EL317 – DATA COMMUNICATION NETWORKS

LAB MANUAL

DEPARTMENT OF ELECTRICAL ENGINEERING,

FAST-NU, LAHORE

Created by: Dr. Saima Zafar

Date: September, 2013

Last Updated by: Mr. Waqas Rehman, M. Asim Ahsan

Date: August 2019

Approved by the HoD: Dr. S. M. Sajid

Date: August 2019

Table of Contents

List of Equipment

Sr. No.	Description
1	Workstations (PCs)
2	Network Interface Card
3	Ethernet CAT5 cable
4	RJ45 Connector
5	Wire-shark (software)
6	Visual C++ (software)
7	Network simulator (NS-3) (software)
8	Graphical network simulator (GNS3) (software)

EXPERIMENT 1

INTRODUCTION TO NETWORKING CONCEPTS

OBJECTIVE:

- To get familiarized with network cables and network interface card
- Learn different network topologies

BACKGROUND:

This is an introductory lab; in this lab, we will discuss network topologies, and study the structure of the interface card pins and its role in different types of cabling techniques namely, cross cable and straight cable.

A few terms that will be used frequently:

Node
Any electronic device that is attached to a network; and is capable of sending, receiving, or forwarding information over a communications channel e.g. Printer, computer, digital camera, scanner etc.

Topology
The manner in which the nodes and links are physically arranged in a network.

Understanding the interface card
The interface card for each computer is similar in structure although its shape may be different. Just like all humans are same in structure but different in appearance. This fact has a remarkable implication on the way two computers can communicate with each other. Each interface card has a pair of pins for transmission, we call it Tx+ and Tx-, and similarly we have a pair for reception Rx+ and Rx-.

Ethernet cable structure
An Ethernet cable consists of four pairs of wires. In order to identify the wires at the ends the pairs are colored. The four pairs are blue/white blue, green/white green, orange/orange green, brown/white brown. The wires are usually twisted to reduce cross talk. There are different categories of wires; each having a different set of specifications in relation to speed, performance, noise (cross talk) etc. e.g. CAT5 and CAT6.

CAT5 cable
Category 5 cable includes four twisted pairs in a single cable jacket. This use of balanced lines helps preserve a high signal-to-noise ratio despite interference from both external sources and other pairs (this latter form of interference is called crosstalk). It is most commonly used for 100 Mbit/s networks, such as 100BASE-TX Ethernet, although IEEE 802.3ab defines standards for 1000BASE-T - Gigabit Ethernet over category 5 cable. Cat5 cable typically has three twists per inch of each twisted pair of

24-gauge copper wires within the cables.

The RJ45 connector (8P8C)

An RJ45 connector, shown in figure 1.1 and 1.2, is connected to the ends of the Ethernet cable. 8P8C stands for 8 points 8 contact. The NIC's have a slot for connecting such pins in them.

Figure 1.1. RJ45 connector

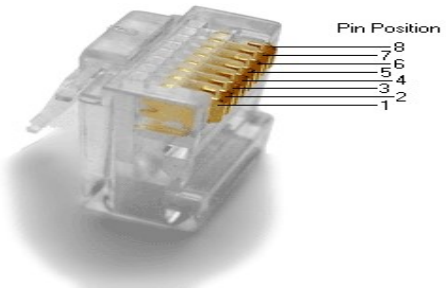

Figure 1.2. RJ45 pin positions

The cross cable

In order to communicate the Tx+ must be connected to the Rx+ and Tx- to Rx-. This means that the wires need to be crossed in order for successful communication. A cable with such crossed connections is known as a cross cable which is shown in figure 1.3. This kind of cable is generally used to connect same kind of devices directly.

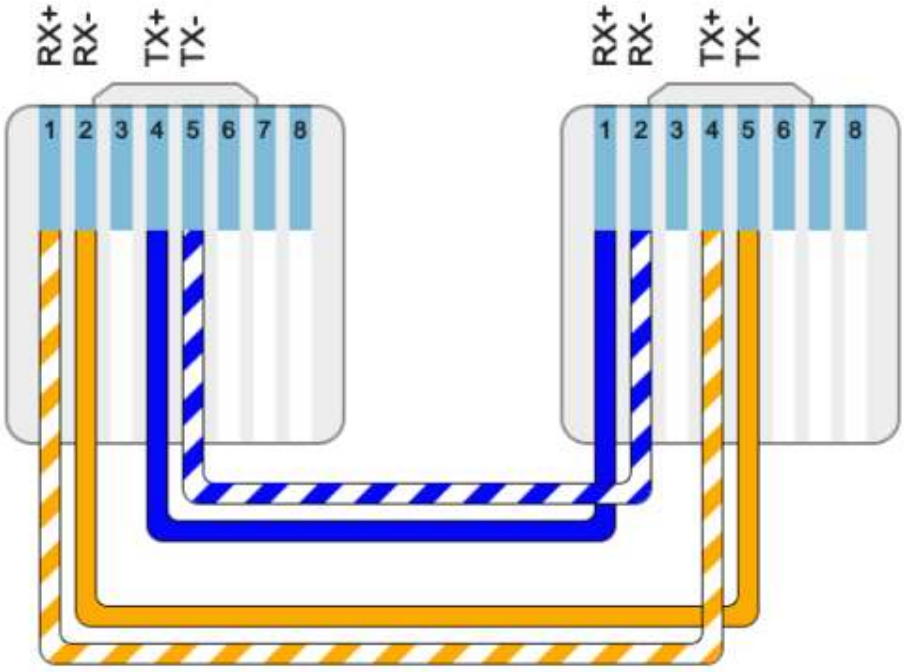

Figure 1.3 Cross cable configuration

The straight cable

As is obvious from the description of cross cable, in a straight cable the wires are not crossed but are wired as it is in a straight fashion. Straight cable is used to connect a computer to a switch or a router. Both straight and cross cable configuration is shown in figure 1.4

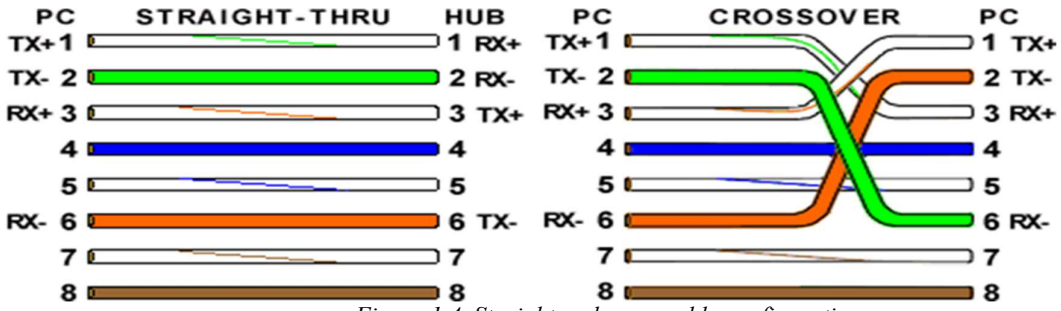

Figure 1.4. Straight and cross cable configuration

POST LAB TASK:

Write a report (Maximum of three pages) on different topologies that are used in networks. Compare them with each other and describe the fundamental differences.

Submit hand written hard copy of your word document in next lab session. Your report should be in your own words.

Note: Your report should be of maximum 2 pages.

EXPERIMENT 2

INTRODUCTION TO NETWORK SIMULATOR VERSION-3 (NS-3)

OBJECTIVE:

- Build and analyze simple topology using point to point link

BACKGROUND:

This is an introductory lab on network simulator version 3. We'll review some terms that are commonly used in networking, but have a specific meaning in ns-3. To explain the concepts and abstractions a code is also provided.

In ns-3 there is no concept of the operating system, the basic level of abstraction in ns-3 is the application level-user program. This abstraction is represented in C++ by the class named 'Application'. This class provides methods for managing the representation of developer's version of user-level application in simulations.

The basic ns-3 computing device abstraction is called node represented in C++ by the class 'Node'. Just as software applications run on computers to perform tasks in the "real world," ns-3 applications run on ns-3 Nodes to drive simulations in the simulated world.

Media over which the data flow in networks are called channels. In ns-3 one connects Node to an object representing a channel. Basic communication sub network abstraction is called the channel represented in C++ by the class 'Channel'. A simple channel can model a wire whereas specialized channels can model complicated things such as an Ethernet switch. Specialized versions of Channel that will be used in the lab are 'CSMA Channel', 'PointToPoint Channel' and 'Wi-Fi Channel'.

Network interface cards are controlled using the network device drivers called net devices referred in UNIX by names as eth0. A net device is installed in the Node in order to enable the Node to communicate with other Nodes in simulation via Channels. The netdevice abstraction represented in C++ by class 'NetDevices'. Several specialized netdevices are 'CsmaNetDevice', 'PointToPointNetDevice' and 'WifiNetDevice'. A Node may be connected to more than on Channel via multiple NetDevices.

In a large simulation network we need to connect many Nodes, NetDevices and Channels. Since connecting NetDevices to Nodes, NetDevices to Channels, assigning IP addresses, etc., are such common tasks in ns-3, we provide what we call 'topology helpers' to make this as easy as possible.

Visit the following link in order to understand the abstractions used in ns-3. It also provides you with the line by line explanation of the code given below:

https://www.nsnam.org/docs/release/3.20/tutorial/html/conceptual-overview.html#a-first-ns-3-scripT

TASK:

1. Understand each and every line of the above code, run the above script and comment on the output.
2. Modify the code so that 4 packets are sent and echoed.
3. Create 2 node with a single interface as shown in figure 2.1:
4. Point to point link
5. Data rate set to 8 Mbps, transmission delay set to 4ms
6. IP set to 192.168.40.0
7. Udp Echo Server Port set to 93
8. Packet size set to 512 bytes

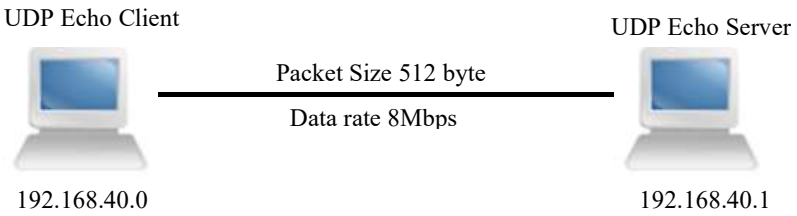

Figure 2.1: Point to Point Network

POST LAB QUESTIONS:

Implement the network topology given in figure 2.2, both the outer nodes, node 1 and node 3, must send number of packets to the center node (node 2) at 5 and 10 seconds respectively. You must set the following parameters.

