NITTE EDUCATION TRUST | NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Visvesvaraya Technological University, Belgaum Approved by UGC, AICTE & Govt. of Karnataka)
Yelahanka, Bangalore-560064

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

LAB MANUAL

VI Semester

Network Programming Lab

Sub Code: 18ISL66

Prepared by:

Mr. Preetham N

Assistant Professor

Mrs. Navya C

Assistant Professor

Scheme: (2018)

TABLE OF CONTENTS

Vision and Mission of the Department

Vision

To build a strong research and teaching environment in the field of Information Technology to meet the ever-evolving global needs and to equip students with the latest knowledge, skills and practical orientation to face challenges in IT profession.

Mission

- To offer comprehensive educational programs in the field of Information Technology producing highly accomplished graduates
- To inculcate among the students, the culture of research and innovation
- To encourage students to participate in co-curricular and extra-curricular activities leading to enhancement of their social and professional skills

Program Educational Objectives (PEOs)

- Graduates will progress in their careers in IT industries of repute.
- Graduates will succeed in higher studies and research.
- Graduates of Information Science and engineering will demonstrate highest integrity with ethical values, good communication skills, leadership qualities and self-learning abilities.

Program Outcomes

PO-1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. .
PO-2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
PO-3	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
PO-4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO-5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO-6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO-7	Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO-8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
PO-9	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO-10	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO-11	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO-12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Objectives (PSOs)

- Student will be able to understand the architecture and working of computer system with relevant system software and apply appropriate system calls.
- Student will be able to apply mathematical methodologies in modelling real world problems for the development of software applications using algorithms, data structures and programming tools.

INTRODUCTION TO NETWORK PROGRAMMING LAB

NETWORKING BASICS

Computer networking is the practice of linking computing devices together with hardware and software that supports data communications across these devices.

KEY CONCEPTS AND TERMS

Packet: A message or data unit that is transmitted between communicating processes.

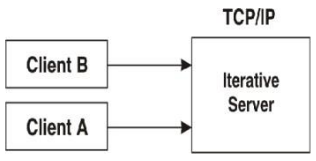
Host: A computer system that is accessed by a user working at a remote location. It is the remote process with which a process communicates. It may also be referred as Peer.

Channel: Communication path created by establishing a connection between endpoints.

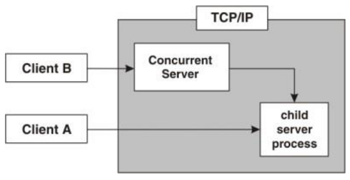
Network: A group of two or more computer systems linked together

Server: In computer networking, a server is a computer designed to process requests and deliver data to other computers over a local network or the Internet.

- **Iterative servers**: This server knows ahead of time about how long it takes to handle each request & server process handles each request itself.



- **Concurrent servers:** The amount of work required to handle a request is unknown, so the server starts another process to handle each request.



Client: A Client is an application that runs on a personal computer or workstation and relies on a server to perform some operations.

Network Address: Network addresses give computers unique identities they can use to communicate with each other. Specifically, IP addresses and MAC addresses are used on most home and business networks.

Protocols: A Protocol is a convention or standard rules that enables and controls the connection, communication, and data transfer between two computing endpoints.

Port An interface on a computer to which you can connect a device. It is a "logical connection place" and specifically, using the Internet's protocol, TCP/IP.

A port is a 16-bit number, used by the host-to-host protocol to identify to which higher-level protocol or application program (process) it must deliver incoming messages.

PORTS	**RANGE**
Well-known ports	1-1023
Ephemeral ports	1024-5000
User-defined ports	5001-65535

