Computer Networks Lab

Manual [CSE 3162]

CONTENTS

Course Objectives

- To understand network using GNS3 Simulation andNS2.
- 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):5marks
  - 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 of60.
- 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 wave forms 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.

LABNO: 1 **Date:**

Iterative TCP and UDP Programs and analysis using Wireshark

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

Prerequisites:

- Knowledge of the C programming language and Linux Networking APIs
- Knowledge of Basic Computer Networking
- Understanding of connection less service.

TCP 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 FROMSERVERIS...");
    while(n){
    n=read(sockfd,buf,sizeof(buf
    )); puts(buf);
}
}
```

- indicates replace this address with your systems IP address

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

LAB EXERCISES:

Q1.1) Write an *iterative* TCP client server 'C' program.
Rewrite the above solved program using **modular programming techniques using** following modules at least.
CreateClientSocket(): Creates a Socket and establishes connection to server at given IP address and port. (At client program)
CreateServerSocket(): Create a Socket and accepts connections from various clients and creates client specific socket for each.
PerformClientTask() : Worker function at client side. PerformServerTask() : Worker function at server side. TerminateSocket() ...etc with suitable arguments list.

Q1.2) 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.

Q1.3) 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 clie nt by echoing back the sentence in uppercase. The process continues until one of them types "QUIT".
Note: Use modular programming techniques and reuse suitable modules in all programs.

Q1.4) Write an iterative TCP client server 'C' program to illustrate encryption and decryption of messages usingTCP.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.

Q1.5) Write an iterative 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.

UDP 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 echois\n");
        puts(buf);
        return 0;
}
```

- indicates replace this address with your systems IP address

LAB EXERCISES:

Q1.6) Write a UDP client-server program where client sends rows of a matrix and the server combines them together as a matrix.

Q1.7) Write a Client program to send a manually crafted HTTP request packet to a Web Server and display all fields received in HTTP Response at client Side.

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.

Q1.8) 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 bytelane. 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.

Q1.9) 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?

Q1.10) 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?

Q1.11) What is the length of the first packet? How many bytes of payload are carried by the first packet?

LAB NO: 2 **Date:**

Concurrent Client-Server Programs and analysis using Wireshark

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 programing 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

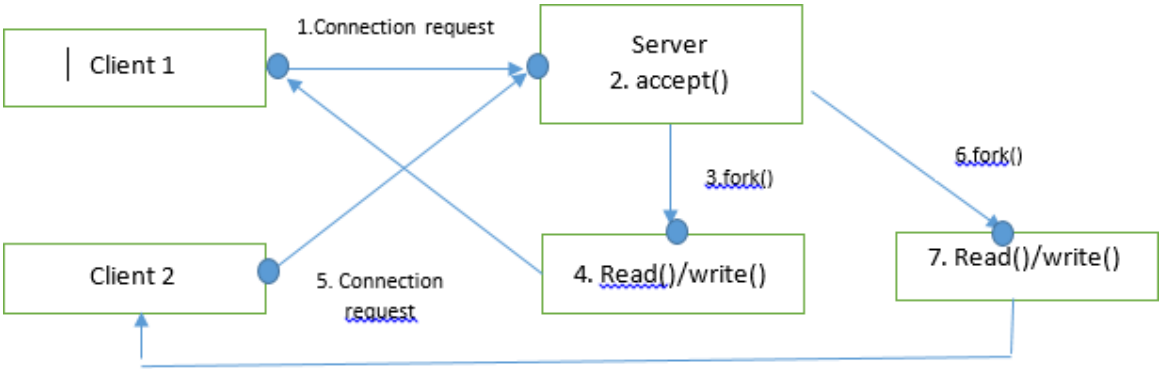


Figure 2.1 TCP concurrent server and clients interactions

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, butrare)
- 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
- voidFD_CLR(intfd,fd_set*fdset):clearelementsofthesetcorrespondingtothefile descriptor passed as fd
- voidFD_SET(intfd,fd_set*fdset):setelementsofthesetcorrespondingtothefile 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,structtimeval *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.

I. 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.

LAB EXERCISES

Q2.1) Rewrite above solved program using suitable modular programming techniques

Q2.2) 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.

Q2.3) 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 performs the operation on integers and sends result back to the client which is displayed on the client screen. Then both the processes terminate.

Q2.4) Write a concurrent TCP daytime server 'C' program. Along with the result sever should also send the process id to client.

Q2.5) 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.

Analyzing TCP packets using Wireshark:

1. Start your web browser and clear the browser's cache memory, but do not access any website yet.
2. Open Wireshark and start capturing.
3. Go back to your web browser and retrieve any file from a website. Wireshark starts capturing packets.
4. After enough packets have been captured, stop Wireshark and save the captured file.
5. 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.

Question:

Q2.6) 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.

Question:

Q2.7) 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 GETmessage)?
2. How many bytes are transmitted in thispacket?
3. How often the receiver generates an acknowledgment? To which acknowledgment rule 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

Q2.8) 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?

Q2.9) From the captured information, answer the following question in your lab report. Using the hexdump, determine the following for any TCP packet:

1. 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.
2. Using the information in the detail pane lane, verify your answers is question1.
3. Does any of the TCP packet header carry options? Explain your answer.
4. What is the size of aTCP packet with no options. What is the size of a TCP packet with options?
5. Is window size in any of the TCP packet zero? Explain your answer.
6. Analyze Interaction between your TCP Client-Server Programs using wireshark

Q2.10) Analyze Interaction between your UDP Client-Server Programs using wireshark

LAB NO: 3 **Date:**

Study of Application Layer Protocols using Wireshark.

Objectives:

- To understand application layer functionality and protocols
- Know the meaning of process-to-process communication

Prerequisites:

- Brief idea about Application layer in Network Software Architecture and its protocols
- Working Knowledge of Wireshark, traceroute, ping tools to capture and investigate some application layer protocols like HTTP, FTP, TELNET,SSH, SMTP, POP3, and DNS.

Lab exercises:

Q3.1) Retrieve web pages using HTTP. Use Wireshark to capture packets for analysis. Learn about most common HTTP messages . Also capture response messages and analyze them. During the lab session, also examine and analyze some HTTP headers.

Q3.2) Use FTP to transfer some files, Use Wireshark to capture some packets. Show that FTP uses two separate connections: a control connection and a data-transfer connection. The data connection is opened and closed for each file transfer activity. Also show that FTP is an insecure file transfer protocol because the transaction is done in plaintext.

Q3.3) Use Wireshark to capture packets exchanged by the TELNET protocol. As in FTP, observe commands and responses during the session in the captured packets. Also show that TELNET is vulnerable to hacking because it send all data including the password in plaintext .

Q3.4) Use Wireshark to capture packets exchanged by the SSH protocol. As in TELNET, observe commands and responses during the session in the captured packets. Also show that SSH is not vulnerable to hacking because it sends all data encrypted including the password .

Q3.5) Investigate SMTP protocol in action. Sent an e-mail and, using Wireshark investigate the contents and format of the SMTP packet exchanged between client and the server. Explain the three phases in this SMTP session .

Q3.6) Investigate the state and behavior of the POP3 protocol. Retrieve the mails stored in your mailbox at the POP3 server can observe and analyze the states of the POP3 and the type and the contents of the messages exchanged, by analyzing the packets through Wireshark.

Q3.7) Analyze the behavior of the DNS protocol. In addition to Wireshark, several network utilities are available for finding some information stored in the DNS servers. Use dig utilities (which has replaced nslookup). Set Wireshark to capture the packets sent by this utility.

