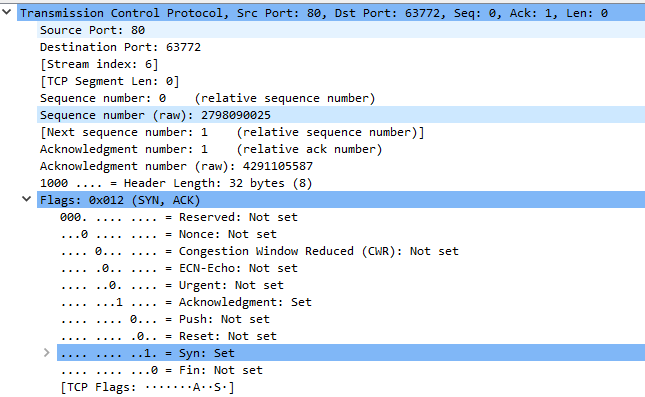
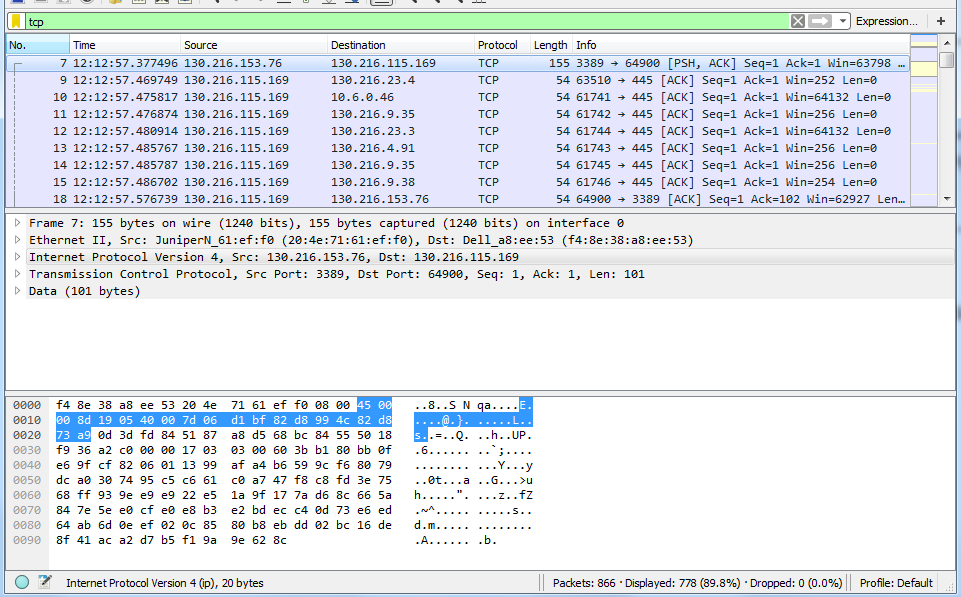


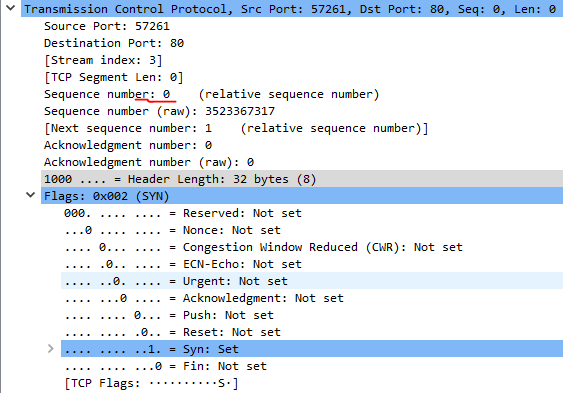
Figure 2: TPC Header

* 1. Clear out the previous filter and set your filter to “tcp”, you should see a window similar to this:



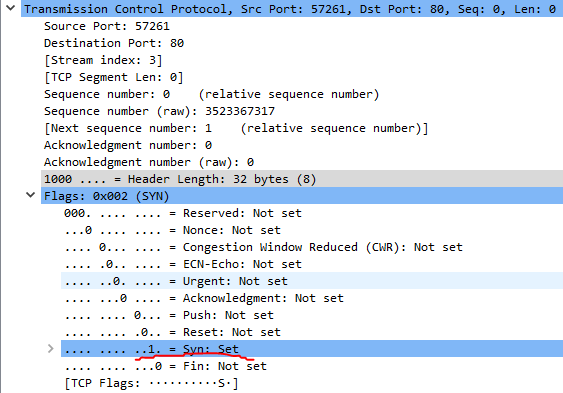
### Questions

* + 1. What is the relative sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu?