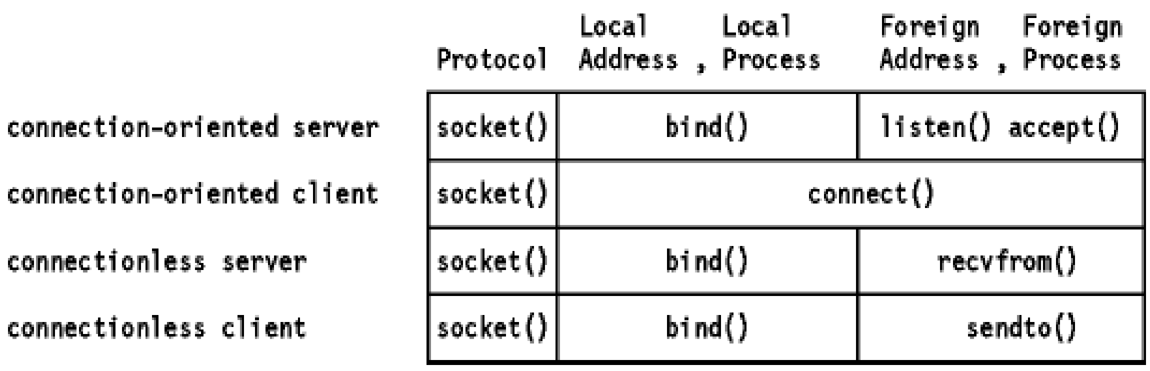
Connection: It defines the communication link between two processes.

Association: Association is used for 5 tuple that completely specifies the two processes that make up a connection.

{ Protocol, local-address, local-process, foreign-address, foreign- process}

The local address and foreign address specify the network ID & Host-ID of the local host and the foreign host in whatever format is specified by protocol suite.
The local process and foreign process are used to identify the specific processes on each system that are involved in a connection.
We also define Half association as either

{ *protocol, local-address, local process}* or **{ *protocol, local-address, local process}*** which specify each half of a connection. This half association is called a Socket or transport address.



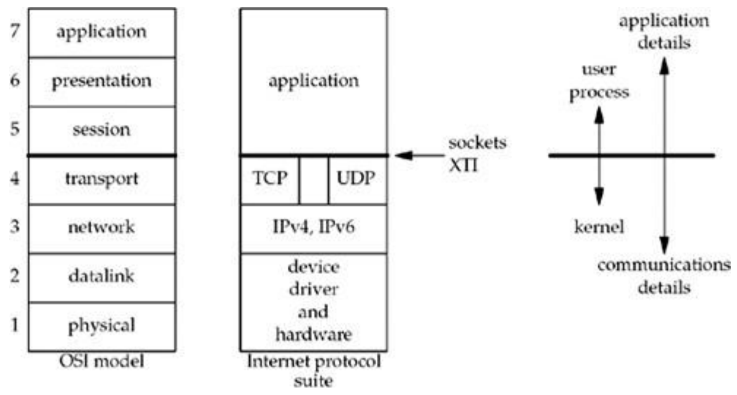
	Protocol	Local Address ,	Local Process	Foreign Address ,	Foreign Process
connection-oriented server	socket()	bind()		listen() accept()	
connection-oriented client	socket()	connect()			
connectionless server	socket()	bind()		recvfrom()	
connectionless client	socket()	bind()		sendto()	

OSI Model

A common way to describe the layers in a network is to use the International Organization for Standardization (ISO) open systems interconnection (OSI) model for computer communications. This is a seven-layer model, which we show in Figure below along with the approximate mapping to the Internet protocol suite.

We consider the bottom two layers of the OSI model as the device driver and networking hardware that are supplied with the system. The network layer is handled by the IPv4 and IPv6 protocols. The transport layers that we can choose from are TCP and UDP

Layers in OSI model and Internet protocol suite.



The upper three layers of the OSI model are combined into a single layer called the **application**. This is the Web client (browser) or whatever application we are using. With the Internet protocols, there is rarely any distinction between the upper three layers of the OSI model.

Syllabus

Department: *Information Science and Engineering*	***Course Type:*** *Core*
Course Title*: Network Programming Lab*	***Course Code:****18ISL66*
L-T-P:*0-0-2*	***Credits:****1*
Total Contact Hours:*26*	***Duration of SEE:3*** *hrs*
SEE Marks*: 50*	***CIE Marks:*** *50*

Pre-requisite:

- Knowledge on Computer Networks
- NS 3.29 simulator, OS: Ubuntu/ Fedora Core.