1. IP address of Node 1 and Node 3 should be 192.168.1.1 and 192.168.5.2 respectively.
2. Point to point link between Node 1 and Node 2
    a. Data rate set to 8 Mbps, transmission delay set to 8ms
3. Point to point link between Node 2 and Node 3
    a. Data rate set to 8 Mbps, transmission delay set to 4ms

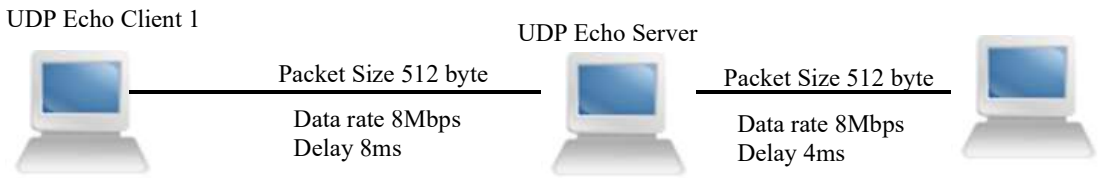

Figure 2.2

EXPERIMENT 3

SIMPLE TOPOLOGY IN NS-3

OBJECTIVE:

- Build and analyze simple topology using point to point link

BACKGROUND:

In this lab you are required to create a simple topology by modifying the code of experiment 2, in which two nodes were created and a point-to-point link was established between the two.
Modify that scenario to add another node shown in figure 3.1:

1. Create a third node (node 2), and set it up with an Internet stack.
2. Create a point-to-point link from node 1 to node 2.
3. The devices on this new link need addresses. Assign addresses on this link.
4. Create a new echo server and client, install the client on node 0 and the server on node 2, and set up the necessary parameters for communication.
5. Modify the echo clients to send six packets each instead of a single packet.
6. Since we have more than just a single link, we need to set up routing. This can be done with the line: **Ipv4GlobalRoutingHelper :: PopulateRoutingTables();**

Put this line after you assign the IP addresses on the links.

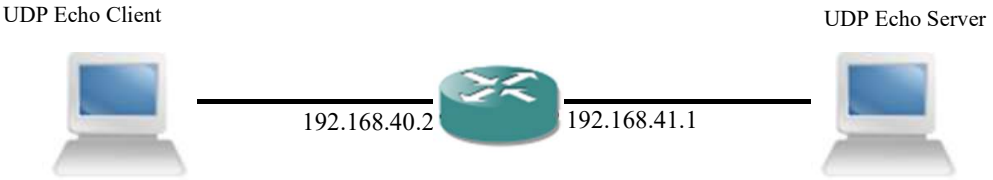

Figure 3.1

POST LAB QUESTIONS:

Modified above code to implement the following topology shown in figure 3.2

1. Set node 3 and node 4 as 'UdpEchoServer' having port number 93 and 94 respectively.
2. Set node 1 as 'UdpEchoClient' and must send packet to both Node 3 and 4.

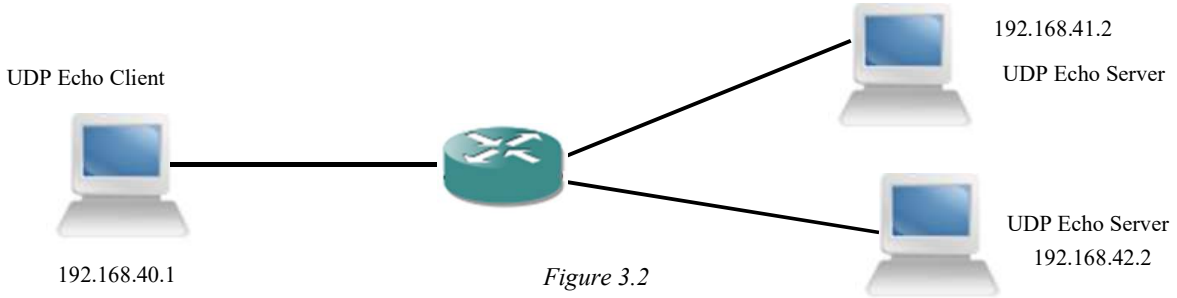

Figure 3.2

EXPERIMENT 4

BUILDING TOPOLOGIES IN NS-3

OBJECTIVE:

- Build and analyze the network using CSMA channel

BACKGROUND:

You have implemented simple (point to point topology) topology in lab 2 and 3. In today's lab we are going to implement point to point, CSMA (LAN) and Wi-Fi Network as shown in figure 4.1.

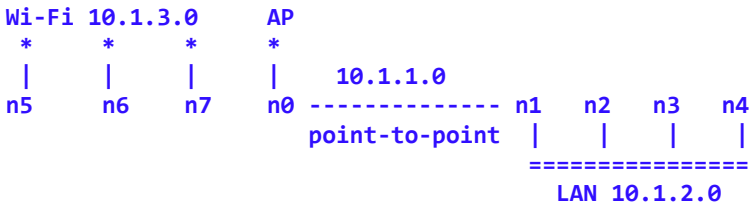

Figure 4.1

Step 1

First of all, you have to create simple point to point topology between two nodes, which you have already implemented in last two labs.

Step 2

Now we are going to extend our point-to-point example (the link between the nodes n0 and n1 below) by hanging a bus network off of the right side. This can be done by implement CSMA channel. Our task is to create four nodes on LAN (CSMA channel) – one required node and three extra nodes.

Step 3

You can see that we are adding a new network device to the node on the left side of the point-to-point link that becomes the access point for the wireless network. A number of wireless STA nodes are created to fill out the new 10.1.3.0 network as shown on the left side of the illustration.

Visit the following link in order to understand the code. It also provides you with the line by line explanation of the code given below:
https://www.nsnam.org/docs/release/3.20/tutorial/html/building-topologies.html

TASK:

Modified above code to implement the following topology shown in figure 4.2

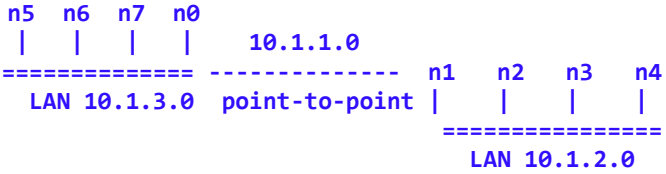

Figure 4.2

1. Set node 3 as 'UdpEchoServer' port set to 93.
2. Set node 6 as 'UdpEchoClient'

POST LAB QUESTIONS:

Modified above code to implement the following topology shown in figure 4.3

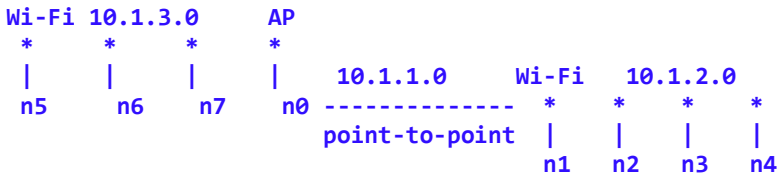

Figure 4.3

1. Set node 3 as 'UdpEchoServer' port set to 93.
2. Set node 6 as 'UdpEchoClient'.

EXPERIMENT 5

BUILDING TOPOLOGIES IN NS-3

OBJECTIVE:

- Implement the star network topology

BACKGROUND:

You have implemented point to point, CSMA and Wi-Fi topologies. In today's lab we are going to strengthen those skills with implementation of the Star topology shown in figure 5.1:

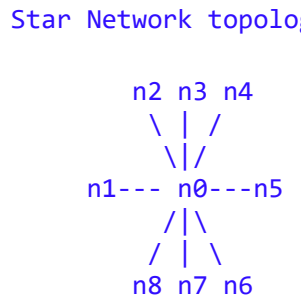

Figure 5.1: Star Network

Each link is a point to point link. The central node (n0) is the packet sink. The other nodes generate traffic, which is sent to the sink. The code for this topology is in the examples folder (examples/tcp/star.cc). In this particular example, pcap tracing has been enabled, while logging is not enabled. Enable logging by inserting the following lines of code in the beginning of main:

```
LogComponentEnable ("Star", LOG_LEVEL_INFO);
LogComponentEnable ("PacketSink", LOG_LEVEL_INFO);
LogComponentEnable ("OnOffApplication", LOG_LEVEL_INFO);
NS_LOG_INFO ("Star Topology Simulation");
```

Now running the code will show transmission and reception of packets in the terminal window.

TASK:

Modify the above topology as shown below in figure 5.2:

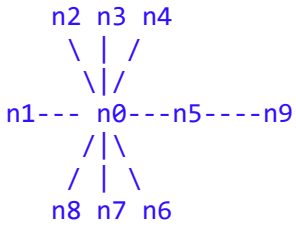

Figure 5.2: Modified Star Network

Do not install any applications on nodes 0 and 5. Make node 9 (n9) the packet sink and send all packets (from n1-n4 and n6-n8) to this sink. All links are point to point links.

POST LAB QUESTION:

Modify the star topology as below in the figure 5.3

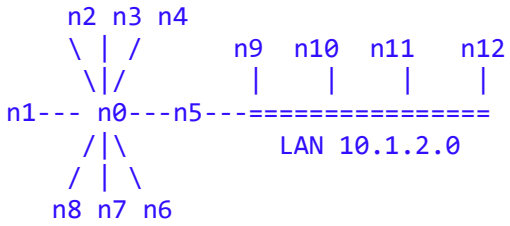

Figure 5.3

Do not install any application layer on n0 and n5. Make n12 packet sink and send all packets from n1-n4 and n6-n11 to this sink.

