PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

Laboratory Manual

UE20EC304 Computer Communication Networks Lab

By

Prof M Rajasekar

Associate Professor

Department of Electronics and Communications Engineering

Aug- Dec 2022

Contents

Chapter 1

Lab 1: Introduction to Wireshark, GNS3 and Python

The experiments in this laboratory course have been designed to help the students better understand the concepts that they would be learning in the course UE19EC301 Computer Communication Networks through hands-on experience i.e. computer simulation. The experiments in this lab cover the important concepts and protocols related to the application layer, transport layer and network layer. Students are advised to explore the link layer functionalities using open ended experiments.

This lab course begins with an introduction to the concept of a protocol stack. The process of sending data from one host to another host across a computer network and the way how it is parsed into several actions in various layers of the protocol stack are explained. The role of IPv4 addresses and services provided by each layer is explained briefly.

Introduction to the various software used during the lab sessions is also given.

1.1 Overview

1.1.1 Concept of protocol stack

Application developers have produced a variety of messages (e.g., email, messages containing webpage, etc.), each having its own representation and associated process or program for interpreting the messages and executing the tasks based on contents of the message (e.g., a web browser is a end-user process which runs the HTTP protocol for exchanging messages with a web server which stores webpages). Sending messages from host to host is a complex process as there is a jungle of network components (e.g., routers and switches) which have to be traversed before the message reaches the destination. A host may be running multiple application processes with each communicating with a corresponding process running on a different host on the other side of the internet (sometimes separated by thousands of miles). Further, there are millions of users which are also sending messages to their respective destination hosts via the common network components. Hence, there is a need to execute the above complex process as several subtasks where each subtask has its own functions. These subtasks are referred to as layers which are named as application layer (one which generates and formats the message), transport layer (one which multiplexes and demultiplexes the messages leaving and arriving at the hosts respectively), network layer (one which identifies the path of network components leading to the destination host), link layer (one which pushes the data onto the physical medium connecting the host with the first network component on the path to the destination) and the physical layer (one which carries the data as modulated carrier signals). At the sending host, an application message along with its header is encapsulated into a transport layer segment. The transport layer segment along with its header is encapsulated into an IP datagram. The IP datagram along with its header is

encapsulated into a frame. The frame along with its header is converted into bits and the modulated carrier signal is transmitted at the physical layer. The reverse happens at the receiving host, where each layer starting with the link layer which removes its header after some pre-processing and passes the remaining payload to its upper layer. This process is repeated till the message is retrieved by the corresponding application layer process running at the receiving host.

1.1.2 Software tools

The various software tools used in this lab include **Wireshark, GNS3 and Python**.

Wireshark is the most popular packet sniffer and packet Analyser. Wireshark can run over any network adapter (real or virtual) and sniffs the packets that are sent or received across it. The contents of the packet can be extracted and analysed as to what information was added by each layer to aid the actions taken by network components and destination host who receive these packets.

GNS3 is a very popular network emulator which can be used to virtually design computer networks. GNS3 is used by several companies such as AT&T, CISCO, Intel, etc. The configuration of the network components can also be exported to a real network. GNS3 offers a wide range of configuration commands for the network layer and link layer. A virtual network design in GNS3 can be integrated with a real network and data can be exchanged with destination hosts on the public internet via GNS3 network. Wireshark can be incorporated into GNS3 to Analyse the packets exchanged between any pair of devices (e.g., routers and hosts).

Python is a powerful, simple and elegant programming tool. Various open source libraries are available online. Students will use Python 2.7 or above to write socket programs and demonstrate key application layer and transport layer concepts.

1.2 Procedures

1.2.1 Wireshark installation

Wireshark 1.12.4 can be downloaded from www.wireshark.org. Procedure for installation on Windows is simple. Double click the .exe file and follow the instructions. Remember to check ***WinPcap*** in the options for checkbox while installing; otherwise the network adapters will not be displayed upon installation. For Linux installation see the end of the chapter.

1.2.2 Familiarizing and running Wireshark

Upon running Wireshark the following window is displayed. The available network adapters can be observed in Fig. 1 under *Interface list*. The network adapter which is connected to the internet is chosen and the start button (displayed as green shark fin) is chosen. A web browser is opened and any URL (e.g. http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-

file1.html) is entered. Wireshark would start displaying various packets exchanged over the network adapter. Three sub-windows appear within the main window (top-down), namely,

(a) *packets listing window* which shows the various packets exchanged along with their labels, source and destination IP addresses (or host names), epoch time and size of the packet,

(b) *packet headers window* which shows the content of any packet selected in the packets listing window, such as protocol headers arranged according to the protocol stack and the application layer message (if any), and

(c) the *packet content window* which shows the raw data corresponding to the packet chosen. The various sub-windows can be observed from Fig. 2.