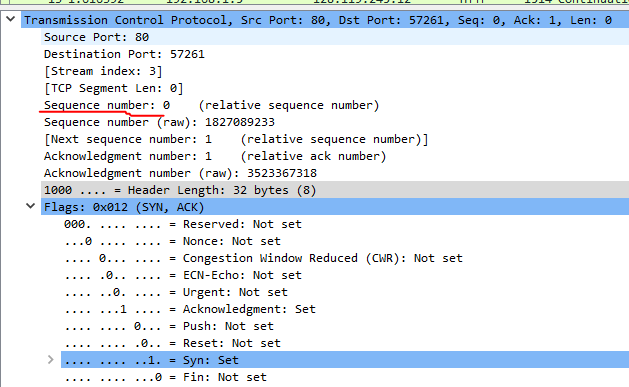
The relative sequence number of the segment is 0.

* + 1. What is it in the segment that identifies the segment as a SYN segment?



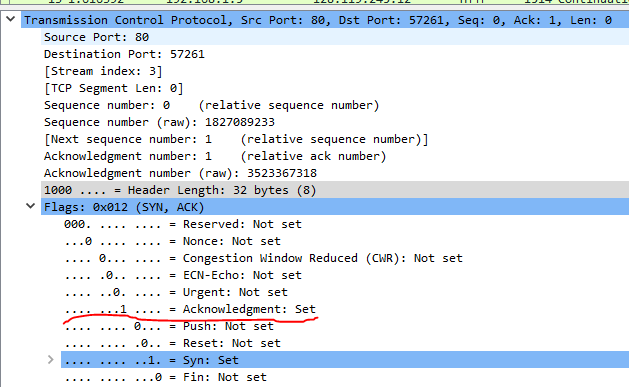
The flag for Syn is set to 1 which indicates that the segment is a SYN segment.

* + 1. What is the relative sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN?



The relative sequence number is 0.

* + 1. What is the value of the Acknowledgement field in the SYNACK segment?



The value of the Acknowledgement field is 1.

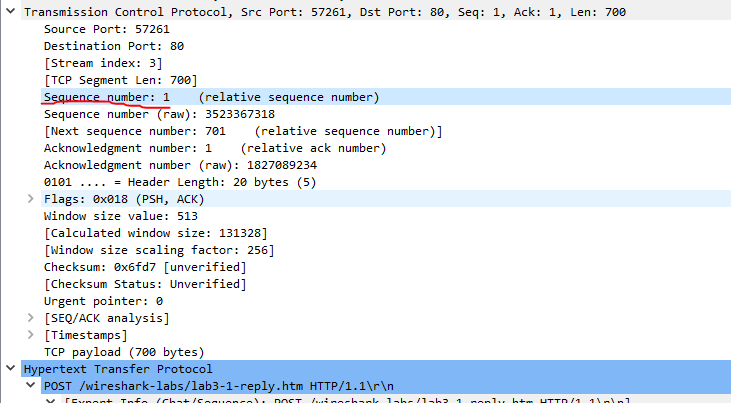
* + 1. How did gaia.cs.umass.edu determine the value in 1.2.4?

The value of the acknowledgement field corresponds to the sequence number + 1 for the segment it is acknowledging. The segment that is being responded to has a sequence number of 0, hence the 1 in the acknowledgement field.

* + 1. What is it in the segment that identifies the segment as a SYNACK segment?