EXPERIMENT 6

INTRODUCTION TO CALLBACKS IN NS-3

OBJECTIVE:

- Implement the callbacks for data tracing

BACKGROUND:

The purpose of using the simulator is to gather data for analysis. While the simulator provides options for data collection in the form of both pcap and Logging, these are difficult to customize. To deal with this issue, NS-3 provides a set of tracing tools. Tracing can include many methods, from simple std::cout to file output. In this lab, we will consider one of the most important methods, callbacks.

Put simply, callbacks are event triggered function calls. NS-3 is a discrete event simulator; most events can be configured for call backs. The callback itself is written along lines similar to that of c++ functions; it can then be used to output information, keep counts and update files. , please go through the tutorial provided at http://www.nsnam.org/docs/tutorial/html/tracing.html#. The examples use fourth and fifth and sixth.cc, it is a good idea to practically implement and follow the tutorial up to "Using Mid-Level Helpers".

TASK:

Start with examples/wireless/wifi-app.cc. Understand the callbacks implemented in the code. Make additions/modifications to count the total number of packets received correctly and the total number of packets received with errors. Is it possible to modify the code so that statistics at each node are collected individually?

POST LAB QUESTIONS:

Understand each line of the code file main-callbacks.cc, run the script. Also change the parameter and observe the change in the output.

EXPERIMENT 7

NETWORK ANALYSIS

OBJECTIVE:

- To understand different computer network tools like ping, tracert and their usage

BACKGROUND:

IP Address

The IP address is a unique 32-bit address used to identify each machine on the network. Every device connected to the internet has its own unique identifying number, which is called its **IP address** or **Internet Protocol address**. An example of a computer IP address would be 192.168.1.10, where the number 10 in this case corresponds to the computer's address on the local network.

IP Config Command

This command gives IP information about interfaces within that machine.

Go to:
Start → Run → cmd
>ipconfig

Write down the output:

Ping

Ping (Packet Internet Groper) is a computer network tool used to test whether a particular host is reachable across an IP network. Ping works by sending a small packet of information containing an ICMP ECHO_REQUEST to a specified computer, which then sends an ECHO_REPLY packet in return. ICMP is used by hosts and routers to communicate network layer information to each other. The most common use of ICMP is for error reporting. ICMP messages are carried as IP payloads. The Ping program sends an ICMP type 8 code 0 message to the specified host. The destination host seeing the echo request sends back a type 0 code 0 ICMP echo reply.

Ping places a unique sequence number on each packet it transmits, and reports which sequence numbers it receives back. Thus, you can determine if packets have been dropped, duplicated, or reordered. Ping checksums each packet it exchanges. You can detect some forms of damaged packets.

Ping places a timestamp in each packet, which is echoed back and can easily be used to compute how long each packet exchange took - the Round Trip Time (RTT).

Write Output of the following statements:

>ping ***ipaddress***

>ping -t *ipaddress* (keep pinging until manually stopped)

>ping -n *count ipaddress* (ping with n packets)

Tracert

Traceroute is a computer network tool used to determine the route taken by packets across an IP network.

Paste the output of following command

>**tracert** *hostname [use any website e.g., www.google.com]*
(Note: Min, Max and average hop times are displayed in-order)

Briefly explain what do you understand from the output?

Produce route trace using the site http://www.net.princeton.edu/traceroute.html for the following URLs:

1. www.nu.edu.pk
2. www.mit.edu

Paste your output here:

MAC Address

A MAC address is also sometimes referred to as an **E**thernet **H**ardware **A**ddress (EHA), **hardware address** or **physical address**.

A **MAC address** is essentially a unique, fixed **serial number** of your computer's **network card** in a globally agreed format. This network card can also be referred to as a Network adapter or **N**etwork **I**nterface **C**ard (NIC).

The standard form for MAC addresses is six groups of two hexadecimal digits (characters **0 – 9** and **A – F**) separated by a hyphen (-) or a colon (:):

12 – 34 – 56 – 78 – 90 – AB

To find out your computers MAC Address open the command prompt and type ipconfig /all and then press the ENTER. The window should then display a whole load of information about all of the network cards in your machine.

Write down MAC address of system you are using.

The MAC address for each network adapter is displayed as its **physical address**.

ARP

Address resolution Protocol (ARP) is a network layer protocol used to convert an IP address into a physical address. A host wishing to obtain a physical address broadcasts an ARP request onto the TCP/IP network. The host on the network that has the IP address in the request then replies with its physical hardware address.

Write down the output of the following command.

- **arp –a** *(Display the ARP cache tables for all interfaces)*

RARP

Reverse Address Resolution Protocol (RARP) is a network layer protocol used to obtain an IP address for a given hardware address.

POST LAB QUESTIONS:

- Why we need both IP and MAC addresses?
- What is the purpose of "Maxium hop limit" in tracert command?
- Why is a "rarp" command not needed?

EXPERIMENT 8

WIRE-SHARK: NETWORK ANALYSIS

OBJECTIVE:

- Learn the use of Wireshark to understand different Internet Protocol

BACKGROUND:

Wireshark (*Ethereal*) is a packet sniffing tool. As the name suggests, a packet sniffer captures 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 as shown in figure 8.1. A packet sniffer 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.

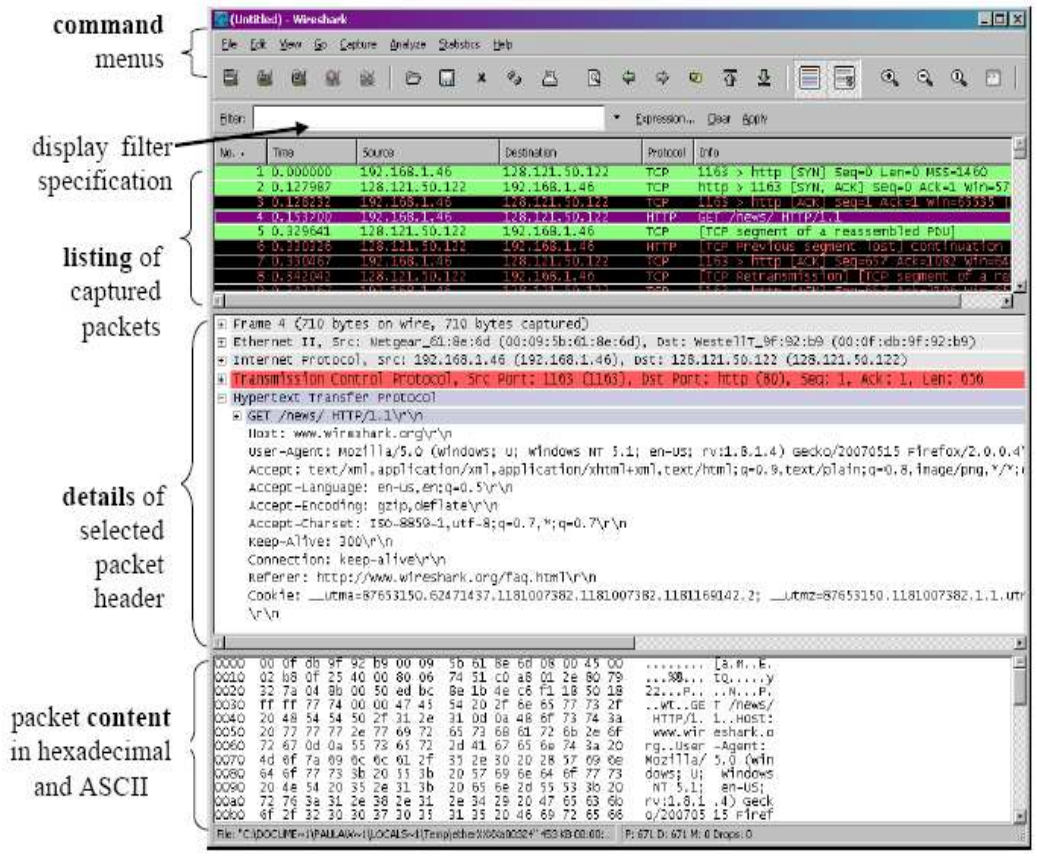

Figure 8.1

1. Start WireShark . Open your browser.

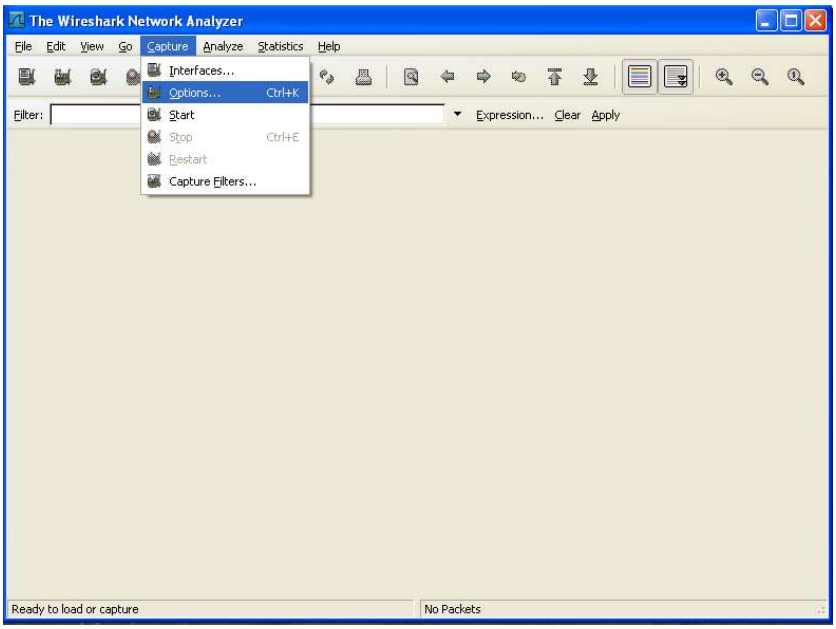

Figure 8.2

2. In the Options uncheck "Hide Capture Info Dialog illustrated in figure 8.2. Change your Interface to the appropriate one from the .drop-down list. Then Press the capture Start button to start capturing as shown in figure 8.3.

