



MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGG.

CERTIFICATE

This is to certify that Ms./Mr.

Reg. No.: Section:

Roll No.: has satisfactorily completed the lab exercises prescribed for Computer Networks Lab [CSE 3113] of Third Year B.Tech. Degree in Computer Science and Engg. at MIT, Manipal, in the academic year 2017-2018.

Date:

Signature
Faculty in Charge

CONTENTS

| LAB NO. | TITLE | PAGE NO. | REMARKS |
|---------|--|----------|---------|
| | Course Objectives and Outcomes | v | |
| | Evaluation plan | v | |
| | Instructions to the Students | vi | |
| 1. | Basic Computer Networking related commands and configurations | 1-5 | |
| 2. | Socket Programming in ‘C’ using TCP – Iterative Client – Server Programs | 6-17 | |
| 3. | Socket Programming in ‘C’ using TCP – Concurrent Servers and I/O Multiplexing | 18-26 | |
| 4. | Network Data Analysis using tcpdump and Introduction to NS2 Network Simulator | 27-31 | |
| 5. | Socket Programming in ‘C’ using UDP and Network Monitoring and Analysis with Wireshark | 32-40 | |
| 6. | Computer Network Design using HUB in GNS3 | 41-59 | |
| 7. | Computer Network Design using SWITCH and ROUTERS in GNS3 | 60-69 | |
| 8. | Study of Domain Name Server | 70-73 | |
| 9. | Study of DHCP Server | 74-77 | |
| 10. | Introduction To NS2: Wired Network | 78-96 | |
| 11. | Measuring Performance of Protocols with NS2 | 97-99 | |
| 12. | Design of VLANs Using GNS3 | 100-105 | |
| 13. | References | 106 | |
| 14. | Appendix | 107 | |

Course Objectives

- To understand network using GNS3 Simulation and NS2.
- To develop skills in network monitoring and analysis using various tools.
- To understand development of network applications.
- To design and deploy computer networks

Course Outcomes

At the end of this course, students will be able to

- Learn to use Network Related commands and configuration files in Linux Operating System.
- Learn to Develop Network Application Programs.
- Analyze Network Traffic using network Monitoring Tools.

Evaluation plan

- Internal Assessment Marks : 60%
 - ✓ Continuous evaluation component (for each evaluation): 5 marks.
 - ✓ The assessment will depend on punctuality, program execution, maintaining the observation note and answering the questions in viva voce.
 - ✓ Total marks of the 12 evaluations is marks out of 60.
- End semester assessment of 2 hour duration: 40%.

INSTRUCTIONS TO THE STUDENTS

Pre-Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session.
2. Be in time and follow the institution dress code.
3. Must Sign in the log register provided.
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum.

In-Lab Session Instructions

- Follow the instructions on the allotted exercises.
- Show the program and results to the instructors on completion of experiments.
- On receiving approval from the instructor, copy the program and results in the Lab record
- Prescribed textbooks and class notes can be kept ready for reference if required.

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- Observation book should be complete with proper design, logical diagrams, truth tables and waveforms related to the experiment they perform.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- The exercises for each week are divided under three sets:
 - Solved exercise
 - Lab exercises – to be completed during lab hours
 - Additional Exercises – to be completed outside the lab or in the lab to enhance the skill
- In case a student misses a lab class, he/ she must ensure that the experiment is completed during the repetition class with the permission of the faculty concerned but credit will be given only to one day's experiment(s).
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and/or combinations of the questions.
- A sample note preparation is given as a model for observation.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

Basic Computer Networking Related Commands and Configurations

Objectives:

In this lab, student will be able to

1. Connect the computers in Local Area Network and configure related settings
2. Study and practice Networking related commands and configurations

I. Unix Utilities to test Internet connection and to diagnose congestion between computers:

ifconfig: The **ifconfig** command is used to configure a network interface. The following options are used for the reconfiguration of the IP address and network mask.

ifconfig -a : Shows the states of all interfaces in the system.

ifconfig *<interface name>* *down* : Disables the network interface, where interface name is the name of the Ethernet interface.

ifconfig *<interface name>* *<new IP address>* *up* : Assigns a new IP address to the interface and brings it up.

ifconfig *<interface name>* *netmask* *<new netmask>* : Assigns a new network mask for the interface.

```
projectlab@PL-01: ~  
projectlab@PL-01:~$ ifconfig  
enp2s0    Link encap:Ethernet  HWaddr 6c:f0:49:90:4d:fb  
          inet addr:172.16.59.2  Bcast:172.16.59.255  Mask:255.255.255.0  
          inet6 addr: fe80::1688:82c6:397e:1f33/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:2457 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:1471 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:3143221 (3.1 MB)  TX bytes:93825 (93.8 KB)  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING  MTU:65536  Metric:1  
          RX packets:5720 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:5720 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1  
          RX bytes:493352 (493.3 KB)  TX bytes:493352 (493.3 KB)  
  
projectlab@PL-01:~$
```

Figure 1.1: ifconfig command

Ping: Ping (also written as PING or ping) is a utility that you use to determine whether or not a specific IP address is accessible. Ping works by sending a packet to a specified address and waiting for a reply. Ping is used primarily to troubleshoot Internet connections and there are many freeware and shareware Ping utilities available for download.

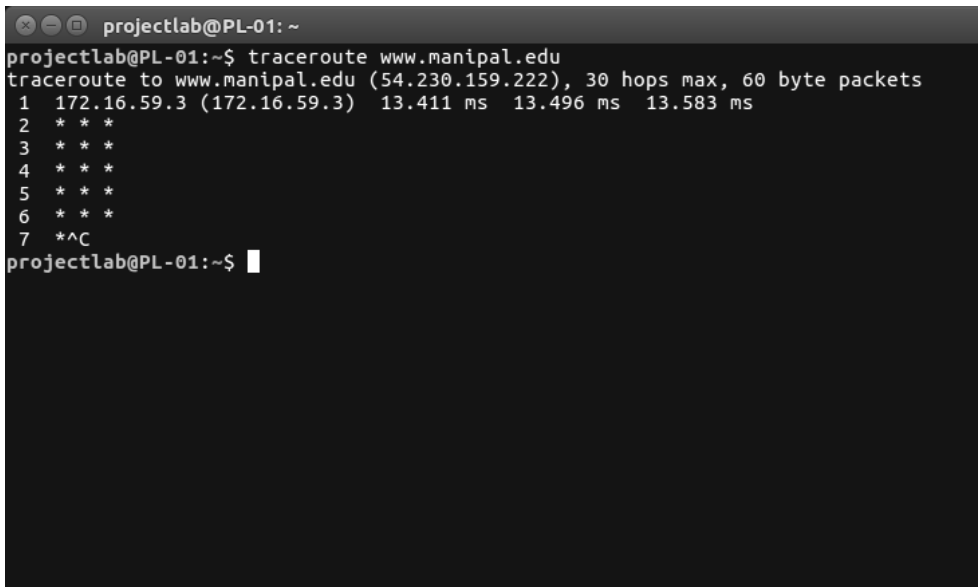
```
projectlab@PL-01: ~  
projectlab@PL-01:~$ ping www.manipal.edu  
PING d315btbdcz2b03.cloudfront.net (54.230.190.48) 56(84) bytes of data.  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=1 ttl=248 time=301 ms  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=3 ttl=248 time=293 ms  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=4 ttl=248 time=297 ms  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=5 ttl=248 time=295 ms  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=6 ttl=248 time=294 ms  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=7 ttl=248 time=299 ms  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=8 ttl=248 time=297 ms  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=9 ttl=248 time=295 ms  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=10 ttl=248 time=296 ms  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=12 ttl=248 time=297 ms  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=14 ttl=248 time=295 ms  
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=15 ttl=248 time=296 ms  
^C  
--- d315btbdcz2b03.cloudfront.net ping statistics ---  
15 packets transmitted, 12 received, 20% packet loss, time 14023ms  
rtt min/avg/max/mdev = 293.914/296.801/301.028/1.978 ms
```

Figure 1.2: ping command

Traceroute: Traceroute is a utility that traces a packet from your computer to an Internet host, but it will show you how many hops the packet requires to reach the host and how long each hop takes. If you're visiting a Web site and pages are appearing slowly, you can use traceroute to figure out where the longest delays are occurring. Traceroute utilities work by sending packets with low time-to-live (TTL) fields. The TTL value specifies how many hops the packet is allowed before it is returned. When a packet can't reach its destination because the TTL value is too low, the last host returns the packet and identifies itself. By sending a series of packets and incrementing the TTL value with each successive packet, traceroute finds out who all the intermediary hosts are.

```
projectlab@PL-01:~$ sudo apt install traceroute
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure 1.3: Install traceroute command



```
projectlab@PL-01:~$ traceroute www.manipal.edu
traceroute to www.manipal.edu (54.230.159.222), 30 hops max, 60 byte packets
 1  172.16.59.3 (172.16.59.3)  13.411 ms  13.496 ms  13.583 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  **^C
projectlab@PL-01:~$
```

Figure 1.4: traceroute command

II. SOLVED EXERCISE

Connect the computers in local area network

1. Right click on the network manager applet,
 - Go to **Edit connections →wired tab→add**
2. Put the mac address of the interface you will be configuring. The ifconfig command can show you what the mac address is:

\$ ifconfig

```
eth0    Link encap:Ethernet HWaddr 00:30:1b:b9:53:94
HWaddr 00:30:1b:b9:53:94 = mac address
```

3. Then click the ipv4 settings tab. set method to manual.
4. click add to add IP address

```
# example for computer one would be
address | netmask      | gateway
10.0.0.1 | 255.255.255.0 |
# example for computer two would be
10.0.0.2 | 255.255.255.0 |
```

5. See if you can ping each other from computer one.

\$ ping 10.0.0.2

```
ping 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=128 time=0.457 ms
```

I. LAB EXERCISES

Note: Use **man** command to explore various options

1. What is the IP of the machine you are using? Compare it with the IP of your neighbors.
Are the IPs of your neighbors same? Why or Why not?
2. Use the ping command for the following URLs and record the success or failure statistics along with the average round trip time.
 - a) google.com
 - b) facebook.com
3. Based on output of **ifconfig -a** command, identify the following
 - Host Id
 - MAC address of your system[physical address]
 - Subnet mask

4. In the LAN, compare your result of Q3, with your neighbor computers. What similarities do you see in the MAC address?
5. With the help **man** pages, comment on the results of **ping** with different options.
6. Ping the computer's loopback IP address. Type the following command:

ping 127.0.0.1

The address 127.0.0.1 is reserved for loopback testing. If the ping is successful, then TCP/IP is properly installed and functioning on this computer.

7. Assign static IP address/mask with **ifconfig** command to your computer.

II. ADDITIONAL EXERCISES

1. Configure your computer to receive Network Settings from DHCP Server of the Institution.
2. Configure DNS Server address for your host with static IP address.
3. Learn about configuring a host behind proxy server of your Institution
4. Learn about Internet Browser's Network Configurations.

Socket Programming in 'C' using TCP – Iterative Client – Server Programs

Objectives:

- To familiarize with application level programming with sockets.
- To understand principles of Inter-Process Communication with Unix TCP Sockets.
- To Learn to write Network programs using C programming language

Prerequisites:

- Knowledge of the C programming language and Linux Networking APIs
- Knowledge of Basic Computer Networking

I. Sockets

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else. To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as `read()` and `write()` work with sockets in the same way they do with files and pipes.

Types of Sockets

There are four types of sockets available to the users. The first two are most commonly used and the last one is rarely used.

- **Stream Sockets:** Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order – "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.
- **Datagram Sockets:** Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in

Stream Sockets – you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).

- **Raw Sockets:** These provide users access to the underlying communication protocols, which support socket abstractions. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.

Types of Servers

There are two types of servers you can have:

- **Iterative Server:** This is the simplest form of server where a server process serves one client and after completing the first request, it takes request from another client. Meanwhile, another client keeps waiting.
- **Concurrent Servers:** This type of server runs multiple concurrent processes to serve many requests at a time because one process may take longer and another client cannot wait for so long. The simplest way to write a concurrent server under Unix is to fork a child process to handle each client separately.

How to implement a Client

The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. Both the processes establish their own sockets. The steps involved in establishing a socket on the client side are as follows:

- Create a socket with the *socket()* system call.
- Connect the socket to the address of the server using the *connect()* system call.
- Send and receive data. There are a number of ways to do this, but the simplest way is to use the *read()* and *write()* system calls.

How to implement a Server Program in C using Linux APIs?

The steps involved in establishing a socket on the server side are as follows:

- Create a socket with the *socket()* system call.
- Bind the socket to an address using the *bind()* system call. For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the *listen()* system call.

- Accept a connection with the *accept()* system call. This call typically blocks the connection until a client connects with the server.
- Send and receive data using the *read()* and *write()* system calls.

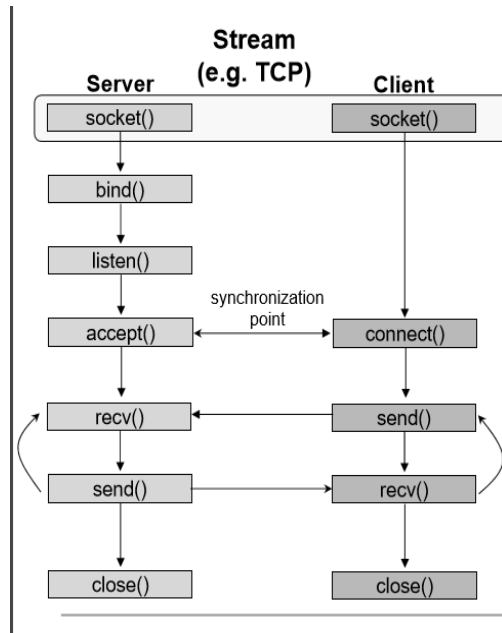


Figure 2.1: TCP client server interactions

Basic data structures used in Socket programming:

Various structures are used in Unix Socket Programming to hold information about the address and port, and other information. Most socket functions require a pointer to a socket address structure as an argument. Structures defined in this chapter are related to Internet Protocol Family.

○ Socket Descriptor

- A simple file descriptor in Unix. Data type is integer.

○ Socket Address

- This construct holds the information for socket address.

Table 2.1: System calls used in socket programming

| Primitive | Meaning |
|-----------|---|
| Socket | Create a new communication endpoint |
| Bind | Attach a local address to a socket |
| Listen | Announce willingness to accept connections |
| Accept | Block caller until a connection request arrives |
| Connect | Actively attempt to establish a connection |
| Send | Send some data over the connection |
| Receive | Receive some data over the connection |
| Close | Release the connection |

syntax

```
struct sockaddrs {  
    unsigned short sa_family; // address family, AF_xxx or //PF_xxx  
    char sa_data[14];        // 14 bytes of protocol address  
};
```

- AF stands for Address Family and PF stands for Protocol Family.

Table 2.2: Address Family

| Name | Purpose |
|-------------------|-------------------------|
| AF_UNIX, AF_LOCAL | Local communication |
| AF_INET | IPv4 Internet protocols |
| AF_INET6 | IPv6 Internet protocols |
| AF_IPX | IPX - Novell protocols |

○ **struct sockaddr_in**

- This construct holds the information about the address family, port number, Internet address, and the size of the struct sockaddr.

- struct sockaddr_in


```

      {
          short int sin_family; // Address family unsigned short int sin_port; //
          Port number
          struct in_addr sin_addr; // Internet address
      };
      
```
- The IP address structure, in_addr, is defined as follows


```

      struct in_addr
      {
          unsigned long int s_addr;
      };
      
```

Some of The System Calls Used For Conversion

Some systems (like x8086) are Little Endian i.e. least significant byte is stored in the higher address, whereas in Big endian systems most significant byte is stored in the higher address. Consider a situation where a Little Endian system wants to communicate with a Big Endian one, if there is no standard for data representation then the data sent by one machine is misinterpreted by the other. So standard has been defined for the data representation in the network (called Network Byte Order) which is the Big Endian.

The system calls that help us to convert a short/long from Host Byte order to Network Byte Order and vice versa are:

- htons() -- "Host to Network Short"
- htonl() -- "Host to Network Long"
- ntohs() -- "Network to Host Short"
- ntohl() -- "Network to Host Long"

To ensure correct byte ordering of the 16-bit port number, your server and client need to apply these functions to the port address.

For example

```

server_address.sin_addr.s_addr= htonl(INADDR_ANY);
server_address.sin_port = htons(9734);

```


IP address is a 32bit integer-not convenient for humans. So, the address is written in dotted decimal representation.

- **inet_addr()** converts the Internet host address from the standard numbers-and-dots notation into binary data. It returns nonzero if the address is valid, zero if not.
- **inet_aton()** is also used for same purpose.

System calls used

1. Socket creation in C using *socket()*

Syntax:

int sockid = socket(family, type, protocol);

where

- *sockid* is socket descriptor, an integer (like a file-handle)
- *family* is the communication domain, like PF_INET for IPv4 protocols and Internet addresses or PF_UNIX for Local communication and File addresses.
- *Type* defines communication type such as SOCK_STREAM or SOCK_DGRAM.
- *protocol* specifies protocol used. It take values like IPPROTO_TCP or IPPROTO_UDP but usually set to 0 (i.e., use default protocol).

If the return value *sockid* is negative values, it means there is problem in socket creation.

NOTE: socket call does not specify where data will be coming from, nor where it will be going to – it just creates the interface!

2. Assign address to socket using *bind()*

bind() associates and reserves a port for use by the socket.

Syntax:

int status = bind(sockid, &addrport, size);

- *Sockid* is a integer describing socket descriptor
- *addrport* is struct sockaddr which contains the (IP) address and port of the machine ,, for TCP/IP server, internet address is usually set to INADDR_ANY, i.e., chooses any incoming interface
- *size* specifies the size (in bytes) of the addrport structure
- *Status* will be assigned -1 returns on failure.

3. Listening to connection requests using *listen()*

This system call instructs TCP protocol implementation to listen for connections

Syntax:

int status = listen(sockid, queueLimit);

- *Sockid* is socket descriptor which is created using *socket()*
- *QueueLimit* is an integer which specifies number of active participants that can “wait” for a connection
- *Status* will be assigned -1 when returns on failure.

Note: The listening socket (*sockid*) is never used for sending and receiving. It is used by the server only as a way to get new sockets.

4. Establish Connection using *connect()*

The client establishes a connection with the server by calling *connect()*

Syntax:

int status = connect(sockid, &foreignAddr, addrlen);

- *sockid* is socket descriptor to be used in connection
- *foreignAddr* is struct *sockaddr* which contains address of the passive participant
- *addrlen* is *sizeof(foreignAddr)*
- *Status* will be assigned -1 when returns on failure

Note: *connect()* is blocking where as *listen()* is non blocking.

5. Accept incoming Connection using *accept()*

The server gets a socket for an incoming client connection by calling *accept()*

Syntax:

int newsockid = accept(sockid, &clientAddr, &addrLen);

- *newsockid* is an *__integer*, the new socket is created in server which is client specific and this new socket is used for data-transfer between server and client.
- *Sockid* is the socket created using *socket* system call, which is used only to listen to incoming requests from client.
- *clientAddr* is in the form of *struct sockaddr*, address of the active participant.
- *addrLen* is size of *clientAddr* parameter

Note: *accept()* is blocking, it waits for connection before returning and dequeues the next connection on the queue for socket (*sockid*).

6. Exchanging data with stream socket

Application running in server and client(s) can transfer data using *send()* and *receive()* system call.

Syntax:

int count = send(sockid, msg, msgLen, flags);

- *Sockid* is the new socket descriptor created by *accept* in server side and socket in client side, depending on where it is used.
- *Msg* is an array holding message to be transmitted
- *msgLen* holds length of message (in bytes) to transmit
- *flags* are integer, special options, usually set 0
- Return value *count* has number of bytes transmitted and is set to -1 on error

Syntax:

int count = recv(sockid, recvBuf, bufLen, flags);

- *recvBuf* stores received message
- *bufLen* holds number of bytes

7. Closing the socket using *close()*

When finished using a socket, the socket should be closed.

Syntax:

int status= close(sockid);

- *sockid*: the file descriptor (socket being closed)
- *status*: 0 if successful, -1 if error

Closing a socket closes a connection (for stream socket) and frees up the port used by the socket.

II. SOLVED EXERCISE:

Write an iterative TCP client server program where client sends a message to server and server echoes back the message to client. Client should display the original message and echoed message.

Note: As socket is also a file descriptor, we can use read and write system calls to receive and send data.

Program:

Server code:

// Make the necessary includes and set up the variables:

```
#include<stdio.h>
```

```
#include<string.h>
```

```
#include<sys/types.h>
```

```
#include<sys/socket.h>
```

```
#include<netinet/in.h>
```

```
#define PORTNO 10200
```

```
int main()
```

```
{
```

```
    int sockfd,newsockfd,portno,clilen,n=1;
```

```
    struct sockaddr_in seraddr,cliaddr;
```

```
    int i,value;
```

// create an unnamed socket for the server

```
sockfd = socket(AF_INET,SOCK_STREAM,0);
```

//Name the socket

```
seraddr.sin_family = AF_INET;
```

```
seraddr.sin_addr.s_addr = inet_addr("172.16.59.10");// **
```

```
seraddr.sin_port = htons(PORTNO);
```

```
bind(sockfd,(struct sockaddr *)&seraddr,sizeof(seraddr));
```

//Create a connection queue and wait for clients

```
listen(sockfd,5);
```

```
while (1) {
```

```
    char buf[256];
```

```
    printf("server waiting");
```

//Accept a connection

```
clilen = sizeof(clilen);
```

```
newsockfd=accept(sockfd,(struct sockaddr *)&cliaddr,&clilen);
```

//Read and write to client on client_sockfd (Logic for problem mentioned here)

```

n = read(newsockfd,buf,sizeof(buf));
printf(" \nMessage from Client %s \n",buf);
n = write(newsockfd,buf,sizeof(buf));
}
}

```

**** - indicates replace this address with your systems IP address**

Client Code:

//Make the necessary includes and set up the variables