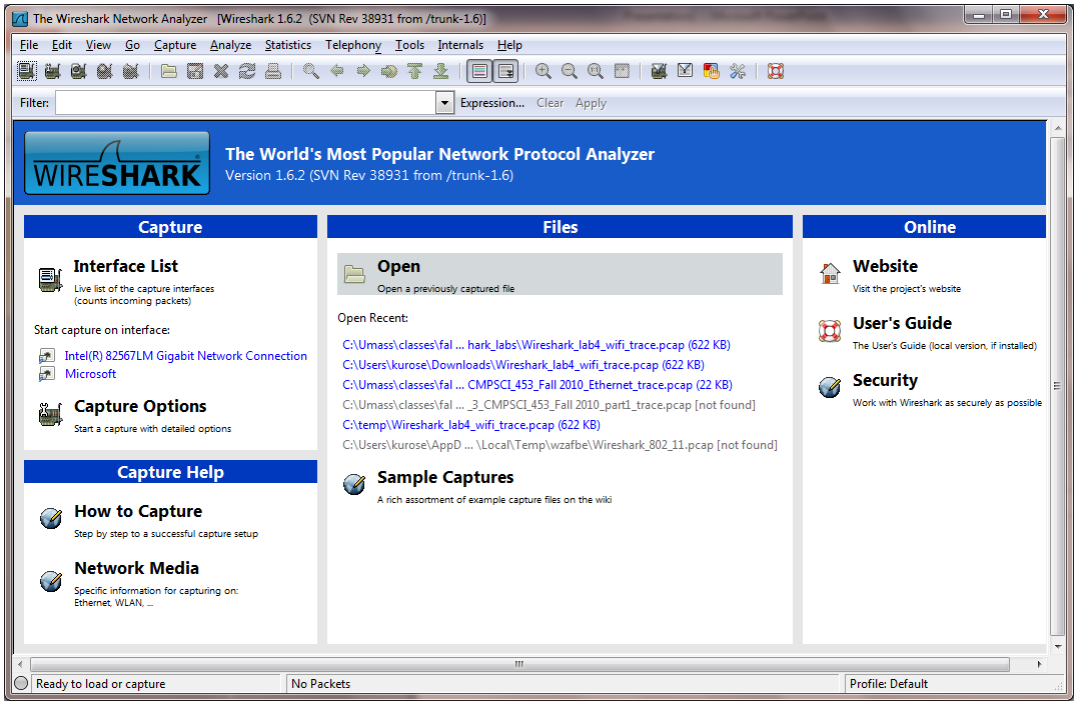


Fig. 1: Initial Wireshark Screen

Packets can also be chosen according to the protocol type by entering the packet type in the search bar. For example, enter *http* into the search bar and only packets containing HTTP messages are displayed in the packets listing window. Selecting the HTTP GET message will display the HTTP message along with the HTTP header. Besides, even the transport layer header, datagram header (i.e., network layer header), link layer header can be observed as in Fig. 3.

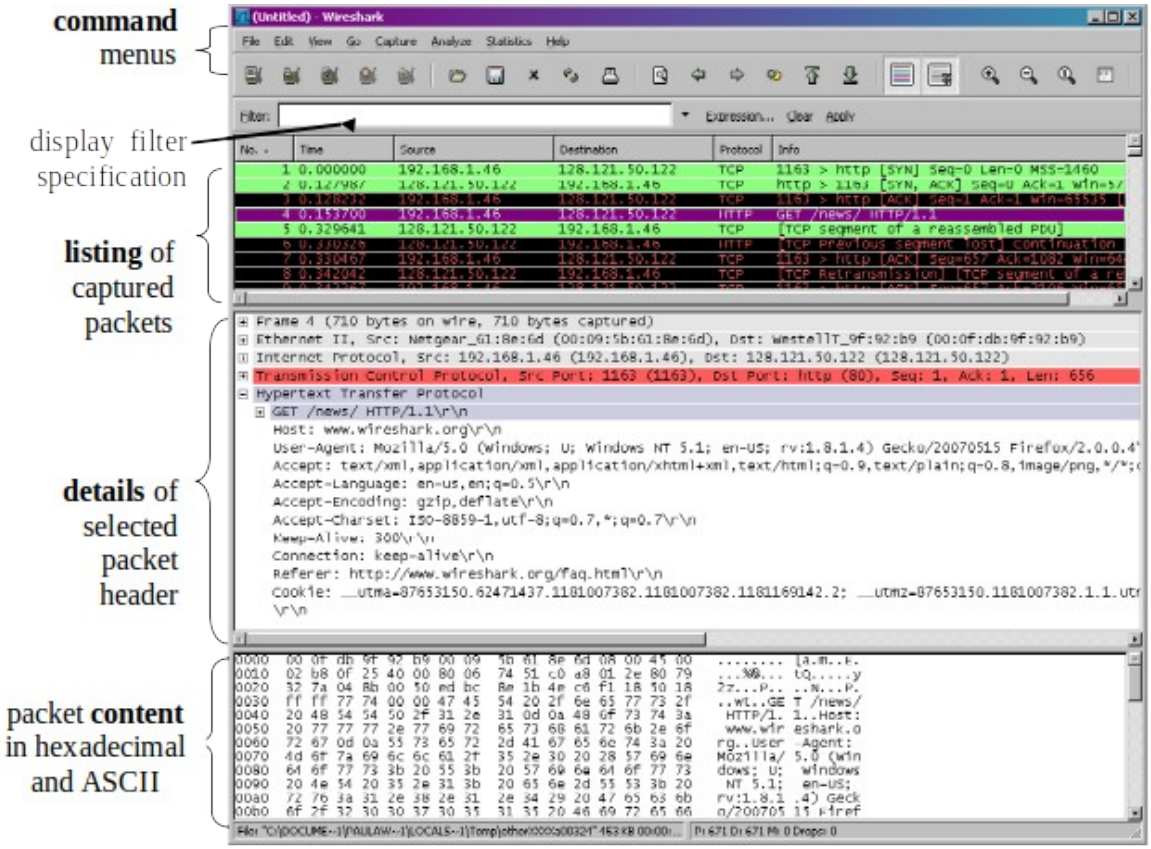


Fig. 2: Wireshark Graphical User Interface, during packet capture and analysis

Fig. 3: Wireshark window upon entering http in search bar

1.2.3 GNS3 installation

GNS3 1.1.3 is downloaded from www.gns3.com and installed by selecting the .exe file. Select GNS3, WinPCAP, Wireshark, Dynamips, VPCS during the installation (in the choose components window) and follow the instructions.

Download the IOS images for CISCO routers C7200 and C3725 from the internet by searching in Google **or** using the link https://mega.nz/folder/nJR3BTjJ#N5wZsncqDkdKyFQLELU1wQ or from the google drive link https://drive.google.com/drive/folders/18Aqx2n6XlNTc-jxp7ThigRCMc1ac4u5?usp=sharing.

Once either .image or .bin file is downloaded, install them by following the instructions as given in https://www.computernetworkingnotes.com/ccna-study-guide/how-to-add-install-or-import-ios-in-gns3.html **or** Start GNS3 and select the Edit from the Menu and then select Preferences. Choose IOS routers. Add a new router IOS by locating it in the PC.