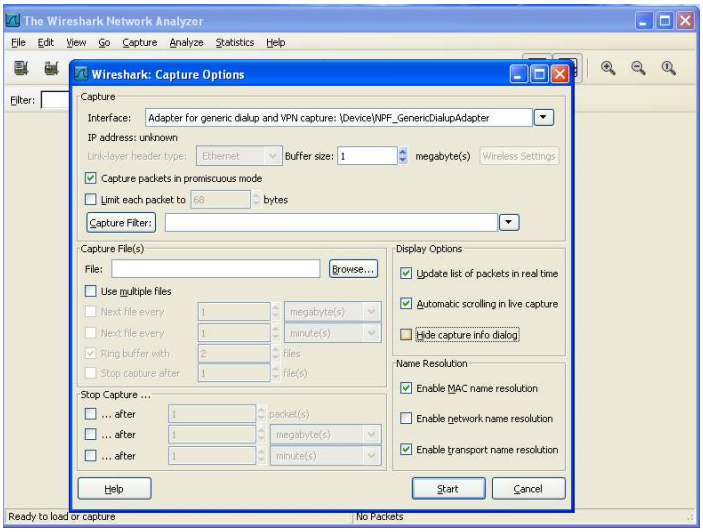

Figure 8.3

3. While WireShark is running enter the URL
   http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html

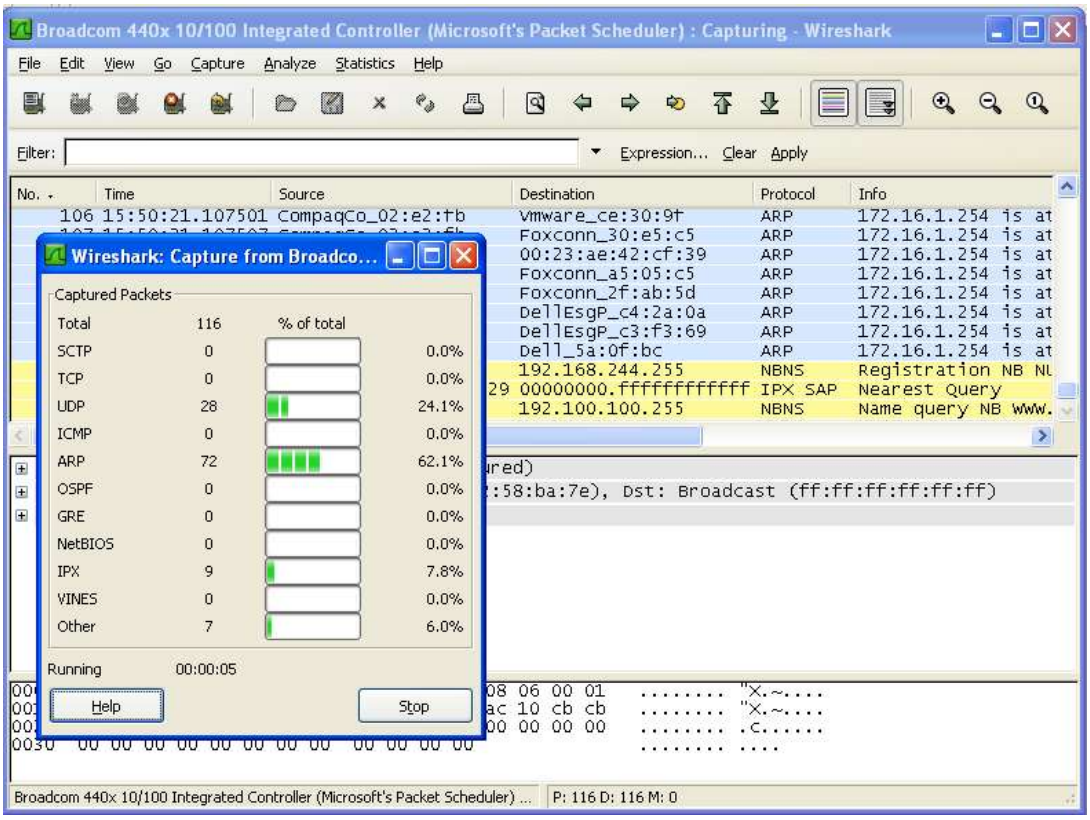

Figure 8.4

4. After your browser has displayed the INTRO-wireshark-file1.html page, stop Wireshark packet capture by selecting stop in the Wireshark capture window as shown in figure 8.4. This will cause the Wireshark capture window to disappear and the main Wireshark window to display all packets captured since you began packet capture.

   - List up to 10 protocols that appear in the protocol column

   - How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received?

- What is the Internet address of the gaia.cs.umass.edu? What is the Internet address of your computer?

- Write down the sequence number of the first TCP packet.

POST LAB QUESTIONS:

- What do you mean by TCP three-way handshaking? Identify SYN, SYN-ACK and ACK packets generated for TCP connection setup.
- What is the purpose of FIN and ACK flags in TCP header?

EXPERIMENT 9

Wireshark Lab- Analysis of Hyper Text Transfer Protocol (HTTP)

OBJECTIVE:

- Understand the Hyper Text Transfer Protocol using Wireshark

BACKGROUND:

In this exercise, we'll explore several aspects of the HTTP protocol: the basic GET/response interaction, HTTP message formats, retrieving large HTML files, retrieving HTML files with embedded objects, and HTTP authentication and security.

Note: For each of the five scenarios given below you have to *note down on the manual* the prints of relevant packets dump from Wireshark.

1. The Basic HTTP GET/response interaction

Let's begin our exploration of HTTP by downloading a very simple HTML file - one that is very short, and contains no embedded objects. Do the following:

1. Start up your web browser.

2. Start up the WireShark packet sniffer, as described in the introductory lab (but don't yet begin packet capture).

3. Wait a bit more than one minute (we'll see why shortly), and then begin Wireshark packet capture.

4. Enter the following to your browser
http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html
Your browser should display the very simple, one-line HTML file.

5. Stop Wireshark packet capture. Enter "http" (just the letters, not the quotation marks) in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.

Your Wireshark window should look similar to the window shown in Figure 1.

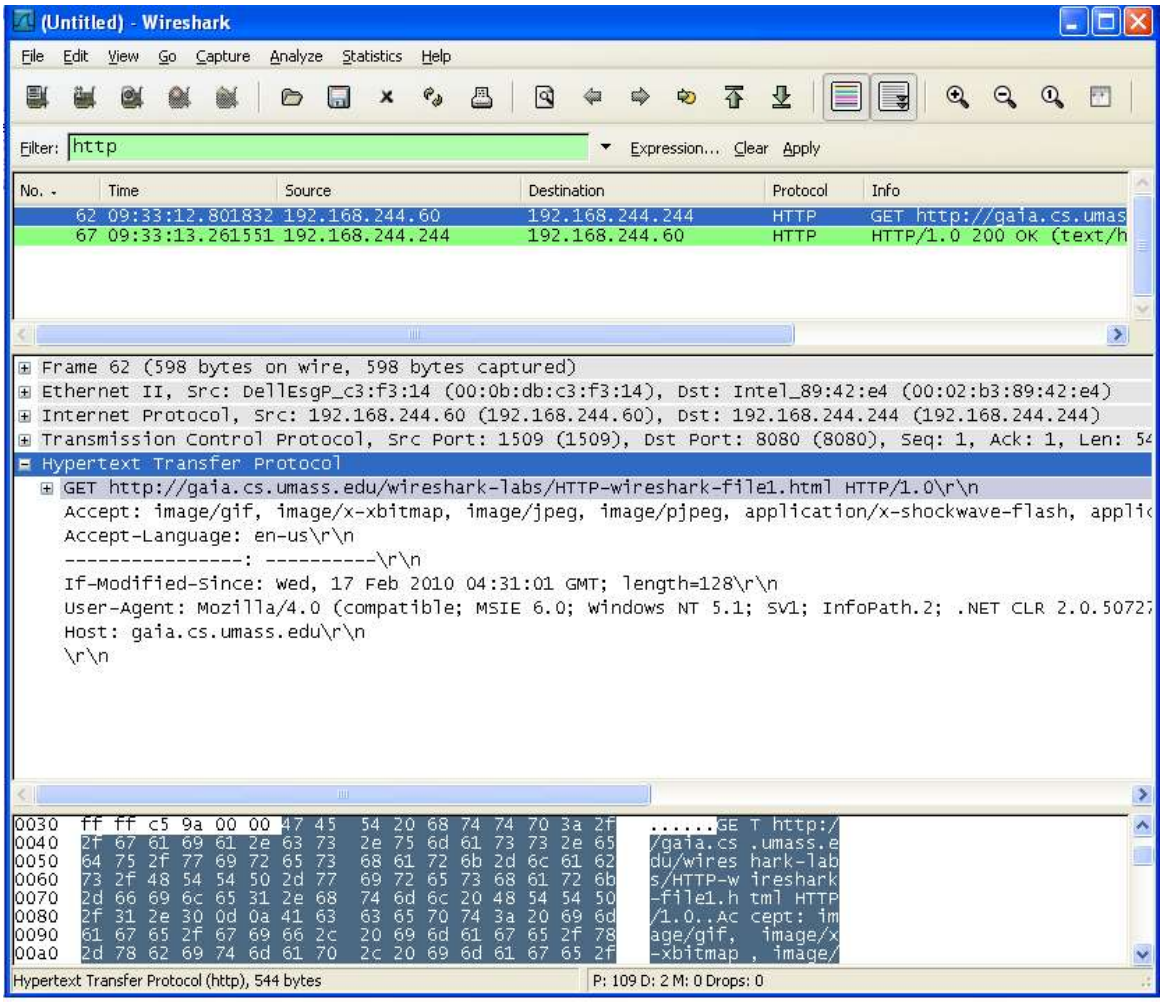

Figure 9.1

The example in Figure 9.1 shows in the packet-listing window that two HTTP messages were captured: the GET message (from your browser to the gaia.cs.umass.edu web server) and the response message from the server to your browser. The packet-contents window shows details of the selected message (in this case the HTTP GET message, which is highlighted in the packet-listing window). Recall that since the HTTP message was carried inside a TCP segment, which was carried inside an IP datagram, which was carried within an Ethernet frame, Wireshark displays the Frame, Ethernet, IP, and TCP packet information as well. We want to minimize the amount of non-HTTP data displayed (we're interested in HTTP here, and will be investigating these other protocols is later labs), so make sure only HTTP line is selected (which means that all information about the HTTP message is displayed).