LAB No: 4 **Date:**

STUDY OF NETWORK DEVICES IN GNS3

Objectives:

- Study of network simulatorGNS3.
- To Learn Static IP address Assignment.
- To study the characteristics of HUB device.
- To study the functions of ROUTER device
- To study the functions of SWITCH device

Prerequisites

- Knowledge of ARP Protocol
- Network Mask

LAB EXERCISES:

Q4.1. Design network configuration shown in Figure 4.1 for all parts. Connect all four VMs to a single Ethernet segment via a single hub as shown in Figure 4.1. Configure the IP addressesfor the PCs as shown inTable4.1.

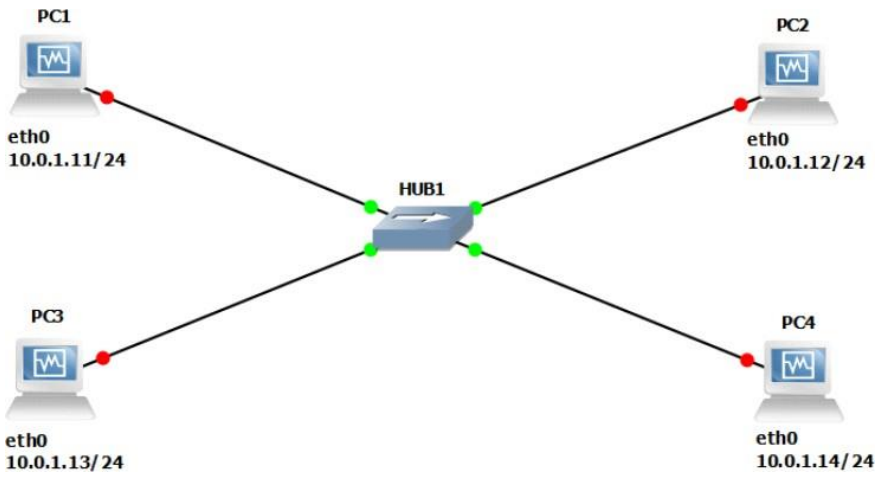


Figure 4.1: Network Design

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 4.1: IP Address of PCs

a. On PC1, view theARPcache with showarp

b. StartWiresharkon PC1-Hub1 link with a capture filter set to the IP address ofPC2.

c. Issue a ping command from PC1toPC2: **PC1% ping 10.0.1.13 –c3**

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 (~2minutes).

show arp

e. Save the results of Wireshark.

1. 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 4.2 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 4.2: 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 address10.0.1.11.

c. From PC3, issue a ping command to the duplicate IP address, 10.0.1.11, bytyping

PC3% ping 10.0.1.11 –c 5

d. Stop Wireshark, save all ARP packets and screenshot the ARPcache of PC3using 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 asgiven inTable6.1.

Q4.2) To test the effects of changing the netmask of a network configuration. Design the configuration as Q4.1 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

- Run Wireshark on PC1-Hub1 link and capture the packets for the following scenarios

i. From PC1 ping PC3.

ii. From PC1 pingPC2.

iii. From PC1 pingPC4.

iv. From PC4 pingPC1.

v. From PC2 ping PC4.
vi. From PC2 ping PC3.

- 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.

- When you are done with the exercise, reset the interfaces to their original values as

  givenTable4.1. (Note that /24 corresponds to network mask 255.255.255.0. and /28 to

  network mask 255.255.255.240).

Q4.3) EXERCISES Based On Q4.1

• 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 address10.0.1.12.

Q4.4) Based On Q4.2

• 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.

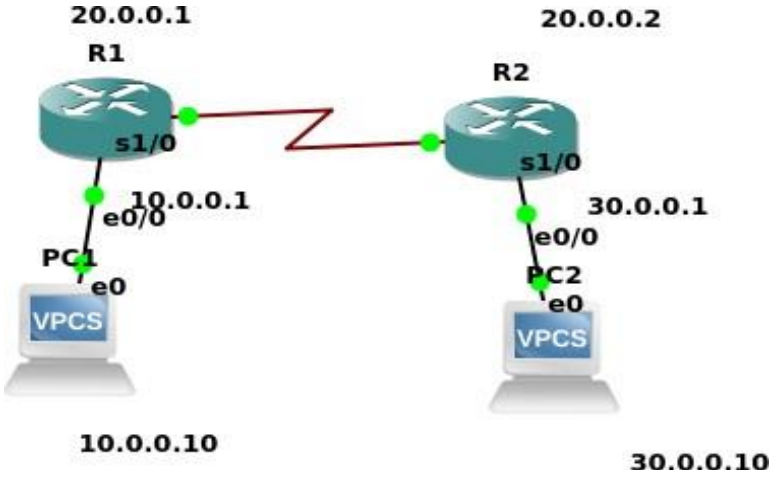
Q4.5) Based On Q4.3

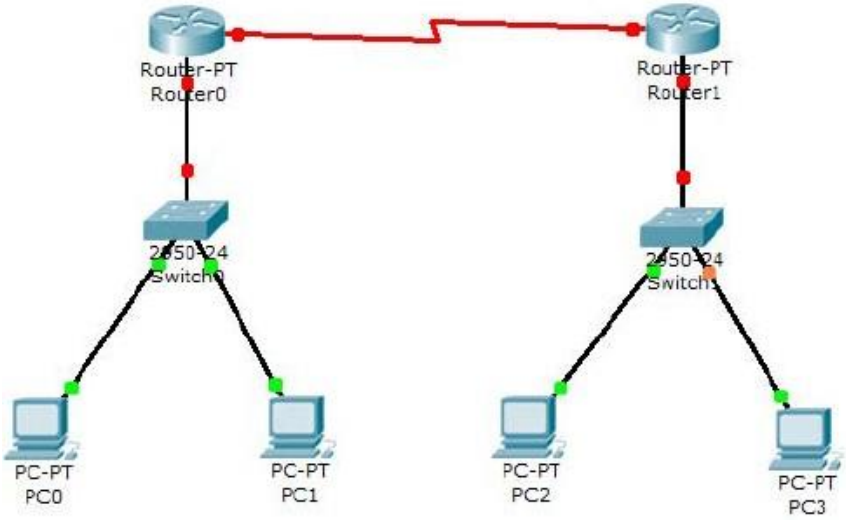
• Use your output data and ping results to explain what happened in each of the ping commands.

• Which ping operations were successful and which were unsuccessful? Why?

Q4.6) Configure the below network topology as shown in Figures below.

i. check the connectivity by pinging from PC1 toPC2.
ii. Analyse ARP exchanges between various network components.





LAB No 5 **Date:**

Study of Subnetting and Supernetting using GNS3

Objectives:

- **Understanding significance of Netmask value**
- **Study of Subnetting/Supernetting**

NETWORK PREFIXES and ROUTING

Q5.1) 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 5.1.

The network includes one router, four hosts and two hubs. The IP addresses of all devices are given in Figure 5.2. 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.