```

#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<string.h>

```

```

int main()

```

```

{

```

```

    int len,result,sockfd,n=1;
    struct sockaddr_in address;
    char ch[256],buf[256];

```

//Create a socket for the client

```

sockfd = socket(AF_INET, SOCK_STREAM, 0);

```

//Name the socket as agreed with the server

```

address.sin_family=AF_INET;
address.sin_addr.s_addr=inet_addr("172.16.59.10");  **
address.sin_port=htons(10200);
len = sizeof(address);

```

//Connect your socket to the server's socket

```

result=connect(sockfd,(struct sockaddr *)&address,len);
if(result==-1)
{

```

```

        perror("\nCLIENT ERROR");
        exit(1);
    }
    //You can now read and write via sockfd (Logic for problem mentioned
    here)
    printf("\nENTER STRING\t");
    gets(ch);
    ch[strlen(ch)]='\0';
    write(sockfd,ch,strlen(ch));
    printf("STRING SENT BACK FROM SERVER IS .....");
    while(n){
        n=read(sockfd,buf,sizeof(buf));
        puts(buf);
    }
}
}

```

**** - indicates replace this address with your systems IP address**

Steps to execute the program

1. Open two terminal windows and open a text file from each terminal with .c extension using command:
\$gedit filename.c
2. Type the client and server program in separate text files and save it before exiting the text window.
3. First compile and run the server using commands mentioned below
 - a. ***\$gcc filename -o executablefileName //renaming the a.out file***
 - b. ***\$/ executablefileName***
4. Compile and run the client using the same instructions as listed in 3a & 3b.

Note: The ephemeral port number has to be changed every time the program is executed.

III. LAB EXERCISES:

Write an iterative TCP client server 'C' program

1. DayTime Server: Where client sends request to time server to send current time. Server responds to the request and sends the current time of server to client. [Hint: read man pages of `asctime()` and `localtime()`]. Display server process id at client side along with time.
2. Write a TCP client which sends sentences to a server program. Server displays the sentence along with client ip and ephemeral port number. Server then responds to the client by echoing back the sentence in uppercase. The process continues until one of them types "QUIT".

IV. ADDITIONAL EXERCISES:

Write an iterative TCP client server 'C' program

1. To illustrate encryption and decryption of messages using TCP. The client accepts message to be encrypted through standard input device. The client will encrypt the string by adding 4(random value) to ASCII value of each alphabet. The encrypted message is sent to the server. The server then decrypts the message and displays both encrypted and decrypted form of the string. Program terminates after one session.
2. Where the client accepts a sentence from the user and sends it to the server. The server will check for duplicate words in the string. Server will find number of occurrence of duplicate words present and remove the duplicate words by retaining single occurrence of the word and send the resultant sentence to the client. The client displays the received data on the client screen. The process repeats until user enter the string "Stop". Then both the processes terminate.

Socket Programming in 'C' using TCP – Concurrent Servers and I/O Multiplexing

Objectives:

- To implement concurrent server to handle multiple requests by client at a time.
- To Learn use of synchronous I/O multiplexing concepts in Server-Client Programs

Prerequisites:

- Understanding of Multiprocessing/Multitasking Concepts of Operating System.
- Knowledge of the C programming language and Linux Networking APIs

I. Introduction to Socket Programming in 'C' using TCP/IP – Concurrent Servers

There might need to consider the case of multiple, simultaneous clients connecting to a server. The fact that the original socket is still available and that sockets behave as file descriptors gives you a method of serving multiple clients at the same time. If the server calls fork to create a second copy of itself, the open socket will be inherited by the new child process. It can then communicate with the connecting client while the main server continues to accept further client connections. This is, in fact, a fairly easy change to make to your server program, which is shown in the following

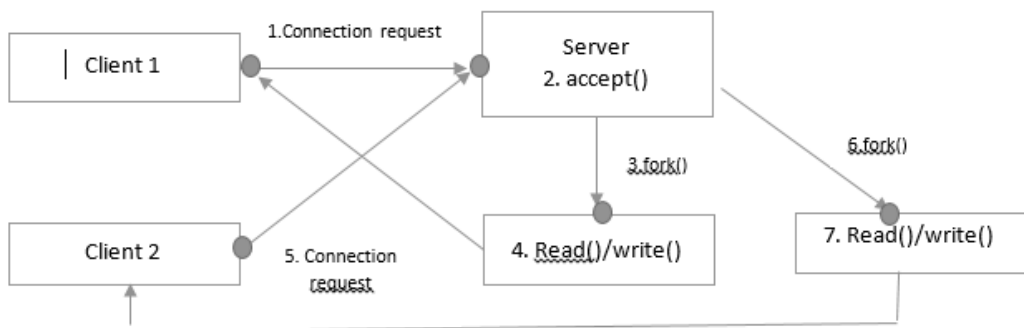


Figure 3.1: TCP concurrent server and clients interactions

Socket Programming in 'C' using TCP/IP – IO multiplexing using *select ()* system call

I/O multiplexing is the capability to tell the kernel that we want to be notified if one or more I/O conditions are ready, like input is ready to be read, or descriptor is capable of taking more output.

I/O multiplexing is typically used in networking applications in the following scenarios:

- When a client is handling multiple descriptors (normally interactive input and a network socket)
- When a client to handle multiple sockets at the same time (this is possible, but rare)
- If a TCP server handles both a listening socket and its connected sockets
- If a server handles both TCP and UDP
- If a server handles multiple services and perhaps multiple protocols.

select() gives you the power to monitor several sockets at the same time. It'll tell you which ones are ready for reading, which are ready for writing, and which sockets have raised exceptions, if you really want to know that. And also a server can deal with multiple clients by waiting for a request on many open sockets at the same time. The select function operates on data structures, *fd_set*, that are sets of open file descriptors. Several macros are defined for manipulating these sets:

- `void FD_ZERO(fd_set *fdset):` initializes an *fd_set* to the empty set, *FD_SET*
- `void FD_CLR(int fd, fd_set *fdset):` clear elements of the set corresponding to the file descriptor passed as *fd*
- `void FD_SET(int fd, fd_set *fdset) :` set elements of the set corresponding to the file descriptor passed as *fd*
- `int FD_ISSET(int fd, fd_set *fdset):` returns nonzero if the file descriptor referred to by *fd* is an element of the *fd_set* pointed to by *fdset*.

Syntax:

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds, struct timeval *timeout);

The *select()* function indicates which of the specified file descriptors is ready for reading, ready for writing, or has an error condition pending. If the specified condition is false for all of the specified file descriptors, *select()* blocks up to the specified timeout interval, until the specified condition is true for at least one of the specified file

descriptors. The `nfds` argument specifies the range of file descriptors to be tested. The `select()` function tests file descriptors in the range of 0 to `nfds-1`. `readfds`, `writefds` and `errorfds` arguments point to an object of type `fd_set`. `readfds` specifies the file descriptors to be checked for being ready to read. `writefds` specifies the file descriptors to be checked for being ready to write, `errorfds` specifies the file descriptors to be checked for error conditions pending. On successful completion, the objects pointed to by the `readfds`, `writefds`, and `errorfds` arguments are modified to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively. For each file descriptor less than `nfds`, the corresponding bit will be set on successful completion if it was set on input and the associated condition is true for that file descriptor. The timeout is an upper bound on the amount of time elapsed before `select` returns. It may be zero, causing `select` to return immediately. If the timeout is a null pointer, `select()` blocks until an event causes one of the masks to be returned with a valid (non-zero) value. If the time limit expires before any event occurs that would cause one of the masks to be set to a non-zero value, `select()` completes successfully and returns 0.

We have seen how `fork()` can be used to handle multiple clients. But forking a new process is expensive, it duplicates the entire state (memory, stack, file/socket descriptors, ...). Threads decrease this cost by allowing multitasking within the same process. Both process and thread incurs overhead as they include creation, scheduling and context switching and also as their numbers increases, this overhead increases. Solution is `select` system call.

II. SOLVED EXERCISE ON CONCURRENT SERVERS

Write a TCP concurrent Echo server and simple client.

Server Program

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#define PORTNO 10200
int main()
{
```

```

int sockfd,newsockfd,portno,clilen,n=1;
char buf[256];
struct sockaddr_in seraddr,cliaddr;
int i,value;
sockfd = socket(AF_INET,SOCK_STREAM,0);
seraddr.sin_family = AF_INET;
seraddr.sin_addr.s_addr = inet_addr("172.16.59.10"); /**
seraddr.sin_port = htons(PORTNO);
bind(sockfd,(struct sockaddr *)&seraddr,sizeof(seraddr));
// Create a connection queue, ignore child exit details, and wait for clients
listen(sockfd,5);
while(1){
    //Accept the connection
    clilen = sizeof(clilen);
    newsockfd=accept(sockfd,(struct sockaddr *)&cliaddr,&clilen);
    //Fork to create a process for this client and perform a test to see
    whether //you're the parent or the child:
    if(fork()==0){
        // If you're the child, you can now read/write to the client on
        newsockfd.
        n = read(newsockfd,buf,sizeof(buf));
        printf(" \nMessage from Client %s \n",buf);
        n = write(newsockfd,buf,sizeof(buf));
        close(newsockfd);
        exit(0);
    }
    //Otherwise, you must be the parent and your work for this client is
    finished
    else
        close(newsockfd);
}
}

```

**** - indicates replace this address with your systems IP address**

Client Program remains same.

Steps for execution

1. Open three terminals
2. Terminal 1: \$gcc server.c -o server
./server
3. Terminal 2 : \$gcc client.c -o client1
./client1
4. Terminal 3: \$gcc client.c -o client2
./client2

III. SOLVED EXERCISE ON IO MULTIPLEXING USING *select()*

Write a TCP program to handle multiple clients.

Program: //echo server

//Begin as usual with the includes and declarations and then initialize inputs to handle input //from the keyboard

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
#include <stdio.h>
```

```
#include <netinet/in.h>
```

```
#include <sys/time.h>
```

```
#include <sys/ioctl.h>
```

```
#include <unistd.h>
```

```
int main()
```

```
{
```

```
int server_sockfd, client_sockfd;
```

```
int server_len, client_len;
```

```
struct sockaddr_in server_address;
```

```
struct sockaddr_in client_address;
```

```
int result;
```

```
fd_set readfds, testfds;
```

//Create and name a socket for the server

```
server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
server_address.sin_family = AF_INET;
```

```
server_address.sin_addr.s_addr = inet_addr("172.16.56.10");/**
```

```
server_address.sin_port = htons(10200);
```

```

server_len = sizeof(server_address);
bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
//Create a connection queue and initialize readfds to handle input from server_sockfd
listen(server_sockfd, 5);
FD_ZERO(&readfds);
FD_SET(server_sockfd, &readfds);
//Now wait for clients and requests. Because you have passed a null pointer as the timeout //parameter, no timeout will occur. The program will exit and report an error if select returns
//a value less than 1
while (1) {
    char ch;
    int fd ,nread;
    testfds = readfds;
    printf("server waiting\n");

    result = select(FD_SETSIZE, &testfds, (fd_set *)0,(fd_set *)0, (struct timeval *) 0);
    //After this time, test result. If there has been an error, the program exits:
    if(result < 1) {
        perror("server error");
        exit(1);
    }

    //Once you know you've got activity, you can find which descriptor it's on by checking //each in turn using FD_ISSET
    for(fd = 0; fd < FD_SETSIZE; fd++) {
        if(FD_ISSET(fd,&testfds)) {

            //If the activity is on server_sockfd, it must be a request for a new connection, and you //add the associated client_sockfd to the descriptor set:
            if(fd == server_sockfd) {
                client_len = sizeof(client_address);
                client_sockfd = accept(server_sockfd, (struct sockaddr *)&client_address,
                    &client_len);
            }
        }
    }
}

```

```

        FD_SET(client_sockfd, &readfds);
        printf("adding client on fd %d\n", client_sockfd); }
//If it isn't the server, it must be client activity. If close is received, the client has
gone
//away, and you remove it from the descriptor set. Otherwise, you "serve" the
client
//as in the previous examples.

```

```

    else
    {
        ioctl(fd, FIONREAD, &nread);
        if(nread == 0) {
            close(fd);
            FD_CLR(fd, &readfds);
            printf("removing client on fd %d\n", fd);
        }
        else {
            read(fd, &ch, 1);
            sleep(5);
            printf("serving client on fd %d\n", fd);
            ch++;
            write(fd, &ch, 1);

```

```

}}}}}}

```

**** - indicates replace this address with your systems IP address**

Sample Input and Output

```
student@oslab-01: ~  
student@oslab-01:~$ ./select  
server waiting  
adding client on fd 4  
server waiting  
serving client on fd 4  
server waiting  
adding client on fd 5  
serving client on fd 4  
server waiting  
serving client on fd 5  
server waiting  
█
```

```
student@oslab-01: ~  
student@oslab-01:~$ ./c  
ENTER STRING   cb  
STRING SENT BACK FROM SERVER IS .....c  
@  
b  
@  
█
```

```
student@oslab-01: ~  
student@oslab-01:~$ ./c  
ENTER STRING   n  
STRING SENT BACK FROM SERVER IS .....n  
@  
█
```

IV. LAB EXERCISES

1. Write a TCP concurrent client server program where server accepts integer array from client and sorts it and returns it to the client along with process id.

2. Implement concurrent Remote Math Server

To perform arithmetic operations in the server and display the result at the client. The client accepts two integers and an operator from the user and sends it to the server. The server then receives integers and operator. The server will perform the operation on integers and sends result back to the client which is displayed on the client screen. Then both the processes terminate.

3. Implement simple TCP daytime server using select().

V. ADDITIONAL EXERCISES

1. Write a concurrent TCP daytime server 'C' program. Along with the result server should also send the process id to client.
2. Write a concurrent TCP client server 'C' program where the client accepts a sentence from the user and sends it to the server. The server will check for duplicate words in the string. Server will find number of occurrence of duplicate words present and remove the duplicate words by retaining single occurrence of the word and send the resultant sentence to the client. The client displays the received data on the client screen. The process repeats until user enter the string "Stop". Then both the processes terminate.

Network Data Analysis using tcpdump

Objectives

- Understanding the network analysis tool – tcpdump

Network Analysis tool

Network Analysis tools are used to identify problems in the network, as well as to help understand the behavior of network protocols. We will use the following tools extensively in the experiments.

Tcpdump:

Tcpdump is a network traffic sniffer built on the packet capture library libpcap. While started, it captures and displays packets on the LAN segment. By analyzing the traffic flows and the packet header fields, a great deal of information can be gained about the behavior of the protocols and their operation within the network. Problems in the network can also be identified. A packet filter can be defined in the command line with different options to obtain a desired output.

Basics

Below are a few options you can use when configuring `tcpdump`.

Options

- **-i any** : Listen on all interfaces just to see if you're seeing any traffic.
- **-I eth0** : Listen on the eth0 interface.
- **-D** : Show the list of available interfaces
- **-n** : Don't resolve hostnames.
- **-nn** : Don't resolve hostnames *or* port names.
- **-q** : Be less verbose (more quiet) with your output.
- **-t** : Give human-readable timestamp output.
- **-tttt** : Give maximally human-readable timestamp output.
- **-X** : Show the packet's *contents* in both hex and ascii.
- **-XX** : Same as `-X`, but also shows the ethernet header.

- **-v, -vv, -vvv** : Increase the amount of packet information you get back.
- **-c** : Only get *x* number of packets and then stop.
- **-s** : Define the *snaplength* (size) of the capture in bytes. Use `-s0` to get everything, unless you are intentionally capturing less.
- **-S** : Print absolute sequence numbers.
- **-e** : Get the ethernet header as well.
- **-q** : Show less protocol information.
- **-E** : Decrypt IPSEC traffic by providing an encryption key.

Expressions

In `tcpdump`, *Expressions* allow you to trim out various types of traffic and find exactly what you're looking for. Mastering the expressions and learning to combine them creatively is what makes one truly powerful with `tcpdump`.

There are three main types of expression: `type`, `dir`, and `proto`.

- Type options are: `host`, `net`, and `port`.
- Direction lets you do `src`, `dst`, and combinations thereof.
- Proto(col) lets you designate: `tcp`, `udp`, `icmp`, `ah`, and many more.

Examples

So, now that we've seen what our options are, let's look at some real-world examples that we're likely to see in our everyday work.

BASIC COMMUNICATION

By looking at all interfaces.

```
# tcpdump -i any
```

SPECIFIC INTERFACE

```
# tcpdump -i eth0
# tcpdump -tttnnvvS
```

FIND TRAFFIC BY IP

One of the most common queries, this will show you traffic from 1.2.3.4, whether it's the source or the destination.

```
# tcpdump host 1.2.3.4
```

FILTERING BY SOURCE AND DESTINATION

```
# tcpdump src 2.3.4.5  
# tcpdump dst 3.4.5.6
```

FINDING PACKETS BY NETWORK

To find packets going to or from a particular network, use the `net` option. You can combine this with the `src` or `dst` options as well.

```
# tcpdump net 1.2.3.0/24
```

SHOW TRAFFIC RELATED TO A SPECIFIC PORT

You can find specific port traffic by using the `port` option followed by the port number.

```
# tcpdump port 3389  
  
# tcpdump src port 1025
```

SHOW TRAFFIC OF ONE PROTOCOL

```
# tcpdump icmp
```

SHOW ONLY IP6 TRAFFIC

```
# tcpdump ip6
```

FIND TRAFFIC USING PORT RANGES

You can also use a range of ports to find traffic.

```
# tcpdump portrange 21-23
```

FIND TRAFFIC BASED ON PACKET SIZE

```
# tcpdump less 32
```

```
# tcpdump greater 64
```

```
# tcpdump <= 128
```

WRITING CAPTURES TO A FILE

It's often useful to save packet captures into a file for analysis in the future. These files are known as PCAP (PEE-cap) files, and they can be processed by hundreds of different applications, including network analyzers, intrusion detection systems, and of course by `tcpdump` itself. Here we're writing to a file called *capture_file* using the `-w` switch.

```
# tcpdump port 80 -w capture_file
```

READING PCAP FILES

You can read PCAP files by using the `-r` switch. Note that you can use all the regular commands within `tcpdump` while reading in a file; you're only limited by the fact that you can't capture and process what doesn't exist in the file already.

```
# tcpdump -r capture_file
```

Advanced

FROM SPECIFIC IP AND DESTINED FOR A SPECIFIC PORT

Let's find all traffic from 10.5.2.3 going to any host on port 3389.