By looking at the information in the HTTP GET and response messages, answer the following questions. When answering the following questions, print out the GET and response messages and indicates where in the message you've found the information.

Now answers the following questions

1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?

2. What languages (if any) does your browser indicate that it can accept to the server?

4. What is the status code returned from the server to your browser?

5. When was the HTML file that last modified at the server?

6. How many bytes of content are being returned to your browser?

7. By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one.

2. The HTTP CONDITIONAL GET/Response Interaction

Most web browsers perform object caching and thus perform a conditional GET when retrieving an HTTP object. Before performing the steps below, make sure your browser's cache is empty. (To do this for Internet Explorer, select *Tools->Internet Options->Delete File;* these actions will remove cached files from your browser's cache.) Now do the following:

• Start up your web browser, and make sure your browser's cache is cleared, as discussed above.

• Start up the Wireshark packet sniffer

• Enter the following URL into your browser
http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html
Your browser should display a very simple five-line HTML file.

• Quickly enter the same URL into your browser again (or simply select the refresh button on your browser)

• Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.

Now answer the following questions

8. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE" line in the HTTP GET?

9. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?

10. Close the browser and open the link again. Capture this packet. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE:" line in the HTTP GET? If so, what information follows the "IF-MODIFIED-SINCE:" header?

3. Retrieving Long Documents

In our examples thus far, the documents retrieved have been simple and short HTML files. Let's next see what happens when we download a long HTML file. Do the following:

• Start up your web browser, and make sure your browser's cache is cleared, as discussed above.

• Start up the Wireshark packet sniffer

• Enter the following URL into your browser
http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html
Your browser should display the rather lengthy US Bill of Rights.

• Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed.

In the packet-listing window, you should see your HTTP GET message, followed by a multiple-packet response to your HTTP GET request. This multiple-packet response deserves a bit of explanation. The HTTP response message consists of a status line, followed by header lines, followed by a blank line, followed by the entity body. In the case of our HTTP GET, the entity body in the response is the *entire* requested HTML file. In our case here, the HTML file is rather long, and at 4500 bytes is too large to fit in one TCP packet. The single HTTP response message is thus broken into several pieces by TCP, with each piece being contained within a separate TCP segment (see Figure 1.22 in the text). Each TCP segment is recorded as a separate packet by Wireshark, and the fact that the single HTTP response was fragmented across multiple TCP packets is indicated by the "Continuation" phrase displayed by Wireshark. We stress here that there is no "Continuation" message in HTTP!

Now answer the following questions

12. How many HTTP GET request messages were sent by your browser?

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

4. HTML Documents with Embedded Objects

Now that we've seen how Wireshark displays the captured packet traffic for large HTML files, we can look at what happens when your browser downloads a file with embedded objects, i.e., a file that includes other objects (in the example below, image files) that are stored on another server(s). Do the following:

• Start up your web browser, and make sure your browser's cache is cleared, as discussed above.

• Start up the Wireshark packet sniffer

• Enter the following URL into your browser

http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html

Your browser should display a short HTML file with two images. These two images are referenced in the base HTML file. That is, the images themselves are not contained in the HTML; instead the URLs for the images are contained in the downloaded HTML file. As discussed in the textbook, your browser will have to retrieve these logos from the indicated web sites. Our publisher's logo is retrieved from the www.awl.com web site. The image of our book's cover is stored at the manic.cs.umass.edu server.

• Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed.

Now answer the following questions

14. How many HTTP GET request messages were sent by your browser? To which Internet addresses were these GET requests sent?

15. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.

POST LAB QUESTIONS:

HTTP Authentication

Let's try visiting a web site that is password-protected and examine the sequence of HTTP message exchanged for such a site. The URL http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html is password protected. The username is "wireshark-students" (without the quotes), and the password is "network" (again, without the quotes). Open this URL in your browser and capture packets.

Answer the following questions

- What is the server's response (status code and phrase) in response to the initial HTTP GET message from your browser?
- When your browser's sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?
- Analyze the second HTTP GET packet and check if you can see the username and password.
- Is your password on this site secured? If not, explain briefly what additional measures can be taken to secure a website.

Submit two to three pages report with all the relevant figures and details. Your report must contain detailed procedure how you performed this experiment. You must answer these questions with details and relevant figures.

EXPERIMENT 10

ANALYSIS OF DOMAIN NAME SYSTEM (DNS) PROTOCOL

OBJECTIVE:

- Understand the Domain Name System Protocol

THEORY:

Domain Name System (DNS) translates hostnames to IP addresses, fulfilling a critical role in the Internet infrastructure. In this lab, we'll take a closer look at the client side of DNS. Recall that the client's role in the DNS is relatively simple – a client sends a *query* to its local DNS server, and receives a *response* back. Much can go on "under the covers," invisible to the DNS clients, as the hierarchical DNS servers communicate with each other to either recursively or iteratively resolve the client's DNS query. From the DNS client's standpoint, however, the protocol is quite simple – a query is formulated to the local DNS server and a response is received from that server. Before beginning this lab, you'll probably want to review DNS by reading Section 2.5 of the text. In particular, you may want to review the material on **local DNS servers**, **DNS caching**, **DNS records and messages**, and the **TYPE field** in the DNS record.

1. nslookup

In this lab, we'll make extensive use of the *nslookup* tool, which is available in most Linux/Unix and Microsoft platforms today. To run *nslookup* in Linux/Unix, you just type the *nslookup* command on the command line. To run it in Windows, open the Command Prompt and run *nslookup* on the command line.

In it is most basic operation, *nslookup* tool allows the host running the tool to query any specified DNS server for a DNS record. The queried DNS server can be a root DNS server, a top-level-domain DNS server, an authoritative DNS server, or an intermediate DNS server (see the textbook for definitions of these terms). To accomplish this task, *nslookup* sends a DNS query to the specified DNS server, receives a DNS reply from that same DNS server, and displays the result.

```
C:\>nslookup www.mit.edu
Server:  dns-prime.poly.edu
Address:  128.238.29.22

Name:    www.mit.edu
Address:  18.7.22.83


C:\>nslookup -type=NS mit.edu
Server:  dns-prime.poly.edu
Address:  128.238.29.22

Non-authoritative answer:
mit.edu nameserver = bitsy.mit.edu
mit.edu nameserver = strawb.mit.edu
mit.edu nameserver = w20ns.mit.edu

bitsy.mit.edu   internet address = 18.72.0.3
strawb.mit.edu  internet address = 18.71.0.151
w20ns.mit.edu   internet address = 18.70.0.160

C:\>nslookup www.aiit.or.kr bitsy.mit.edu
Server:  BITSY.MIT.EDU
Address:  18.72.0.3

Non-authoritative answer:
Name:    www.aiit.or.kr
Address:  218.36.94.200
```

Figure 10.1

Figure 10.1 shows the results of three independent *nslookup* commands (displayed in the Windows Command Prompt). In this example, the client host is located on the campus of Polytechnic University in Brooklyn, where the default local DNS server is dns-prime.poly.edu. When running *nslookup*, if no DNS server is specified, then *nslookup* sends the query to the default DNS server, which in this case is dns-prime. poly.edu.

Details of each command:

- **nslookup www.mit.edu**

In words, this command is saying "*Please send me the IP address for the host www.mit.edu.*" As shown in the screenshot, the response from this command provides two pieces of information: (1) the name and IP address of the DNS server that provides the answer; and (2) the answer itself, which is the host name and IP address of www.mit.edu. Although the response came from the local DNS server at Polytechnic University, it is quite possible that this local DNS server iteratively contacted several other DNS servers to get the answer, as described in Section 2.5 of the textbook.

- **nslookup –type=NS mit.edu**

In this example, we have provided the option "-type=NS" and the domain "mit.edu". This causes *nslookup* to send a query for a type-NS record to the default local DNS server. In words, the query is saying, *"Please send me the host names of the authoritative DNS for mit.edu."* (When the –type option is not used, *nslookup* uses the default, which is to query for type A records; see Section 2.5.3 in the text.) The answer, displayed in the above screenshot, first indicates the DNS server that is providing the answer (which is the default local DNS server) along with three MIT name servers. Each of these servers is indeed an authoritative DNS server for the hosts on the MIT campus. However, *nslookup* also indicates that the answer is "non-authoritative," meaning that this answer came from the cache of some server rather than from an authoritative MIT DNS server. Finally, the answer also includes the IP addresses of the authoritative DNS servers at MIT. (Even though the type-NS query generated by *nslookup* did not explicitly ask for the IP addresses, the local DNS server returned these "for free" and *nslookup* displays the result.)

- **nslookup www.aiit.or.kr bitsy.mit.edu**

In this example, we indicate that we want to the query sent to the DNS server bitsy.mit.edu rather than to the default DNS server (dns-prime.poly.edu). Thus, the query and reply transaction takes place directly between our querying host and bitsy.mit.edu. In this example, the DNS server bitsy.mit.edu provides the IP address of the host www.aiit.or.kr, which is a web server at the Advanced Institute of Information Technology (in Korea).

Do the following (and write down the results)

1. Run *nslookup* to obtain the IP address of a Web server in Asia, e.g. **www.lums.edu.pk**. Paste screenshot of your result here:

2. Run *nslookup* to determine the authoritative DNS servers for a university in Europe (e.g., **cam.ac.uk**)

3. Run *nslookup* so that one of the DNS servers obtained in Question 2 is queried for the mail servers for Yahoo! mail. If server does not respond, just type the command you used.

2. ipconfig

ipconfig (for Windows) and *ifconfig* (for Linux/Unix) are among the most useful little utilities in your host, especially for debugging network issues. Here we'll only describe *ipconfig*, although the Linux/Unix *ifconfig* is very similar. *ipconfig* can be used to show your current TCP/IP information, including your address, DNS server addresses, adapter type and so on. For example, if you want to see all this information about your host, simply enter:
`ipconfig \all` into the Command Prompt, as shown in figure 10.2.