After adding the router, click on the Ok button. For Linux installation see the end of the chapter.

1.2.4 Simple network configuration using GNS3

Draw the following simple network by dragging and dropping the components from the left pane as shown in Fig. 4. Right click on the router and select configure. Select slots, choose slot 2 and add NM-4T to add 4 serial ports to the router (will be used in later experiments).

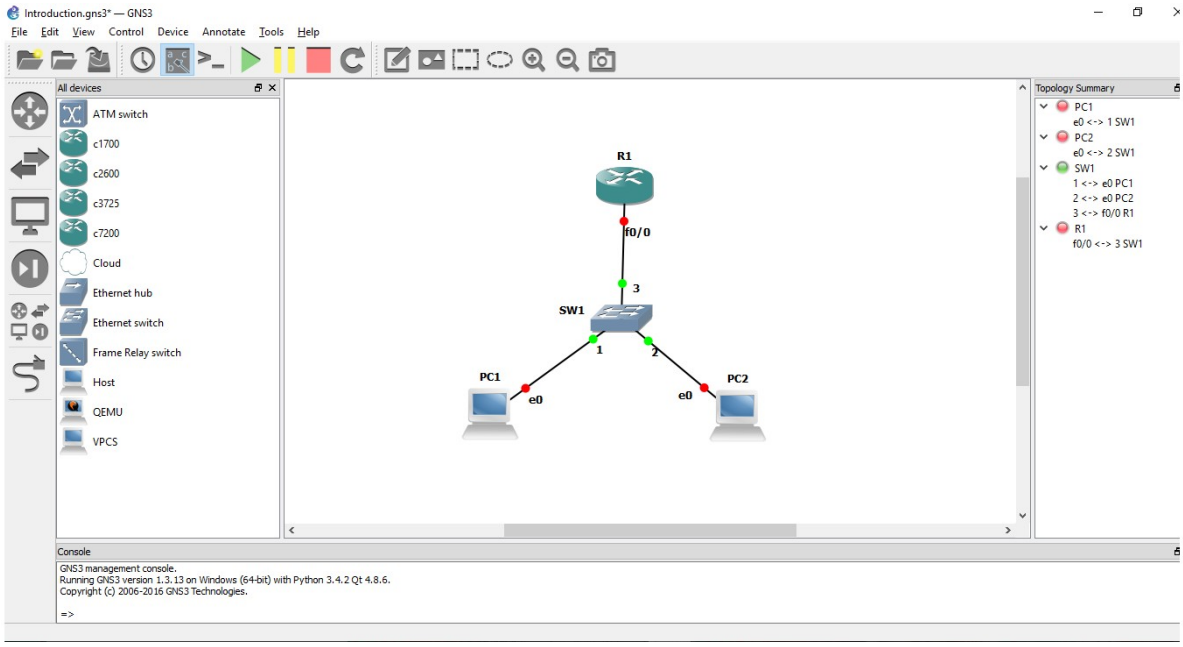


Fig. 4: Simple network in GNS3

Turn on all devices by clicking on the play button. Right click on the router and select Idle_PC, thereafter, choose any value with an asterisk to set the resources for the virtual router. Type the following commands for configuring the router (c3725 router).

R1#configure terminal
R1(config)#interface f0/1
R1(config-if)#ip address 10.0.0.1 255.255.255.0
R1(config-if)#no shutdown

R1(config-if)#exit

Type the command

PC1> ip 10.0.0.2/24 10.0.0.1

in PC1 by double clicking it. Similarly, type

PC1> ip 10.0.0.3/24 10.0.0.1

in PC2 by double clicking it. Next, type the following command in PC1 to display the ping statistics. Observe whether ping was successful. Ping is used to check if a destination device (clients/router) are available.

PC1> ping 10.0.0.3

Right click on any port and select Start capture. Repeat ping operation and observe the packets exchanged.

1.2.5 Python installation

Python installation can be done by downloading Python 2.7 or above from www.python.org. After installing python, open Python IDLE from which you can choose the editor for writing python programs with the .py.

Alternatively, you can install Sypder 3 or above from https://www.anaconda.com/download/

Linux installation

1. Open Ubuntu software center and install Spyder 3
2. To install GNS 3:
    a. Open terminal and type the following
    b. sudo add-apt-repository ppa:gns3/ppa
    c. sudo apt-get update
    d. sudo apt-get install dynamips gns3
3. To import router C3725 open https://www.sysnettechsolutions.com/en/gns3/gns3-supported-ios-images-download/. Download c3725-adventerprisek 9-mz.124-15.T14. After extracting the file, open GNS3 and add ios image by right clicking on the router images.
4. To enable Wireshark capture type the following in the terminal
    a. sudo dpkg-reconfiugure wireshark-common
    b. select *yes*
    c. sudo chmod 777 /usr/bin/dumpcap

Chapter 2

Lab 2: Analyse GET and Conditional GET under HTTP using Wireshark

2.1 Objective

To explore HTTP GET request/response by downloading simple HTML files which contain no embedded objects. We will also explore the conditional GET request/response and Analyse the differences between the GET and conditional GET messages.

2.2 Procedure

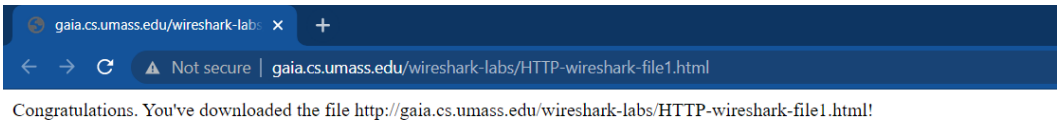
1. Start up your web browser.

2. Start up the Wireshark packet sniffer, as described in Lab 1 (do not begin packet capture). Enter "*http*" (just the letters, without quotes) in the display-filter-specification window, so that only captured HTTP messages will be displayed in the packet-listing window.

3. Wait for a while and then begin Wireshark packet capture.

4. Clear the cache in your browser (To do this under Firefox, select *Tools->Clear Recent History* and check the Cache box, or for Internet Explorer, select *Tools->Internet Options->Delete File;* these actions will remove cached files from your browser's cache.)