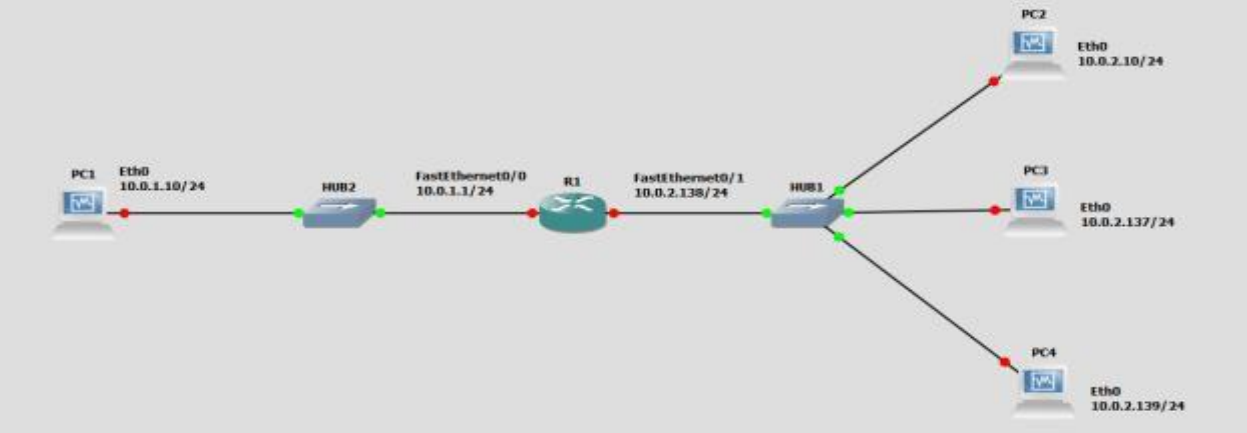


Figure 5.1: Network topology

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

Figure 5.2

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.

(1) Configure the hosts and the router to conform to the topology shown in Figure 5.2, using the IP addresses as given in Figure 5.2 . Note that PC2, PC3, and PC4 have different netmasks.

(2) Add Router1 as default gateway on all hosts. (PC1, PC2, PC3, andPC4).

(3) Issue *ping* commands from PC1

i) Clear the ARP table on all PCs.

ii) 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 (-c2).

iv) Save the output of the ping command at PC1 and the output of Wiresharkon PC1 and PC3.

(4) 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.

(5) Issue *ping* commands from PC3 toPC4

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.

- Issue a ping command from PC3 to PC4 for at least three sends (-c 3). Save the output of the ping command and the output of Wireshark on PC3. 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.

- 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.

Lab 6 Mid-Semester Mini Project Exam

LAB No 7 **Date:**

Study of DHCP/DNS using GNS3

Objectives:

- Understand DHCP Service
- Analyzing DHCP Packets
- To illustrate the significance of Domain Name Server

To Study the information exchanged between DNS and Clients

Introduction

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.

LAB EXERCISES

Q7.1) 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:

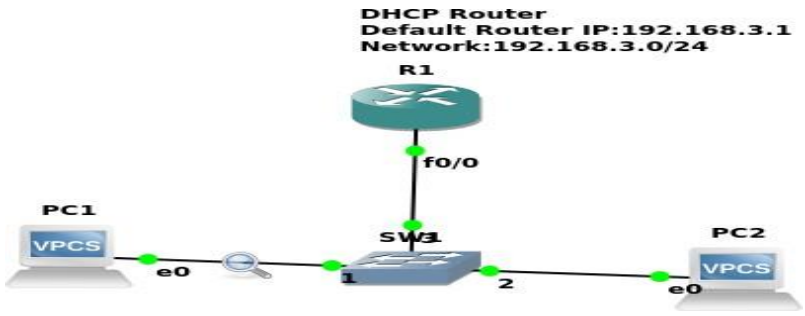


Figure 7.1 :Network Topology for DHCP Configuration

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.

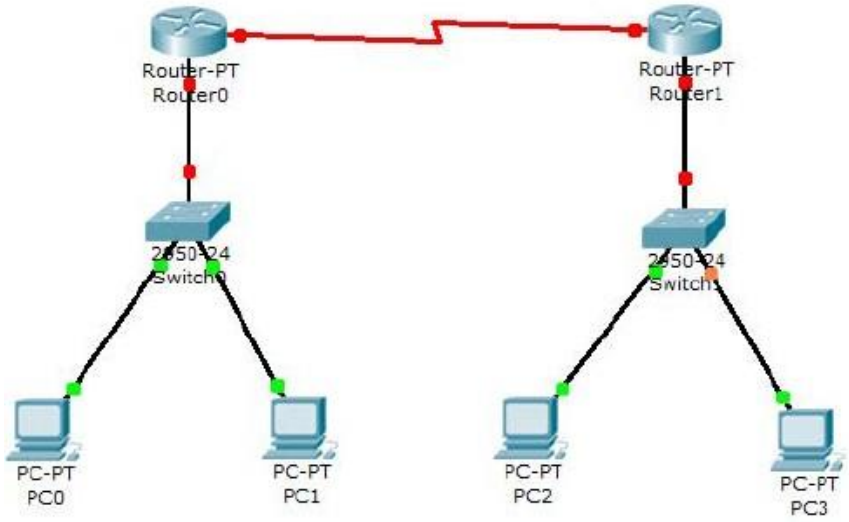
8	8.50813900	0.0.0.0	255.255.255.255	DHCP	342	DHCP	Discover - Transaction ID 0x1028062c
9	8.53260900	192.168.3.1	192.168.3.3	DHCP	342	DHCP	Offer - Transaction ID 0x1028062c
10	8.53274900	0.0.0.0	255.255.255.255	DHCP	357	DHCP	Request - Transaction ID 0x1028062c
11	8.56323600	192.168.3.1	192.168.3.3	DHCP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

Figure 7.2: DHCP

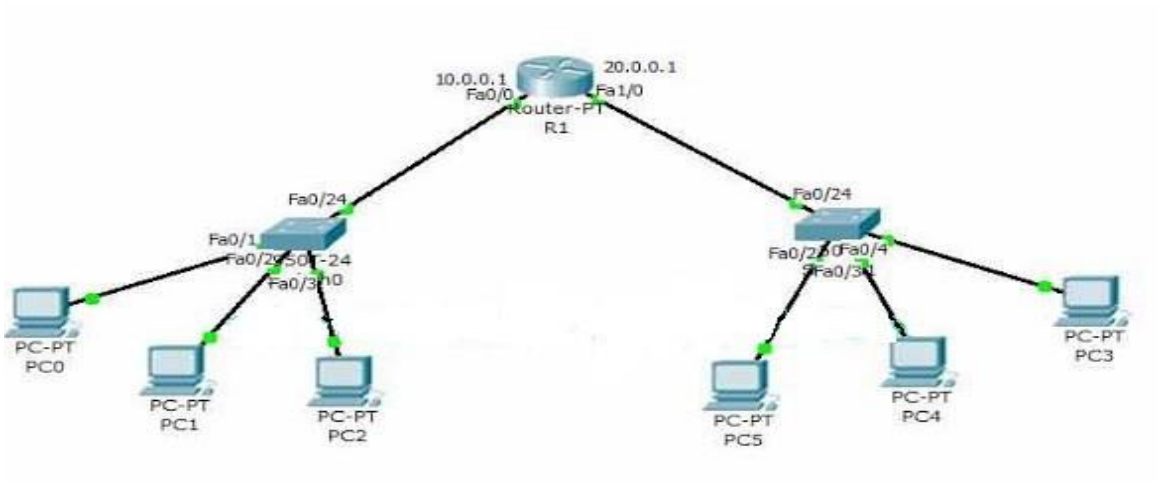
In Wireshark we see the following information with regards to DHCP:

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.