```
tcpdump -nnvvS src 10.5.2.3 and dst port 3389
```

I. LAB EXERCISES:

1. While **tcpdump host *your_host*** is running in one command window, run ping 127.0.0.1 from another command window. From the ping output, is the 127.0.0.1 interface on? Can you see any ICMP message sent from your host in the tcpdump output? Why?
2. While **tcpdump host *your_host*** is running to capture traffic from your machine, execute telnet 128.238.66.200. Note there is no host with this IP address in the current configuration of the lab network. Save the tcpdump output of the first few packets for the lab report. After getting the necessary output, terminate the telnet session. From the saved tcpdump output, describe how the ARP timeout and retransmission were performed. How many attempts were made to resolve a non-existing IP address?
3. Explain briefly the purposes of the following tcpdump expressions.
 - a. `tcpdump udp port 520`
 - b. `tcpdump -x -s 120 ip proto 89`
 - c. `tcpdump -x -s 70 host ip addr1 and (ip addr2 or ip addr3)`
 - d. `tcpdump -x -s 70 host ip addr1 and not ip addr2`
4. Set up a simple wired network with two nodes n0 and n1 for UDP_CBR traffic. Provide duplex link between these nodes having propagation delay of 10msec and capacity of 10Mb by considering n0 as source node and n1 as sink node.

II. ADDITIONAL EXERCISES:

1. Set up a simple wired network with two nodes n0 and n1 for TCP_FTP traffic. Provide duplex link between these nodes having propagation delay of 10msec and capacity of 10Mb by considering n0 as source node and n1 as sink node.

Socket Programming in 'C' using UDP and Network Monitoring and Analysis with Wireshark

Objectives:

- Understand UDP Client-Server Socket-Programming
- To monitor network data using wireshark tool.

Perquisites:

- Understanding of connectionless service.

I. Introduction to User Datagram Protocol

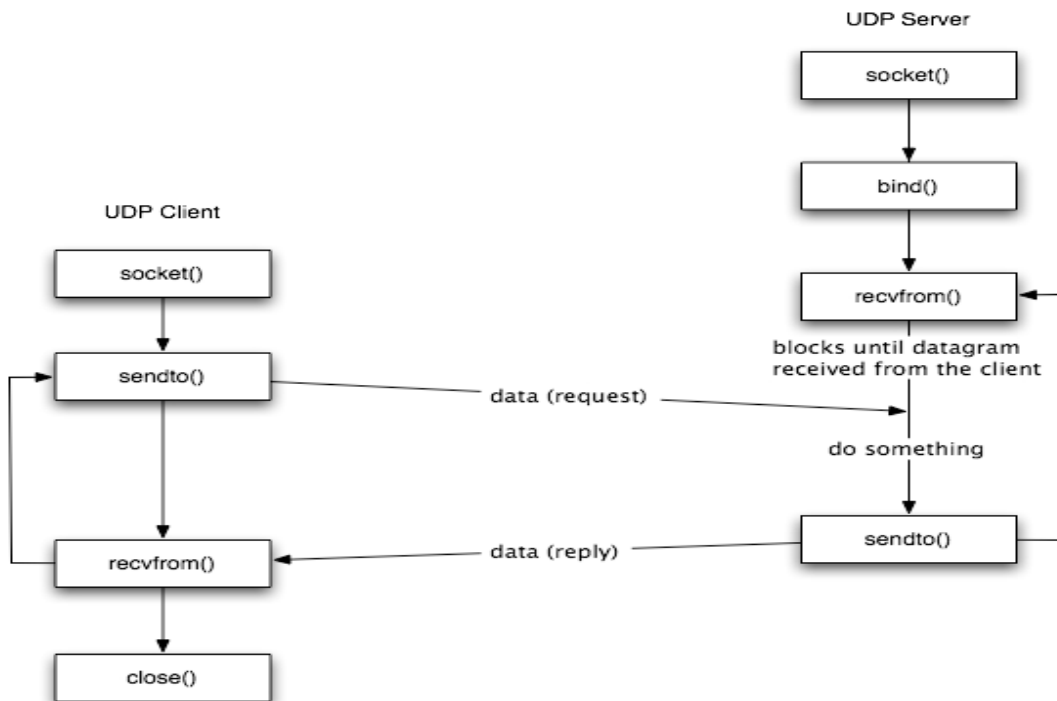


Figure 5.1: Interaction between UDP client and Server

System calls used in UDP:

1. Socket creation using `socket()`: Only difference in this call is its second parameter. For TCP, we use stream sockets as data is transferred in the form of stream where as in UDP, data is transmitted in the form of datagrams. So, the type of sockets used for UDP transmission is `DATAGRAM` socket. The system call should be used as follows

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

2. Exchange of data using `sendto()` and `recvfrom()`: As UDP does not establish the connection, while sending datagram, it should specify the address of the receiver.

Syntax:

int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);

- *sockfd*: It is a socket descriptor returned by the `socket` function.
- *msg*: It is a pointer to the data you want to send.
- *len*: It is the length of the data you want to send (in bytes).
- *flags*: It is set to 0.
- *to*: It is a pointer to `struct sockaddr` for the host where data has to be sent.
- *tolen*: It is set it to `sizeof(struct sockaddr)`.
- Return value will be number of bytes sent or -1 for error.

Similarly for receiving data, `recvfrom` is used and syntax is as follows

Syntax:

int recvfrom(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *from, int fromlen);

- *sockfd*: It is a socket descriptor returned by the `socket` function.
- *buf*: It is the buffer to read the information into.
- *len*: It is the maximum length of the buffer.
- *flags*: It is set to 0.
- *from*: It is a pointer to `struct sockaddr` for the host where data has to be read.
- *fromlen*: It is set it to `sizeof(struct sockaddr)`.
- Return value will be number of bytes recieved or -1 for error.

INTRODUCTION TO WIRESHARK

Wireshark is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.

Purposes:

- Network administrators use it to troubleshoot network problems
- Network security engineers use it to examine security problems
- Developers use it to debug protocol implementations
- People use it to learn network protocol internals

Getting started with wireshark:

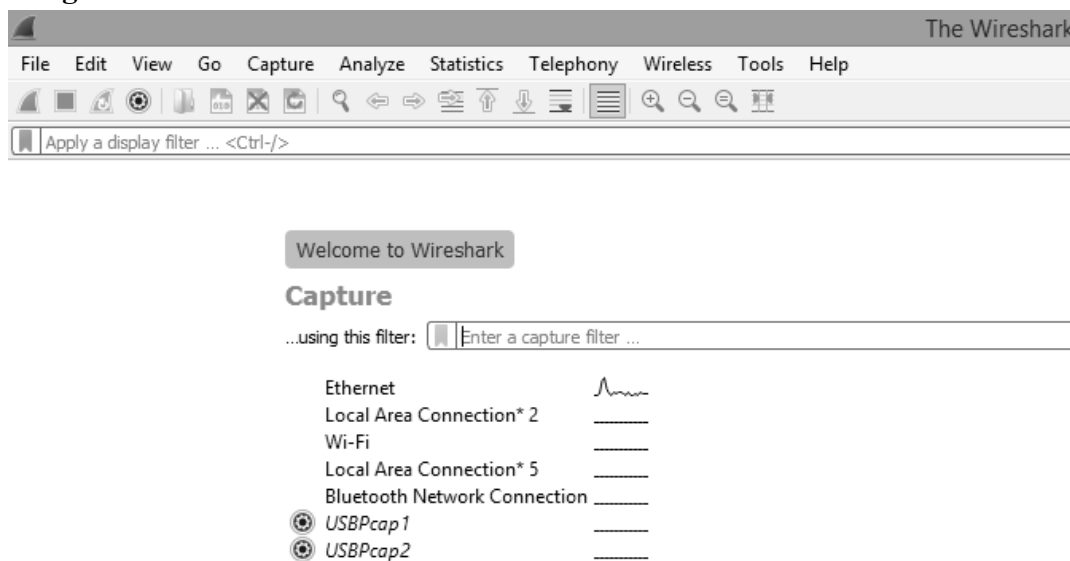


Figure 5.2: Welcome screen of Wireshark

- Choose the interface as “Ethernet” .Once interface is selected, if computer is connected to network and if there is some network traffic, main window will be displayed. Click on “start capturing packets” button on main toolbar.

The capture is split into 3 parts:

1. Packet List Panel – this is a list of packets in the current capture. It colours the packets based on the protocol type. When a packet is selected, the details are shown in the two panels below.

2. Packet Details Panel – this shows the details of the selected packet. It shows the different protocols making up the layers of data for this packet. Layers include Frame, Ethernet, IP, TCP/UDP/ICMP, and application protocols such as HTTP.
3. Packet Bytes Panel – shows the packet bytes in Hex and ASCII encodings.

Wireshark uses two types of filters, capture filters and display filters. Capture filters are used to decide which packets should be kept. Only packets that meet filter criteria will be kept. Display filters work after the capture is completed. They restrict which packets are shown, but they don't actually discard any information. Capture filters would be more useful on very busy networks when you need to limit the amount of data your machine needs to process. On the other hand, display filters don't actually save any memory; display filters let you temporarily focus an analysis without losing any underlying information.

Capture filters can be set in two different places. Go to the Capture menu and select "Options" and you will find a selection for capture filters. Alternatively, Go to the Capture menu and select "Capture Filters". From the "Capture Filters" dialog box you will see a help menu that will explain how the function works. Display filters can be entered at the top of the display screen.

Display filters can be set as: Right click on the Source IP address field in the Packet Details Panel. Select Prepare a Filter->Selected. Wireshark automatically generates a Display Filter, and applies it to the capture. The filter is shown in the Filter Bar, below the button toolbar. Only packets captured with a Source IP address of the value selected should be displayed. This same process can be performed on most fields within Wireshark, and can be used to include or exclude traffic.

SOLVED EXERCISE

Write a UDP Echo client server program.

Server Program:

```
#include<stdio.h>
#include<fcntl.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<unistd.h>
int main()
{
    int sd;
    char buf[25];
    struct sockaddr_in sadd,cadd;
    //Create a UDP socket
    sd=socket(AF_INET,SOCK_DGRAM,0);
    //Construct the address for use with sendto/recvfrom... */
    sadd.sin_family=AF_INET;
    sadd.sin_addr.s_addr=inet_addr("172.16.56.10");//**
    sadd.sin_port=htons(9704);
    int result=bind(sd,(struct sockaddr *)&sadd,sizeof(sadd));
    int len=sizeof(cadd);
    int m=recvfrom(sd,buf,sizeof(buf),0,(struct sockaddr *)&cadd,&len);
    printf("the server send is\n");
    puts(buf);
    int n=sendto(sd,buf,sizeof(buf),0,(struct sockaddr *)&cadd,len);
    return 0;
}
```

**** - indicates replace this address with your systems IP address**

Client Program

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<unistd.h>
int main()
{
    int sd;
    struct sockaddr_in address;
    sd=socket(AF_INET,SOCK_DGRAM,0);
    address.sin_family=AF_INET;
    address.sin_addr.s_addr=inet_addr("172.16.56.10");//**
    address.sin_port=htons(9704);
    char buf[25],buf1[25];
    printf("enter buf\n");
    gets(buf);
    int len=sizeof(address);
    int m=sendto(sd,buf,sizeof(buf),0,(struct sockaddr *)&address, len);
    int n=recvfrom(sd,buf,sizeof(buf),0,(struct sockaddr *)&address,&len);
    printf("the server echo is\n");
    puts(buf);
    return 0;
}
```

**** - indicates replace this address with your systems IP address**

LAB EXERCISES:

1. Write a UDP client-server program where client sends rows of a matrix and the server combines them together as a matrix.
2. **Analyzing UDP datagrams using Wireshark:**
 - Start your web browser and clear the browser's cache memory, but do not access any website yet.
 - Open Wireshark and start capturing.
 - Go back to your web browser and retrieve any file from a website. Wireshark starts capturing packets.
 - After enough packets have been captured, stop Wireshark and save the captured file.
 - Using the captured file, analyze TCP & UDP packets captured

Note: DNS uses UDP for name resolution , HTTP uses TCP.

Using the captured information, answer the following questions in your lab report.

- A.** In the packet list pane, select the first DNS packet. In the packet detail pane, select the **User Datagram Protocol**. The UDP hexdump will be highlighted in the packet byte lane.

Using the hexdump, Answer the following:

- a. the source port number.
 - b. the destination port number.
 - c. the total length of the user datagram.
 - d. the length of the data.
 - e. whether the packet is directed from a client to a server or vice versa.
 - f. the application-layer protocol.
 - g. whether a checksum is calculated for this packet or not.
- B.** What are the source and destination IP addresses in the DNS query message? What are those addresses in the response message? What is the relationship between the two?
- C.** What are the source and destination port numbers in the query message? What are those addresses in the response message? What is the relationship between the two? Which port number is a well-known port number?
- D.** What is the length of the first packet? How many bytes of payload are carried by the first packet?

2b. Analyzing TCP packets using Wireshark:

Start your web browser and clear the browser's cache memory, but do not access any website yet.

- Open Wireshark and start capturing.
- Go back to your web browser and retrieve any file from a website. Wireshark starts capturing packets.
- After enough packets have been captured, stop Wireshark and save the captured file.
- Using the captured file, select only those packets that use the service of TCP. For this purpose, type **tcp** (lowercase) in the *filter field* and press **Apply**. The packet list pane of the Wireshark window should now display a bunch of packets.

Part I: Connection-Establishment Phase

Identify the TCP packets used for connection establishment. Note that the last packet used for connection establish may have the application-layer as the source protocol.

Questions

Using the captured information, answer the following question in your lab report about packets used for connection establishment.

1. What are the socket addresses for each packet?
2. What flags are set in each packet?
3. What are the sequence number and acknowledgment number of each packet?
4. What are the window size of each packet?

Part II: Data-Transfer Phase

The data-transfer phase starts with an HTTP GET request message and ends with an HTTP OK message.

Questions

Using the captured information, answer the following question in your lab report about packets used for data transfer.

1. What TCP flags are set in the first data-transfer packet (HTTP GET message)?
2. How many bytes are transmitted in this packet?
3. How often the receiver generates an acknowledgment? To which acknowledgment rule (defined in Page 200 in the textbook) does your answer corresponds to?

4. How many bytes are transmitted in each packet? How are the sequence and acknowledgment numbers related to number of bytes transmitted?
5. What are the original window sizes that are set by the client and the server? Are these numbers expected? How do they change as more segments are received by the client?
6. Explain how the window size is used in flow control?
7. What is purpose of the HTTP OK message in the data transfer phase?

Part III: Connection Termination Phase

The data-transfer phase is followed by the connection termination phase. Note that some packets used in the connection-termination phase may have the source or sink protocol at the application layer. Find the packets used for connection termination.

Questions

Using the captured information, answer the following question in your lab report about packets used for connection termination.

1. How many TCP segments are exchanged for this phase?
2. Which end point started the connection termination phase?
3. What flags are set in each of segments used for connection termination?

I. ADDITIONAL EXERCISES

1. From the captured information, answer the following question in your lab report.
 - i) Using the hexdump, determine the following for any TCP packet:
The source port number, the destination port number, the sequence number, the acknowledgment number, the header length, the set flags, the window size and the urgent pointer value.
 - ii) Using the information in the detail pane lane, verify your answers is question 1.
 - iii) Does any of the TCP packet header carry options? Explain your answer.
 - iv) What is the size of a TCP packet with no options. What is the size of a TCP packet with options?
 - v) Is window size in any of the TCP packet zero? Explain your answer.
2. Analyze Interaction between your TCP Client-Server Programs using wireshark
3. Analyze Interaction between your UDP Client-Server Programs using wireshark

Basic Computer Network Design using HUB in GNS3

Objectives:

- Study of network simulator GNS3.
- To Learn Static IP address Assignment.
- To study the characteristics of HUB device.

I. Introduction to GNS3

GNS3 is a free graphical network simulator capable of emulating a number of network devices. Supported devices include Cisco routers and firewalls, Juniper routers, and frame-relay switches. With this software, users get an easy to use interface that allows them to build complex labs consisting of a variety of supported Cisco routers. GNS3 works by using real Cisco IOS images which are emulated using a program called Dynamips.

Some Supported GNS3 Features

- Design of high quality and complex network topologies
- Emulation of many Cisco router platforms and PIX firewalls
- Simulation of simple Ethernet, ATM and Frame Relay switches
- Connection of the simulated network to the real world
- Packet capture using Wireshark

II. Screen Layout

The following figure shows a screenshot of the GNS graphical user interface:

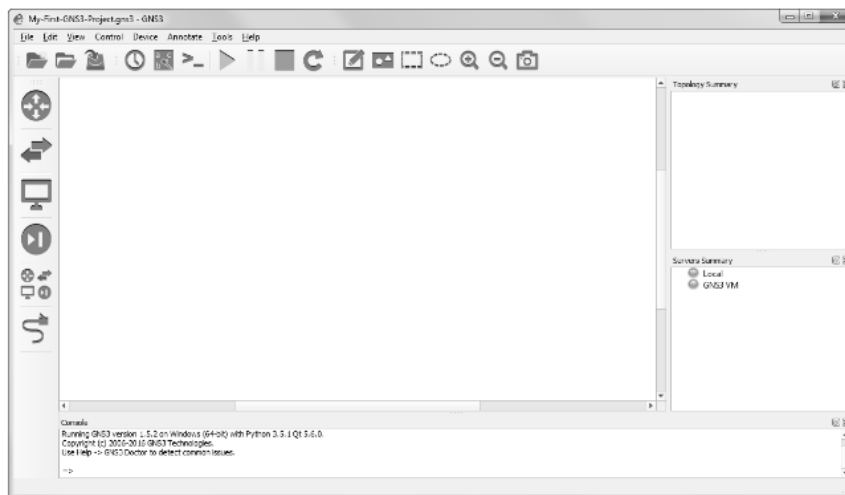


Figure 6.1: GNS graphical user interface

GNS3 Workspace: The GNS3 workspace is the area of GNS3 where you create topologies by adding devices and links.

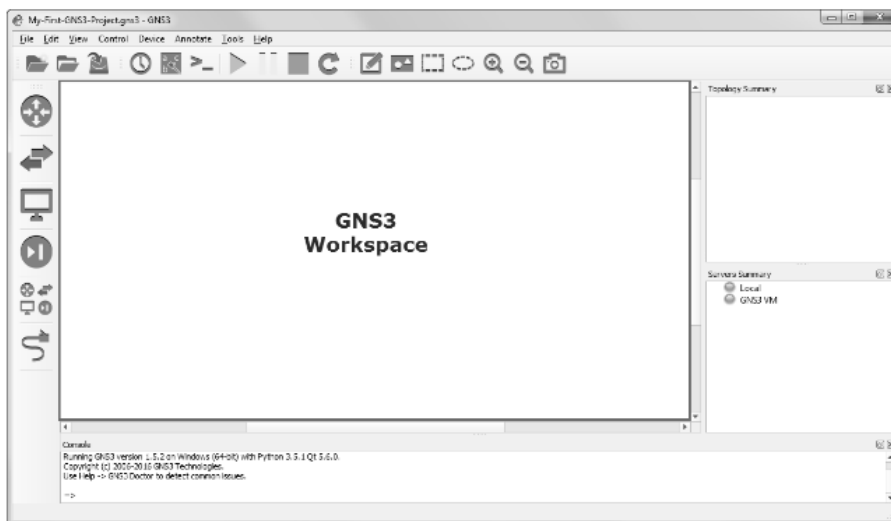


Figure 6.2: GNS workspace

Devices Toolbar: The devices toolbar allows you to add devices to your network topology. You do this by dragging devices from the Toolbar to the GNS3 workspace.

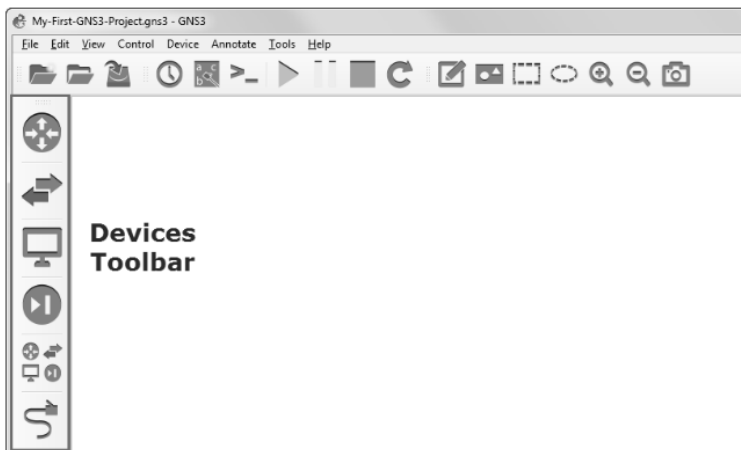


Figure 6.3: GNS Devices Toolbar

The devices toolbar is grouped into different types by default:

Routers:

GNS3 requires one or more Cisco IOS images to run on your virtual Dynamips routers, and GNS3 does not provide them. Images can be copied from a router you own or through a Cisco connection online (CCO) account, if you have a contract with Cisco. For the lab purpose, we have downloaded Cisco 3600 IOS image and Cisco 3700 IOS image from the net.

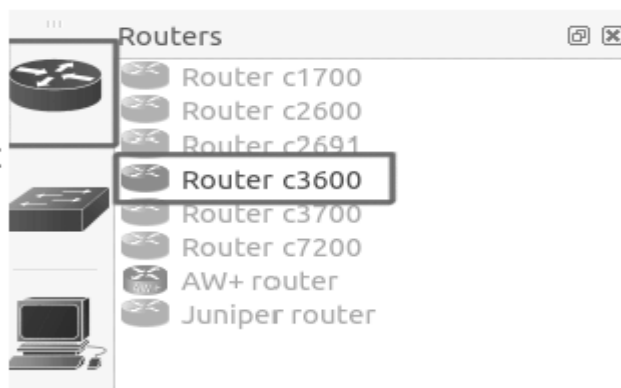


Figure 6.4: GNS Router

Switches:

Switching is done with only “Ethernet Switch” unless specified otherwise in the lab. When connecting to a switch select unused ports for new connections.

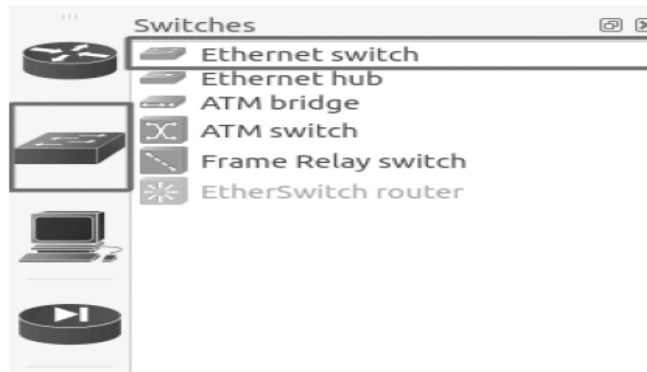


Figure 6.5: GNS Switch

Virtual PCs:

VPCS are lightweight linux machine emulators. Each one runs within a single process, it has many limitations such as lacking a full linux command list, as well as only having a single interface.

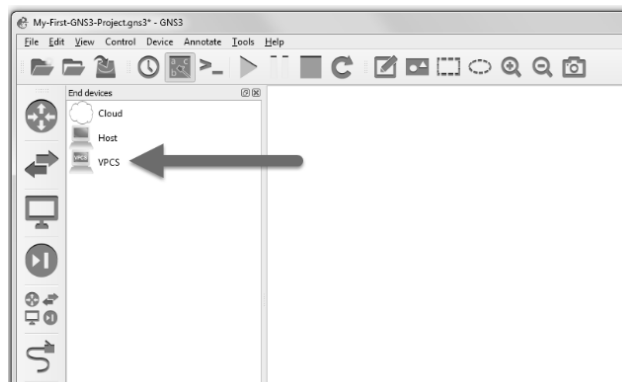


Figure 6.6: GNS VPCS

Add Links:

Adding links is simple.

1. Click the icon shown to the right.

2. Click on the target and select which port to attach a virtual connection to, do the same with the next target.



Figure 6.7: GNS Links

III. Using the GNS3 Toolbar

The GNS3 toolbar contains several groups of icons that are roughly organized by function and offer a simple way to get things done. The first group deals with projects, the second with links, the third with devices and snapshots, and the fourth with additional ways to visually organize your projects.

First Toolbar Group

The first group of toolbar icons, shown in Figure 6.8, deals with actions that affect entire projects.

From left to right, these icons are as follows:

New blank project Creates a new project folder and allows you to choose what to name your project.

Open project Opens a previously saved project. To open a project, choose the project folder name and select the file named `<project_name>.gns3`.

Save project Saves a complete project to the GNS3 *projects* folder. By default, a PNG image file of your workspace is saved with your project.



Figure 6.8: First toolbar group

Second Toolbar Group

The buttons in the second group of toolbar icons, shown in Figure 6.9, allow you to create project snapshots, show or hide interface labels, and connect to your devices using the virtual console port on your devices.

From left to right, these icons are as follows:

Snapshot Creates a snapshot of your devices, links, and IOS configurations to record the state of your workspace at that time. You can save more than one snapshot and revert to a saved snapshot at any time. Options are Create, Delete, Restore, and Close.

Show interface labels Shows or hides interface names used by a link. These labels are abbreviated and displayed with devices in your workspace (for example, f0/0 is displayed for FastEthernet0/0).

Console connect to all devices Opens a console connection to all running routers in your workspace.

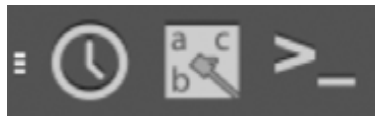


Figure 6.9: Second toolbar group

Third Toolbar Group

The third group of toolbar icons, shown in Figure 6.10, primarily deals with controlling devices.