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



6. Stop Wireshark packet capture.

2.3 Analyses

The *packet-listing window* will show 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 OK 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 the boxes at the far left of the Frame, Ethernet, IP and TCP information have a plus sign or a right-pointing triangle (which means there is hidden, undisplayed information), and the HTTP line has a minus sign or a down-pointing triangle (which means that all information about the HTTP message is displayed).

(*Note:* You should ignore any HTTP GET and response for favicon.ico. If you see a reference to this file, it is your browser automatically asking the server if it (the server) has a small icon file that should be displayed next to the displayed URL in your browser. We'll ignore references to this pesky file in this lab.).

By looking at the information in the HTTP GET and response messages, answer 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?
3. What is the IP address of your computer? Of the gaia.cs.umass.edu server?
4. What is the status code returned from the server to your browser?
5. When was the HTML file that you were retrieving 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.

Next we Analyse the Conditional GET message and response. Repeat the procedure given in 2.2, but this time enter the 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.

(*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-2 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author's computers.)

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. 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?

11. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

Provide screenshots of the packet-contents window for each GET message and response. We are only interested in the HTTP protocol so make sure it is clearly visible in the screenshots.

Chapter 3

Lab 3: Analyse the downloading of embedded objects in a web-page using Wireshark

3.1Objective

Having seen how web pages with simple text-based HTML files were downloaded, we will see how a webpage having embedded objects (image files) are downloaded.

3.2 Procedure

1. Start up your web browser, and make sure your browser's cache is cleared, as discussed above. Start up the Wireshark packet sniffer.

2. Enter the following URL into your browser http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html

3. 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.aw-bc.com web site. The image of our book's cover is stored at the manic.cs.umass.edu server.

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

3.3 Analyses

1. How many HTTP GET request messages were sent by your browser (excluding the object favicon.ico)? To which Internet addresses were these GET requests sent?

2. Fill the following table:

Object	**Source IP Address**	**Destination IP Address**	**Source Port No**	**Destination Port No**	**Transport Layer Protocol**

3. How many hosts were responding to your browser? What are the host names?
4. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.
5. What is the total delay incurred in downloading the webpage along with the embedded objects? Provide a snapshot of the packet listing window with only http packets.
6. What type of TCP connection is used by HTTP for each object? How can you tell?
7. What was the throughput across the HTTP sessions?

Chapter 4

Lab 4: Analyse the DNS query and response using Wireshark

4.1 Objective

Your experiment will be conducted in four parts. First, you will query for the IP address of the given host name. Second, you will query for the canonical host name of the given host name (it may be the mnemonic host name). Third, you will query for the authoritative DNS servers' host name of the given host name. And finally, you will query for the mail server's canonical host name using the given host name.

4.2 Procedure

1. Windows: Open command prompt and type *ipconfig /all* to determine the local DNS IP address and your host IP address. Ubuntu: In terminal, type *nmcli dev show enp2s0*
2. Windows: To view the DNS records stored in your system, type *ipconfig /displaydns*. And to clear all DNS records, type *ipconfig /flushdns*
3. Open and start Wireshark. Issue commands to query DNS records from command prompt. Save the Wireshark files after the DNS response for packet analysis. Repeat this step for each of the four types of queries.
4. For querying IP address of *ieeexplore.ieee.org*, type *nslookup –type=A ieeexplore.ieee.org* in command prompt.
5. For querying canonical hostname of *ieeexplore.ieee.org*, type *nslookup –type=CNAME www.ieee.org* in command prompt.
6. For querying the names of authoritative DNS servers of *www.pes.edu*, type *nslookup –type=NS www.pes.edu* in command prompt.
7. For querying the mail server alias host names of *mail.google.com*, type *nslookup –type=MX mail.google.com* in command prompt.

4.3 Analyses

1. What was the transport protocol used for DNS queries? Give source and destination port numbers for each query in the experiment.
2. Under the Type A query, what is the length of the DNS query message? (Hint: use UDP length, UDP header size and DNS header size)

3. What is the IP address returned for the Type A query? What was the TTL value? How many answers were returned?
4. What is the canonical name returned for the Type CNAME query? What was the TTL value? What was the additional information observed in the response?
5. What is the authoritative DNS servers' name returned for the Type NS query? What was the TTL value? Was any additional records observed in the response?
6. What is the canonical host name returned for the Type MX query? What was the TTL value? What was the name of the authoritative DNS server observed in the response?
7. Given that the DNS query and response use the same header format, how can you tell the difference between a DNS query and response?
8. Did the client request for recursive query? How can you tell?
9. Did the DNS server support recursive query? How can you tell?
10. What is the throughput under Type NS combining the DNS query and its response?

Chapter 5

Lab 5: Analyse the TCP fragmentation when downloading large files from a web-server using Wireshark

5.1 Objective

When a large file is downloaded from a web server, it is usually sent over multiple TCP segments which contain parts of the application layer message (not necessarily the application header). We want to Analyse how the sending host broke the original payload into segments. We want to see how the TCP segments were reassembled at the client to display the whole HTML page.

5.2 Procedure

5 Start up your web browser, and make sure your browser's cache is cleared.

6 Start up the Wireshark packet sniffer

7 Enter the following URL into your browser
http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html.

8 Your browser should display the rather lengthy US Bill of Rights.

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

5.3 Analyses

Answer the following questions:

1. How many HTTP GET request messages were sent by your browser?
2. How many data-containing TCP segments were needed to carry the single HTTP response?
3. What is the status code and phrase associated with the response to the HTTP GET request?
4. Fill the following table:

Frame Type	Frame No	Source Port No	Destination Port No
SYN			
SYNACK			
ACK			

HTTP Request			
First segment of object			
Last segment of object			
FIN			
ACK for FIN			

5. What is the sequence number of the data containing TCP segments returned by the server? Indicate the size of each TCP segment. [Note: Actual segments can be viewed by right clicking on sequence number, selecting protocol preferences and un-ticking the relative sequence numbers]

6. Indicate the sequence number of the TCP segment containing the header lines of the HTTP response message. Provide a snapshot of the raw data and the TCP fields.