Q7.2) Configure DHCP for the below configuration



Q7.3) Configure DHCP for the below configuration



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'simpossible for us to remember all the IP addresses of the servers we access everyday.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.

LAB EXERCISES

Q7.4) Configure the below topology to setup DNS server. R1 will use R2 as DNS server to make DNS resolutions.

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

Figure 7.3 : 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 loop-back interface we just created is working. This will show us that the host name 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.

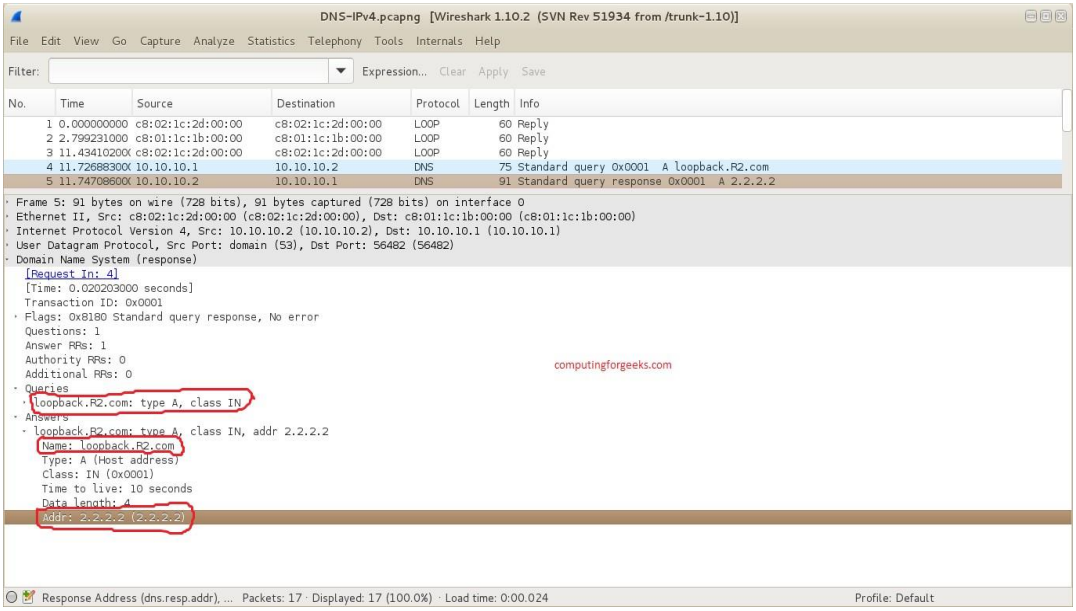
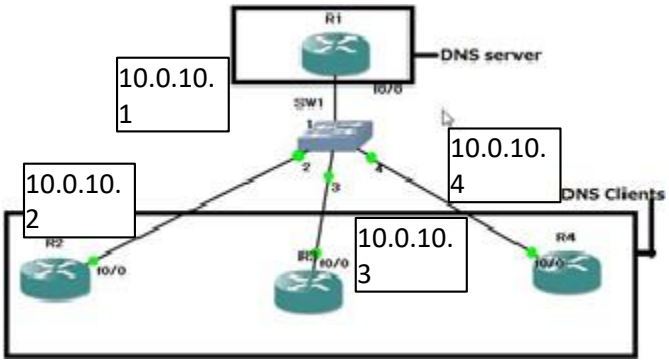
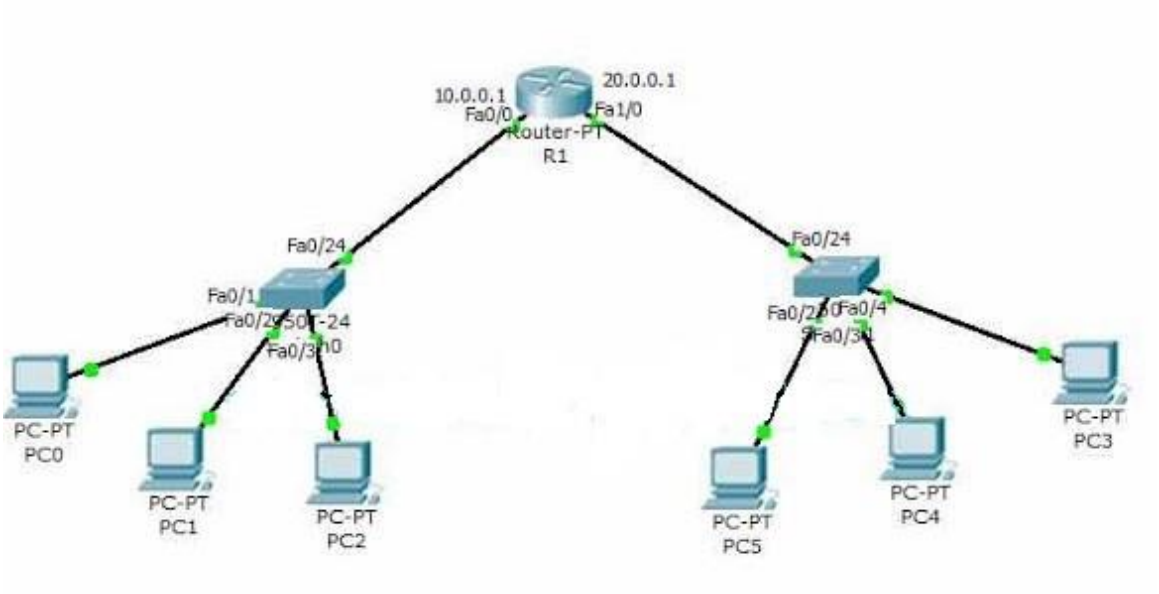


Figure 7.4 : Observation in WIRESHARK

LAB EXERCISE

Q7.5) Configure the topology shown below DNS Server and DNS Client. Test the setup. Analyze the Interaction.




10.0.0.1
Fa0/0
20.0.0.1
Fa1/0
Router-PT
R1
Fa0/24
Fa0/24
Fa0/1
Fa0/4
PC-PT
PC0
PC-PT
PC1
PC-PT
PC2
PC-PT
PC5
PC-PT
PC4
PC-PT
PC3

LAB No: 08 **Date:**

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 8.1 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.

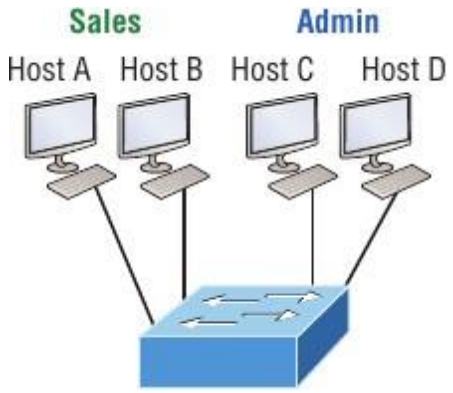


Fig 8.1 *One switch, one LAN: Before VLANs, there were no separations between hosts.*

If we want to separate the host's data, we could either buy another switch or create virtual LANs, as shown in Figure 8.2

