COMPUTER NETWORKS (CS F303)
LAB-SHEET – 1
Topic: Packet Sniffing using Wireshark and Traceroute

Objectives

- This is an exercise in learning by doing and thinking.
- Do NOT consult a book or a website or a senior – if you do so, you are not learning :(
- However, you are encouraged to consult/discuss/argue with other students in your class.
- Some of the material is taken up from Wikipedia and the Wireshark labs which come along with your textbook, i.e. Computer Networking: A Top Down Approach by Kurose and Ross.

One's understanding of network protocols can often be greatly deepened by "seeing protocols in action" and by "playing around with protocols" – observing the sequence of messages exchanged between two protocol entities, delving down into the details of protocol operation, and causing protocols to perform certain actions and then observing these actions and their consequences. This can be done in simulated scenarios or in a "real" network environment such as the Internet. In the Wireshark labs you'll be doing in this course, you'll be running various network applications in different scenarios using your own computer. You'll observe the network protocols in your computer "in action," interacting and exchanging messages with protocol entities executing elsewhere in the Internet. Thus, you and your computer will be an integral part of these "live" labs. You'll observe, and you'll learn, by doing.

The basic tool for observing the messages exchanged between executing protocol entities is called a **packet sniffer**. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a *copy* of packets that are sent/received from/by application and protocols executing on your machine.

Figure 1 shows the structure of a packet sniffer. At the right of Figure 1 are the protocols (in this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The packet sniffer, shown within the dashed rectangle in Figure 1 is an addition to the usual software in your computer, and consists of two parts. The **packet capture library** receives a copy of every link-layer frame that is sent from or received by your computer. You know that that messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable. In Figure 1, the assumed physical media is an Ethernet, and so all upper-layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you all messages sent/received from/by all protocols and applications executing in your computer.

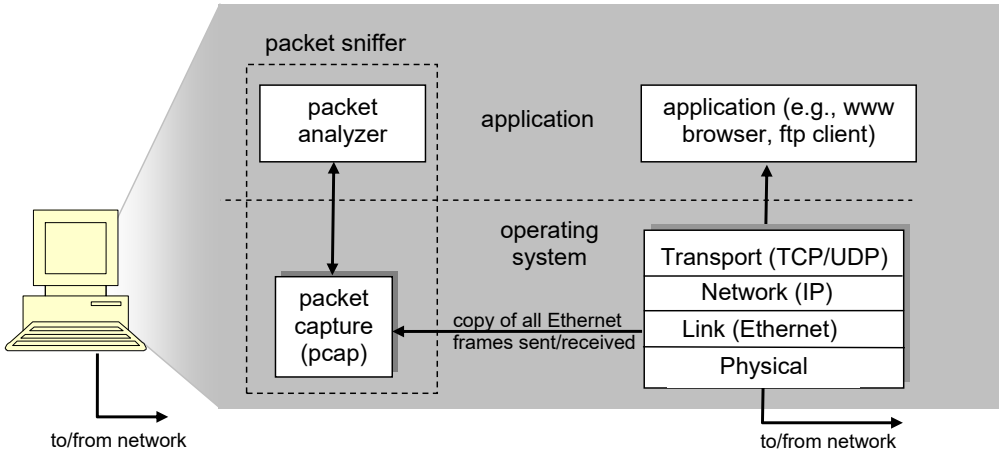

Figure 1: Packet sniffer structure

The second component of a packet sniffer is the **packet analyzer**, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer must "understand" the structure of all messages exchanged by protocols. For example, suppose we are interested in displaying the various fields in messages exchanged by the HTTP protocol in Figure 1. The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment (as HTTP uses TCP as underlying transport layer protocol) within the IP datagram. Finally, it understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment. Finally, it understands the HTTP protocol.

There are many packet sniffers available, for example Wireshark packet sniffer, Ethereal Network Analyzer, Snoop Analyzer Standard, Network Probe, etc. We will be using the Wireshark packet sniffer [http://www.wireshark.org/] for these labs, allowing us to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. (Technically speaking, Wireshark is a packet analyzer that uses a packet capture library in your computer). Wireshark is a free network protocol analyzer that runs on Windows, Linux/Unix, and Mac computers. It operates in computers using Ethernet, serial (PPP and SLIP), 802.11 wireless LANs, and many other link-layer technologies (if the OS on which it's running allows Wireshark to do so).