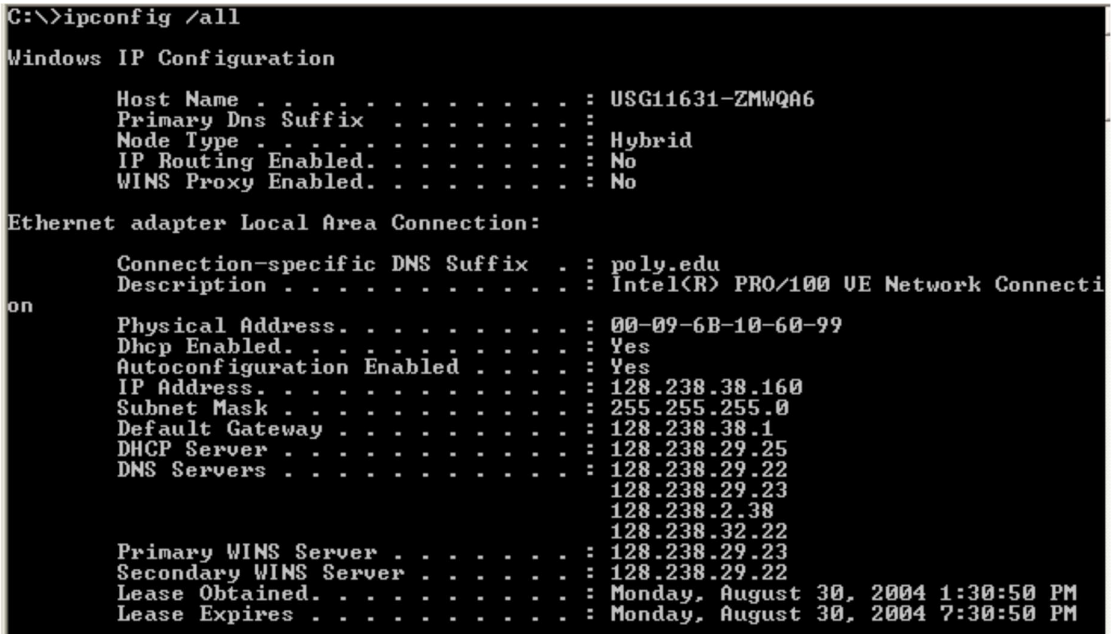

Figure 10.2

ipconfig is also very useful for managing the DNS information stored in your host. We learned that a host can cache DNS records it recently obtained.

To see these cached records, after the prompt C:\> provide the following command:

`ipconfig /displaydns`

Each entry shows the remaining Time to Live (TTL) in seconds.

To clear the cache, enter `ipconfig /flushdns`

Flushing the DNS cache clears all entries and reloads the entries from the hosts file.

3. Tracing DNS with Wireshark

Let's first capture the DNS packets that are generated by ordinary Web surfing activity.

• Use *ipconfig* to empty the DNS cache in your host.
• Open your browser and empty your browser cache. (With Internet Explorer, go to Tools menu and select Internet Options; then in the General tab select Delete Files.)
• Start packet capture in Wireshark.
• With your browser, visit the Web page: **http://www.ietf.org**
• Stop packet capture.

Answer the following questions:

4. Locate the DNS query and response messages. Are they sent over UDP or TCP?

5. What is the destination port for the DNS query message? What is the source port of DNS response message?

6. To what IP address is the DNS query message sent? Use *ipconfig* to determine the IP address of your local DNS server. Are these two IP addresses the same?

7. Examine the DNS query message. What "Type" of DNS query is it? Does the query message contain any "answers"?

8. Examine the DNS response message. How many "answers" are provided? What does each of these answers contain?

9. Consider the subsequent TCP SYN packet sent by your host. Does the destination IP address of the SYN packet correspond to any of the IP addresses provided in the DNS response message?

10. This web page contains images. Before retrieving each image, does your host issue new DNS queries?

Important Note

For questions 11 and onwards, if you are unable get results from nslookup commands, you may use the trace files given in a zipped folder at the following link. http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip

- **Using a trace file**

Download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract files. The traces in this zip file were collected by Wireshark running on one of the author's computers, while performing the steps indicated in the Wireshark lab. Once you have downloaded the trace, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting any trace file.

Now let's use *nslookup* and capture its packets.

• Start packet capture.
• Do an *nslookup* on www.mit.edu
• Stop packet capture.

(For Questions 11-15, dns-ethereal-trace-2 can be used only if you are unable get results from nslookup)

11. What is the destination port for the DNS query message? What is the source port of DNS response message?

12. To what IP address is the DNS query message sent? Is this the IP address of your default local DNS server?

13. Examine the DNS query message. What "Type" of DNS query is it? Does the query message contain any "answers"?

14. Examine the DNS response message. How many "answers" are provided? What does each of these answers contain?

15. Provide a screenshot.

Now repeat the previous experiment, but instead issue the command:
nslookup –type=NS mit.edu

(For Questions 16-19, dns-ethereal-trace-3 can be used only if you are unable get results from nslookup)

16. To what IP address is the DNS query message sent? Is this the IP address of your default local DNS server?

17. Examine the DNS query message. What "Type" of DNS query is it? Does the query message contain any "answers"?

18. Examine the DNS response message. What MIT name servers does the response message provide? Does this response message also provide the IP addresses of the MIT name servers?

19. Provide a screenshot.

Now repeat the previous experiment, but instead issue the command:
nslookup www.aiit.or.kr bitsy.mit.edu

(For Questions 20-23, dns-ethereal-trace-4 can be used only if you are unable get results from nslookup)

20. To what IP address is the DNS query message sent? Is this the IP address of your default local DNS server? If not, what does the IP address correspond to?

21. Examine the DNS query message. What "Type" of DNS query is it? Does the query message contain any "answers"?

22. Examine the DNS response message. How many "answers" are provided? What does each of these answers contain?

POST LAB QUESTIONS:

- Why does HTTP and DNS use TCP and UDP respectively?
- If the webpage having 10 reference objects and base html located at different location, then how may HTTP and DNS request will be generated?

EXPERIMENT 11

INTRODUCTION TO GRAPHICAL NETWORK SIMULATOR 3 (GNS3)

OBJECTIVE:

- Learn the usage of GNS-3
- Build a simple network topology

BACKGROUND

GNS3 is a Graphical Network Simulator that allows emulation of complex networks. GNS3 provides a graphical user interface to design and configure virtual networks, it runs on traditional PC hardware and may be used on multiple operating systems, including Windows, Linux, and Mac OS X. It allows simulation using Cisco Internetwork Operating Systems. It allows you to run a Cisco IOS in a virtual environment on computer. It is possible to simulate a large number of router platforms and PIX firewalls using GNS3. However GNS3 will provide around 1,000 packets per second throughput in a virtual environment. A normal router will provide a hundred to a thousand times greater throughput. GNS3 does not take the place of a real router, but is meant to be a tool for learning and testing in a lab environment. Using GNS3 in any other way would be considered improper.
Visit the following link in order to understand the main interface of the software and learn to build a network topology in GNS3:

http://www.csd.uoc.gr/~hy435/material/GNS3-0.5-tutorial.pdf

TASK 1:
Build the following class C network shown in figure 11.1 with its default subnet mask and ping PC5 from PC1.

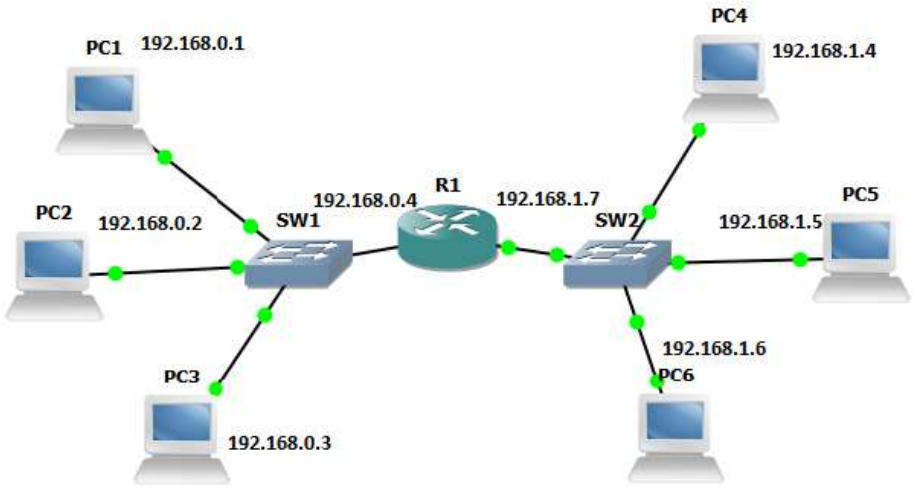

Figure 11.1

TASK 2:

Implement the following topology shown in the figure 11.2 with default subnet mask.

- Ping from HostA to HostB and Server
- Ping from HostB to HostA and Server
- Ping from server to HostA and HostB

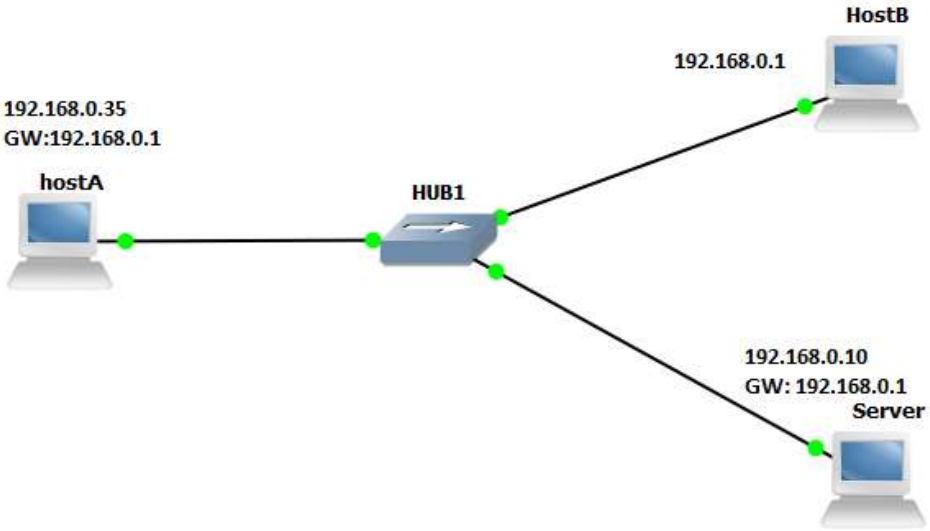

Figure 11.2

Now change the default subnet mask to 255.255.255.224 and perform the following:

- Ping from HostA to HostB and Server
- Ping from HostB to HostA and Server
- Ping from server to HostA and HostB

Q. What is the effect of changing the subnet mask on the network?

POST LAB QUESTIONS:

- Modify the Task 1 and use only the Class B for IP assignment and ping PC4, PC5 and PC6 from PC1, PC2 and PC3 respectively.