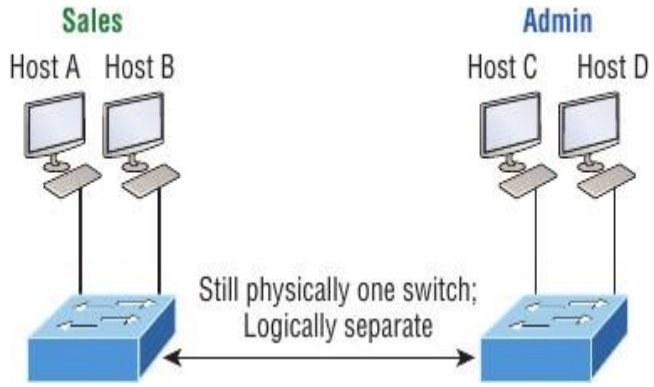


Fig 8.2 *One switch, two virtual LANs (logical separation between hosts): Still physically one switch, but this switch acts as many separate devices.*

In Figure 8.2 , 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 8.3

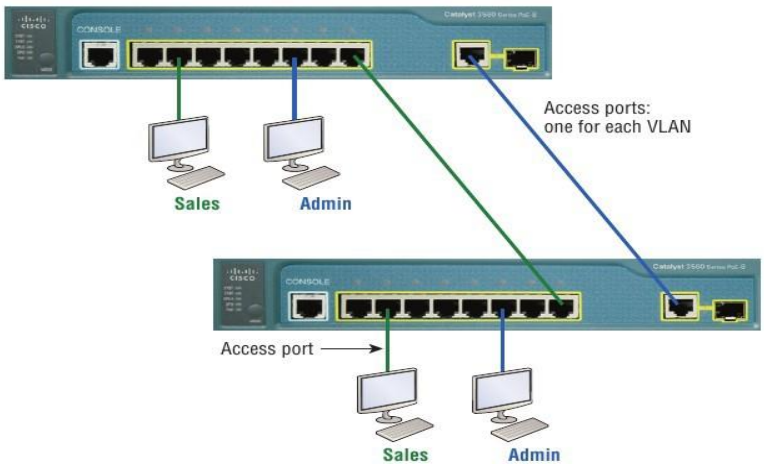


***Fig 8.3** Access Ports*

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 8.3, 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 8.4. 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.Trunklinks 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.

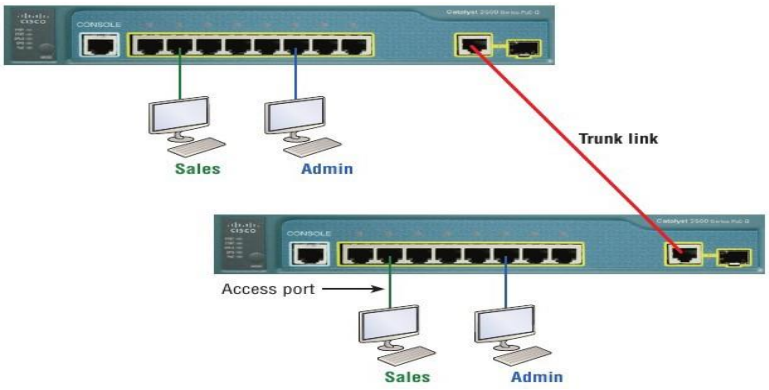


Fig 8.4 *VLANs can span across multiple switches by using trunk links, which carry traffic for multiple VLANs.*

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 8.4, 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. IEEE802.1q

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 towork.

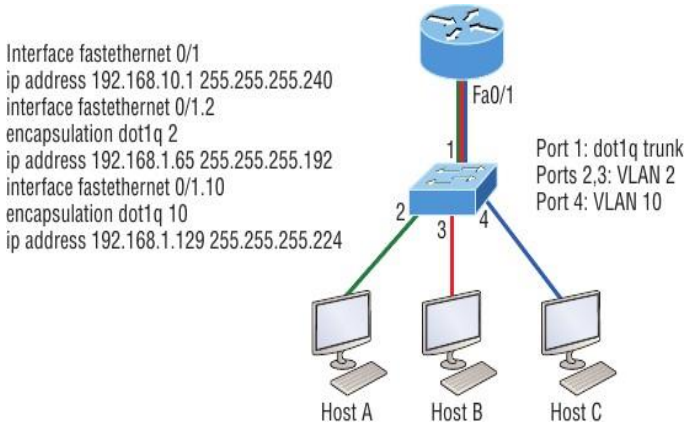
Unlike ISL, which encapsulates the frame with control information, 802.1q

inserts an 802.1q field along with tag control information

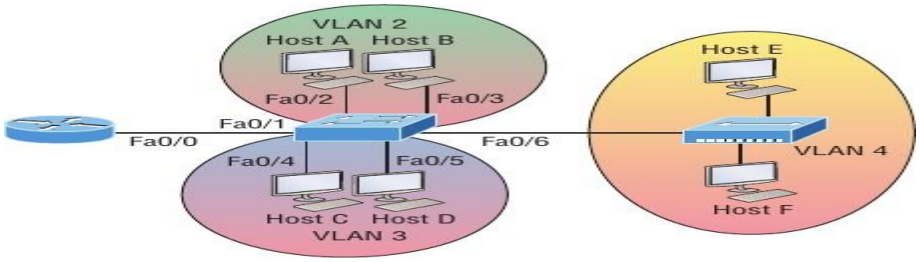
LAB EXERCISE

Q8.1) Configure following inter-VLAN example in GNS3 and verify the working using wireshark tool.

1.



Q8.2) Configure following inter-VLAN example in GNS3 and verify the working using wireshark tool.



LAB No: 9 **Date:**

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

9.1 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 9.1. 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.

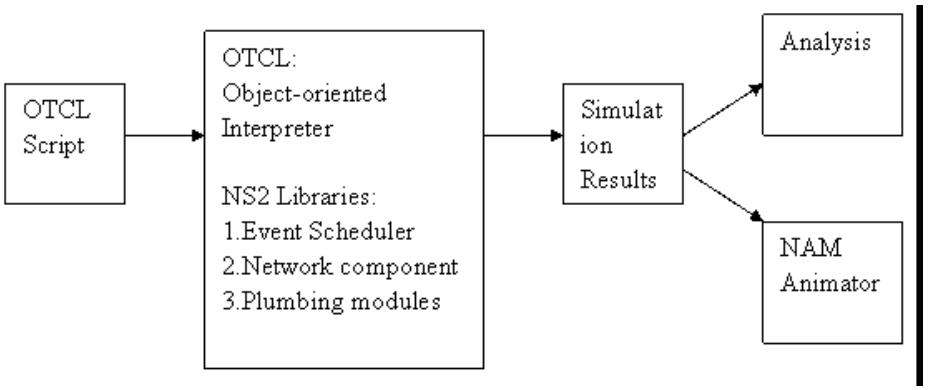


Fig. 9.1 A simplified user's view

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++.

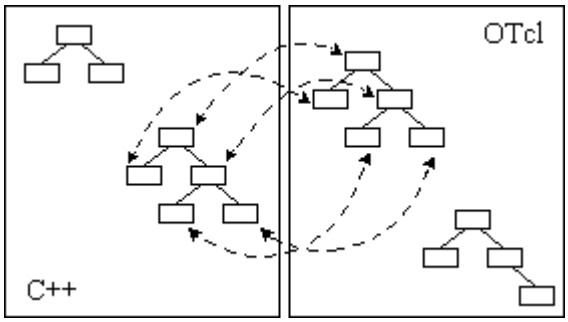


Fig. 9.2 C++ and OTcl: The Duality

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.

- **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. 9.3 shows a screenshot of nam window where the most important functions are being explained.