Running Wireshark

When you run the Wireshark program, you'll get a startup screen, as shown in Figure-2:

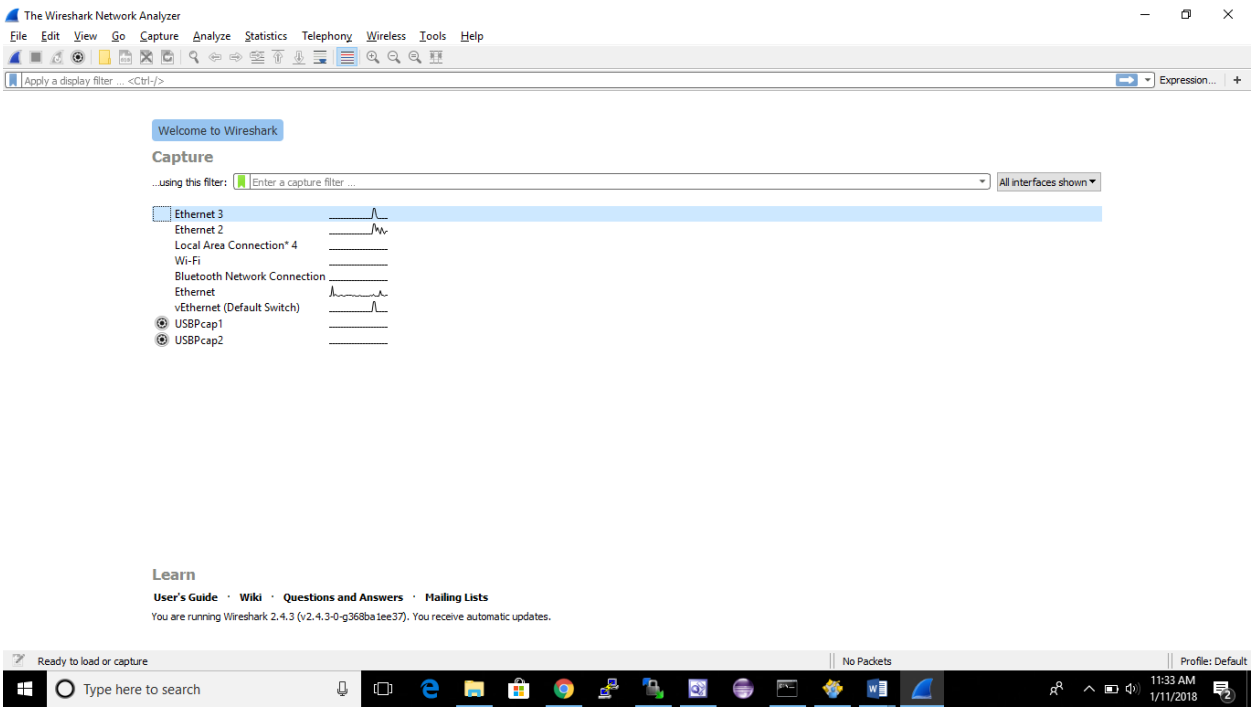

Figure-2: Start up screen of Wireshark.

Take a look at the screen – you'll see an "Interface list". This is the list of network interfaces on your computer. Once you choose an interface, Wireshark will capture all packets on that interface. In the example above, there is an Ethernet interface (Gigabit network Connection) along with some other. Use is for this lab.

Click on "Ethernet" interface to start packet capture (i.e., for Wireshark to begin capturing all packets being sent to/from that interface), a screen like the one shown in Figure-3 will be displayed, showing information about the packets being captured. Once you start packet capture, you can stop it by using the Capture pull down menu and selecting Stop.