Figure 6.10: Third toolbar group

From left to right, these four icons are as follows:

Start/Resume all devices Starts all stopped devices or resumes all suspended devices in your workspace.

Suspend all devices Places all suspend-capable devices in a suspended state.

Stop all devices Stops all devices.

Reload all devices Reloads all devices. Be sure to save your router configurations and project before reloading or else you might lose your configurations.

Fourth Toolbar Group

The final group of toolbar icons, shown in Figure 6.11, provides tools to present your network layouts more clearly. You can add objects such as rectangles and ellipses to your project, and even generate a screenshot of your workspace.



Figure 6.11: Fourth toolbar group

From left to right, the icons in the last toolbar group are as follows:

Add a note Creates text annotations in your workspace. Double-click text to modify it, and right-click the text object to change the Style attributes (such as font size and color). You can also rotate text objects from 0 to 360 degrees.

Insert a picture Adds images and logos to your projects. GNS3 supports PNG, JPG, BMP, XPM, PPM, and TIFF file formats.

Draw a rectangle Draws dynamically sizable rectangles. You can right-click a rectangle object to change the Style attributes for border and border color. Rectangle objects can be rotated from 0 to 360 degrees.

Draw an ellipse Draws dynamically sizable ellipses. You can right-click an ellipse object to change the border style and color.

Zoom in Zooms in your workspace to see details.

Zoom out Zooms out of your workspace for a bigger bird's-eye view.

Screenshot Generates a screenshot of your workspace. The image can be saved as a PNG, JPG, BMP, XPM, PPM, or TIFF file and by default is saved in your *GNS3/projects* folder.

Objects (notes, pictures, and shapes) that you add to your workspace can be grouped into layers. To raise or lower an object, right-click the object and select **Raise one layer** or **Lower one layer**. This feature allows you to manipulate objects in a layer without affecting other layers. You can display layer positions for your objects by choosing **View->Show Layers** from the menu, which is useful during advanced layer manipulation. By adding shapes and colors with this toolbar, you can divide network

components into logical groups. With text, you can add notes and reminders about how your project is configured.

IV. Address Resolution Protocol (ARP) and Reverse Address Resolution Protocol (RARP)

The ARP and RARP protocols perform the translation between IP addresses and MAC layer addresses. We will discuss ARP for broadcast LANs, particularly Ethernet LANs

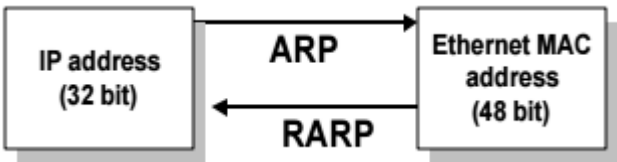


Figure 6.12: Information Exchange

- ARP Packet**

| Hardware Type | | Protocol Type |
|---|-----------------|---------------------------------|
| Hardware length | Protocol length | Operation Request 1, Reply 2 |
| Sender hardware address (For example, 6 bytes for Ethernet) | | |
| Sender protocol address (For example, 4 bytes for IP) | | |
| Target hardware address (For example, 6 bytes for Ethernet) (It is not filled in a request) | | |
| Target protocol address (For example, 4 bytes for IP) | | |

Figure 6.13: ARP Packet

- Encapsulation of ARP Packet**

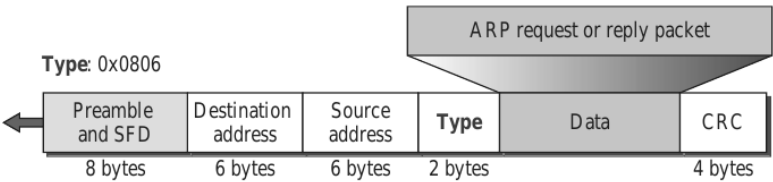


Figure 6.14: ARP Packet

Illustration of Address Translation with ARP

ARP Request: Argon broadcasts an ARP request to all stations on the network: “What is the hardware address of Router137?”

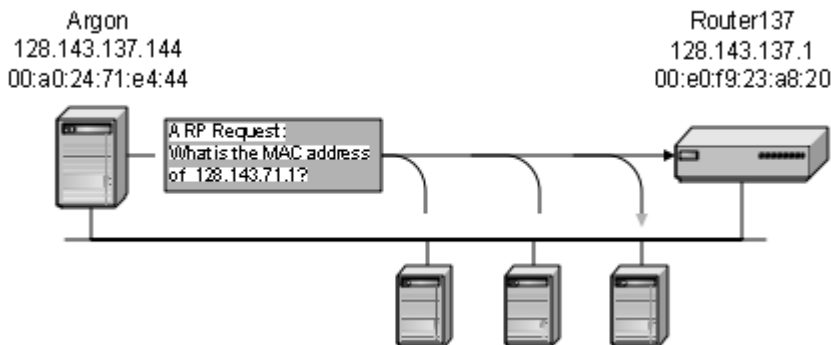


Figure 6.15: ARP Request

ARP Reply: Router 137 responds with an ARP Reply which contains the hardware address

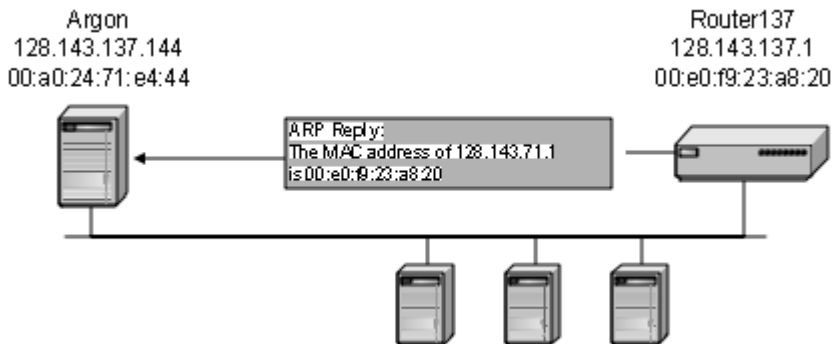


Figure 6.16: ARP Reply

Address Translation done for the above illustration:

ARP Request from Argon:

Source hardware address: 00:a0:24:71:e4:44

Source protocol address: 128.143.137.144

Target hardware address: 00:00:00:00:00:00

Target protocol address: 128.143.137.1

ARP Reply from Router137:

Source hardware address: 00:e0:f9:23:a8:20

Source protocol address: 128.143.137.1

Target hardware address: 00:a0:24:71:e4:44

Target protocol address: 128.143.137.144

V. SOLVED EXERCISE:

When you first start GNS3, you will be prompted to create a new project:

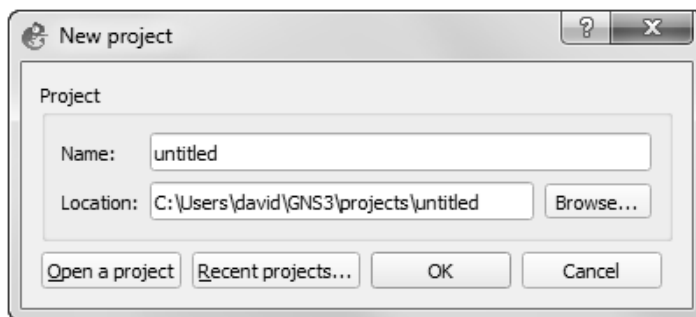


Figure 6.17: New Project Window

Name the project as desired and then click **OK**.

Working with VPCS

1. Drag and drop the **VPCS** node (device) to the GNS3 **Workspace**. An instance of the node becomes available in the **Workspace**. In this example a new VPCS with the name PC1 is now available:

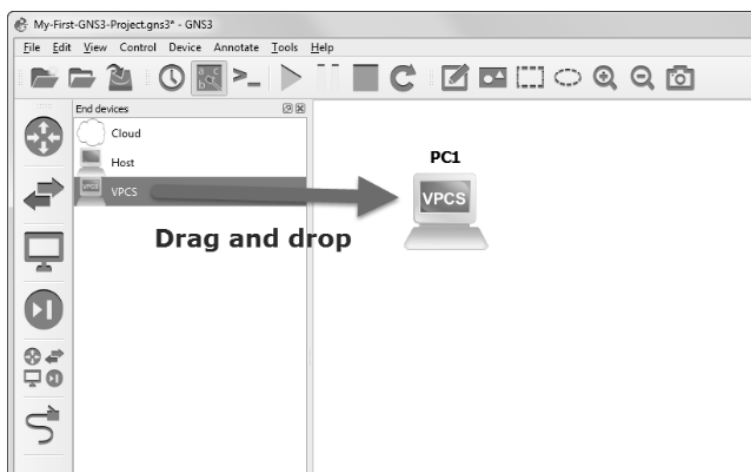


Figure 6.18: Dragging of First VPCS

2. Drag and drop the **VPCS** node again into the GNS3 **Workspace**. In this example, another VPCS was added to the GNS3 workspace (**PC2**):

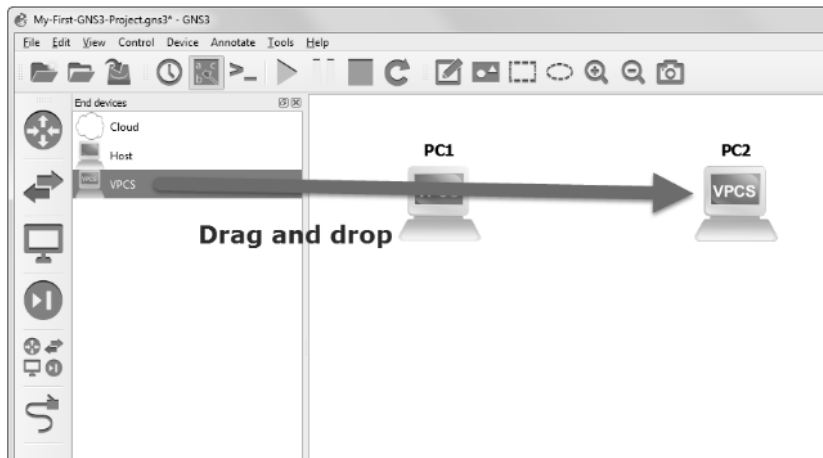


Figure 6.19: Dragging of Second VPCS

3. Click the **Add a Link** button to start adding links to your topology. The mouse cursor will change to indicate that links can be added:

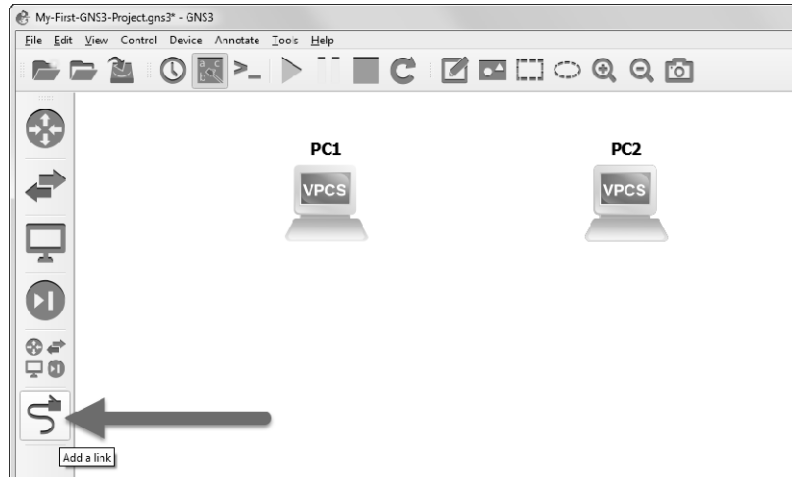


Figure 6.20: Add a Link button

4. Click on **PC1** in your topology to display available interfaces. In this example **Ethernet0** is available (this is device dependent):

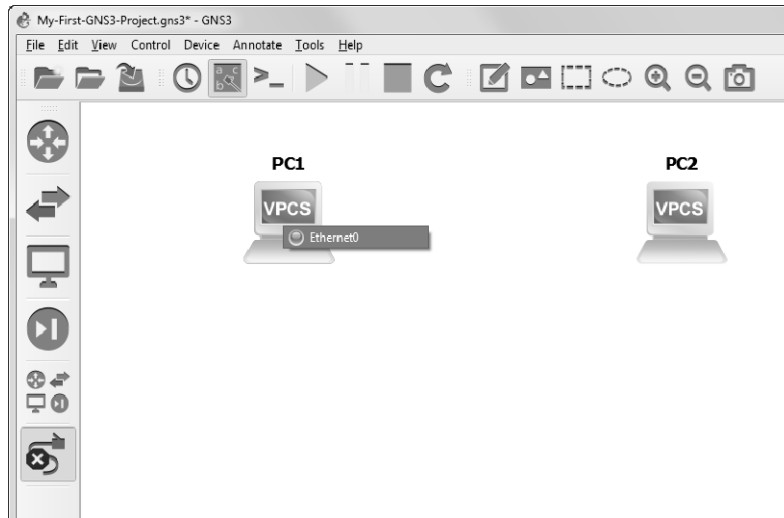


Figure 6.21: Ethernet() of PC0 is displayed

5. Click **Ethernet0** on **PC1** and then select **PC2**:

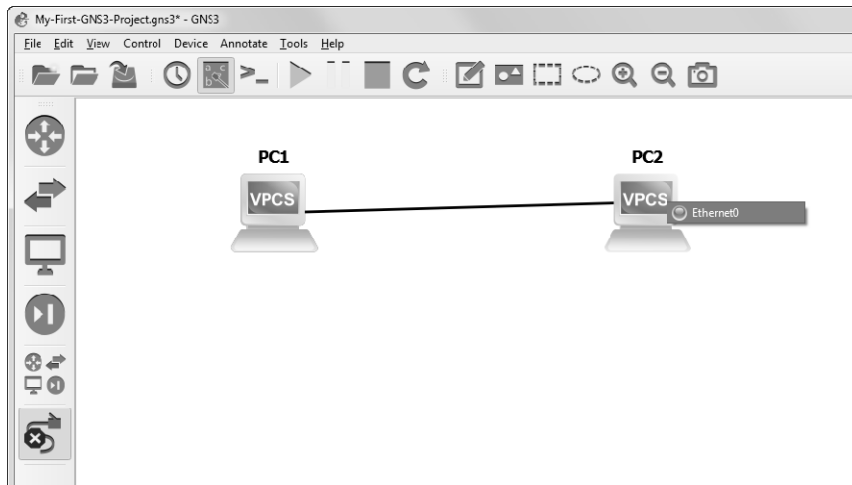


Figure 6.22: Ethernet0 of PC1 is displayed

6. Select **Ethernet0** on **R2** to complete the connection:

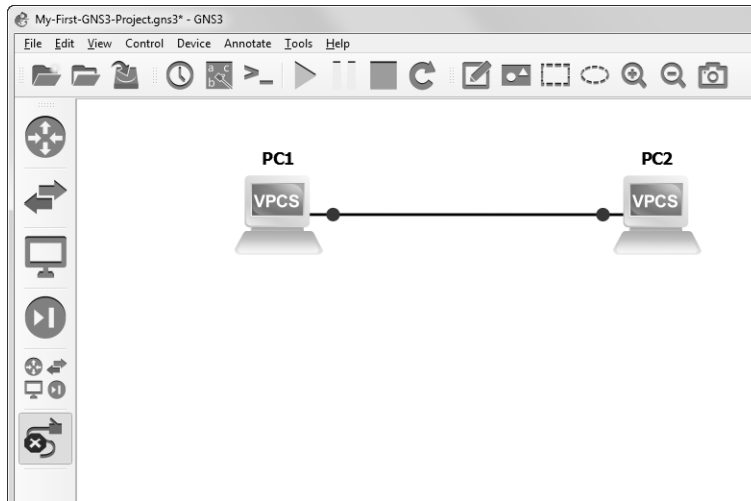


Figure 6.23: Connection Complete

7. Click the **Show/Hide interface labels** button on the **GNS3 Toolbar** to display interface labels in your topology:

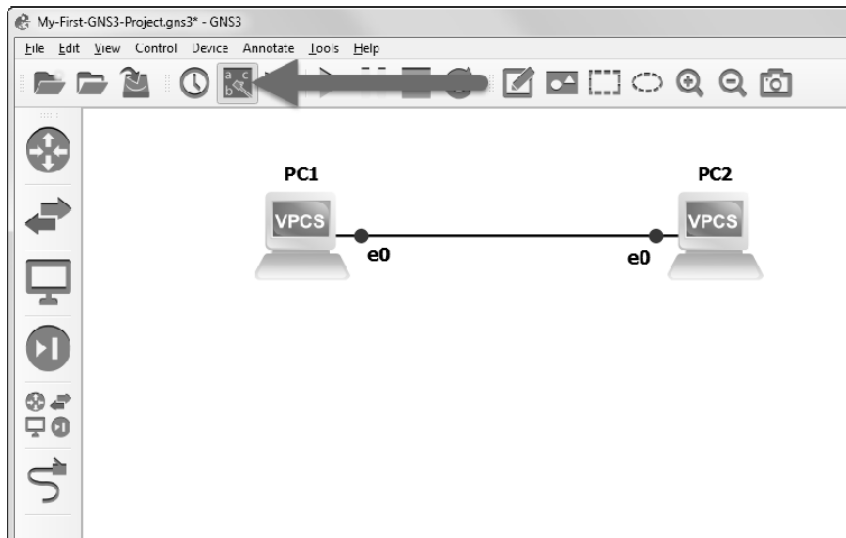


Figure 6.24: Show/Hide Interface Labels Button

8. Power on your network devices by clicking the **Start/Resume** button on the **GNS3** **Toolbar** to start up your network devices:

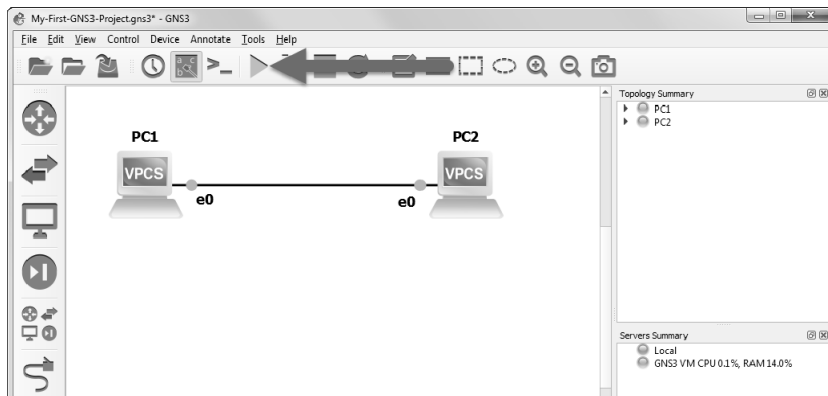


Figure 6.25: Start/Resume button

9. GNS3 indicates that the devices have been powered on by turning the interface connectors from red to green. This can also be seen in the **Topology Summary**:

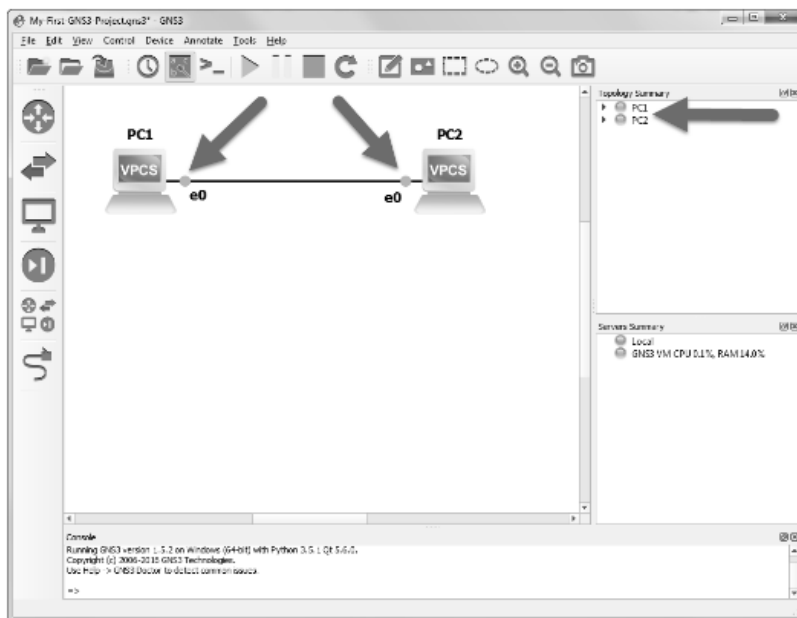


Figure 6.26: Topology Summary

10. You are now ready to configure your devices. Click the **Console connect to all devices** button on the **GNS3 Toolbar** to open a connection to every device in the topology:

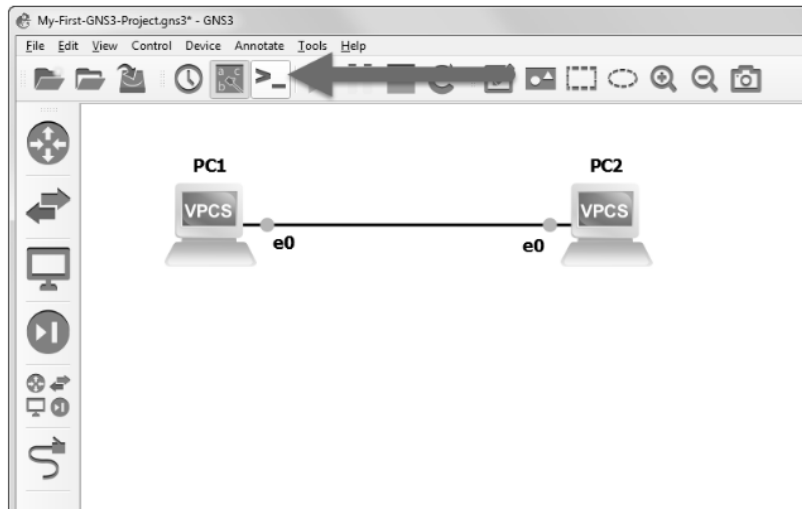


Figure 6.27: Console connect to all devices button

11. A console connection is opened to every device in the topology

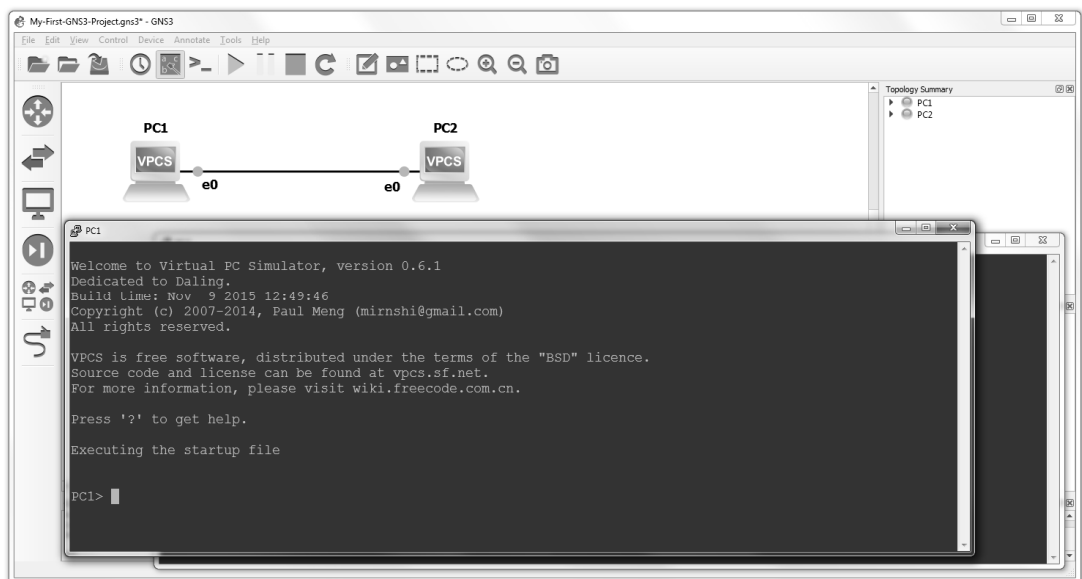


Figure 6.28: Console connection

12. Configure your PCs with IP addresses and default gateways as follows (a default gateway is configured in this example but is not used):

Syntax of ip address configuration:

ip ip-address network-mask default-gateway

PC1> ip 10.1.1.1 255.255.255.0 10.1.1.254

PC2> ip 10.1.1.2 255.255.255.0 10.1.1.254

13. **PC1** should now be able to ping **PC2** (use the key sequence **Ctrl-C** to stop the ping):

PC1> ping 10.1.1.2

Result Pings succeed.