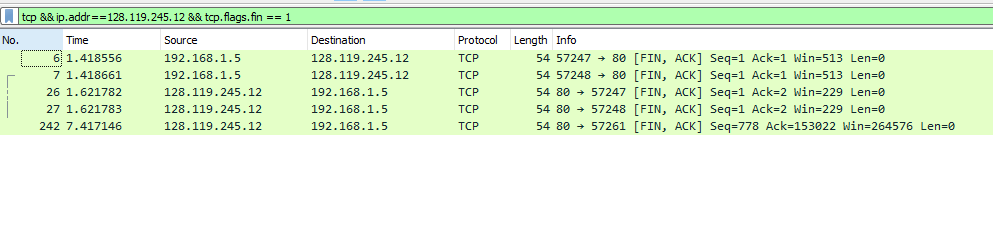
Both the Acknowledgement flag and the Syn flag are set to 1.

* + 1. What is the relative 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.



The relative sequence number is 1.

* + 1. Did you managed to capture the connection close down between yourself and gaia.cs.umass? (Hint what flag would be set in the TCP header?)



Yes, the FIN flag is set in both the sender and receiver, so the connection is closed.

## Exercise 2

**Objectives for Exercise 2:**

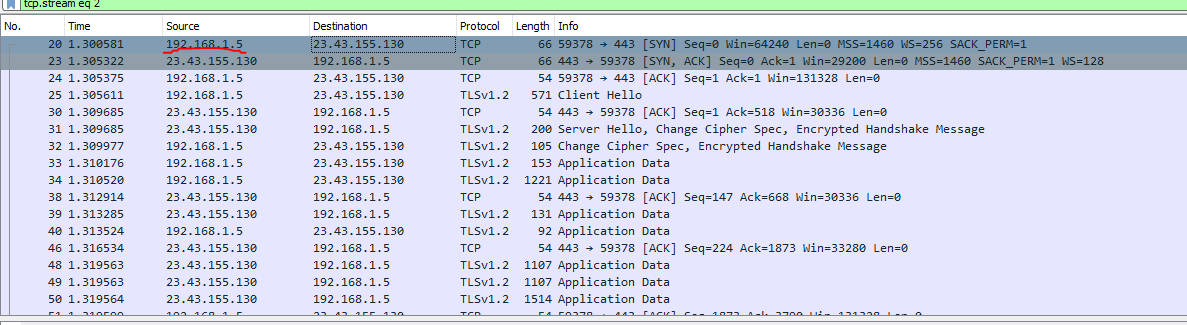
* To investigate the TCP stream.

Following TCP streams make it possible to see the data from a TCP stream in the way that the application layer sees it. For example, trying to make sense of a data stream (the information been sent) or even the password. The stream content is displayed in the same sequence as it appeared on the network. Traffic from A to B is marked in red, while traffic from B to A is marked in blue by default. Looking through the trace packet-by-packet is time-consuming, tedious and prone to error, but through using the Follow TCP Stream feature in Wireshark we can work efficiently. By clicking on each of the messages in the TCP Stream Display window you can see the corresponding packet in the Packet List Pane