7. How much time did it take to fully download the requested object?

Chapter 6

Lab 6: Analyse TCP connections to a web-server using Wireshark

6.1 Objective

Having seen the HTTP GET request/response where objects were fetched from the web server, let us now upload an object (large text file) into the web server. Here we want to Analyse the HTTP POST method. Observe the following: Establishing TCP connections, TCP fragmentation and Closing of TCP connections.

6.2 Procedure

1. Start up your web browser. Go the http://gaia.cs.umass.edu/wiresharklabs/alice.txt and retrieve an ASCII copy of *Alice in Wonderland.* Store this file somewhere on your computer.
2. Next go to http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html.
3. 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.
4. Now start up Wireshark and begin packet capture *(Capture->Options)* and then press *OK* on the Wireshark Packet Capture Options screen (we'll not need to select any options here).
5. 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.
6. Stop Wireshark packet capture. Your Wireshark window should look similar to the window shown below.

6.3 Analyses

Filter the TCP segments by typing "tcp" in the display-filter-specification window. Answer the following questions by inspecting the captured packets.

Fill the following table:

Frame Type	Frame No	Source Port No	Destination Port No
SYN			
SYNACK			
ACK			

HTTP Request			
First segment of object			
Last segment of object			
FIN			
ACK for FIN			

1. What is the method in HTTP request message?
2. What is the content type and content length of HTTP response message?
3. What are the sequence numbers and acknowledgement numbers of the SYN segment, SYNACK segment and ACK segment which established the TCP connection to upload the file onto the web server? What port number was used by the client for the data transfer?

Frame Type	**Sequence Number**	**Acknowledgement No**
SYN		
SYNACK		
ACK		

4. Give the following details during file upload by the client: relative sequence numbers and segment lengths of the first three data carrying TCP segments sent to the server. Based on the packets exchanged between the client and the server, whose sequence number and acknowledgement number incremented?
5. What is the sequence number of the TCP segment in which the HTTP header was sent by the client during the file upload?
6. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. Calculate the Round Trip Time (RTT). Note that the RTT time is the time difference between the time of the POST message and the corresponding ACK.
7. How many TCP segments were required by the client to upload the object onto the web server? Provide a snapshot that displays the information.
8. Plot the Round Trip Time Graph. Note: Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the

“listing of captured packets” window that is being sent from the client to the gaia.cs.umass.edu server. Then select: Statistics→TCP Stream Graph→Round Trip Time Graph.

Chapter 7

Lab 7: Write and Analyse socket programs using Python

7.1 Objective

Demonstrate socket programming using Python.

- In part 1, show the exchange of text messages between a client and a server using UDP sockets.
- In part 2, show the exchange of text messages between a client and a server using TCP sockets.

7.2 Procedure

Students will Analyse the message exchange in Wireshark. Try sending long text messages like RFC text files. Server program should be running prior to executing the client program.

UDP client program

¿ *socket import* ∗ ¿

$serv_{addr} = 10.3.30.204$

$serv_{port} = 8000$

$client_{sock} = socket(AF_{INET}, SOCK_{DGRAM})$

$msg = input(\text{Enter the text message: })$

$client_{sock}.sendto(msg.encode(), (serv_{addr}, serv_{port}))$

$mod_{msg}, s = client_{sock}.recvfrom(2048)$

$print(\text{From Server: }, mod_{msg}.decode())$

UDP server program

¿ *socket import* ∗ ¿

$serv_{addr} = 10.3.30.204$

$serv_{port} = 8000$

$serv_{sock} = socket(AF_{INET}, SOCK_{DGRAM})$

$serv_{sock}.bind((serv_{addr}, serv_{port}))$

$print((\text{The server is ready to receive}))$

while 1 :

$msg, client_{addr} = serv_{sock}.recvfrom(2048)$

$print((\text{Got message from }, client_{addr}))$

$mod_{msg} = msg.upper()$

$serv_{sock}.sendto(mod_{msg}, client_{addr})$

TCP client program

¿ *socket import* ∗ ¿

$serv_{addr} = 10.3.30.204$

$serv_{port} = 8000$
$client_{sock} = socket(AF_{INET}, SOCK_{STREAM})$
$client_{sock}.connect((serv_{addr}, serv_{port}))$
$msg = input(\text{Enter the text message: })$
$client_{sock}.send(msg.encode())$
$mod_{msg} = client_{sock}.revc(2048)$
$print(\text{From Server: }, mod_{msg}.decode())$
$client_{sock}.close()$