14. To capture packets using Wireshark, right-click on the link and select **Start Wireshark**.

VI. LAB EXERCISES:

1. Design network configuration shown in Figure 6.29 for all parts. Connect all four VMs to a single Ethernet segment via a single hub as shown in Figure 6.29. Configure the IP addresses for the PCs as shown in Table 6.1.

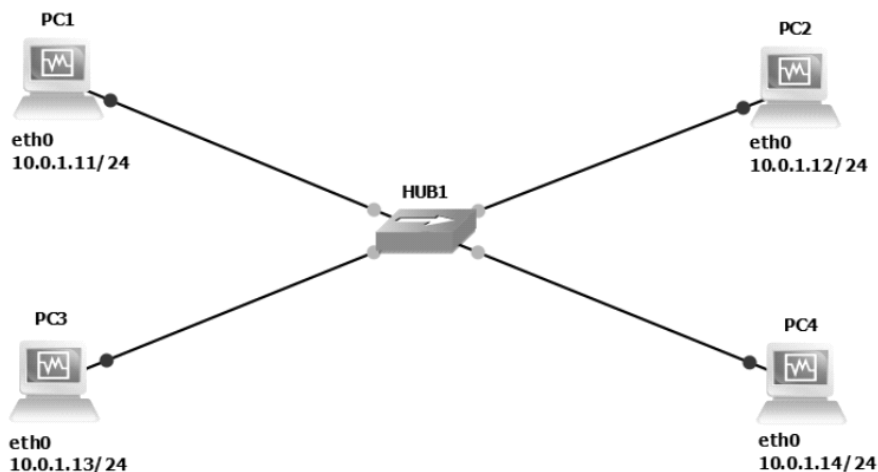


Figure 6.29: Network Design

| VMS | IP Addresses of Ethernet Interface eth0 |
|-----|---|
| PC1 | 10.0.1.11 / 24 |
| PC2 | 10.0.1.12 / 24 |
| PC3 | 10.0.1.13 / 24 |
| PC4 | 10.0.1.14 / 24 |

Table 6.1: IP Address of PCs

- a) On PC1, view the ARP cache with `show arp`
- b) Start Wireshark on PC1-Hub1 link with a capture filter set to the IP address of PC2.
- c) Issue a ping command from PC1 to PC2: **PC1% ping 10.0.1.13 -c 3**
Observe the ARP packets in the Wireshark window. Explore the MAC addresses in the Ethernet headers of the captured packets.
Direct our attention to the following fields:
 - The destination MAC address of the ARP Request packets.
 - The Type Field in the Ethernet headers of ARP packets.
- d) View the ARP cache again with the command **arp -a**. Note that ARP cache entries can get refreshed/deleted fairly quickly (~2 minutes).
show arp
- e) Save the results of Wireshark.

2. To observe the effects of having more than one host with the same (duplicate) IP address in a network.

After completing Exercise 1, the IP addresses of the Ethernet interfaces on the four PCs are as shown in Table 6.2 below. Note that PC1 and PC4 are assigned the same IP address.

| VMS | IP Address of eth0 |
|-----|--------------------|
| PC1 | 10.0.1.11 / 24 |
| PC2 | 10.0.1.12 / 24 |
| PC3 | 10.0.1.13 / 24 |
| PC4 | 10.0.1.11 / 24 |

Table 6.2: IP addresses

- a) Delete all entries in the ARP cache on all PCs.
- b) Run Wireshark on PC3-Hub1 link and capture the network traffic to and from the duplicate IP address 10.0.1.11.
- c) From PC3, issue a ping command to the duplicate IP address, 10.0.1.11, by typing
PC3% ping 10.0.1.11 -c 5
- d) Stop Wireshark, save all ARP packets and screenshot the ARP cache of PC3 using the arp -a command:
PC3% arp - a
- e) When you are done with the exercise, reset the IP address of PC4 to its original value as given in Table 6.1.

3. To test the effects of changing the netmask of a network configuration.

- a) Design the configuration as Exercise 1 and replace the hub with a switch, two hosts (PC2 and PC4) have been assigned different network prefixes. Setup the interfaces of the hosts as follows:

VPCS IP Address of eth0 Network Mask

| | | |
|-----|-----------------|-----------------|
| PC1 | 10.0.1.100 / 24 | 255.255.255.0 |
| PC2 | 10.0.1.101 / 28 | 255.255.255.240 |
| PC3 | 10.0.1.120 / 24 | 255.255.255.0 |
| PC4 | 10.0.1.121 / 28 | 255.255.255.240 |

- b) Run Wireshark on PC1-Hub1 link and capture the packets for the following scenarios
 - i) From PC1 ping PC3.
 - ii) From PC1 ping PC2.
 - iii) From PC1 ping PC4.
 - iv) From PC4 ping PC1.
 - v) From PC2 ping PC4.
 - vi) From PC2 ping PC3.
- c) Save the Wireshark output to a text file (using the “Packet Summary” option from “Print”), and save the output of the ping commands. Note that not all of the above scenarios are successful. Save all the output including any error messages.

- d) When you are done with the exercise, reset the interfaces to their original values as given Table 6.1 (Note that /24 corresponds to network mask 255.255.255.0. and /28 to network mask 255.255.255.240).

VII. ADDITIONAL EXERCISES

Based On Lab Question 1

- What is the destination MAC address of an ARP Request packet?
- What are the different Type Field values in the Ethernet headers that you observed?
- Use the captured data to analyze the process in which ARP acquires the MAC address for IP address 10.0.1.12.

Based On Lab Question 2

- Explain how the ping packets were issued by the hosts with duplicate addresses.
- Did the ping command result in error messages?
- How can duplicate IP addresses be used to compromise the data security?
- Give an example. Use the ARP cache and the captured packets to support your explanation.

Based On Lab Question 3

- Use your output data and ping results to explain what happened in each of the ping commands.
- Which ping operations were successful and which were unsuccessful? Why?

Computer Network Design using SWITCH and ROUTERS in GNS3

Objectives:

- To Learn about IP address Assignment for different subnetworks
- To study the functions of ROUTER device
- To study the functions of SWITCH device

I. Introduction to Router Configuration

Setting Up of IOS Router:

Adding IOS Images to GNS3

Before you start creating projects using IOS routers, add at least one IOS image to GNS3 for example in Figure 7.1, c3745 router image has been selected.

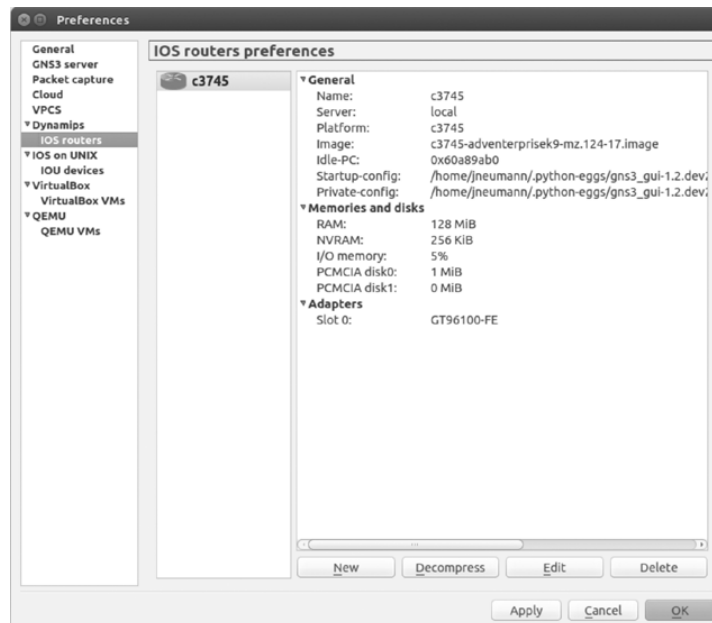


Figure 7.1: IOS routers preferences

Click **New** to start the wizard and then click the **Browse** button to locate your image file. After selecting your image file, you'll be asked whether you would like to decompress the IOS image

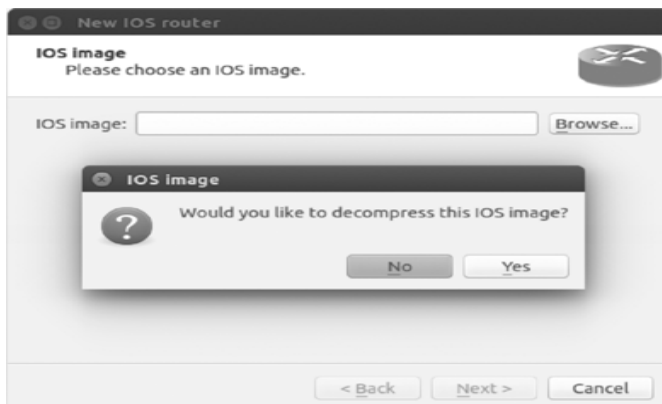


Figure 7.2: Deciding whether to decompress the IOS image

It's a good idea to let GNS3 decompress your image files; otherwise, your routers will have to decompress the images every time a router loads. Decompressing the images ahead of time will make your routers boot much faster. After decompressing your image, click **Next**, and GNS3 will attempt to recognize the router platform that your IOS belongs to, as shown in Figure 7.3.



Figure 7.3: Name and platform screen

GNS3 has determined that my image file belongs to a c3745 router platform and has automatically named it *c3745*.

In general, from here, you can just click through all the configuration settings to configure a basic router model, but the wizard provides opportunities for you to customize router memory and other features during this process. For now, click **Next** to continue. You should be presented with the Memory screen, shown in Figure 7.4.

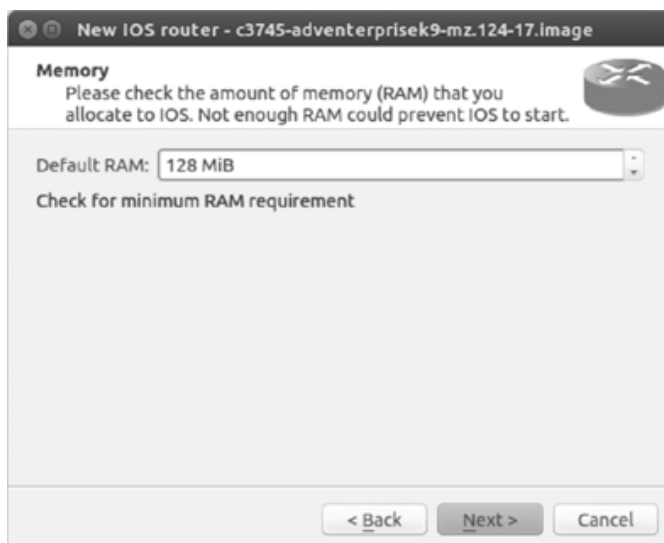


Figure 7.4: IOS Memory screen

Your routers should run fine with the default memory setting. When you're done, click **Next**, and you will be presented with the Network adapters screen, as shown in Figure 7.5.

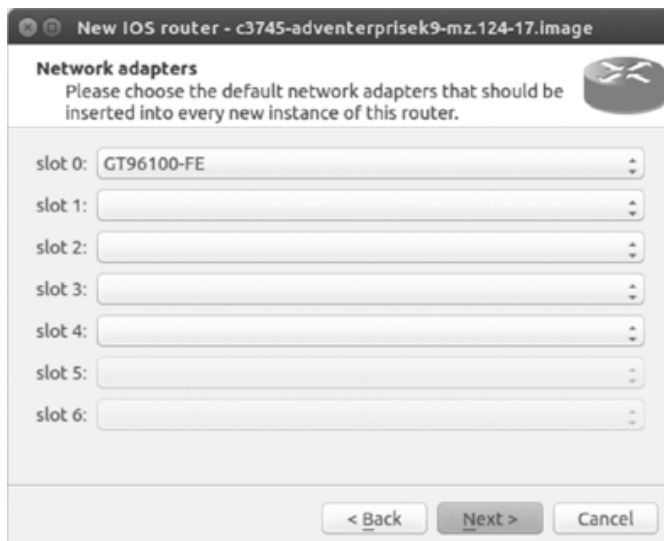


Figure 7.5: Network adapters screen

The default setting configures your router with the same standard options that are provided with a real model of the same Cisco router. If you would like to add more interfaces, use the drop-down menu next to the available slots and choose the desired network modules. The slot options will be limited to actual options that are available in the real version of the Cisco router. When you're done, click **Next** and choose any WIC modules that you would like to install. Then click **Next** again to display the Idle-PC screen, shown in Figure 7.6.



Figure 7.6: Idle-PC screen

If you start a router in GNS3 without an Idle-PC setting, your computer's CPU usage will quickly spike to 100 percent and remain there. This happens because Dynamips doesn't yet know whether your virtual router is doing something that requires system resources, so it overcompensates by giving it all the resources it can. GNS3 will run sluggishly until this is corrected, and if CPU usage is left at 100 percent for a long time, your PC's processor could overheat.

You can easily fix this by having GNS3 look for places in the IOS program code where an idle loop exists (idle loops cause the CPU to spike); the result of this calculation is called an *Idle-PC value*. When the proper Idle-PC value is applied, Dynamips should periodically *sleep* the router when these idle loops are executed, which greatly reduces CPU usage.

To have GNS3 automatically find a value, click the **Idle-PC finder** button, and GNS3 will attempt to search for a value. If GNS3 finds a suitable value, then you're done; click **Finish**. If it's unsuccessful, leave the field blank and click **Next** to save the router without an Idle-PC configuration.

To Start with the Lab exercise create a topology as shown in Figure 7.7:

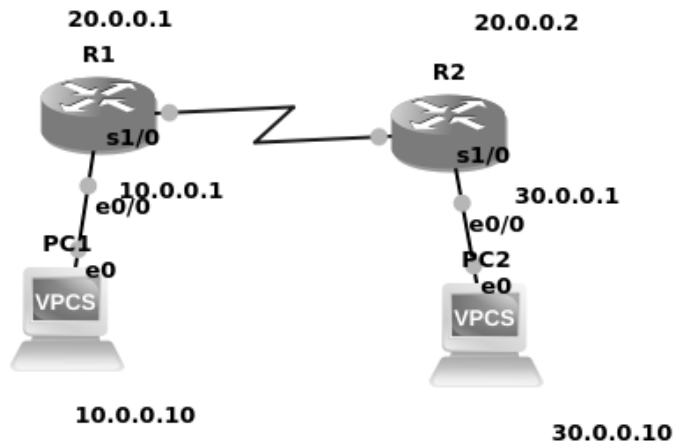


Figure 7.7: Network Topology

you will get a red light first.

II. LAB EXERCISES

1. Switching Cisco IOS Command Modes

This exercise demonstrates how to log into a router and how to work with the different Cisco IOS command modes. It is important to understand the different modes so you know where you are and what commands are accepted at any time.

- Connect the Ethernet interfaces of the Linux PCs and the Cisco router as shown in Figure 7.7. Do not turn on the Linux PCs yet.
- Right-click on Router1 and choose Start.
- Right-click on Router1 and choose Console. Wait a few seconds until the router is initialized. If everything is fine, you should see the prompt shown below. This is the User EXEC mode. If the prompt does not appear, try to restart GNS3 and repeat the setup again.

Router1>

- iv) To see which commands are available in this mode, type?

Router1>?

- v) To view and change system parameters of a Cisco router, you must enter the Privileged EXEC mode by typing:

Router1>enable

Router1#

- vi) Type the following command to disable the Privileged EXEC mode

Router1# disable

NOTE: The Cisco routers in GNS3 sometimes start up in Privileged instead of the User EXEC mode.

- vii) To modify system wide configuration parameters, you must enter the global configuration mode. This mode is entered by typing:

Router1#configure terminal

Router1(config)#

or

Router1#conf t

Router1(config)#

- viii) To make changes to a network interface, enter the interface configuration mode, with the command:

Router1(config)#interface FastEthernet0/0

Router1(config-if)#

The name of the interface is provided as an argument. Here, the network interface that is configured is FastEthernet0/0.

- ix) To return from the interface configuration to the global configuration mode, or from the global configuration mode to the Privileged EXEC mode, use the exit command:

Router1(config-if)#exit

Router1(config)#exit

Router1#

The exit command takes you one step up in the command hierarchy. To directly return to the Privileged EXEC mode from any configuration mode, use the end command:

Router1(config-if)#end

Router1#

- x) To terminate the console session from the User EXEC mode, type logout or exit:

Router1>logout

Router con0 is now available

Press RETURN to get started

2. Configuring a Cisco Router via the console

The following exercises use basic commands from the Cisco IOS that are needed to configure a Cisco router.

- i) Right-click on Router1 and choose Start.
- ii) Right-click on Router1 and choose Console. Wait some seconds until the initial console window is set up. When the router is ready to receive commands, proceed to the next step.
- iii) Configure Router 1 and Router 2 with the IP addresses given in Figure 7.7.

Note: In IOS Mode under Global Configuration, we can enable or disable IP Forwarding. When it is disabled it also deletes the contents of the routing table.

Router1(config)#ip routing

Router1(config)#no ip routing

In IOS Mode under Interface Configuration, we can enable or disable a network interface

Router1(config-if)#no shutdown

Router1(config-if)#shutdown

Tip: “no ip routing” is used to guarantee that the routing cache is empty, not routing table.

In Router 1

Interface Fastethernet0/0 in global configuration mode

R1(config)#inter f 0/0

R1(config-if)#ip address 10.0.0.1 255.0.0.0

R1(config-if)#no shutdown

R1(config-if)#exit

Interface Serial 2/0

R1(config)#inter s2/0

R1(config-if)#ip address 20.0.0.1 255.0.0.0

R1(config-if)#clock rate 64000

R1(config-if)#encapsulation ppp

R1(config-if)#no shutdown

R1(config-if)#exit

In Router 2

Interface Fastethernet 0/0

R2(config)#inter f0/0

R2(config-if)#ip address 30.0.0.1 255.0.0.0

R2(config-if)#no shutdown

R2(config-if)#exit

Interface Serial 2/0

R2(config)#inter s2/0

R2(config-if)#ip address 20.0.0.2 255.0.0.0

R2(config-if)#encapsulation ppp

R2(config-if)#no shutdown

R2(config-if)#exit

Tip to save Time: It will be tiring to manually type in the configuration data for a router, everytime you set a lab, you can save time by saving all this configurations in an excel file and simply copying and pasting in the router console window.

- iv) When you are done, use the following command to check the changes you made to the router configuration, and save the outputs:

R1# show interfaces

R1#show running-config

- v) Assign ip address for both PC's as mentioned in Figure 7.7 with appropriate ip and subnetmask and default gateway.

3. Setting static routing table entries on a Cisco router

In this exercise, you will add static routes to the routing table of Router1. The routing table must be configured so that it conforms to the network topology shown in Figure 7.7. The routes are configured manually, which is also referred to as static routing.

The IOS command to configure static routing is `ip route`. The command can be used to show, clear, add, or delete entries in the routing table. The commands are summarized in the list below.

IOS MODE: PRIVILEGED EXEC

`show ip route`

Displays the contents of the routing table.

`clear ip route *`

Deletes all routing table entries

`show ip cache`

Displays the routing cache.

IOS MODE: GLOBAL CONFIGURATION

`ip route cache / no ip route cache`

Enables or disables route caching. By default, the route cache is enabled by the router.

`ip route destination mask gw_address`

`no ip route destination mask`

Adds or deletes a static routing table entry to the destination with netmask mask. The argument gw_address is the ip address of the next hop router.