2.1 Start Wireshark, open [www.cisco.com](http://www.cisco.com) stop Wireshark.

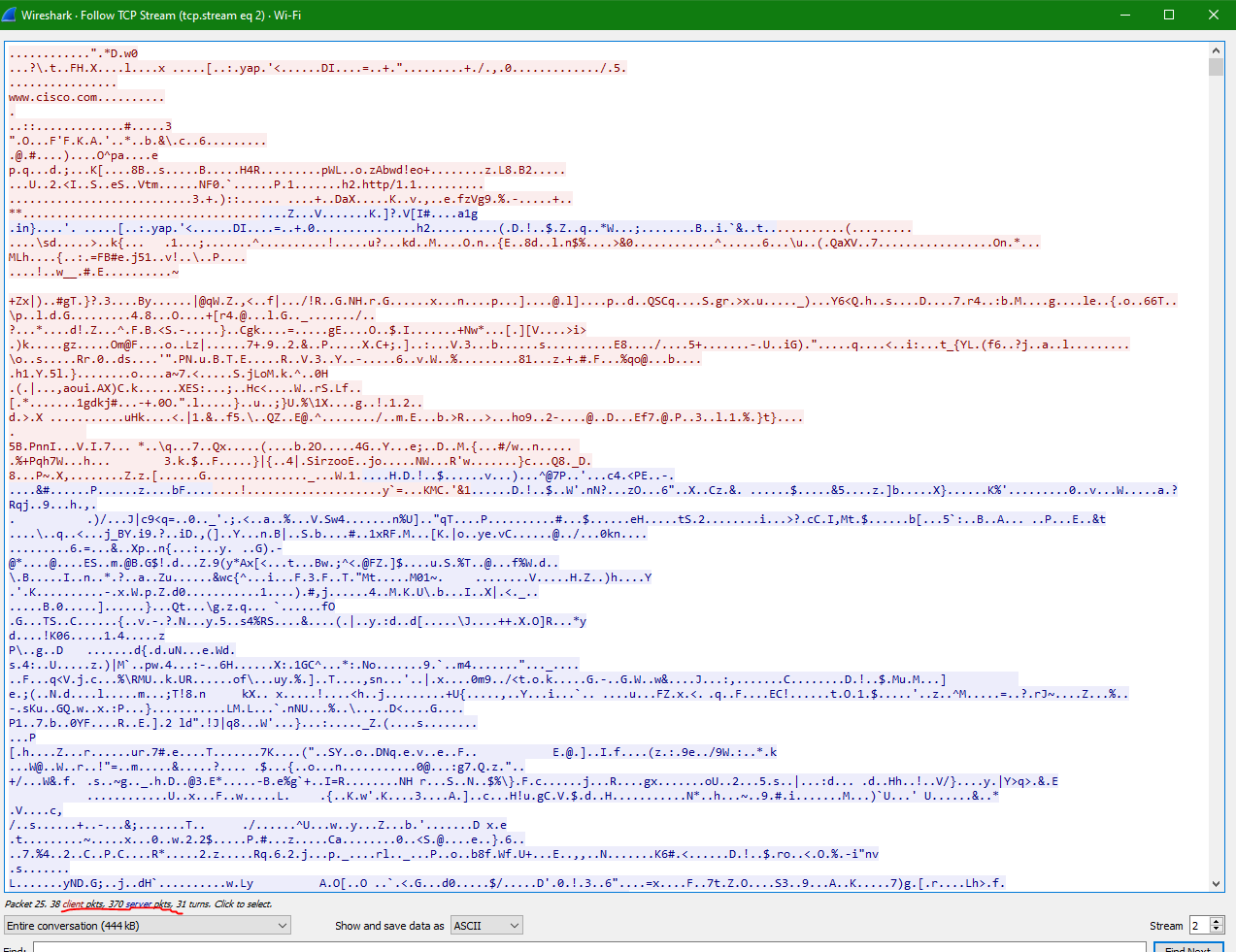
* Identify which communication (packets) are as a result of visiting [www.cisco.com](http://www.cisco.com). Right-click any of the TCP packets in the display window and choose “Follow TCP Stream”. The follow TCP Stream in Wireshark is the way in which the application layer sees the data transmission. Following TCP streams provides a different view on network traffic: instead of individual packets, one can see data flowing between client and server. In case you are trying to make sense of a data stream, or just for filtering purpose. TCP stream will make troubleshooting easier. Notice that the text displayed in the window is in two colours, red text (source to destination), and blue text (destination to source).

Questions

2.1.1 In the TCP Stream you got in 2.1 what is the IP address of the side that initiated the communication? 

The ip address that initiated the communication is 192.168.1.5

2.1.2 In the TCP Stream you got in 2.1 how many turns did the message go between client and server. Attach screenshot as evidence.

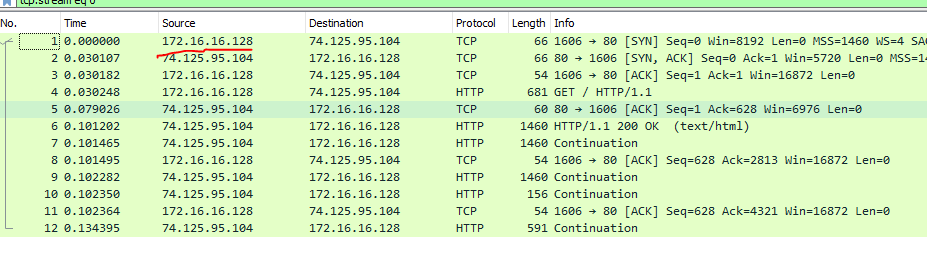


In total there were 31 turns of messages between the client and server.