TCP server program

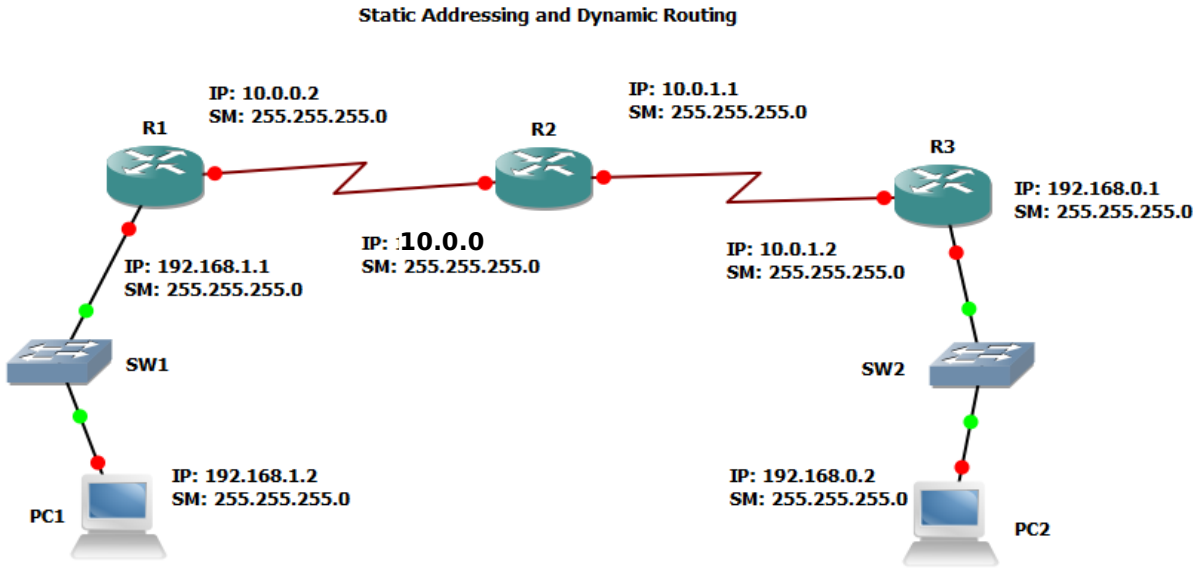
¿ socket import * ¿
$serv_{addr} = 10.3.30.204$
$serv_{port} = 8000$
$serv_{sock} = socket(AF_{INET}, SOCK_{STREAM})$
$serv_{sock}.bind((serv_{addr}, serv_{port}))$
$serv_{sock}.listen(1)$
$print((\text{The server is ready to receive}))$
$while\ 1:$
$conn_{sock}, client_{addr} = serv_{sock}.accept()$
$print((\text{Got connection from }, client_{addr}))$
$msg = conn_{sock}.revc(2048)$
$mod_{msg} = msg.upper()$
$conn_{sock}.send(mod_{msg})$
$conn_{sock}.close()$

Chapter 8

Lab 8: Static addressing and dynamic routing

8.1 Objective

Design a simple network to demonstrate static addressing and dynamic routing using GNS3



8.2 Procedure

1. Configure the router interfaces as shown in the Section 1.2.4. For serial interface choose the labels as s1/0, s1/1 and so on.

2. For dynamic routing write the sample code based on the subnets directly connected to a router as follows. Example for R2

   R2(config)# router rip
   R2(config-router)# version 2
   R2(config-router)# network 10.0.0.0
   R2(config-router)# network 10.0.1.0
   R2(config-router)# end

3. For PC assign IP address as shown in Section 1.2.4.

8.3 Analyses

1. Show the IP addresses of the active interfaces of R1, R2 and R3

2. Show the routing tables in R1, R2 and R3.

3. Show the ping operation by pinging PC2 from PC1. Show packet capture and write port numbers, IP addresses of each Echo request and reply. Explain ping statistics.

4. Show ping operation over TCP by pinging PC2 from PC1. Show packet capture and write port numbers, IP addresses of each Echo request and reply.

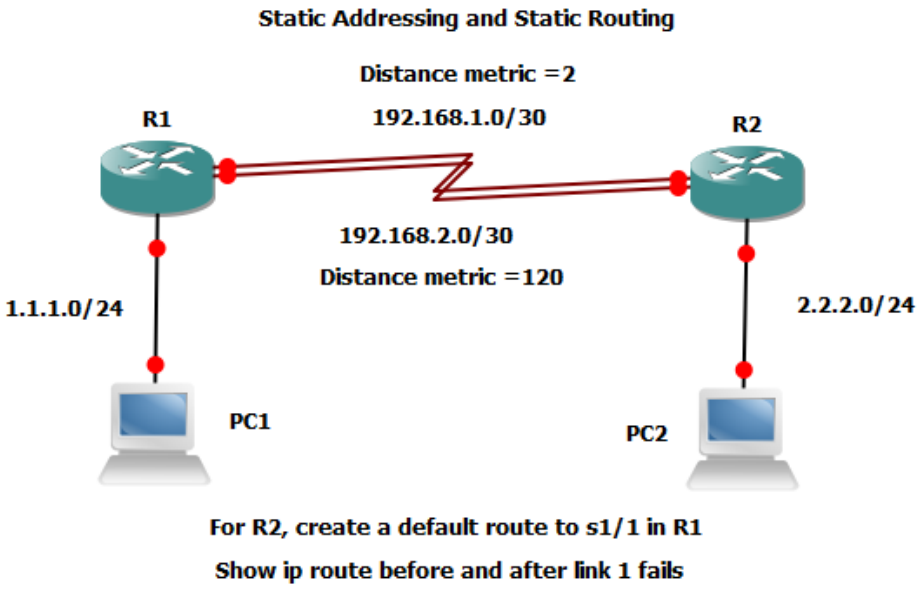
5. For above question, what is the sequence number of each SYN segment?

Chapter 9

Lab 9: Static addressing and static routing

9.1 Objective

Design a simple network to demonstrate static addressing and static routing using GNS3. Assign two static routes from R1 to R2 such that one of them has a higher link cost (say 120). Assign a default route from R2 to R1. Show the effect of link failure on routing (delete low cost link)



9.2 Procedure

1. Configure the router interfaces as shown in the Section 1.2.4. For serial interface choose the labels as s1/0, s1/1 and so on.

2. For default route write R2(config)# ip route 0.0.0.0 0.0.0.0 192.168.1.1 permanent, where 192.168.1.1 is assigned to s1/1 of R1.

3. For static routes from R1, use the command given in step 2 however, specify the actual network address, subnet mask and interface address for forwarding via each link. Assign link costs instead of *permanent*.