Note: Whenever an IP address is configured for a network interface on a router, routing table entries for the directly connected network are added automatically.

By default, Routers know only directly connected networks here Router 1 know only 10.0.0.0 and 20.0.0.0 it doesn't know the 30.0.0.0 like this R2 doesn't know about 10.0.0.0. So we are going to add Static route to this both router.

R1(config)#ip route Destination Network| Destination N/W SubnetMask [Next Hop Address

In Router R1, just give this command, in this case Destination is 30.0.0.0 and its subnet mask is 255.0.0.0 next hop address is 20.0.0.2

R1(config)#ip route 30.0.0.0 255.0.0.0 20.0.0.2

In Router R2

R2(config)#ip route 10.0.0.0 255.0.0.0 20.0.0.1

Now both routers know all networks.

- i) Issue a ping command from PC1 to PC2, Router1 and PC4, respectively
- ii) Save the captured Wireshark output.
- iii) Use the saved data to answer the following questions:
 - What is the output on PC1 when the ping commands are issued?
 - Which packets, if any, are captured by Wireshark?
 - Do you observe any ARP packets? If so, what do they indicate?

III. ADDITIONAL EXERCISE

1. Configure the below network topology as shown in Figure 7.8 and check the connectivity by pinging from PC0 to PC2.

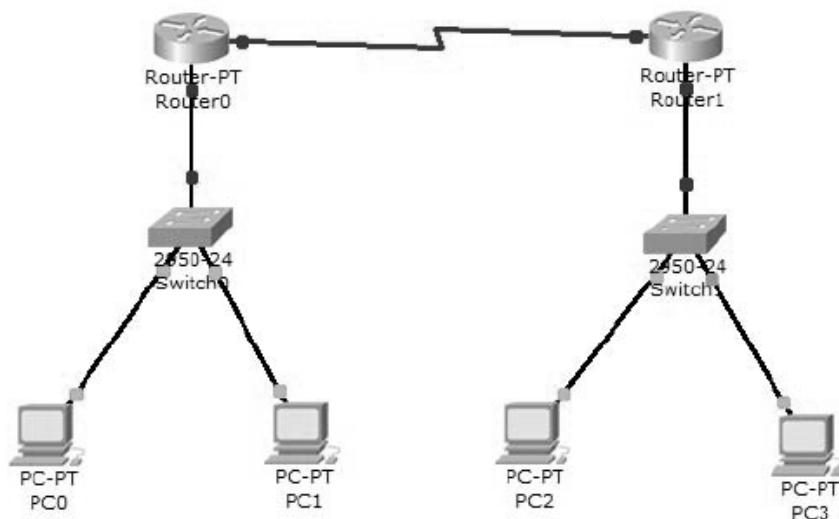


Figure 7.8: Additional Lab Topology

Study of Domain Name Server

Objectives:

- To illustrate the significance of Domain Name Server
- To Study the information exchanged between DNS and Clients

I. Introduction

DNS (Domain Name Servers)

Computers and other network devices on the Internet use an IP address to route the client request to required website. It's impossible for us to remember all the IP addresses of the servers we access every day. Hence we assign a domain name for every server and use a protocol called DNS to turn a user-friendly domain name like "howstuffworks.com" into an Internet Protocol (IP) address like 70.42.251.42 that computers use to identify each other on the network. In other words, DNS is used to map a host name in the application layer to an IP address in the network layer. DNS is a client/server application in which a domain name server, also called a DNS server or name server, manages a massive database that maps domain names to IP addresses. Client requests for address resolution which is defined as mapping a name to an address or an address to a name. It can be done in a recursive fashion or in an iterative fashion.

II. LAB EXERCISES

1. Configure the below topology to setup DNS server. R1 will use R2 as DNS server to make DNS resolutions.

First, let's begin with R1. We'll setup hostname and IP related information.

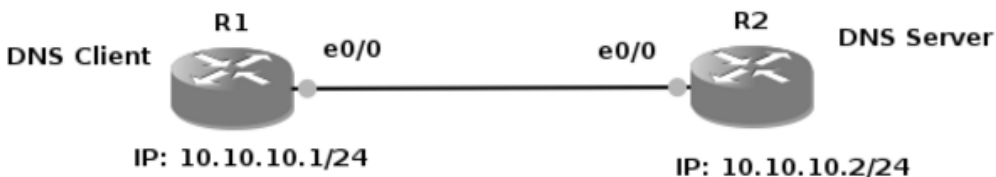


Figure 8.1: Network Topology for DNS Configuration

R1 IP configurations:

```
Enable  
configure terminal  
hostname R1  
interface e0/0  
ip address 10.10.10.1 255.255.255.0  
no shut  
do wr  
end
```

R2 IP and Hostname Configurations:

```
enable  
config t  
hostname R2  
int e0/0  
ip address 10.10.10.2 255.255.255.0  
no shut  
do wr  
end
```

Setting up R2 as DNS Server

```
config t  
ip dns server  
ip host loopback.R2.com 2.2.2.2
```

We mapped loopback.R2.com to ip address 2.2.2.2. Currently, we don't have 2.2.2.2, we could create loopback interface on R2 and assign ip 2.2.2.2.

```
interface loopback 1  
ip address 2.2.2.2 255.255.255.255  
end
```

Let's verify that loopback interface we just created is working. This will show us that the hostname correctly setup locally on R2.

```
ping loopback.R2.com
```

Now it's time to setup R1 to resolve hostnames using R2. On R1 type;

config terminal
ip domain lookup
ip name-server 10.10.10.2

Set R1 to use R2 as default gateway to get to loopback interface on R2. So that after R1 resolve **loopback.R2.com**, it can reach 2.2.2.2 through its default route (R2).
on R1 type:

config t
ip route 0.0.0.0 0.0.0.0 10.10.10.2
end

This tells our router that to get to any network not in it's routing table, it's next hop is 10.10.10.2 which is our router R2.

Now on R1, do a ping to **loopback.R2.com** and you should get a success message.

ping loopback.R2.com repeat 3

If you captured the traffic, you'll see DNS query and Answer as shown in Wireshark capture screen shot below.

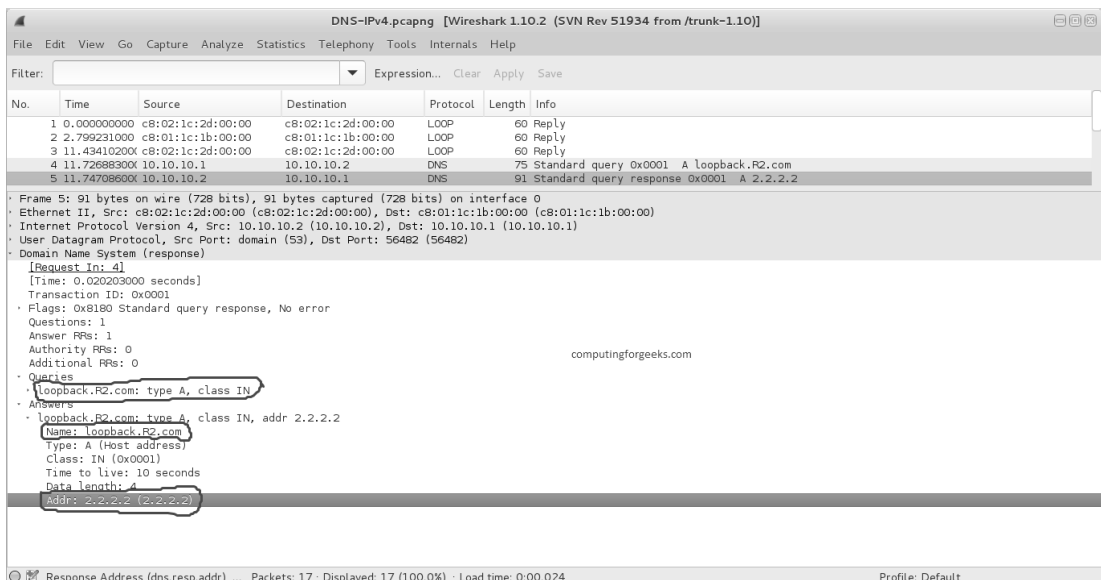


Figure 8.2 : Observation in WIRESHARK

LAB EXERCISE

1. Configure the below DNS Server and DNS Client. Test the setup. Analyze the Interaction.

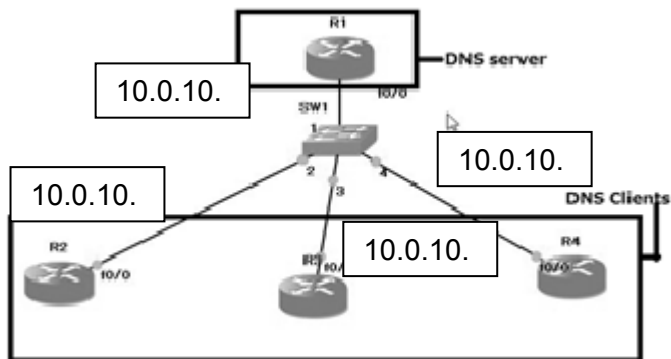


Figure 8.3: Network Topology

Study of DHCP Server

Objectives:

- Understand DHCP Service
- Analyzing DHCP Packets
- Understanding significance of Netmask value

I. DHCP OVERVIEW

The Dynamic Host Configuration Protocol (DHCP) is based on the Bootstrap Protocol (BOOTP), which provides the framework for passing configuration information to hosts on a TCP/IP network. DHCP adds the capability to automatically allocate reusable network addresses and configuration options to Internet hosts. DHCP consists of two components: a protocol for delivering host-specific configuration parameters from a DHCP server to a host and a mechanism for allocating network addresses to hosts. DHCP is built on a client/server model, where designated DHCP server hosts allocate network addresses and deliver configuration parameters to dynamically configured hosts.

II. LAB EXERCISES

Configure two VMs that will be used to test connectivity from end to end and R1 will serve as a DHCP server to distribute IP addresses. The diagram below details the current setup:

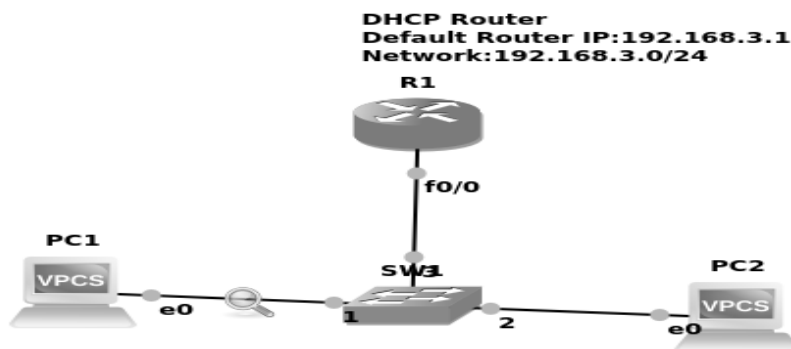


Figure 9.1: Network Topology for DHCP Configuration

1. In order to configure our router as a DHCP server the following commands were used.

R1(config)#IP dhcp pool NAME

R1(dhcp-config)#Network 192.168.3.0 255.255.255.0

R1(dhcp-config)#Default-router 192.168.3.1

The commands above create a DHCP pool, adds the network that we want to assign IP addresses from, and specifies the default gateway for this subnet.

Note: There are many other parameters that go into configuring a DHCP server but this will suffice for our test environment.

That should be it for the DHCP configuration.

2. The next thing that you want to do is configure the fastEthernet 0/0 interface which will connect to our switch.

R1(config)#Interface fastEthernet 0/0

R1(config-if)#No shutdown

R1(config-if)#ip address 192.168.3.1 255.255.255.0

The commands above will turn the interface on and assign an IP address.

3. Turn on the VPCS. In PC1 and PC2 type **dhcp**

PC1>dhcp

PC2>dhcp

4. Let's analyze some of the traffic patterns using Wireshark.

In Wireshark we see the following information with regards to DHCP:

| | | | | | | | | |
|----|------------|-------------|-----------------|------|-----|------|----------|-----------------------------|
| 8 | 8.50813900 | 0.0.0.0 | 255.255.255.255 | DHCP | 342 | DHCP | Discover | - Transaction ID 0x1028062c |
| 9 | 8.53260900 | 192.168.3.1 | 192.168.3.3 | DHCP | 342 | DHCP | Offer | - Transaction ID 0x1028062c |
| 10 | 8.53274900 | 0.0.0.0 | 255.255.255.255 | DHCP | 357 | DHCP | Request | - Transaction ID 0x1028062c |
| 11 | 8.56323600 | 192.168.3.1 | 192.168.3.3 | DHCP | 342 | DHCP | ACK | - Transaction ID 0x1028062c |

We see a discover message followed by an offer, request, and an acknowledgement. This is the process that clients go through in order to obtain an IP address via DHCP. The mnemonic for the steps above is DORA and it should help in memorizing the order of the steps.

2. NETWORK PREFIXES and ROUTING

In this exercise you study how the network prefixes (netmasks) play a role when hosts determine if a datagram can be directly delivered or if it must be sent to a router.

This part uses the network setup shown in Figure 9.2. The network includes one router, four hosts and two hubs. The IP addresses of all devices are given in Table 9.1. Here, each host has only a default route. In other words, the routing table at a host only knows about the directly connected networks and the default gateway.

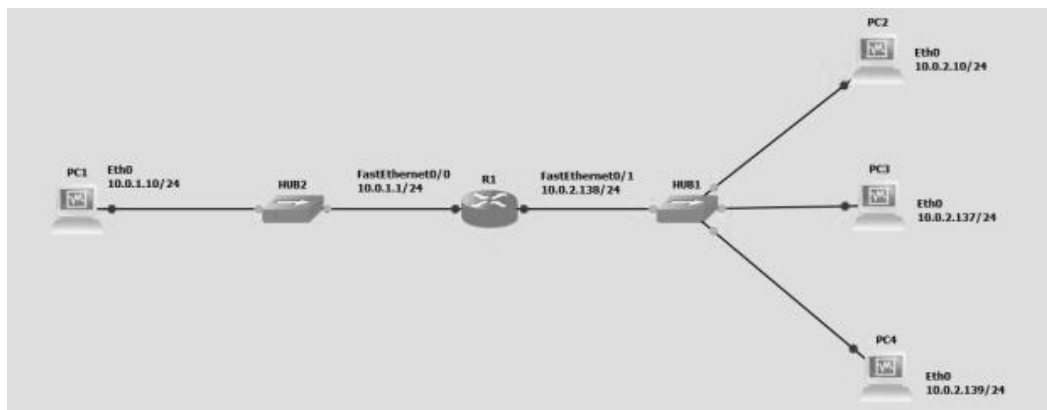


Figure 9.2: Network topology

| Linux PC | Ethernet Interface eth0 | Ethernet Interface eth1 |
|---------------|-------------------------|-------------------------|
| PC1 | 10.0.1.10 / 24 | Disabled |
| PC2 | 10.0.2.10 / 24 | Disabled |
| PC3 | 10.0.2.137 / 29 | Disabled |
| PC4 | 10.0.2.139 / 24 | Disabled |
| Cisco Routers | FastEthernet0/0 | FastEthernet0/1 |
| Router1 | 10.0.1.1 / 24 | 10.0.2.138 / 24 |

Table 9.1

Exploring the role of prefixes at hosts

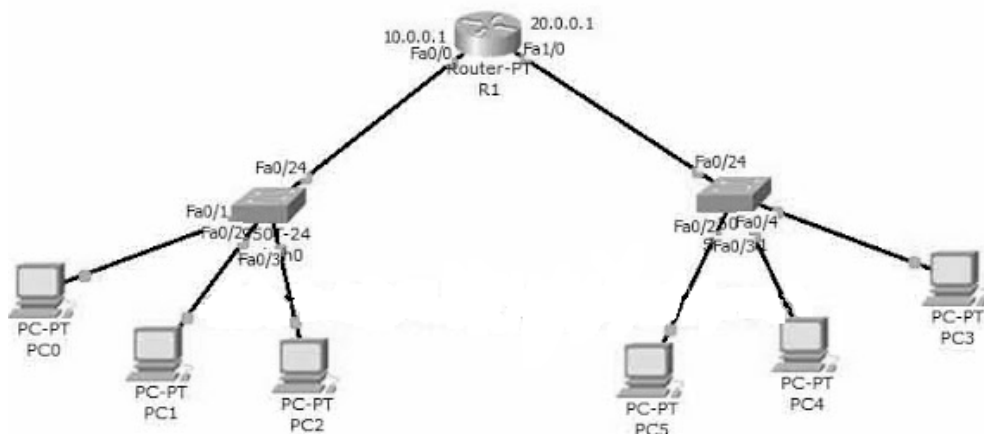
In this exercise, you explore how hosts that are connected to the same local area network, but that have different netmasks, communicate or fail to communicate.

- Configure the hosts and the router to conform to the topology shown in Figure 9.2, using the IP addresses as given in Table 9.1. Note that PC2, PC3, and PC4 have different netmasks.
- Add Router1 as default gateway on all hosts. (PC1, PC2, PC3, and PC4).
- Issue *ping* commands from PC1
 - Clear the ARP table on all PCs.
 - Start Wireshark on PC1 and on PC3, and set the capture filter to capture ICMP and ARP packets only.

- iii) Issue a ping command from PC1 to PC3 for at least two sends (-c 2).
 - iv) Save the output of the ping command at PC1 and the output of Wireshark on PC1 and PC3.
 - v) Save the ARP tables, routing tables, and routing caches of each host. Please note that these are the tables entries from Step 3 after the ping commands are issued.
- d. Issue *ping* commands from PC3 to PC4
 - i) Clear the ARP table on all PCs.
 - ii) Start Wireshark on PC3, and set the capture filter to capture ICMP and ARP packets only.
 - iii) Check the ARP table, routing table, and routing cache of each host. Save the output. Please note that these are the table entries from Step 4 before the ping is issued.
 - iv) Issue a ping command from PC3 to PC4 for at least three sends (-c 3) .
 - v) Save the output of the ping command and the output of Wireshark on PC3.
 - vi) Save the ARP table, routing table, and routing cache of PC3. Please note that these are the table entries from Step 4 after the ping commands are issued.
- 5. Repeat Step 4, but this time issues a ping from PC3 to PC2. Note that once an entry is made in the routing cache, you cannot repeat the previous experiment to obtain the same results. You have to wait until the routing cache is reset or you can delete all the routing caches on all devices.

III. ADDITIONAL EXERCISE

1. Configure DHCP for the below configuration



Introduction To NS2: Wired Network

Objectives

- To implement simple network scenario using TCL script in NS-2.
- To interpret different agents and their applications like FTP over TCP and CBR over UDP.

Prerequisite

- Basic idea about network topology, communication among nodes, events and traffic

I. Introduction

The Network Simulator -2 (NS - Version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkely written in C++ and OTcl targeted at networking Research. It implements network protocols such as TCP and UDP, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ and it also supports for simulation of multicast protocols over wired and wireless (local and satellite) networks. NS-2 can run under the environment of both UNIX and Windows operating systems. The user can choose to install it partly or completely, however many supporting components are desirable during installation for successful running of NS simulation. For beginners it is suggested to make a complete installation that automatically installs all necessary components at once and it requires about 320 MB disk space. With the higher degree of familiarization and expertise the user can go for partial installation of NS2, for the faster simulation.

A simplified user's view is depicted in Fig 10.1. NS2 is Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network setup (plumbing) module libraries. To setup and run a simulation network, a user should write an OTcl script that initiates an event scheduler, sets up the network topology using the network objects and the plumbing functions in the library informing traffic sources to start and stop transmitting packets through the event scheduler. Another major component of NS2 beside network objects is the event

scheduler. An event in NS is a packet ID that is unique for a packet with scheduled time and the pointer to an object that handles the event. In NS, an event scheduler keeps track of simulation time and fires all the events in the event queue scheduled for the current time by invoking appropriate network components by an Event Scheduler which initiate the appropriate action associated with packet pointed by the event.

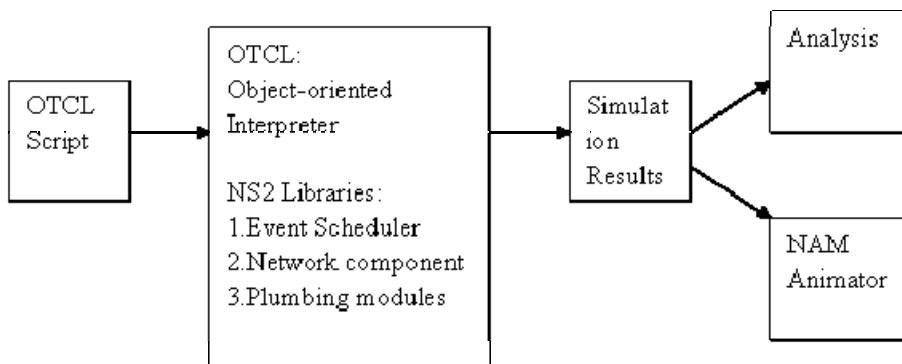


Figure 10.1: A simplified user's view

Network components communicate with one another passing packets, however this does not consume actual simulation time. All the network components that need to spend some simulation time handling a packet (i.e. need a delay) use the event scheduler by issuing an event for the packet and waiting for the event to be fired to itself before doing further action handling the packet. For example, TCP needs a timer to keep track of a packet transmission time out for retransmission (transmission of a packet with the same TCP packet number but different NS packet ID). Timers use event schedulers in a similar manner that delay does. The only difference is that timer measures a time value associated with a packet and does an appropriate action related to that packet after a certain time goes by, and does not simulate a delay.

NS is written not only in OTcl but in C++ also. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time (not simulation time), the event scheduler and the basic network component objects in the data path are written and compiled using C++. These compiled objects are made available to the OTcl interpreter through an OTcl linkage that creates a matching OTcl object for each of the C++ objects and makes the control functions and the configurable variables specified by the C++ object act as member functions and member variables of the corresponding OTcl object. Fig. 10.2 shows an

object hierarchy example in C++ and OTcl. One thing to note in the figure is that for C++ objects that have an OTcl linkage forming a hierarchy, there is a matching OTcl object hierarchy very similar to that of C++.