Cos	**Course Outcome Description**	**Blooms Level**
1	Apply basic functions of byte-oriented and bit-oriented protocols and error detection and correction methods during data transmission.	L3
2	Develop client-server applications using TCP and UDP.	L4
3	Apply link state and distance vector routing algorithms for effective routing of packets and leaky bucket algorithm for congestion control.	L3
4	Apply Point-to-Point protocol in networks with a client and a server, many clients and a server, in star topology and bus topology.	L3
5	Apply NetAnim software to demonstrate the graphical scenario of the network models.	L3

Teaching Methodology:

- Instruction classes
- Demonstrations

Assessment Methods:

- Rubrics for evaluating the regular laboratory classes for 30 marks.
- Two internal assessment tests of 20 marks each and average of the two will be taken.
- Final examination is for 50 Marks.

Course Outcome to Programme Outcome Mapping:

Cos	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	2	3	2	3				1			2		2
CO2	3	2	3	2	3				1			2		2
CO3	3	2	3	2	3				1			2		2
CO4	3	3	2	2	3				1			2		2
CO5	3	3	2	2	3				1			2		2
18ISL66	3	2.4	2.6	2	3				1			2		2

PART – A			
1a	Framing	Design a C program in which sender module should count the no of bytes in the frame and receive module should display each frame received.	CO1
1b		Design a C program to implement bit stuffing, encoding and decoding concept in data link layer.	CO1
2	Error control	Design and implement CRC error detection method used in data link layer.	CO1
3a	Socket Programming	Design a C program to implement client server model (TCP) using socket programming.	CO2
3b		Design a C program to implement client server model (UDP) using socket programming.	CO2
4	Routing Algorithm	Design and implement a C program to route the packet in a network using distance vector algorithm.	CO3
5	Congestion Control	Design a C program for congestion control using leaky bucket algorithm.	CO3
PART – B			
6	Simulate peer-to-peer communication between a client and a server using Point-to-Point protocol. Apply NetAnim software to demonstrate the scenario graphically. Analyze packet parameters by creating trace file using Ascii trace metrics.		CO4, CO5
7	Simulate to implement a bus topology using Point-to-Point protocol between a client and a LAN with 4 nodes. The LAN use CSMA during packet transmission. Apply NetAnim software to demonstrate the scenario graphically. Analyze packet parameters by creating trace file using Ascii trace metrics.		CO4, CO5
8	Simulate peer-to-peer communication between a client and a server using CSMA protocol. Apply NetAnim software to demonstrate the scenario graphically. Analyze packet parameters by creating trace file using Ascii trace metrics.		CO4, CO5
9	Simulate to implement the star topology using Point-to-Point protocol. Apply NetAnim software to demonstrate the scenario graphically. Analyze packet parameters by creating trace file using Ascii trace metrics.		CO4, CO5
10	Simulate packet flow in a network for TCP protocol between a client and a server. Apply NetAnim software to demonstrate the scenario graphically. Analyze packet parameters by creating trace file using Ascii trace metrics.		CO4, CO5

Rubric Evaluation

Performance Indicator	Excellent	Good	Fair	Poor
Fundamental Knowledge (3M)	Student can demonstrate thorough knowledge about the fundamental concepts of computer networks	Student can demonstrate partial knowledge about the fundamental concepts of computer networks	Student can demonstrate fair knowledge about the fundamental concepts of computer networks	Student is not able to demonstrate knowledge about the fundamental concepts of computer networks
Understanding of problem definition (2M)	Student can completely understand the problem definition	Student can partially understand the problem definition	Student can fairly understand the definition of the problem	Student is not able to understand the problem definition
Design of experiment (5M)	Student exhibit's excellent ability to design and analyse the given problem	Students exhibits good ability to design and analyse the given problem	Students exhibits fair ability to design and analyse the given problem	Students exhibits no ability to design and analyse the given problem
Execution (5M)	All conditions handled and extensive testing of the code.	Partial conditions handled and extensive testing of the code.	Not all conditions handled and extensive testing of the code.	No Execution
Documentation (15 M)	Student has followed all the guidelines given by the tutor in writing the lab record and observations	Student has partially followed the guidelines given by the tutor in writing the lab record and observations	The lab record and observations are written in a fair manner without incorporating changes specified by the tutor	Student has not followed the guidelines given by the tutor in writing the lab record and observations

Problem-1 (1a & 1b)

Design and implement a program to handle variable length frames in Data Link Layer.