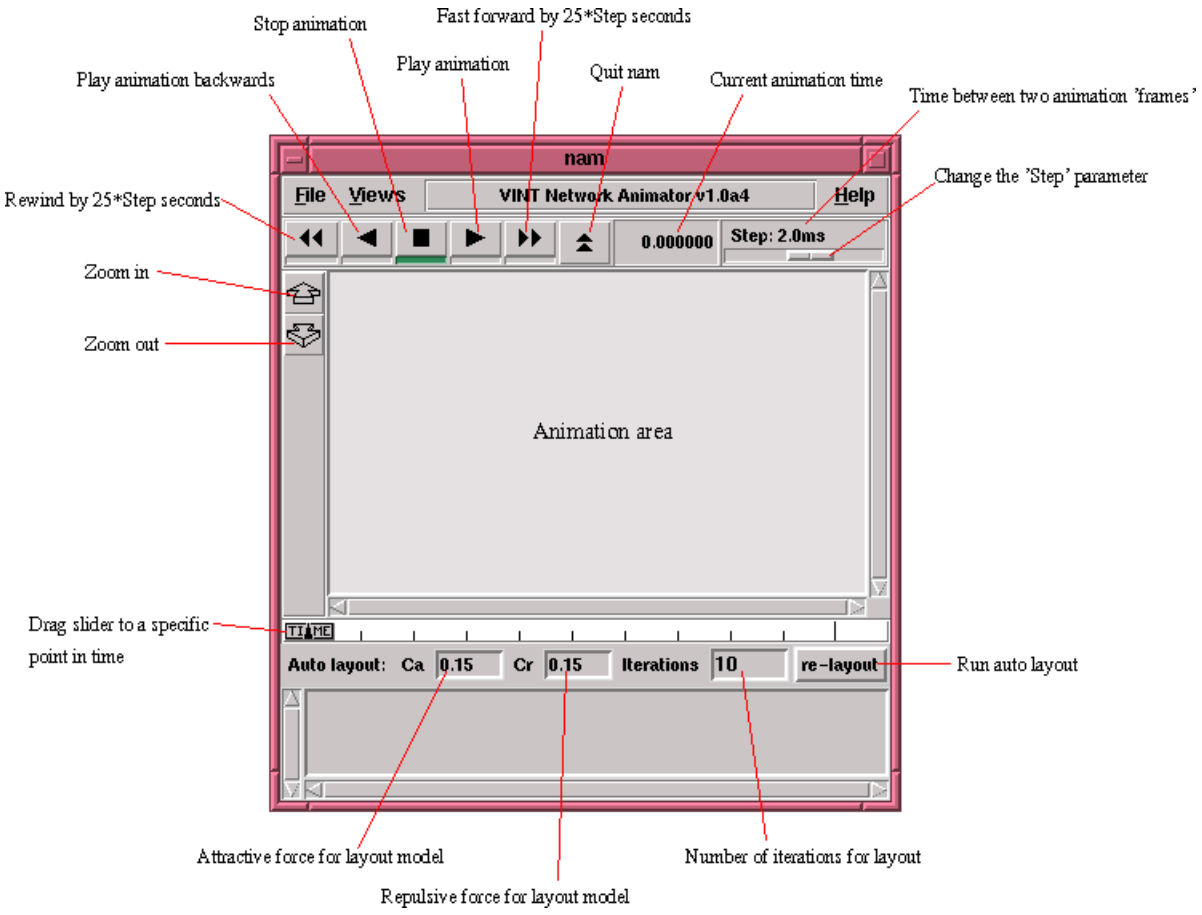


Fig. 9.3 Screenshot of sample nam window

- **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. 9.4 below.

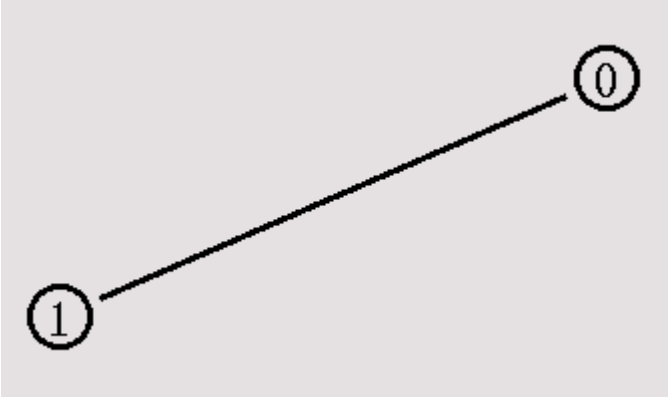


Fig. 9.4 Sample nam window having 2 nodes

- **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.

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.

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

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

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

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"'.

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. 9.5.

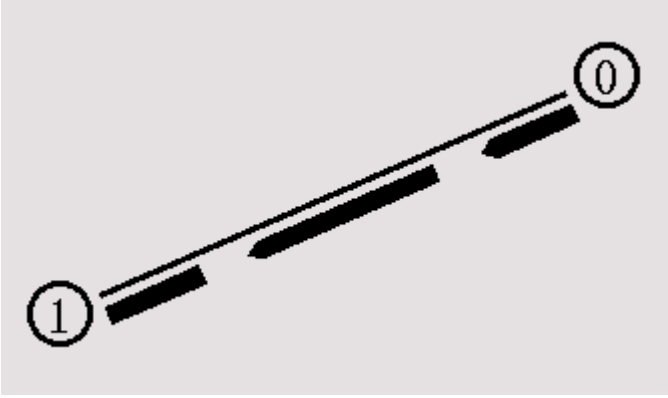


Fig. 9.5 Transmission of data packets from node 0 to node 1

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 'packetsize_' and 'interval_' parameters in the Tcl script and see what happens.

- **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. 9.6.

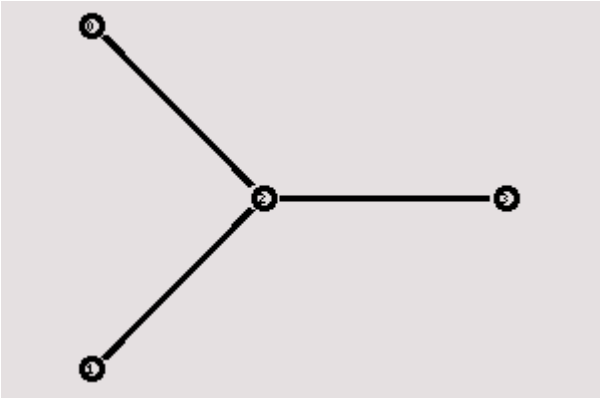


Fig. 9.6 Network topology with 4 nodes in a predefined layout

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.

9.6. 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.

- **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. 9.7.

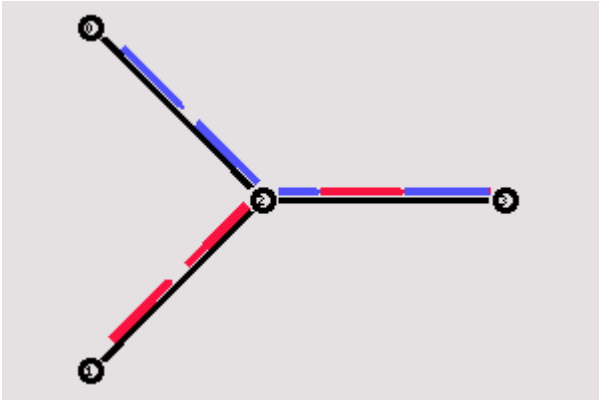


Fig. 9.7 Network topology with 4 nodes with packet flow

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

- **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. 9.8 after a few moments.