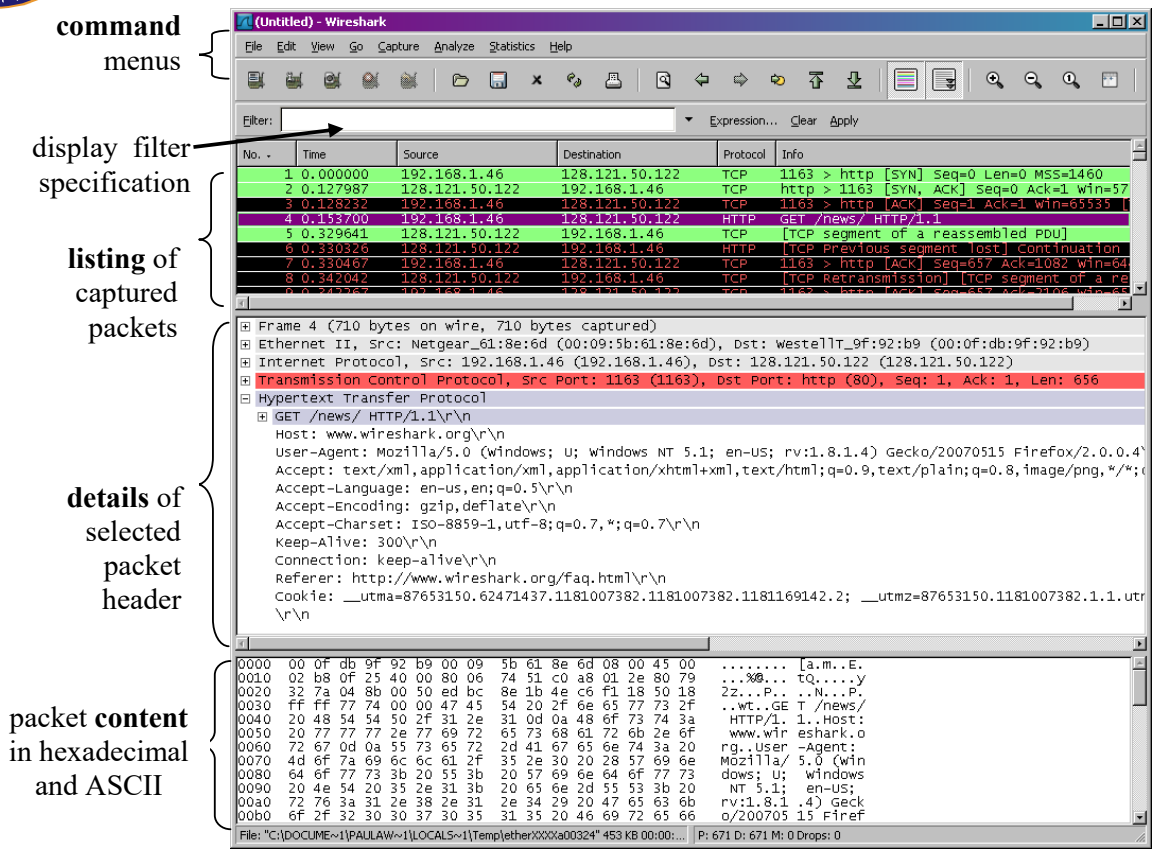

Figure 3: Wireshark Graphical User Interface, during packet capture and analysis

The Wireshark interface has five major components:

- The **command menus** are standard pulldown menus located at the top of the window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data, and exit the Wireshark application. The Capture menu allows you to begin packet capture.
- The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is *not* a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest-level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.
- The **packet-header details window** provides details about the packet selected (highlighted) in the packet-listing window. (To select a packet in the packet-listing window, place the cursor over the packet's one-line summary in the packet-listing window and click with the left mouse button.). These details include information about the Ethernet frame (assuming the packet was sent/received over an Ethernet interface) and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the plus minus boxes to the left of the Ethernet frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be

expanded or minimized. Finally, details about the highest-level protocol that sent or received this packet are also provided.

- The **packet-contents window** displays the entire contents of the captured frame, in both ASCII and hexadecimal format.

Towards the top of the Wireshark graphical user interface, is the **packet display filter field,** into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows). In the example below, we'll use the packet-display filter field to have Wireshark hide (not display) packets except those that correspond to HTTP messages.

Taking Wireshark for a Test Run

Experiment-1

The best way to learn about any new piece of software is to try it out! Do the following

1. Start up your favorite web browser.

2. Start up the Wireshark software. You will initially see a window similar to that shown in Figure 2. Wireshark has not yet begun capturing packets. Select the desired network interface. Packet capture will now begin - Wireshark is now capturing all packets being sent/received from/by your computer!