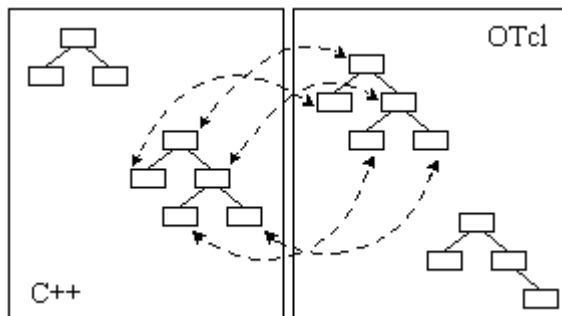


Figure 10.2: C++ and OTcl: The Duality

The reason why NS2 uses two languages is that different tasks have different requirements: For example, simulation of protocols requires efficient manipulation of bytes and packet headers making the run-time speed very important. On the other hand, in network studies where the aim is to vary some parameters and to quickly examine a number of scenarios the time to change the model and run it again is more important. C++ is used for detailed protocol implementation and in cases where every packet of a flow has to be processed. Otcl, on the other hand, is suitable for configuration and setup. Otcl runs quite slowly, but it can be changed very quickly making the construction of simulations easier.

10.2 Starting NS

To start with ns type the command 'ns <tclscript>' (assuming that you are in the directory with the ns executable, or that your path points to that directory), where '<tclscript>' is the name of a Tcl script file which defines the simulation scenario (i.e. the topology and the events).

Everything else depends on the Tcl script. The script might create some output on stdout, it might write a trace file or it might start nam to visualize the simulation. Or all of the above.

Starting nam

You can either start nam with the command 'nam <nam-file>' where '<nam-file>' is the name of a nam trace file that was generated by ns, or you can execute it directly out of the Tcl simulation script for the simulation which you want to visualize. Following Fig. 10.3 shows a screenshot of nam window where the most important functions are being explained.

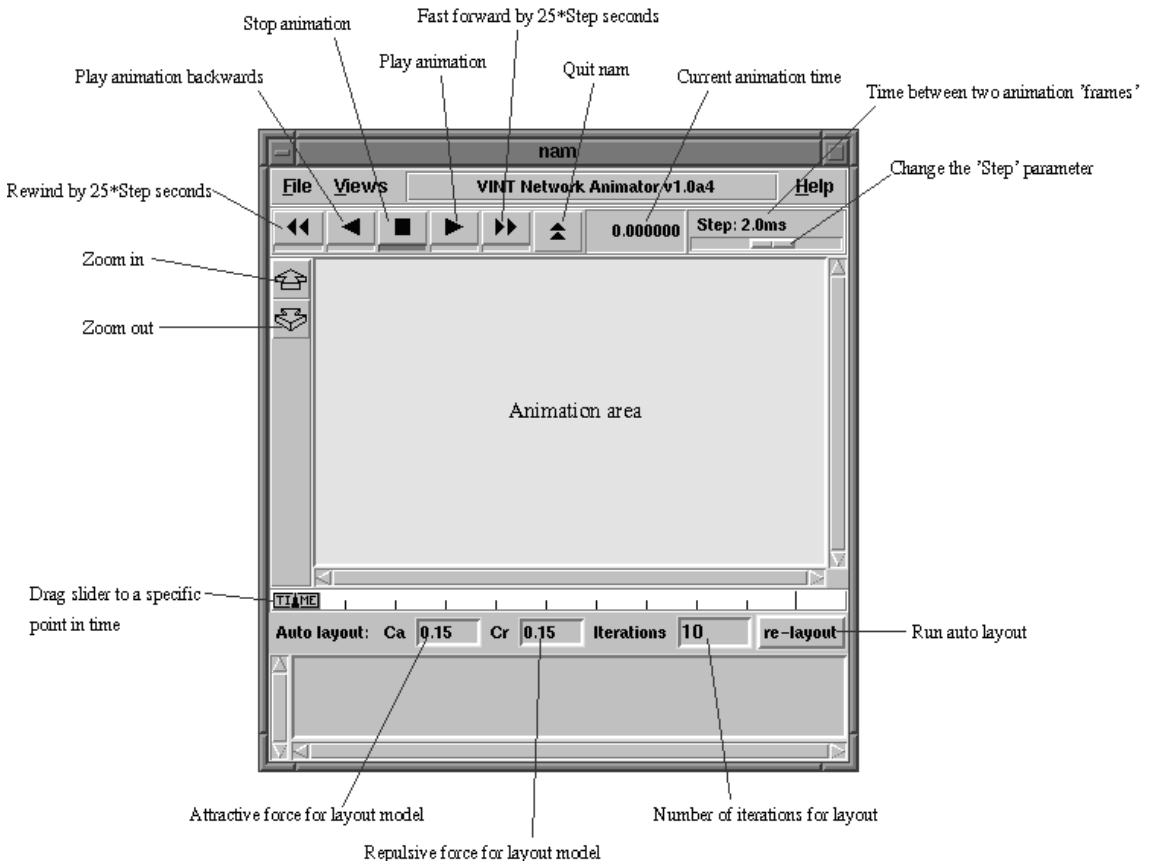


Figure 10.3: Screenshot of sample nam window

10.3 Writing a simple Tcl script

We can write Tcl scripts in any text editor. Let the first example name be 'example1.tcl'.

First step is to create a simulator object. This is done with the command

```
set ns [new Simulator]
```

Next step is to open a file for writing that is going to be used for the nam trace data.

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

The first line opens the file 'out.nam' for writing and gives it the file handle 'nf'. In the second line we tell the simulator object that we created above to write all simulation data that is going to be relevant for nam into this file.

The next step is to add a 'finish' procedure that closes the trace file and starts nam.

```
proc finish {} {
    global ns nf
    $ns flush-trace
    close $nf
    exec nam out.nam &
    exit 0
}
```

The next line tells the simulator object to execute the 'finish' procedure after 5.0 seconds of simulation time.

```
$ns at 5.0 "finish"
```


The last line finally starts the simulation.

```
$ns run
```

We can actually save the file now and try to run it with 'ns example1.tcl'. We are going to get an error message like 'nam: empty trace file out.nam' though, because until now we haven't defined any objects (nodes, links, etc.) or events.

Two nodes, one link

In this section we are going to define a very simple topology with two nodes that are connected by a link. The following two lines define the two nodes. (Note: You have to insert the code in this section **before** the line '\$ns run', or even better, before the line '\$ns at 5.0 "finish"').

```
set n0 [$ns node]  
set n1 [$ns node]
```

A new node object is created with the command '\$ns node'. The above code creates two nodes and assigns them to the handles 'n0' and 'n1'.

The next line connects the two nodes.

```
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
```

This line tells the simulator object to connect the nodes n0 and n1 with a duplex link with the bandwidth 1Megabit, a propagation delay of 10ms and a DropTail queue.

Now we can save the file and start the script with 'ns example1.tcl'. nam will be started automatically and we would see an output that resembles the Fig. 10.4 below.

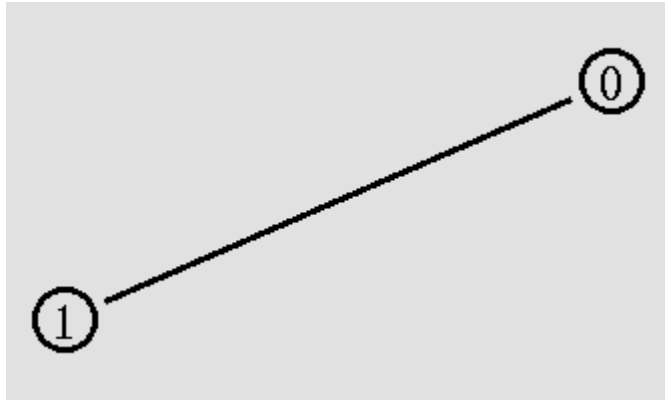


Figure 10.4: Sample nam window having 2 nodes

10.4 Sending data

Of course, this example isn't very satisfying yet, since you can only look at the topology, but nothing actually happens, so the next step is to send some data from node n0 to node n1. In ns, data is always being sent from one 'agent' to another. So the next step is to create an agent object that sends data from node n0, and another agent object that receives the data on node n1.

```
set cbr0 [new Agent/CBR]
$ns attach-agent $n0 $cbr0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
```

Here first two lines create a CBR agent and attach it to the node n0. CBR stands for 'constant bit rate'. The third and the fourth line should be self-explaining. The packetSize is being set to 500 bytes and a packet will be sent every 0.005 seconds (i.e. 200 packets per second). Like this we can also find other relevant parameters for

each agent type. The next lines create a Null agent which acts as traffic sink and attach it to node n1.

```
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
```

Now the two agents have to be connected with each other.

```
$ns connect $cbr0 $null0
```

And now we have to tell the CBR agent when to send data and when to stop sending. Note: It's probably best to put the following lines just before the line '\$ns at 5.0 "finish"'.

```
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
```

This code should be self-explaining again.

Now we can save the file and start the simulation again. When we click on the 'play' button in the nam window, we will see that after 0.5 simulation seconds, node 0 starts sending data packets to node 1 as shown in Fig. 10.5.

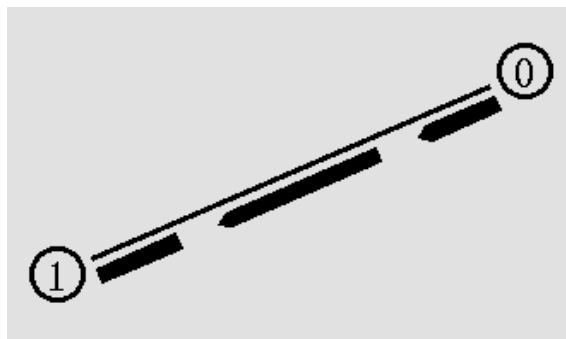


Figure 10.5: Transmission of data packets from node 0 to node 1

Now we can start some experiments with nam and the Tcl script. Click on any packet in the nam window to monitor it, and you can also click directly on the link to get some graphs with statistics. Also observe the changes by making changes in 'packetize_' and 'interval_' parameters in the Tcl script and see what happens.

10.5 Defining topology

Now insert the following lines into the code to create four nodes.

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

The following piece of Tcl code creates three duplex links between the nodes.

```
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail
```

Add the next three lines to your Tcl script and start it again.

```
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
```

We will probably understand what this code does when you look at the topology in the nam window now. It should look like the Fig. 10.6.

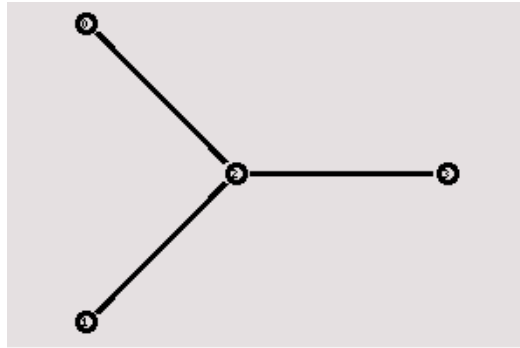


Figure 10.6: Network topology with 4 nodes in a predefined layout

Note that the autolayout related parts of nam are gone, since now we have taken the layout into our own hands. The options for the orientation of a link are right, left, up, down and combinations of these orientations.

10.6 The Events

Now we create two CBR agents as traffic sources and attach them to the nodes n0 and n1. Then we create a Null agent and attach it to node n3.

```
set cbr0 [new Agent/CBR]
$ns attach-agent $n0 $cbr0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005

set cbr1 [new Agent/CBR]
$ns attach-agent $n1 $cbr1
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005

set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

The two CBR agents have to be connected to the Null agent.

```
$ns connect $cbr0 $null0  
$ns connect $cbr1 $null0
```

We want the first CBR agent to start sending at 0.5 seconds and to stop at 4.5 seconds while the second CBR agent starts at 1.0 seconds and stops at 4.0 seconds.

```
$ns at 0.5 "$cbr0 start"  
$ns at 1.0 "$cbr1 start"  
$ns at 4.0 "$cbr1 stop"  
$ns at 4.5 "$cbr0 stop"
```

When the above script is executed, we can will notice that there is more traffic on the links from n0 to n2 and n1 to n2 than the link from n2 to n3 can carry. A simple calculation confirms this: We are sending 200 packets per second on each of the first two links and the packet size is 500 bytes. This results in a bandwidth of 0.8 megabits per second for the links from n0 to n2 and from n1 to n2. That's a total bandwidth of 1.6Mb/s, but the link between n2 and n3 only has a capacity of 1Mb/s, so obviously some packets are being discarded. But which ones? Both flows are black, so the only way to find out what is happening to the packets is to monitor them in nam by clicking on them.

10.7 Marking flows

Add the following two lines to above CBR agent definitions.

```
$cbr0 set fid_ 1  
$cbr1 set fid_ 2
```

The parameter 'fid_' stands for 'flow id'.

Now add the following piece of code to your Tcl script, preferably at the beginning after the simulator object has been created, since this is a part of the simulator setup.

```
$ns color 1 Blue  
$ns color 2 Red
```

This code allows us to set different colors for each flow id as shown in Fig. 10.7.

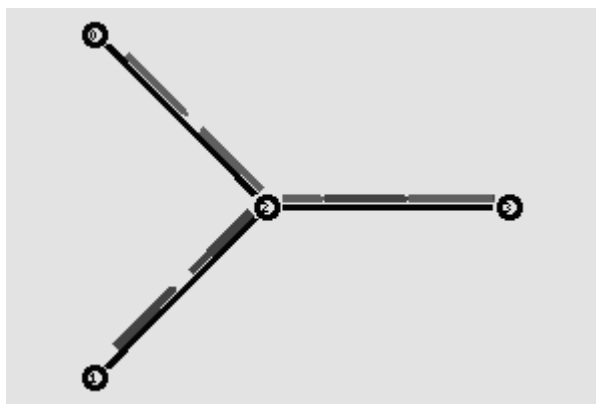


Figure 10.7: Network topology with 4 nodes with packet flow

Now you can start the script again and one flow should be blue, while the other one is red. Watch the link from node n2 to n3 for a while, and we would notice that after some time the distribution between blue and red packets isn't too fair anymore

10.8 Monitoring a queue

Add the following line to your code to monitor the queue for the link from n2 to n3.

```
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

Start ns again and we will see a picture similar to the one shown in Fig. 10.8 after a few moments.

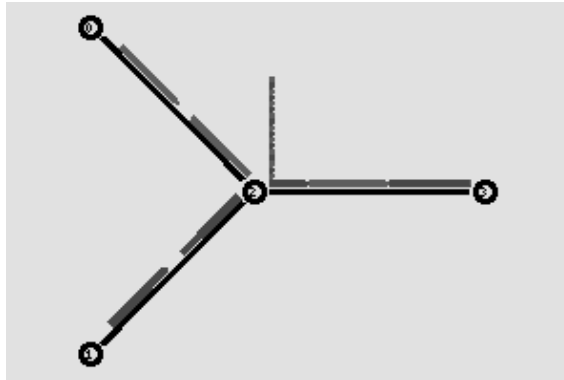


Figure 10.8: Network topology with 4 nodes with packet queue

We can see the packets in the queue now, and after a while we can even see how the packets are being dropped. But you can't really expect too much 'fairness' from a simple DropTail queue. So let's try to improve the queueing by using a SFQ (stochastic fair queueing) queue for the link from n2 to n3. Change the link definition for the link between n2 and n3 to the following line.

```
$ns duplex-link $n3 $n2 1Mb 10ms SFQ
```

The queueing should be 'fair' now. The same amount of blue and red packets should be dropped as shown in Fig. 10.9.

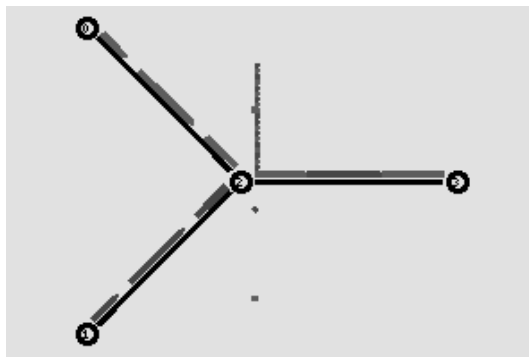


Figure 10.9: Network topology with 4 nodes with packet drop

Steps to create and execute Tcl script

1. Open a text editor and write the codes there and save it with .tcl file extension
gedit filename.tcl
2. To execute the file, open terminal and navigate to the folder where the saved file is present and run:
ns filename
3. Step 2 automatically creates the .nam and .tr files in the same folder. Optionally, to execute the nam file following command can be used in the terminal:
nam filename.nam

10.9 The algorithm for writing a complete Tcl script

Begin:

1. Create instance of network simulator object
2. Turn on Tracing and NAM
3. Define a 'Finish' procedure – Here the instruction to open NAM has to be included
4. Create network
 - i) Create nodes
 - ii) Assign node position (NAM)
 - iii) Create link between nodes (Simplex/Duplex)
 - iv) Define colors for data flow (NAM)
 - v) Set the Queue size for node (Optional)
 - vi) Monitor the queue (NAM)
 - vii) Attach agent for every node - Transport Connection (TCP/UDP)
 - viii) Set up Traffic Application (FTP/CBR)
5. Schedule the events
6. Call finish procedure
7. Print the necessary output data
8. Run the Simulation
9. Visualize using NAM or analyze the trace file

End

II. SOLVED EXERCISE

Sample Tcl script:

Step 1. define global simulator object

```
set ns [new Simulator]
```

Step 2. Define different colors for data flows (for NAM)

```
$ns color 1 green
```

```
$ns color 2 Red
```

Step 3. Opening the NAM trace file

```
set nt [open simulate.nam w]
```

```
$ns namtrace-all $nt
```

Step 4. Opening the Trace file

```
set tr [open simulate.tr w]
```

```
$sn trace-all $tr
```

Step 5. Define a 'finish' procedure

```
proc finish {} {  
    global ns nt tr  
    $ns flush-trace  
    #Close the NAM trace file  
    close $nt  
    close $tr  
    exec nam simulate.nam &  
    exit 0  
}
```

Step 6. Creation of six nodes

```
set n0 [$ns node]
```

```
set n1 [$ns node]
```

```
set n2 [$ns node]
```

```
set n3 [$ns node]
```

```
set n4 [$ns node]
```

```
set n5 [$ns node]
```

Step 7. Creating links between the nodes

```
$sn duplex-link $n0 $n2 2Mb 10ms DropTail
$sn duplex-link $n1 $n2 2Mb 10ms DropTail
$sn duplex-link $n2 $n3 1.7Mb 20ms DropTail
$sn duplex-link $n3 $n4 2Mb 10ms DropTail
$sn duplex-link $n3 $n5 2Mb 10ms DropTail
```

Step8. Setting Queue Size of link (n2-n3) to 10

```
$sn queue-limit $n2 $n3 10
```

#Step9. Provide node positions to visualize in NAM window

```
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n4 $n5 orient right-down
```

#Step 10. Monitoring the queue for link (n2-n3) (n3-n4). (for NAM)

```
$sn duplex-link-op $n2 $n3 queuePos 0.5
```

#Step 11. Setting up a TCP connection

```
set tcp [new Agent/TCP]
$tcp set class_ 1
$ns attach-agent $na $tcp
```

#Step 12. If we setup tcp traffic source then connect it with tcp sink

```
set sink [new Agent/TCPSink]
$ns attach-agent $ne $sink
$ns connect $tcp $sink
$tcp set fid_ 2
```

#Step 13. Setting up a FTP over TCP connection

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
```

#Step 14. Setup a UDP connection

```
set udp [new Agent/UDP]  
$sn attach-agent $nb $udp
```

#Step 15. If we setup udp traffic source then connect it with null

```
set null [new Agent/Null]  
$ns attach-agent $nf $null  
$ns connect $udp $null  
$udp set fid_ 1
```

#Step 16. Setting up a CBR over UDP connection

```
set cbr [new Application/Traffic/CBR]  
$cbr attach-agent $udp  
$cbr set type_ CBR  
$cbr set packet_size_ 1000  
$cbr set rate_ 1mb  
$cbr set random_ false
```

#Step 17. Scheduling events for the CBR and FTP agents

```
$sn at 0.1 "$cbr start"  
$sn at 1.0 "$ftp start"  
$sn at 4.0 "$ftp stop"  
$sn at 4.5 "$cbr stop"
```

```
$ns at 5.0 "finish"
```

#Step 18. Run the simulation

```
$ns run
```

OUTPUT:

(i) Nam file:

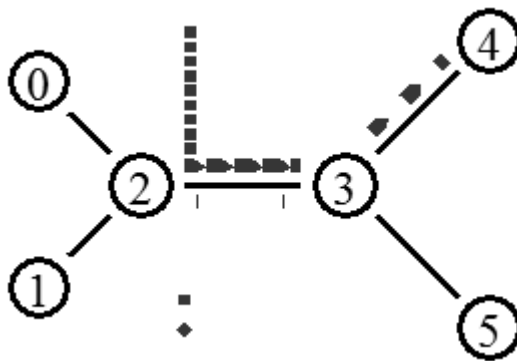


Figure 10.10: Nam output

(ii) Trace file:

```
File Edit Format View Help
t 0.1 1 2 cbr 1000 ----- 2 1.0 3.1 0 0
- 0.1 1 2 cbr 1000 ----- 2 1.0 3.1 0 0
+ 0.108 1 2 cbr 1000 ----- 2 1.0 3.1 1 1
- 0.108 1 2 cbr 1000 ----- 2 1.0 3.1 1 1
r 0.114 1 2 cbr 1000 ----- 2 1.0 3.1 0 0
+ 0.114 2 3 cbr 1000 ----- 2 1.0 3.1 0 0
- 0.114 2 3 cbr 1000 ----- 2 1.0 3.1 0 0
+ 0.116 1 2 cbr 1000 ----- 2 1.0 3.1 2 2
- 0.116 1 2 cbr 1000 ----- 2 1.0 3.1 2 2
r 0.122 1 2 cbr 1000 ----- 2 1.0 3.1 1 1
+ 0.122 2 3 cbr 1000 ----- 2 1.0 3.1 1 1
- 0.122 2 3 cbr 1000 ----- 2 1.0 3.1 1 1
+ 0.124 1 2 cbr 1000 ----- 2 1.0 3.1 3 3
- 0.124 1 2 cbr 1000 ----- 2 1.0 3.1 3 3
r 0.13 1 2 cbr 1000 ----- 2 1.0 3.1 2 2
+ 0.13 2 3 cbr 1000 ----- 2 1.0 3.1 2 2
- 0.13 2 3 cbr 1000 ----- 2 1.0 3.1 2 2
+ 0.132 1 2 cbr 1000 ----- 2 1.0 3.1 4 4
- 0.132 1 2 cbr 1000 ----- 2 1.0 3.1 4 4
r 0.138 1 2 cbr 1000 ----- 2 1.0 3.1 3 3
+ 0.138 2 3 cbr 1000 ----- 2 1.0 3.1 3 3
- 0.138 2 3 cbr 1000 ----- 2 1.0 3.1 3 3
r 0.138706 2 3 cbr 1000 ----- 2 1.0 3.1 0 0
+ 0.14 1 2 cbr 1000 ----- 2 1.0 3.1 5 5
- 0.14 1 2 cbr 1000 ----- 2 1.0 3.1 5 5
r 0.146 1 2 cbr 1000 ----- 2 1.0 3.1 4 4
+ 0.146 2 3 cbr 1000 ----- 2 1.0 3.1 4 4
- 0.146 2 3 cbr 1000 ----- 2 1.0 3.1 4 4
r 0.146706 2 3 cbr 1000 ----- 2 1.0 3.1 1 1
+ 0.148 1 2 cbr 1000 ----- 2 1.0 3.1 6 6
- 0.148 1 2 cbr 1000 ----- 2 1.0 3.1 6 6
r 0.154 1 2 cbr 1000 ----- 2 1.0 3.1 5 5
+ 0.154 2 3 cbr 1000 ----- 2 1.0 3.1 5 5
- 0.154 2 3 cbr 1000 ----- 2 1.0 3.1 5 5
```

Figure 10.11: Trace file

III. LAB EXERCISES

1. Set up a simple wired network with two nodes n0 and n1 for UDP_CBR traffic. Provide duplex link between these nodes having propagation delay of 10msec and capacity of 10Mb by considering n0 as source node and n1 as sink node.

2. Set up a simple wired network with two nodes n0 and n1 for TCP_FTP traffic. Provide duplex link between these nodes having propagation delay of 10msec and capacity of 10Mb by considering n0 as source node and n1 as sink node.
3. Set up the LAN network with 4 nodes, duplex-link node N1 to node N2 is 2Mb capacity and 20ms delay, duplex-link node N3 to node N4 is 1Mb capacity and 10ms delay. N1(source) and N2(sink), N3(source) and N4(sink). Change the color of the flow between the nodes. Duration of the simulation is 5 seconds.

IV. ADDITIONAL EXERCISE

1. Consider a network with seven nodes n0, n1, n2, n3, n4, n5 and n6 forming a circular topology. n0 is a TCP source, which transmits packets to node n6 (a TCP sink) through the node n5. Node n1 is another traffic source, and sends UDP packets to node n3 through n2. The duration of the simulation time is 10 seconds. Write a TCL script to simulate this scenario.

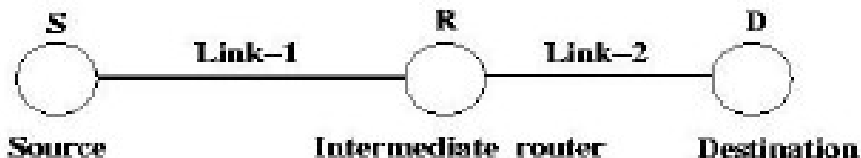
Measuring Performance of Protocols with NS2

Objectives:

- To understand performance of protocols on Network

I. LAB EXERCISE

1. Metric Mahadeva wants to understand performance of protocols on networks. Since this is his first foray in this area, he decides to keep the topology and protocol simple. His smart friend Raju Topiwala gave him the following topology to use and suggested that he use UDP since its much simpler than TCP. UDP just adds sequence numbers to packets and pretty much does nothing more (as far as this experiment is concerned).



There is a source node, a destination node, and an intermediate router (marked "S", "D", and "R" respectively). The link between nodes S and R (Link-1) has a bandwidth of 1Mbps and 50ms latency. The link between nodes R and D (Link-2) has a bandwidth of 100kbps and 5ms latency.

Mahadeva wants to understand the following:

- If the source data rate (rate at which source injects data into the network) varies, what happens to the packets in the network?
- At what point does it get congested?
- How do the throughput and loss vary as a function of the source data rate.

Help him to write a ns2 tcl script (name it ns-udp.tcl) that helps him conduct this experiment. Help him also interpret the results of the experiment.

Guidance:

1. Vary the source data rate (termed Offered Load hence forth) to take on values of 40kbps, 80 kbps, 120kbps and 160kbps. Note each value will result in one run. Thus you have 4 runs and hence 4 trace files. Ensure proper naming of the trace files (Eg: udp-40k.tr, udp-80k.tr, udp-120k.tr and udp-160k.tr). Include a tcl file of one of the runs as ns-udp.tcl in the directory.
2. Routers have limited buffer space and drop packets during congestion. Use the “queue-limit” command to model this. Limit the queue at the bottleneck link (link-2) between node "R" and node "D" to 10 packets. It is fun to monitor the queue at Link-2 (between R and D). You can do so using the duplex-link-op command.
3. Let each experiment run for at least 10sec.
4. **Metrics:** For each run, calculate/measure the following metrics by processing the output trace file.
 - a) Offered Load: The rate at which source traffic is being injected into the network. (You set this value in the tcl script but confirm via trace that you got it correct.)
 - b) Packet Loss: The number of packets that were sent by the sender, but didn't reach the destination. Express it in percentage.
 - c) Throughput: The rate at which bits are being received at the destination? Eg: If 100 packets of size 100 bytes were received in a duration of 100 seconds, we say the throughput is $100 * 100 * 8 / 100 = 800\text{bps}$ or 0.8kbps. Express it in kbps.

II. LAB REPORT

Plot the following graphs. Ensure the axis are properly labeled, with proper legend and correct units.

1. Offered Load vs percentage packet loss
2. Offered load vs throughput

In the report, comment on what you observe in each graph and the reasons for the same.

III. ADDITIONAL EXERCISE

This does involve bash scripting or using C code. This is a continuation of the previous exercise. If the source data rate (rate at which source injects data into the network) varies, how does the delay vary as a function of the source data rate. That is plot the Offered Load vs Average end-to-end delay.

Average end-to-end delay for a run: If t_1 is the time the packet was generated at the source and t_2 was the time the packet was received at the destination. Delay of a packet is $t_2 - t_1$. Calculate average of these values for packets that reached the destination. Express it in ms. Then plot the Offered Load vs Average end-to-end delay for the different runs.

Design of VLANs Using GNS3

Objectives:

- To understand Virtual Lan(VLAN) Concepts

We can solve many of the problems associated with layer 2 switching with VLANs. VLANs work like this: Figure 12.1 shows all hosts in this very small company connected to one switch, meaning all hosts will receive all frames, which is the default behavior of all switches.

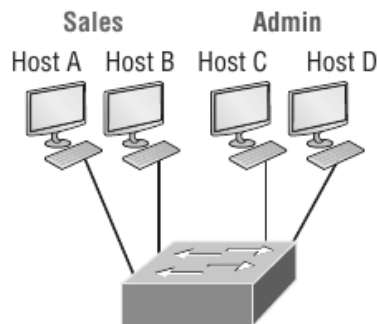


Figure 12.1: One switch, one LAN: Before VLANs, there were no separations between hosts

If we want to separate the host's data, we could either buy another switch or create virtual LANs, as shown in Figure 12.2

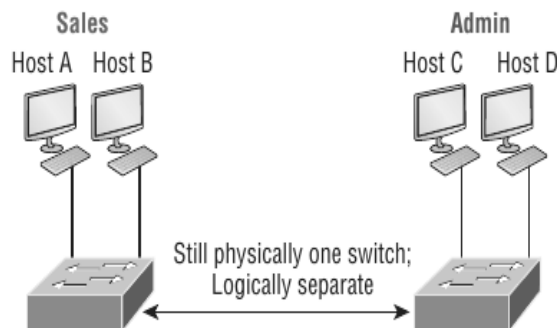


Figure 12.2: One switch, two virtual LANs (logical separation between hosts): Still physically one switch, but this switch acts as many separate devices

In Figure 12.2, we configured the switch to be two separate LANs, two subnets, two broadcast domains, two VLANs—they all mean the same thing—without buying another switch. We can do this 1,000 times on most Cisco switches, which saves thousands of Rupees and more!

There are two different types of ports in a switched environment. Let's take a look at the first type in Figure 12.3.

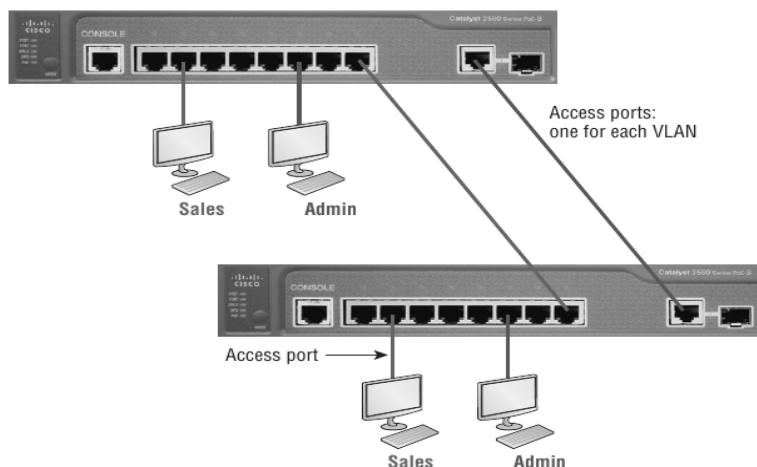


Figure 12.3: Access Ports

Notice there are access ports for each host and an access port between switches—one for each VLAN.

Access ports

An access port belongs to and carries the traffic of only one VLAN. Traffic is both received and sent in native formats with no VLAN information (tagging) whatsoever. Anything arriving on an access port is simply assumed to belong to the VLAN assigned to the port. Because an access port doesn't look at the source address, tagged traffic—a frame with added VLAN information—can be correctly forwarded and received only on trunk ports.

added VLAN information—can be correctly forwarded and received only on trunk ports.

With an access link, this can be referred to as the configured VLAN of the port. Any device attached to an access link is unaware of a VLAN membership—the device just assumes it's part of some broadcast domain. But it doesn't have the big picture, so it doesn't understand the physical network topology at all.

Another good bit of information to know is that switches remove any VLAN information from the frame before it's forwarded out to an access-link device. Remember that access-link devices can't communicate with devices outside their VLAN unless the packet is routed. Also, you can only create a switch port to be either an access port or a trunk port—not both. So you've got to choose one or the other and know that if you make it an access port, that port can be assigned to one VLAN only. In Figure 12.3, only the hosts in the Sales VLAN can talk to other hosts in the same VLAN. This is the same with Admin VLAN, and they can both communicate to hosts on the other switch because of an access link for each VLAN configured between switches.

Trunk ports

The term trunk port was inspired by the telephone system trunks, which carry multiple telephone conversations at a time. So it follows that trunk ports can similarly carry multiple VLANs at a time as well.

A trunk link is a 100, 1,000, or 10,000 Mbps point-to-point link between two switches, between a switch and router, or even between a switch and server, and it carries the traffic of multiple VLANs—from 1 to 4,094 VLANs at a time. But the amount is really only up to 1,001 unless you're going with something called extended VLANs.

Instead of an access link for each VLAN between switches, we'll create a trunk link demonstrated in Figure 12.4. Trunking can be a real advantage because with it, you get to make a single port part of a whole bunch of different VLANs at the same time. This is a great feature because you can actually set ports up to have a server in two separate broadcast domains simultaneously so your users won't have to cross a layer 3 device (router) to log in and access it.

Another benefit to trunking comes into play when you're connecting switches. Trunk links can carry the frames of various VLANs across them, but by default, if the links between your switches aren't trunked, only information from the configured access VLAN will be switched across that link.

It's also good to know that all VLANs send information on a trunked link unless you clear each VLAN by hand.

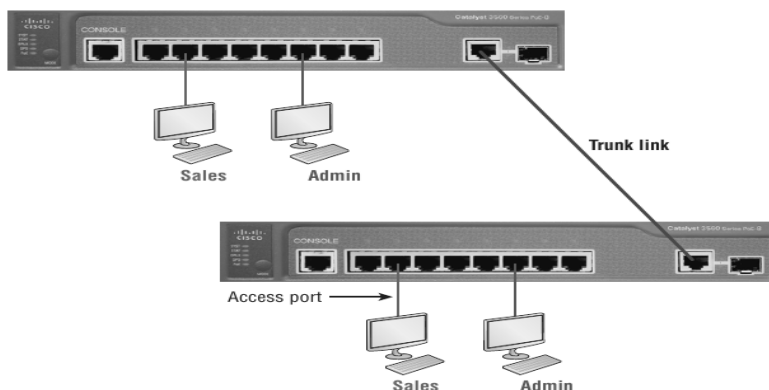


Figure 12.4: VLANs can span across multiple switches by using trunk links, which carry traffic for multiple VLANs

Frame Tagging

As you now know, you can set up your VLANs to span more than one connected switch. You can see that going on in Figure 12.4, which depicts hosts from two VLANs spread across two switches. This flexible, power-packed capability is probably the main advantage to implementing VLANs, and we can do this with up to a thousand VLANs and thousands upon thousands of hosts!

All this can get kind of complicated—even for a switch—so there needs to be a way for each one to keep track of all the users and frames as they travel the switch fabric. And this just happens to be where frame tagging enters the scene.

This frame identification method uniquely assigns a user-defined VLAN ID to each frame.

Here's how it works: Once within the switch fabric, each switch that the frame reaches must first identify the VLAN ID from the frame tag. It then finds out what to do with the frame by looking at the information in what's known as the filter table. If the frame reaches a switch that has another trunked link, the frame will be forwarded out of the trunk-link port.

Once the frame reaches an exit that's determined by the forward/filter table to be an access link matching the frame's VLAN ID, the switch will remove the VLAN identifier. This is so the destination device can receive the frames without being required to understand their VLAN identification information.

Another great thing about trunk ports is that they'll support tagged and untagged traffic simultaneously if you're using 802.1q trunking. The trunk port is assigned a default port VLAN ID (PVID) for a VLAN upon which all untagged traffic will travel. This VLAN is also called the native VLAN and is always VLAN 1 by default, but it can be changed to any VLAN number. Similarly, any untagged or tagged traffic with a NULL (unassigned) VLAN ID is assumed to belong to the VLAN with the port default PVID. Again, this would be VLAN 1 by default. A packet with a VLAN ID equal to the outgoing port native VLAN is sent untagged and can communicate to only hosts or devices in that same VLAN. All other VLAN traffic has to be sent with a VLAN tag to communicate within a particular VLAN that corresponds with that tag.

VLAN Identification Methods:

1. Inter-Switch Link (ISL)

Inter-Switch Link (ISL) is a way of explicitly tagging VLAN information onto an Ethernet frame. This tagging information allows VLANs to be multiplexed over a trunk link through an external encapsulation method. This allows the switch to identify the VLAN membership of a frame received over the trunked link.

2. IEEE 802.1q

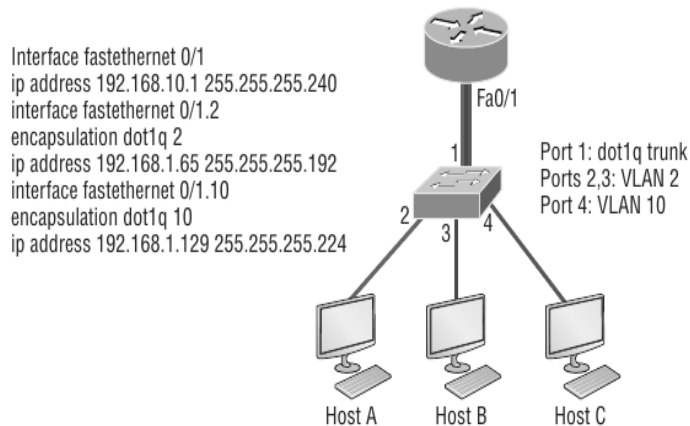
Created by the IEEE as a standard method of frame tagging, IEEE 802.1q actually inserts a field into the frame to identify the VLAN. If you're trunking between a Cisco switched link and a different brand of switch, you've got to use 802.1q for the trunk to work.

Unlike ISL, which encapsulates the frame with control information, 802.1q inserts an 802.1q field along with tag control information

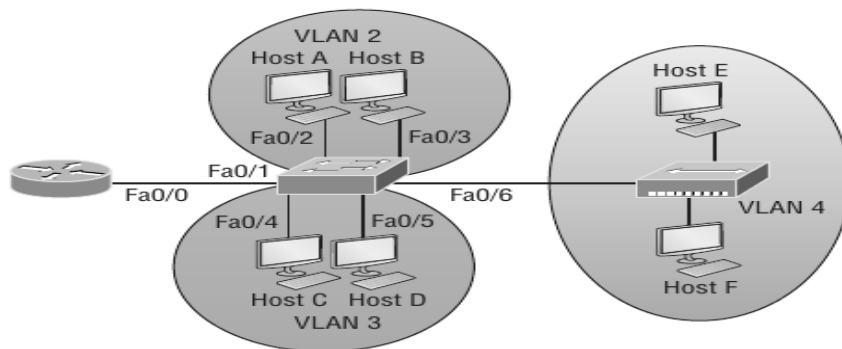
LAB EXERCISE

Configure following inter-VLAN example in GNS3 and verify the working using wireshark tool.

1.



2.



References:

1. W. Richard Stevens, “UNIX Network Programming, Volume 1: The Sockets Networking API”, Third Edition, Addison-Wesley Professional Computing, 2003.
2. Introduction and Reference Guide to Wireshark,
<https://thepracticalsysadmin.com/wireshark-reference-guide>.
3. Marc Greis tutorial for NS2, www.isi.edu/nsnam/ns/tutorial/, 2004.
4. Teerawat Issariyakul and Ekram Hossain. Introduction to Network Simulator NS2: Second Edition, Springer, 2011.
5. Jason C. Nuemann, “The Book of GN3”, No Starch Press, 2015.
6. GNS3 Documentation, <https://www.gns3.com>.

APPENDIX

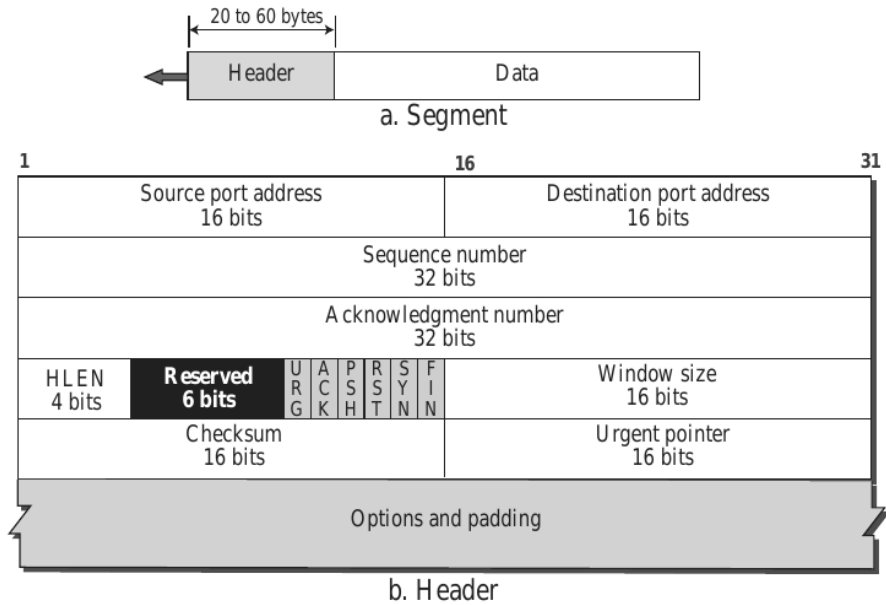


Figure 1: TCP Segment Format

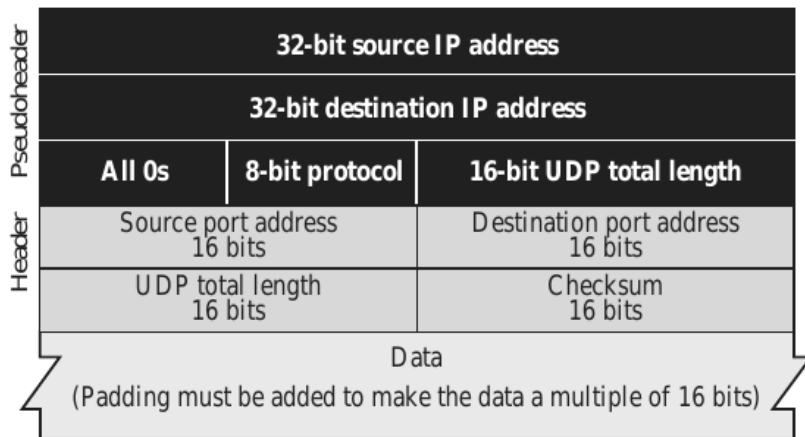


Figure 2: UDP Header and Pseudoheader

| Identification | Flags |
|--|---|
| Number of question records | Number of answer records (All 0s in query message) |
| Number of authoritative records (All 0s in query message) | Number of additional records (All 0s in query message) |

Figure 3: Format of DNS message

| | | | | | | | |
|----|--------|----|----|----|----|----------|-------|
| QR | OpCode | AA | TC | RD | RA | Three 0s | rCode |
|----|--------|----|----|----|----|----------|-------|

Figure 4: Flag fields of DNS message