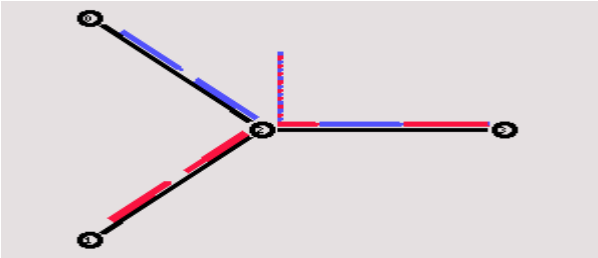


Fig. 9.8 Network topology with 4 nodes with packet queue

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. 9.9.

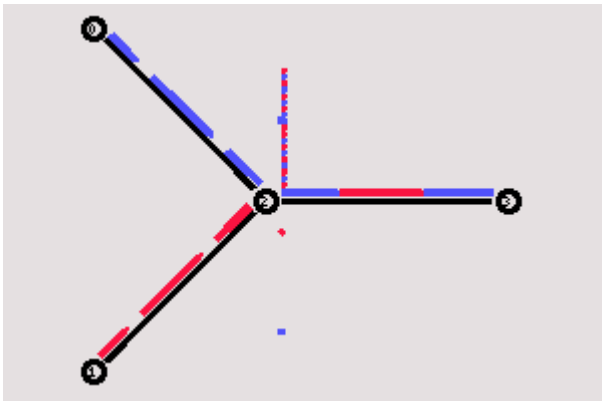


Fig. 9.9 Network topology with 4 nodes with packet drop

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

- **The algorithm for writing a complete Tcl script**

Begin:

- Create instance of network simulator object
- Turn on Tracing and NAM
- Define a 'Finish' procedure – Here the instruction to open NAM has to be included
  - Create network
    - Create nodes
  - Assign node position (NAM)
  - Create link between nodes (Simplex/Duplex)
  - Define colors for data flow (NAM)
  - Set the Queue size for node (Optional)
  - Monitor the queue (NAM)
  - Attach agent for every node - Transport Connection (TCP/UDP)
  - Set up Traffic Application (FTP/CBR)
    - Schedule the events
    - Call finish procedure
    - Print the necessary output data
    - Run the Simulation

- Visualize using NAM or analyze the trace file

End

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:

- **Nam file**

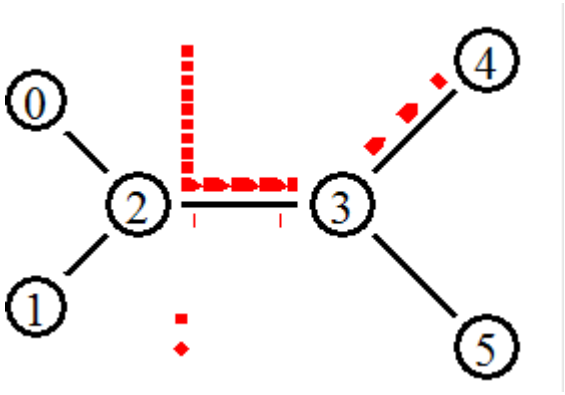


Fig. 10.10 Nam output

- **Trace file:**

Fig. 9.11 Trace file

LAB EXERCISES

- 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.
- 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.

- 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.
- 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.

LAB No 10 **Date:**

Measuring Performance of Protocols with NS2

Objectives:

- To understand performance of protocols on Network

QUEUE MANAGEMENT

Queue management schemes can broadly be divided into two groups: schemes that use the instantaneous queue size, like Drop Tail, and schemes that advocate an element of averaging of the queue size, like RED, before dropping or marking decisions are made. We focus on Drop Tail and RED. Drop Tail Drop Tail is perhaps the simplest queue management policy; it drops all incoming packets after the buffer is full.

Random Early Detection (RED): The goals of the RED algorithm, as per RFC 2309, are to reduce queuing delay and packet loss, to maintain high link utilization, to better accommodate bursty sources, and to provide a low-delay environment for interactive services by maintaining a small queue size. In this paper we use drops, instead of ECN marks, as the feedback signal to the end- systems. After the arrival of each packet, the RED algorithm calculates the average queue size avgas follows:

$$avg = (1 - w_q) * \overline{avg} + w_q * q \ ,$$

Where **wq** is the queue weight, **q** is the instantaneous queuesize, and **avg** is the previous average queue size. If the avg is less than **minth** then packets are enqueued. If the **avg** is more than **maxth** then all

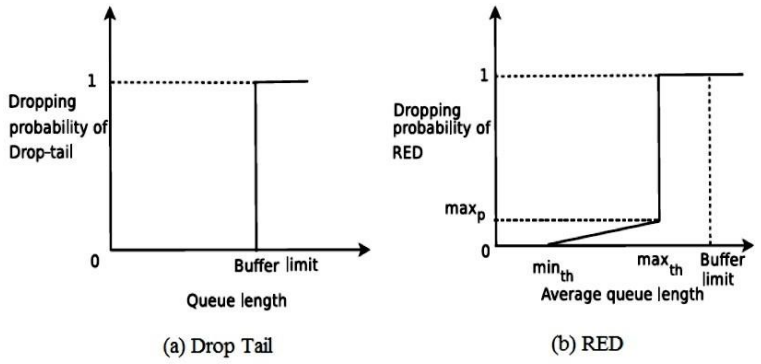
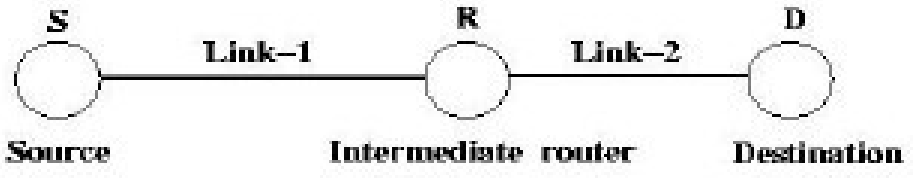


Fig.1. Drop functions of Drop Tail and RED

the incoming packets are dropped. If the **avg** is in between minth and **maxth**, then packets are dropped with a probability **pa**. The drop functions for DropTail and RED are shown in Fig.1.

LAB EXERCISE

10.1 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 it's 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:

2.If the source data rate (rate at which source injects data into the network) varies, what happens to the packets in the network?

3. At what point does it get congested?

4. 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:

3. 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.

4. 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.

5. Let each experiment run for at least 10sec.

6. Metrics: For each run, calculate/measure the following metrics by processing the output trace file.

- **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.)

- **Packet Loss**: The number of packets that were sent by the sender, but didn't reach the destination. Express it in percentage.
- **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 = 800bps or 0.8kbps. Express it in kbps.

LAB REPORT

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

- Offered Load vs percentage packet loss
- Offered load vs throughput

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

10.2 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 t1 is the time the packet was generated at the source and t2 was the time the packet was received at the destination. Delay of a packet is t2-t1. 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.

10.3 Setup a LAN network with 4 nodes N1, N2, N3 and N4 with the following configurations. Change the color of the flow between the nodes. Simulation duration is 5 seconds.

i. Node N1 to node N2 is 2Mb capacity and 20ms delay. Node N3 to node N4 is 1Mb and 10ms delay. N1 (source), N2 (sink), N3 (source), N4 (sink). Communication links are half-duplex and DropTail type of queue management.
ii. Node N1 to node N2 is 2Mb capacity and 20ms delay. Node N3 to node N4 is 1Mb and 10ms delay. N1 (source), N2 (sink), N3 (source), N4 (sink). Communication links are half-duplex and RED type of queue management.
iii. Node N1 to node N2 is 2Mb capacity and 20ms delay. Node N3 to node N4 is 1Mb and 10ms delay. N1 (source), N2 (sink), N3 (source), N4 (sink). Communication links are full-duplex and RED type of queue management.