3. Once you begin packet capture, a window similar to that shown in Figure 3 will appear. This window shows the packets being captured. By selecting *Capture* pulldown menu and selecting *Stop*, you can stop packet capture. But don't stop packet capture yet. Let's capture some interesting packets first. To do so, we'll need to generate some network traffic. Let's do so using a web browser, which will use the HTTP protocol.

4. While Wireshark is running, enter the URL:
   https://icms.bits-pilani.ac.in/lab-1-webpages/lab1_first_page.htm
   and have that page displayed in your browser. In order to display this page, your browser will contact the HTTP server and exchange HTTP messages with the server in order to download this page. The Ethernet frames containing these HTTP messages (as well as all other frames passing through your Ethernet adapter) will be captured by Wireshark.

5. After your browser has displayed the lab1_first_page.htm, stop Wireshark packet capture by selecting stop in the Wireshark capture window. The main Wireshark window should now look similar to Figure 3. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! The HTTP message exchanges with the web server should appear somewhere in the listing of packets captured. But there will be many other types of packets displayed as well (see, e.g., the many different protocol types shown in the *Protocol* column in Figure 3). Even though the only action you took was to download a web page, there were evidently many other protocols running on your computer that are unseen by the user. We'll learn much more about these protocols as we progress through the text! For now, you should just be aware that there is often much more going on than "meet's the eye"!

6. Type in "http" (without the quotes, and in lower case – all protocol names are in lower case in Wireshark) into the display filter specification window at the top of the main Wireshark window.

Then select *Apply* (to the right of where you entered "http"). This will cause only HTTP message to be displayed in the packet-listing window.

7. Find the HTTP GET message that was sent from your computer to the webserver. When you select the HTTP GET message, the Ethernet frame, IP datagram, TCP segment, and HTTP message header information will be displayed in the packet-header window. Recall that the HTTP GET message that is sent to the gaia.cs.umass.edu web server is contained within a TCP segment, which is contained (encapsulated) in an IP datagram, which is encapsulated in an Ethernet frame. By clicking on '+' and '-' right-pointing and down-pointing arrowheads to the left side of the packet details window, *minimize* the amount of Frame, Ethernet, Internet Protocol, and Transmission Control Protocol information displayed. *Maximize* the amount information displayed about the HTTP protocol. Your Wireshark display should now look roughly as shown in Figure 4. (Note, in particular, the minimized amount of protocol information for all protocols except HTTP, and the maximized amount of protocol information for HTTP in the packet-header window).