Framing:

Datalink layer packs the bits into frames, so that each frame is distinguishable from other. Framing in datalink layer separates a message from one source to a destination, or from other messages to other destination, by adding a sender address and a destination address.

Fixed-Size Framing

In fixed-size framing, there is no need for defining the boundaries of the frames; the size itself can be used as a delimiter.

Variable-Size Framing

In variable-size framing, we need a way to define the end of the frame and the beginning of the next. Historically, two approaches were used for this purpose: **a character-oriented approach and a bit-oriented approach.**

Character-Oriented Protocols

The header, which normally carries the source and destination addresses and other control information

And the trailer, which carries error detection or error correction redundant bits, are also multiples of 8 bits.

To separate one frame from the next, an 8-bit (1-byte) flag is added at the beginning and the end of a frame.

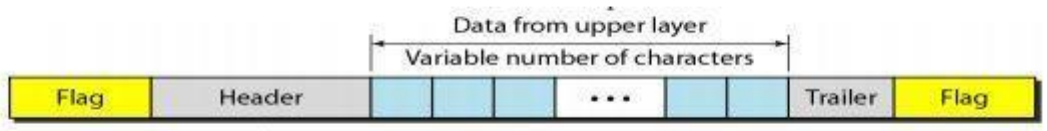


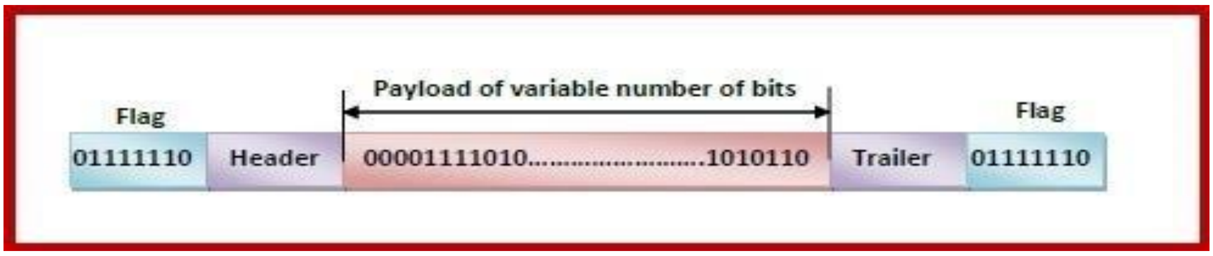
Figure: *A frame in a character-oriented protocol*

Byte stuffing (or character stuffing):

- In this sender's data link layer insert a special escape byte (ESC) just before each "accidental" flag byte in the data.
- The data link layer on the receiving end removes the escape byte before the data are given to the network layer.
- This technique is called **byte stuffing or character stuffing**.

Bit-Oriented Protocols

In a bit-oriented protocol, the data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, and so on. However, in addition to headers (and possible trailers), we still need a delimiter to separate one frame from the other. Most protocols use a special 8-bit pattern flag 01111110 as the delimiter to define the beginning and the end of the frame.



Problem with variable length frames:

However, in addition to headers (and possible trailers), we still need a method to separate one frame from the other. There are two different methods proposed for this

I. **By inserting the number of bytes/bits (length of the frames)---byte stuffing**

II. **By use of a special delimiter flag _______ bit stuffing**

Insertion of length of frames in variable length frames:

Every frame carries its length, which will indicate to the receiver where current frame will end and which is the beginning of next frame.

Sender side Algorithm Read the Frame Insert length of frame as beginning field (Length Including itself) Send Frame
Receiver side Algorithm Receive the frame Extract Length field and read number of bytes specified in Length Field

Use of a special delimiter flag

To separate one frame from the next, a flag is added at the beginning and the end of a frame. Any pattern used for the flag could also be part of the information. If this happens, the receiver, when it encounters this pattern in the middle ofthe data, thinks it has reached the end of the frame. To fix this problem, a byte-stuffing/bit-stuffing strategy was added to.