LAB No 11 **Date:**

Measuring Performance of Congestion control Protocols with NS2

Objectives:

- To understand performance of TCP Tahoe and Reno protocols in Ethernet Network

Congestion protocol

Congestion Protocol uses a congestion window and a congestion policy that avoid congestion. Previously, we assumed that only receiver can dictate the sender's window size. We ignored another entity here, the network. If the network cannot deliver the data as fast as it is created by the sender, it must tell the sender to slow down. In other words, in addition to the receiver, the network is a second entity that determines the size of the sender's window.

Congestion policy in TCP –

I. Slow Start Phase: starts slowly increment is exponential to threshold
II. Congestion Avoidance Phase: After reaching the threshold increment is by 1
III. Congestion Detection Phase: Sender goes back to slow start phase or Congestion avoidance phase.

Slow Start Phase: exponential increment – In this phase after every RTT the congestion window size increments exponentially.

Initially cwnd = 1

After 1 RTT, cwnd = 2^(1) = 2

2 RTT, cwnd = 2^(2) = 4

3 RTT, cwnd = 2^(3) = 8

Congestion Avoidance Phase : additive increment – This phase starts after the threshold value also denoted as *ssthresh*. The size of *cwnd*(congestion window) increases additive. After each RTT cwnd = cwnd + 1.

Initially cwnd = i

After 1 RTT, cwnd = i+1

2 RTT, cwnd = i+2

3 RTT, cwnd = i+3

Congestion Detection Phase: multiplicative decrement – If congestion occurs, the congestion window size is decreased. The only way a sender can guess that congestion has occurred is the need to retransmit a segment. Retransmission is needed to recover a missing packet which is assumed to have been dropped by a router due to congestion. Retransmission can occur in one of two cases: when the RTO timer times out or when three duplicate ACKs are received.

vii. **Case 1 : Retransmission due to Timeout –** In this case congestion possibility is high.
(a) ssthresh is reduced to half of the current window size.
(b) set cwnd = 1
(c) start with slow start phase again.

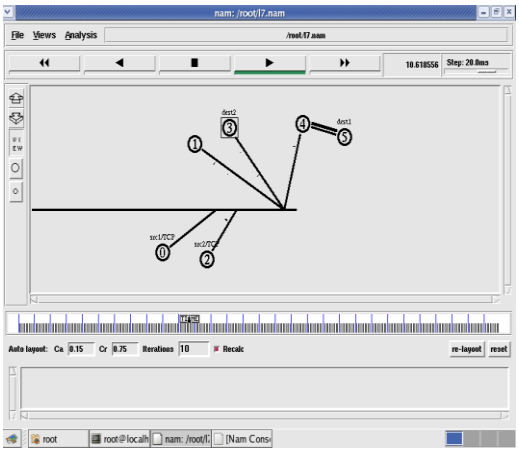
viii. **Case 2 : Retransmission due to 3 Acknowledgement Duplicates –** In this case congestion possibility is less.
(a) ssthresh value reduces to half of the current window size.
(b) set cwnd= ssthresh
(c) start with congestion avoidance phase

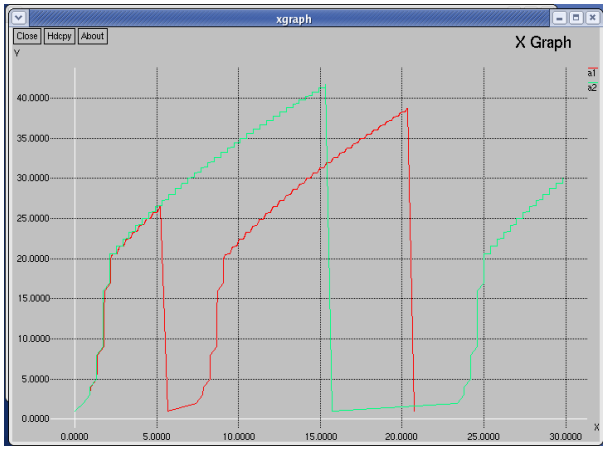
Lab Exercises

11.1 Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination with TCP Tahoe congestion avoidance strategies at transport layer.
11.2 Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination with TCP Reno congestion avoidance strategies at transport layer.

Lab Report

Plot a comparative graph representing congestion window size over a period of time in both TCP Tahoe and Reno.





Lab 12 End Semester Mini Project Evaluation

References:

1. W.Richard Stevens," UNIX Network Programming,Volume1: 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. TeerawatIssariyakul and Ekram Hossain. Introduction to Network Simulator NS2: Second Edition, Springer,2011.
5. Jason C. Nuemann, "The Book of GN3", No StarchPress,2015.
6. GNS3 Documentation, https://www.gns3.com.

APPENDIX:

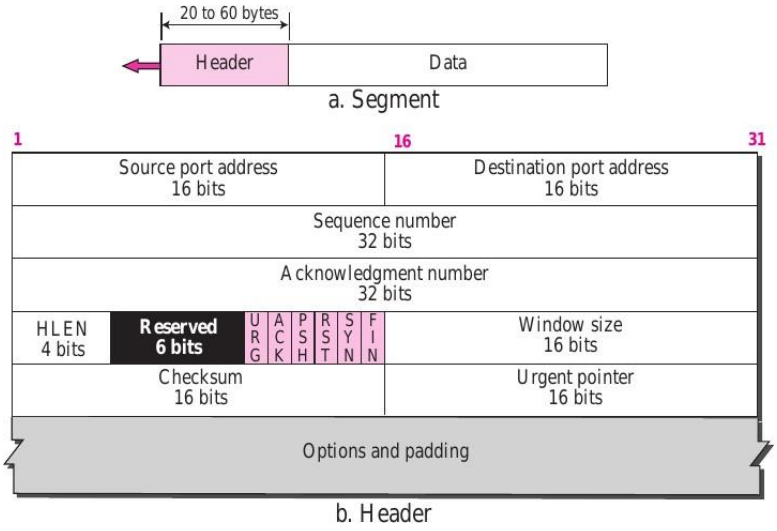


Fig 1: TCP Segment Format

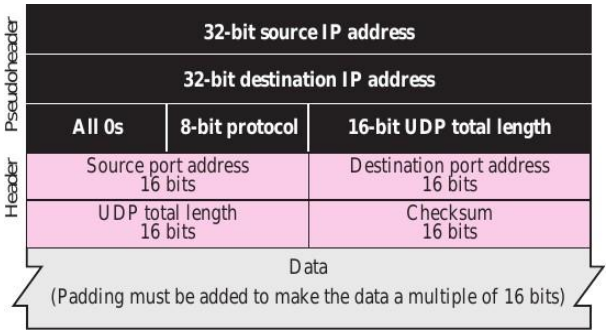


Fig:2 :UDP Header and Pseudo-header

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)

Fig 3: Format of DNS message

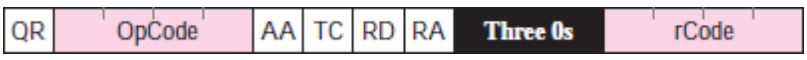


Fig 4: Flag fields of DNS message