4. For PC assign IP address as shown in Section 1.2.4.

9.3 Analyses

1. Show the IP addresses of the active interfaces of R1 and R2

2. Show the routing tables in R1 and R2 before link 1 fails.

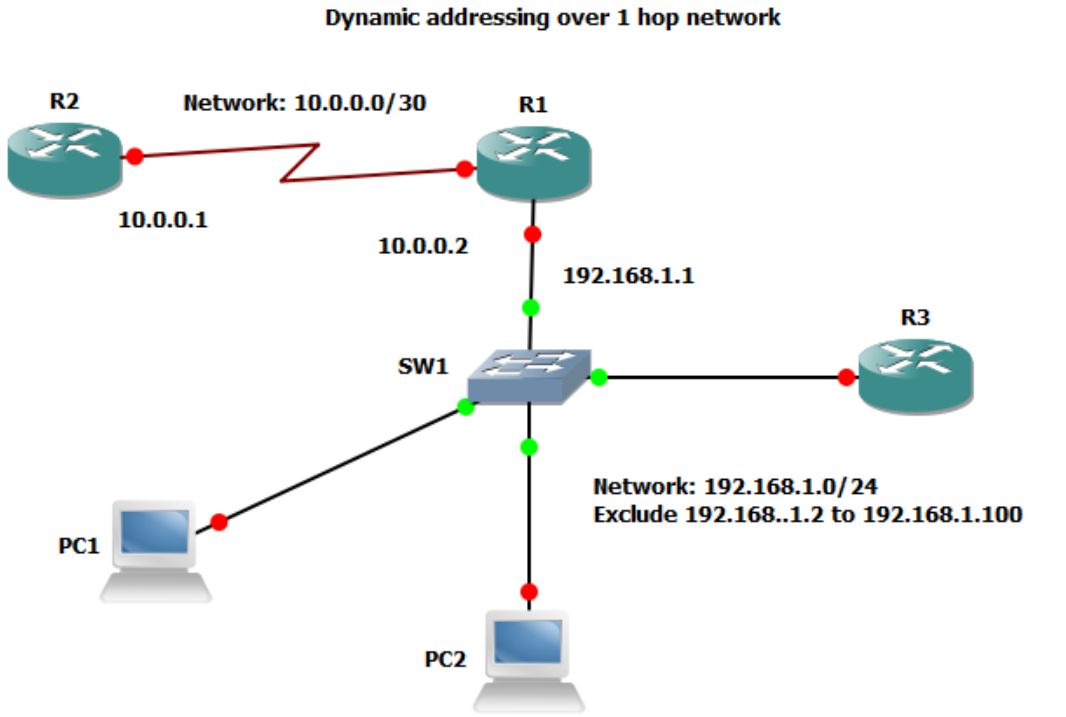
3. Show the routing tables in R1 and R2 after link 1 fails.
4. Show the ping operation by pinging PC2 from PC1. Show packet capture and write port numbers, IP addresses of each Echo request and reply. Explain ping statistics.

Chapter 10

Lab 10: Dynamic addressing and dynamic routing

10.1 Objective

Design a simple 1-hop network to demonstrate dynamic addressing and dynamic routing using GNS3. Configure a DHCP server in R1 with address pool 192.168.1.0 /24 and configure R3 as a DHCP client. Assign addresses using DHCP to the interfaces in PC1, PC2 and R3.



10.2 Procedure

1. Configure the router interfaces of R1 and R2 as shown in the Section 1.2.4.

2. For dynamic routing configure as shown in Section 8.2.

3. For configuring DHCP server in PC1:

```
R1(config)# ip dhcp pool POOL_A
R1(dhcp-config)# ip dhcp excluded-address 192.168.1.5 192.168.1.100
R1(config)# ip dhcp pool POOL_A
R1(dhcp-config)# dns-server 192.168.1.1
R1(dhcp-config)# default-router 192.168.1.1
R1(dhcp-config)# lease 3
R1(dhcp-config)# network 192.168.1.0 /24
```

R1(dhcp-config)# import all

4. To configure R3 as a DHCP client and obtain IP address for f0/0

```
R3(config)# interface f0/0
R3(config-if)#ip dhcp client client-id ascii My_name
R3(config-if)#ip address dhcp
```

5. For PC assign IP address using DHCP
   PC1> ip dhcp -d

10.3 Analyses

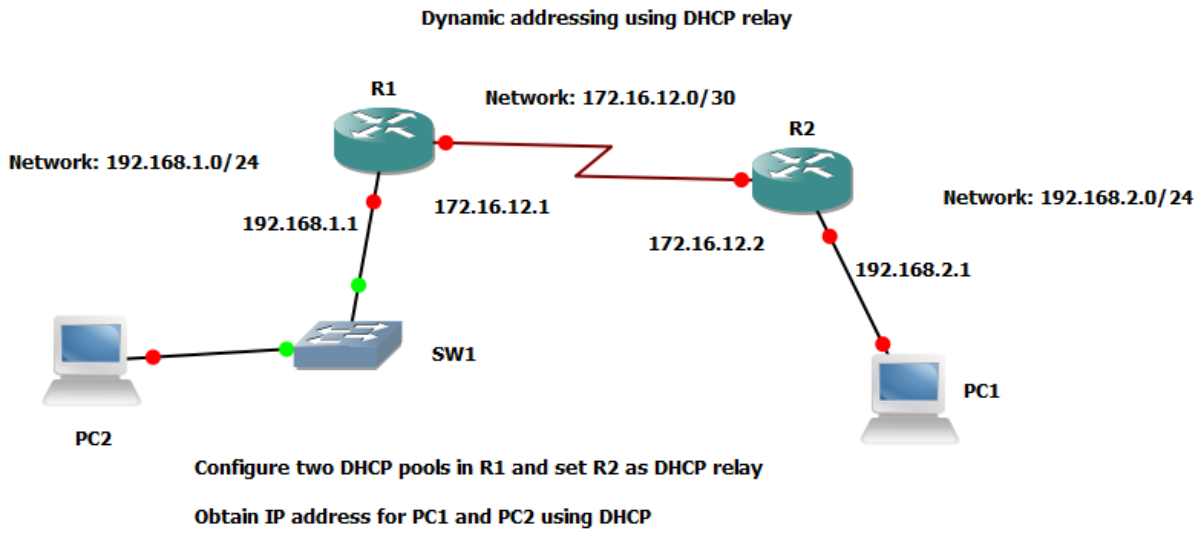
1. Show the IP addresses assigned via DHCP
2. Show the routing tables in R1 and R2.
3. Show DHCP server statistics.
4. Show DHCP server's pool information.
5. Analyse the packets exchanged between PC1 and the DHCP server when obtaining IP address. Write port numbers, IP address and MAC address for each packet observed.
6. Show the ping operation by pinging PC2 from PC1. Show packet capture and write port numbers, IP addresses of each Echo request and reply. Explain ping statistics.