EXPERIMENT 12

BUILDING A NETWORK IN GNS3

OBJECTIVE:

- Implement the Dynamic Host Control Protocol in GNS-3
- Under the concept of subnet

You have to design a network solution for the Fast-NU Labs, Staff and Faculty members. Labs, Staff and faculty members should be on different Sub networks. There are total 50 computers divided in this way Staff 6, Faculty 4, and each lab contains 10(There are 4 Labs)

1. Choose appropriate Network class for assigning IPs to systems.
2. Use network devices appropriately and efficiently because they are costly and your design should be cost effective.
3. Use wires (straight through and cross over where necessary and applicable) – no wireless LAN is required for this submission.
4. You can assign IPs any of two ways
   a. Assign IPs to the machines using static IP allocation. Explain how you applied these IP's
   b. IPs can be assigned to the machines using DHCP servers using following commands.

```
enable    //to enter in privillage mode
config t  // configuration mode
ipdhcp pool mypool
network 192.168.9.0 255.255.255.0
default -router 192.168.9.1
exit
ipdhcp excluded-address 192.168.9.1
showipdhcp binding
```

5. Use ping from the command prompt of computers to check your network design is working

POST LAB QUESTIONS:

- What DORA stands for? Using a timeline diagram, show an example of a client getting its IP address from DHCP server.

EXPERIMENT 13

INTRODUCTION TO SOCKET PROGRAMMING

OBJECTIVE:

- Learn and Understand Communication using socket programming

BACKGROUND:

For C++:
Command: g++ *source_files*… -o *output_file*
For C:
Command: gcc *source_files… -o output_file*
Source files need not be cpp or c files. They can be preprocessed files, assembly files, or object files.

The whole compilation file works in the following way:
Cpp/c file(s) >> Preprocessed file(s) >>Assembly File(s) Generation>> Object file(s) Generation >> Final Executable

Every c/cpp file has its own preprocessed file, assembly file, and object file.

1. For running only the preprocessor, we use -E option.
2. For running the compilation process till assembly file generation, we use –S option.
3. For running the compilation process till object file creation, we use –c option.
4. If no option is specified, the whole compilation process till the generation of executable will run.

A file generated using any option can be used to create the final executable. For example, let's suppose that we have two source files: math.cpp and main.cpp, and we create object files:
g++ main.cpp –c –o main.o
g++ math.cpp –c –o math.o

The object files created using above two commands can be used to generate the final executable.
g++ main.o math.o –o my_executable
The file named "my_executable" is the final exe file. There is specific extension for executable files in Linux.

Socket:

A socket is defined as an endpoint for communication. A pair of processes communicating over a network employs a pair of sockets—one for each process. A socket is identified by an IP address concatenated with a port number. In general, sockets use a client–server architecture. The server waits for incoming client requests by listening to a specified port. Once a request is received, the server accepts a connection from the client socket to complete the connection. Servers implementing specific services (such as telnet, FTP, and HTTP) listen to well-known ports (a telnet server listens to port 23; an FTP server listens to port 21;

and a web, or HTTP, server listens to port 80). All ports below 1024 are considered well known; we can use them to implement standard services.

The communication over the network in TCP/IP model takes place in form of a client server architecture. ie, the client begins the communication and server follows up and a connection is established. Sockets can be used in many languages like Java, C++ , Python etc but here in this lab, we will understand the socket communication in its purest form (i.e in C programming language)

Functions used in Socket Programming:

socket()	Endpoint for communication
bind()	Assign a unique telephone number
listen()	Wait for a caller
connect()	Dial a number
accept()	Receive a call
send(), recv()	Talk
close()	Hang up

Server code:

```
#include <sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdio.h>
#include <unistd.h>
int main()
{
         int sd;
         char* msg="connected";
         char msg1[100];
         struct sockaddr_in my_addr, client_addr;
         my_addr.sin_family=AF_INET;
         my_addr.sin_port=htons(4999);
         my_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
         sd=socket(AF_INET,SOCK_STREAM,0);
         bind(sd,(struct sockaddr *) &my_addr,sizeof(struct
         sockaddr_in));
         listen(sd,5);
         int i=0;
while(i<2)
      {
            int size=sizeof(struct sockaddr);
            int new_sd=accept(sd,(struct sockaddr*)&client_addr,&size);
            send(new_sd, msg,100,0);
            recv(new_sd,msg1,100,0);
            printf("%s\n",msg1);
            ++i;
      }
      return 0;
}
```

Client Code

```
#include <sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdio.h>
#include <unistd.h>

int main()
{
      int sd;
      char msg[100];
      char* msg1="Also Connected";
      struct sockaddr_in server_addr;
      server_addr.sin_family=AF_INET;
      server_addr.sin_port=htons(4999);
      server_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
      sd=socket(AF_INET,SOCK_STREAM,0);
      connect(sd,(struct sockaddr*)
      &server_addr,sizeof(struct sockaddr));
      recv(sd,msg,100,0);
      send(sd, msg1,100,0);
      printf("%s\n",msg);
      return 0;
}
```

TASK#1

Run the code and then comment the whole code.

Procedure

1) Compile both client and server using GCC in Ubuntu terminal
2) First run the server
3) Open a new terminal window
4) Run the clients in the second terminal

TASK#2

Modify the server and client code to make an authentication method to establish the connection with client. The client will send a username and password if both are correct the further connection will continue else the connection will stop.

POST LAB QUESTIONS:

- The above code is making a TCP Connection convert it into UDP.

EXPERIMENT 14

TCP CLIENT/SERVER APPLICATION IN SOCKET PROGRAMMING

OBJECTIVE:

- Make a simple Client- Server Application using Socket API

Example TCP Server

The following is an example of a networked TCP server and associated client. The server opens a passive socket and listens for connection requests from clients. This is a very simple server: it accepts at most one connection request. Once a client connects, the server sends the message "Hello there!" to the client and shuts down.The client is not expected to send anything to the server. It simply connects, reads the server's "reply," and prints it out.

```
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <iostream>
#include <stdio.h> using namespace
std;
int main(int argc, char** argv)
 {
      int server_sock;
      int client_sock;
      int error;

  //create the server socket
  server_sock = socket(PF_INET, SOCK_STREAM, 0);

  // Need an address to bind the serversocket to
  struct sockaddr_in server_address;
  memset(&server_address, 0, sizeof(server_address));
  server_address.sin_addr.s_addr = net_addr("127.0.0.1");
  //htonl(INADDR_ANY);
  server_address.sin_family = PF_INET;
  server_address.sin_port = htons(2222);

  // bind the server socket to a local address
  error = bind(server_sock,(struct sockaddr *)&server_address,
  sizeof(server_address));

  if (error == -1)
  { perror("bind failed.");
   return 1;
  }
```

```
  // listen for a connection
  error = listen(server_sock, 10);
  if (error == -1)
  { perror("listen failed.");
   return 3; }
  cerr << "Now listening for connections..." << endl;

  // accept a connection
  struct sockaddr_in client_address;
  socklen_t client_addr_len;
   error = client_sock = accept(server_sock,(struct sockaddr *)
  &client_address, &client_addr_len);
  if (error == -1)
  { perror("accept failed.");
   return 0; }
  cerr << "Connection request received." << endl;
  write(client_sock, "hello there!", 12);
  close(client_sock);
  close(server_sock);
  return 0;
}
```

TCP Client

This is the client to go along with the server just shown.

```
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <iostream> #include
<stdio.h> using namespace std;

int main(int argc, char** argv)
{
        int client_sock;
         int error;
  //create the client socket
  client_sock = socket(PF_INET, SOCK_STREAM, 0);

  // Need the address of the server to connect to
  struct sockaddr_in server_address;
  memset(&server_address, 0, sizeof(server_address));
  server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
   server_address.sin_family = PF_INET;
   server_address.sin_port = htons(2222);

  error = connect(client_sock, (struct sockaddr *)&server_address,
  sizeof(server_address));

  if (error == -1)
  {
```

```
    perror("Failed to connect.");
    return 1;
   }
   cout << "Here is the message from the server:\n ";
   char ch;
   while (read(client_sock, &ch, 1) > 0)
   {
    write(1, &ch, 1);
   }

   close(client_sock);

   return 0;
}
```

This server and client use 127.0.0.1 for the IP address of the server, and both assume that the server is running at port 2222.

TASK

Modify the server and client above so that server computes and sends back to the client the square of any integer that the client sends. The protocol is binary, and is as follows:

1. Client connects and sends a single integer in binary form.
2. Server reads the integer and sends back a single integer, the square of the received integer, also in binary form.
3. The server closes the connection to the client.

The server will be a single-threaded server that serializes service to its clients. The server will stay in a loop in which it accepts a connection, reads the integer sent through the connection, and sends back the integer square through the same connection. After that, the server closes the connection and goes back to wait for another connection.

The client should ask the user for an integer, read the integer, open a connection to the server, and send the integer. Then it should read the reply and print it (the square) to the screen.

Test your server and client on your own machine by using two different terminals.

POST LAB QUESTIONS:

Modify the server and client of part 1 so that the client can ask the server to compute either a square or a cube. Use a binary protocol as follows. To request computation of the square of *x*, the client sends

<integer 6><string square><integer x>

That is, it sends an integer in binary form for the length of a string, then sends the characters of the string, and then sends the integer argument in binary form. Requesting computation of a cube is similar: the client sends

<integer 4><string cube><integer x>

These are the length of the string "cube", the string "cube" itself, and then the argument in binary form.