* 1. Open the file named “captureTCP1.pcap”.
* Right-click any of the TCP or HTTP packets in the display window and choose “Follow TCP Stream”.

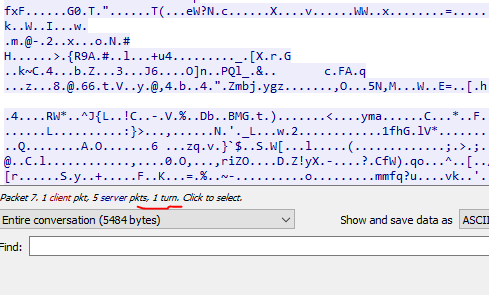
Questions

* + 1. In the TCP Stream you got in 2.1 what is the IP address of the side that initiated the communication?



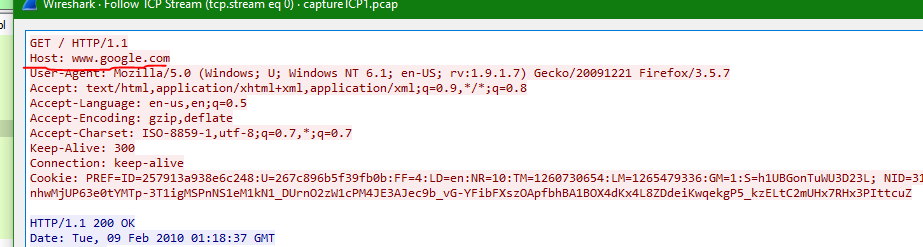
The IP address that initiates the communication is 172.16.16.128 as it is sends the first request with the SYN flag set.

* + 1. How many turns did the message go between client and server,



There was 1 turn between the client and server.

* + 1. what is the name of the originating source host



The name is www.google.com

## Exercise 3 Python Socket Network Programming (using python 3)

**Objective to Exercise 3**

* Gaining insight in to Socket programming
* Getting some skills that you’ll be using for your assignment

Sockets are the real backbones behind web browsing. There is always a server and a client in any application. In this exercise, we will explore the writing the client side of the client-server application program. Firstly import a socket library using

import socket

Make a simple TCP socket

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

The first parameter in the socket instance is AF\_INET and it refers to the address family IPV4. The second parameter on the other hand SOCK\_STREAM means connection oriented TCP protocol. For UDP protocol use SOCK\_DGRAM

Using the socket created, we will connect to a server. We can only connect to a server by knowing its IP address. We want to connect google server. You can get the IP address of google.com in different ways, we will explore 2 ways in this lab. Open command prompt and type the following

$ ping www.google.com

On the other hand we can use python to get the IP address by using the following command

import socket

ip = socket.gethostbyname(‘www.google.com’)

print (ip)

Connecting to the server, you need the following the port number, here since we are connecting over the internet (http) we will use port 80. The created socket instance (s) is used to connect.

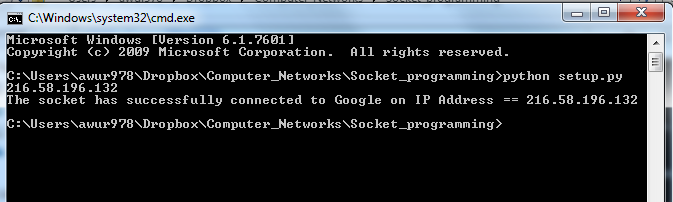
port = 80

s.connect((ip, port))

Finally print your result

print (“The socket has successfully connected to Google on IP Address == %s” %(ip))

Now put everything together in a script and run the program. You will get something like this



Here, google.com is our server and we do not need to write the server side of the application. We will try and send data (message) to the google server using the function *sendall.* Add the following to your script

message = “GET / HTTP/1.1\r\n\r\n”

s.sendall (message.encode())

print (“Message send successfully”)

The message “GET / HTTP/1.1\r\n\r\n” is an http command to fetch the main page of a website

Finally, we will like to receive the message from the server (i.e. google.com main page) by using the function “recv”. Include the following in your code

reply = s.recv(4096)

print (reply)

You should get something similar to the screen shot below

The 4096 is the size of the message buffer, the default is 1024. You can play around with this to see the effect.

Finally, you can close the created socket using

s.close()

Please include the screenshot of your capture as shown below.