Chapter 11

Lab 11: Dynamic addressing using DHCP relay

11.1 Objective

Design a 2-hop network to demonstrate dynamic addressing and dynamic routing using GNS3. Configure a DHCP server in R1 with two pools 192.168.1.0 /24 and 192.168.0.0 /24. Exclude the addresses 192.168.1.1-192.168.1.50 and 192.168.0.1-192.168.0.100 from the address pools. Configure R2 as a DHCP relay. Assign addresses using DHCP to the interfaces in PC1 and PC2.



11.2 Procedure

1. Configure the router interfaces of R1 and R2 as shown in the Section 1.2.4.

2. For dynamic routing configure as shown in Section 8.2.

3. For configuring DHCP server in R1 follow the steps in Section 10.2.

4. Configuring R2 as DHCP relay

   R2(config)# interface f1/0
   R2(config-if)# ip helper-address 172.16.12.1

5. For PC assign IP address using DHCP
   PC1> ip dhcp -d

11.3 Analyses

1. Show the IP addresses assigned via DHCP

2. Show the routing tables in R1 and R2.

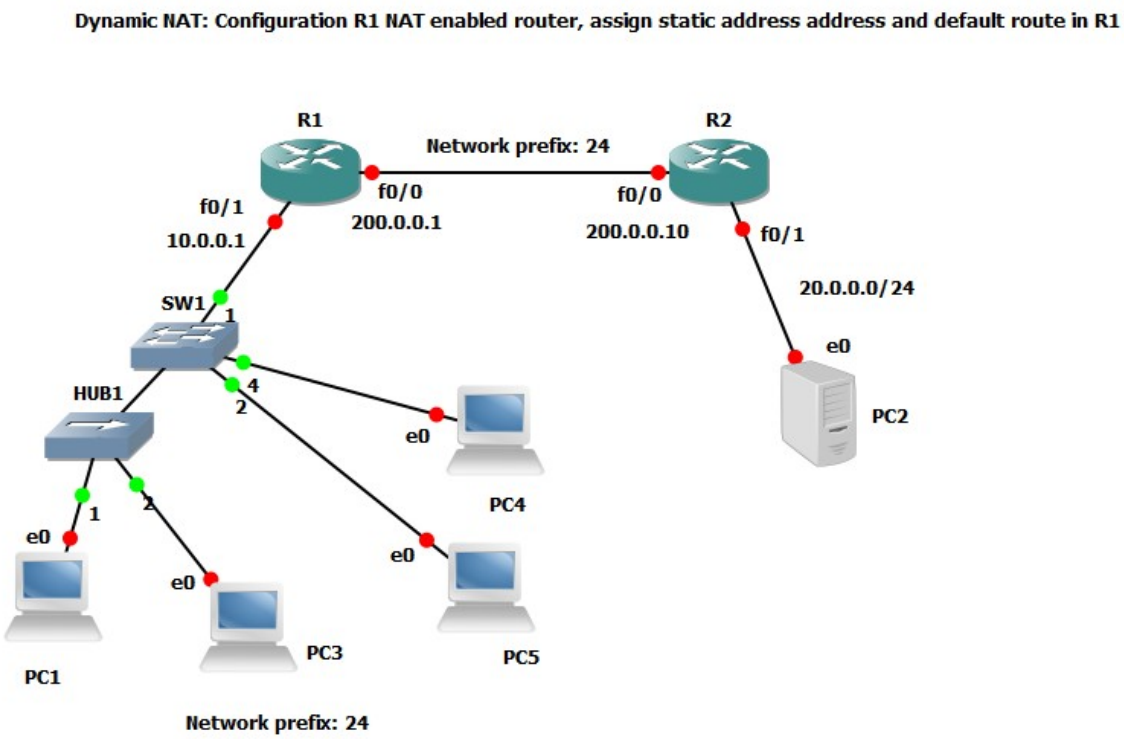
3. Show DHCP server statistics.
4. Show DHCP server’s pool information.
5. Analyse the packets exchanged between PC1 and the DHCP server when obtaining IP address. Write port numbers, IP address and MAC address for each packet observed.
6. Show the ping operation by pinging PC2 from PC1. Show packet capture and write port numbers, IP addresses of each Echo request and reply. Explain ping statistics.

Chapter 12

Lab 12: Dynamic NAT configuration using one NAT enabled router

12.1 Objective

Design a NATed network to demonstrate using GNS3 and explain packet capture. Add a server (PC icon changed) to router R2 on network 20.0.0.0/24. Connect network 10.0.0.0/24 as a private network to NAT enabled router R1. Ping the server from any PC in the private network. Let public address pool include 200.0.0.2 to 200.0.0.5.



12.2 Procedure

1. Configure the router interfaces of R1 and R2 as shown in the Section 1.2.4. While declaring interface f0/1 in R1, configure it as the interface for the private network. While declaring interface f0/0 in R1, configure it as the interface for the public network.

2. For dynamic routing configure as shown in Section 8.2. For static routing follow Section 9.2.

3. For configuring NAT in R1 follow the steps below.

   R1(config)#ip nat pool my_pub_ips 200.0.0.2 200.0.0.5 netmask 255.255.255.0
   R1(config)#access-list 1 permit 10.0.0.0 0.0.0.255

```
R1(config)#ip nat inside source list 1 pool my_pub_ips
R1(config)#ip nat log translations syslog
```

4. For PC assign IP address as given in Section 1.2.4

12.3 Analyses

1. Show the NAT translations
2. Show routing tables in R1 and R2 (you can use static or dynamic routing).
3. Show the ping operation by pinging PC2 from PC1. Show packet capture and write port numbers, IP addresses of each Echo request and reply. Explain ping statistics.
4. Do the opposite of step 3 and explain what happened.