Appendix B: Lab Evaluation Criteria

Labs with projects

1. Experiments and their report 50%
    a. Experiment 60%
    b. Lab report 40%
2. Quizzes (3-4) 15%
3. Final evaluation 35%
    a. Project Implementation 60%
    b. Project report and quiz 40%

Labs without projects

1. Experiments and their report 50%
    a. Experiment 60%
    b. Lab report 40%
2. Quizzes (3-4) 20%
3. Final Evaluation 30%
        i. Experiment 60%
        ii. Lab report, pre and post experiment quiz 40%

Notice:

Copying and plagiarism of lab reports is a serious academic misconduct. First instance of copying may entail ZERO in that experiment. Second instance of copying may be reported to DC. This may result in awarding FAIL in the lab course.

Appendix C: Safety around Electricity

In all the Electrical Engineering (EE) labs, with an aim to prevent any unforeseen accidents during conduct of lab experiments, following preventive measures and safe practices shall be adopted:

- Remember that the voltage of the electricity and the available electrical current in EE labs has enough power to cause death/injury by electrocution. It is around 50V/10 mA that the "cannot let go" level is reached. "The key to survival is to decrease our exposure to energized circuits."
- If a person touches an energized bare wire or faulty equipment while grounded, electricity will instantly pass through the body to the ground, causing a harmful, potentially fatal, shock.
- Each circuit must be protected by a fuse or circuit breaker that will blow or "trip" when its safe carrying capacity is surpassed. If a fuse blows or circuit breaker trips repeatedly while in normal use (not overloaded), check for shorts and other faults in the line or devices. Do not resume use until the trouble is fixed.
- It is hazardous to overload electrical circuits by using extension cords and multi-plug outlets. Use extension cords only when necessary and make sure they are heavy enough for the job. Avoid creating an "octopus" by inserting several plugs into a multi-plug outlet connected to a single wall outlet. Extension cords should ONLY be used on a temporary basis in situations where fixed wiring is not feasible.
- Dimmed lights, reduced output from heaters and poor monitor pictures are all symptoms of an overloaded circuit. Keep the total load at any one time safely below maximum capacity.
- If wires are exposed, they may cause a shock to a person who comes into contact with them. Cords should not be hung on nails, run over or wrapped around objects, knotted or twisted. This may break the wire or insulation. Short circuits are usually caused by bare wires touching due to breakdown of insulation. Electrical tape or any other kind of tape is not adequate for insulation!
- Electrical cords should be examined visually before use for external defects such as: Fraying (worn out) and exposed wiring, loose parts, deformed or missing parts, damage to outer jacket or insulation, evidence of internal damage such as pinched or crushed outer jacket. If any defects are found the electric cords should be removed from service immediately.
- Pull the plug not the cord. Pulling the cord could break a wire, causing a short circuit.
- Plug your heavy current consuming or any other large appliances into an outlet that is not shared with other appliances. Do not tamper with fuses as this is a potential fire hazard. Do not overload circuits as this may cause the wires to heat and ignite insulation or other combustibles.
- Keep lab equipment properly cleaned and maintained.
- Ensure lamps are free from contact with flammable material. Always use lights bulbs with the recommended wattage for your lamp and equipment.
- Be aware of the odor of burning plastic or wire.
- ALWAYS follow the manufacturer recommendations when using or installing new lab equipment. Wiring installations should always be made by a licensed electrician or other qualified person. All electrical lab equipment should have the label of a testing laboratory.
- Be aware of missing ground prong and outlet cover, pinched wires, damaged casings on electrical outlets.
- Inform Lab engineer / Lab assistant of any failure of safety preventive measures and safe practices as soon you notice it. Be alert and proceed with caution at all times in the laboratory.
- Conduct yourself in a responsible manner at all times in the EE Labs.

- Follow all written and verbal instructions carefully. If you do not understand a direction or part of a procedure, ASK YOUR LAB ENGINEER / LAB ASSISTANT BEFORE PROCEEDING WITH THE ACTIVITY.
- Never work alone in the laboratory. No student may work in EE Labs without the presence of the Lab engineer / Lab assistant.
- Perform only those experiments authorized by your teacher. Carefully follow all instructions, both written and oral. Unauthorized experiments are not allowed.
- Be prepared for your work in the EE Labs. Read all procedures thoroughly before entering the laboratory. Never fool around in the laboratory. Horseplay, practical jokes, and pranks are dangerous and prohibited.
- Always work in a well-ventilated area.
- Observe good housekeeping practices. Work areas should be kept clean and tidy at all times.
- Experiments must be personally monitored at all times. Do not wander around the room, distract other students, startle other students or interfere with the laboratory experiments of others.
- Dress properly during a laboratory activity. Long hair, dangling jewelry, and loose or baggy clothing are a hazard in the laboratory. Long hair must be tied back, and dangling jewelry and baggy clothing must be secured. Shoes must completely cover the foot.
- Know the locations and operating procedures of all safety equipment including fire extinguisher. Know what to do if there is a fire during a lab period; "Turn off equipment, if possible and exit EE lab immediately."

Appendix D: Guidelines on Preparing Lab Reports

Each student will maintain a lab notebook for each lab course. He will write a report for each experiment he performs in his notebook. A format has been developed for writing these lab reports.

Lab Report Format

For hardware based labs, the format of the report will include:

1. **Introduction**: Introduce area explored in the experiment.
2. **Objective**: What are the learning goals of the experiment?
3. **Measurements**: In your own words write how the experiment is performed (Do not copy/paste the procedure).
    a. **Issues**: Which technical issues were faced during the performance of the experiment and how they were resolved?
    b. **Graphs**, if any
4. **Conclusions**: What conclusions can be drawn from the measurements?
5. **Applications**: Suggest a real world application where this experiment may apply.
6. Answers to post lab questions (if any).

Sample Lab Report:

Introduction

An RC circuit is a first order circuit that utilizes a capacitor as an energy storage element whereas a resistor as an energy wastage element. RC circuits are building blocks of electronic devices and their thorough understanding is important in comprehending advance engineering systems such as transistors and transmission lines.

An RC circuit can be operated with both DC and AC sources. In this lab we study transient response of RC circuits with a square wave as a DC source. During the DC operation of an RC circuit the voltage across the capacitor or the resistor show energy storing (capacitor charging) and dissipating (capacitor discharging via resistor) mechanisms of the circuit. The capacitor charging or discharging curves then lead to determine time constant of the circuit where the time constant signifies time required by the RC circuit to store or waste energy.

Objective:

To study transient response of a series RC circuit

Measurements:

The circuit used for the experiment is shown in Fig. 1.

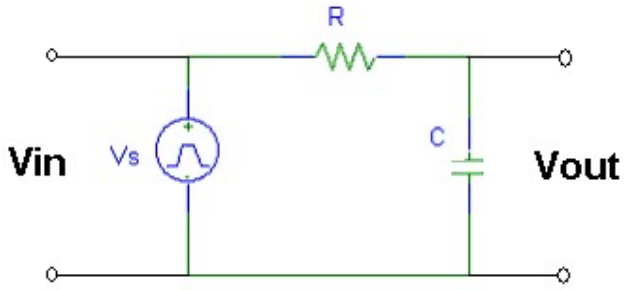

Fig.1. The circuit used in the experiment

Both input (a square wave) and output (voltage across capacitor) waveforms are monitored on an oscilloscope. The capacitor charging is observed during "on" part of the square waveform whereas the capacitor discharging is observed during "off" part of the square waveform (Fig. 2). We measure the time constant from the capacitor charging or discharging curve. While keeping the capacitor value constant, we also measure time constants with various resistor values (Table I).

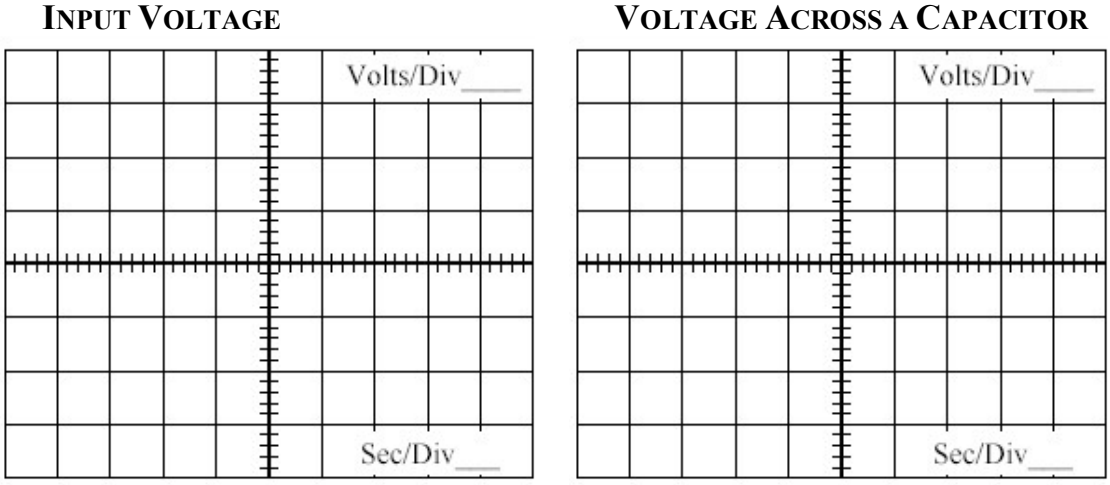

Fig. 2. Input and Output waveforms

TABLE I. Time constant as a function of the resistor values

Resistance (Nominal)	270 Ω	330 Ω	470 Ω	1 kΩ	2.2 kΩ	3.3 kΩ
Resistance (Measured)						
Time constant (Calculated)						
Time constant (Measured)						
Capacitance (Measured)						

Issues:

Mention any issue(s) you encountered during the experiment and how they were resolved

Conclusions:

From the measurements following conclusions can be drawn:

a) The capacitor charging and discharging curves are exponential.
b) The time constant is directly proportional to the resistor value.

Both of the above conclusions are also easily verifiable by solving differential equation for the RC circuit.

Applications:

An RC circuit can be employed for a camera flash. The capacitor discharges through the flash light during a picture taking event.