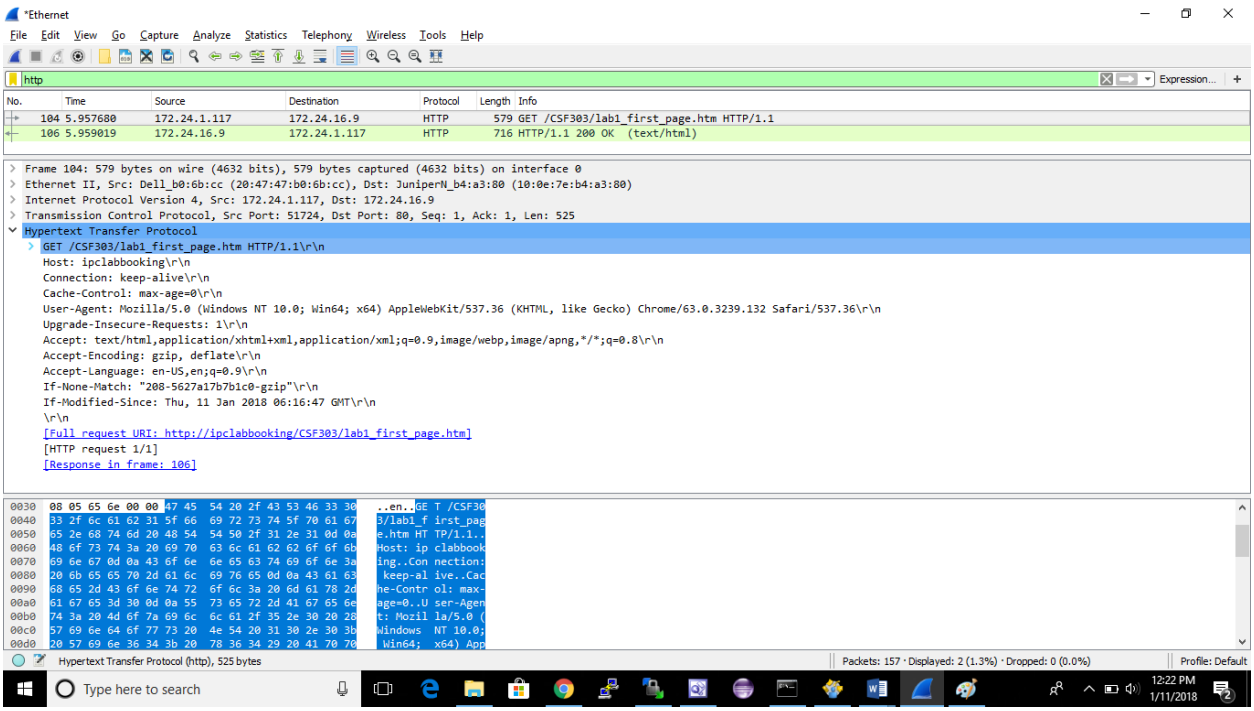

Figure-4: GET message details

8. Save the packet capture and Exit Wireshark

Now try to answer these questions:

1. What does \r\n mean after each line?
2. What indicates the end of HTTP message?
3. What is the version of HTTP used? Is it necessary to pass it in message? Why?
4. What type of connection it is: persistent or non-persistent?
5. What is the source IP address and destination IP address?
6. Which TCP protocol does HTTP use?
7. What is the source and destination port number?
8. How long did it take from when the HTTP GETT message was sent until HTTP OK reply was received?
9. What is the difference between date header and Last-modified header in HTTP OK reply.?
10. Although URL specifies the host, why do you think that it is again the part of HTTP GET message?
11. Does HTTP work on specific port? How can it be verified from GET and OK messages.
12. You can see a line "Response in Frame: *some_number*". What does it mean? Verify your answer.

Experiment-2

1. Clear browsing history. You can do it using tools menu of your Brower.
2. Start Wireshark and apply filter "http" (without quotes) as we are interested in http packets.
3. Type the following URL in your web browser and press enter:
   https://icms.bits-pilani.ac.in/lab-1-webpages/lab1_second_page.htm
4. Stop capturing packets and observe different HTTP packets. Observe that there are two embedded images in the web page.
   Q1. How many HTTP GET requests are generated?
   Q2. What can be concluded from your answer of Q1 above?
   Q3. What is the status code returned in all HTTP response packets?
   Q4. When were the objects embedded in web page last modified at the server?
   Q5. How many bytes of content are being returned to your browser in each HTTP response packet?
5. Now, start Wireshark packet capture again and refresh the webpage, i.e., enter URL:
   https://icms.bits-pilani.ac.in/lab-1-webpages/lab1_second_page.htm
6. Stop capturing packets and observe different HTTP packets again.
   Q1. What is the date header value and Last Modified value in HTTP response packets?
   Q2. Are two embedded images fetched from the server or were locally cached? How to verify your answer? Justify the answer to yourself.
7. Now, start Wireshark packet capture again and enter the following URL:
   https://icms.bits-pilani.ac.in/lab-1-webpages/lab1_third_page.htm
   Stop capturing packets and observe different HTTP packets again.
   Q1. You will observe that one image is not displayed. Why so?
   Q2. Does path of fist image (corresponding to "sixth edition") is same as that in step 7 above?

Experiment 3

1. Clear browsing history.
2. Start Wireshark and apply filter "http" (without quotes) as we are interested in http packets.
3. Type the following two URL's in your web browser and press enter:
   https://icms.bits-pilani.ac.in/lab-1-webpages/lab1_fourth_page.htm
4. https://icms.bits-pilani.ac.in/lab-1-webpages/lab1_fifth_page.htm
   Again observe the http packets exchanged in above two cases.

Q1. You must have observed that in "fifth_page" the size of two images is more than that with respect to "forth_page". In "fifth_page" the images are the one which is cached or the one which are brought from the server? What can be concluded from your observation?

In all the above experiments see whether response from the server was in one single packet or multiple packets. You will have to observe TCP packets for it in each case.

Part 2: Traceroute Program- A handy debugging tool for network administrators

The Traceroute program, written by Van. It lets us see the route that IP datagrams follow from one host to another.

Although there are no guarantees that two consecutive IP datagrams from the same source to the same destination follow the same route, most of the time they do.

Traceroute uses ICMP (*don't worry if you do not understand the ICMP protocols at this point in time*) and the TTL field in the IP header. The TTL field (time-to-live) is an 8-bit field that the sender initializes to some value. The sender initializes to some value. Each router that handles the packet (datagram) is required to decrement the TTL by either one. The purpose of the TTL field is to prevent datagrams from ending up in infinite loops, which can occur during routing transients.

When a router gets an IP datagram, whose TTL is either 0 or 1 it must not forward the datagram. (A destination host that receives a datagram like this can deliver it to the application, since the datagram does not have to be routed. Normally, however, no system should receive a datagram with a TTL of 0.) Instead the router throws away the datagram and sends back to the originating host an ICMP "time exceeded" message. **The key to Traceroute is that the IP datagram containing this ICMP message has the router's IP address as the source address.**

We can now guess the operation of Traceroute. It sends an IP datagram with a TTL of 1 to the destination host. The first router to handle the datagram decrements the TTL, discards the datagram, and sends back the ICMP time exceeded. This identifies the first router in the path. Traceroute then sends a datagram with a TTL of 2, and we find the IP address of the second router. This continues until the datagram reaches the destination host. But even though the arriving IP datagram has a TTL of 1, the destination host won't throw it away and generate the ICMP time exceeded, since the datagram has reached its final destination. **How can we determine when we've reached the destination?**

Traceroute sends UDP datagrams to the destination host, but it chooses the destination UDP port number to be an unlikely value (larger than 30,000), making it improbable that an application at the destination is using that port. This causes the destination host's UDP module to generate an ICMP "port unreachable" error when the datagram arrives. All Traceroute needs to do is differentiate between the received ICMP messages-time exceeded versus port unreachable-to know when it's done.

We're now ready to run traceroute and see the output. To perform the experiment, We'll use a **Traceroute** available online at the website https://www.ultratools.com/tools/traceRoute or you can visit to http://www.traceroute.org/ and choose a traceroute service.

Following is the output we got when, we performed the traceroute for "**google.com**". You can directory write the domain name (like google.com) or the IP address of that domain to run the Traceroute program.

traceroute to 172.217.23.238 (172.217.23.238), 30 hops max, 60 byte packets
1 praha-4d-c1-vl55.masterinter.net (77.93.199.253) 3.650 ms 3.718 ms 3.838 ms
2 vl1387.cr3.r1-8.dc1.4d.prg.masterinter.net (83.167.254.150) 0.148 ms 0.357 ms 0.366 ms
3 72.14.214.168 (72.14.214.168) 0.387 ms 0.376 ms 0.375 ms
4 108.170.245.49 (108.170.245.49) 0.285 ms 0.295 ms 108.170.245.33 (108.170.245.33) 0.274 ms
5 108.170.238.155 (108.170.238.155) 0.365 ms 108.170.238.157 (108.170.238.157) 0.337 ms
108.170.238.155 (108.170.238.155) 0.313 ms
6 prg03s06-in-f14.1e100.net (172.217.23.238) 0.192 ms 0.185 ms 0.235 ms

Lab Exercise-6: Once you run the Traceroute from your computer, the output received by you may not be exactly same as above. (Note: It is suggested that you try it yourself.)
Now try to answer the following.

1. Explain the meaning of first line of the output.
2. The next two lines in the output begin with the TTL, followed by the name of the host or router its IP address and three different time values. What these time values signifies?
3. How we can calculate the per hop time value?

Repeat the above experiment for the BITS Pilani web site and **iitd.ac.in** and observe the output

Lab Exercise-7:
Answer the following:

1. Did you observe the character * in few lines of the output for any of the trace route? If yes, then what does it mean?
2. Did you see the last hop in your output as the destination you are looking for? If no, then what could be the reason for this?

*****