Sender side Algorithm Read the Frame Insert flag at the beginning of frame Patter '011111' (0 followed by 5 1s) should be stuffed with one zero Append flag at the end of the frame Send Frame
Receiver side Algorithm Receive the frame Read data between beginning and end flag Each '0' after '011111' (0 Followed by 5 1s) should be removed Process data

In byte stuffing/bit stuffing, a special byte/bit is added to the data section of the frame when there is a character/binary pattern with the same pattern as the flag. The data section is stuffed with an extra byte/bit. This byte is usually called the escape character (ESC), which has a predefined bit pattern. Whenever the receiver encounters the ESC character, it removes it from the data section and treats the next character as data, not a delimiting flag.

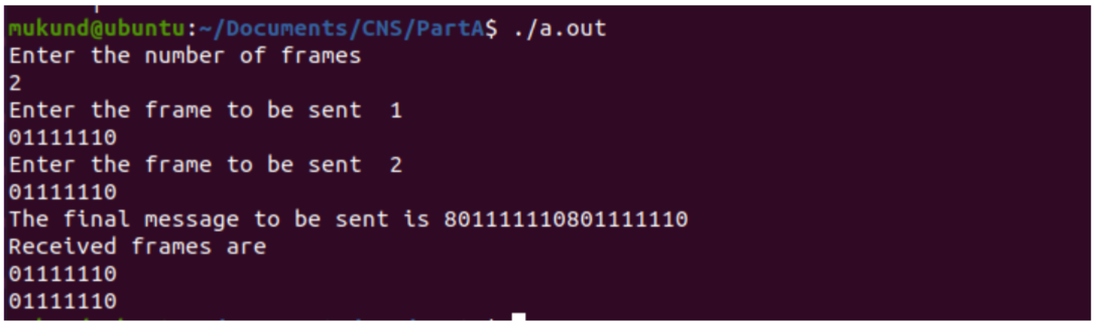
For example, most of the bit-oriented protocols use "01111110" as flag. Here wherever zero followed by five '1's come in data part, one extra 0 will be stuffed as ESC. Receiver will remove all the zeros that come after '011111'.

DATA:	0101111110011111001
Fame sent after bit stuffing:	**01111110** 01011111**0**10011111**0**0001
01111110	

Expected Output:

Output

- Frames Recieved are

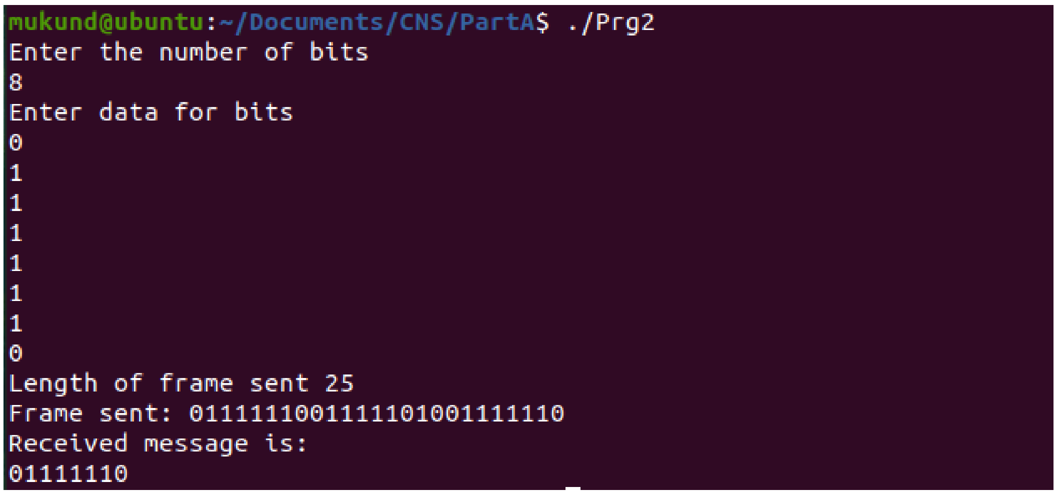


```
mukund@ubuntu:~/Documents/CNS/PartA$ ./a.out
Enter the number of frames
2
Enter the frame to be sent  1
01111110
Enter the frame to be sent  2
01111110
The final message to be sent is 801111110801111110
Received frames are
01111110
01111110
```

Byte stuffing

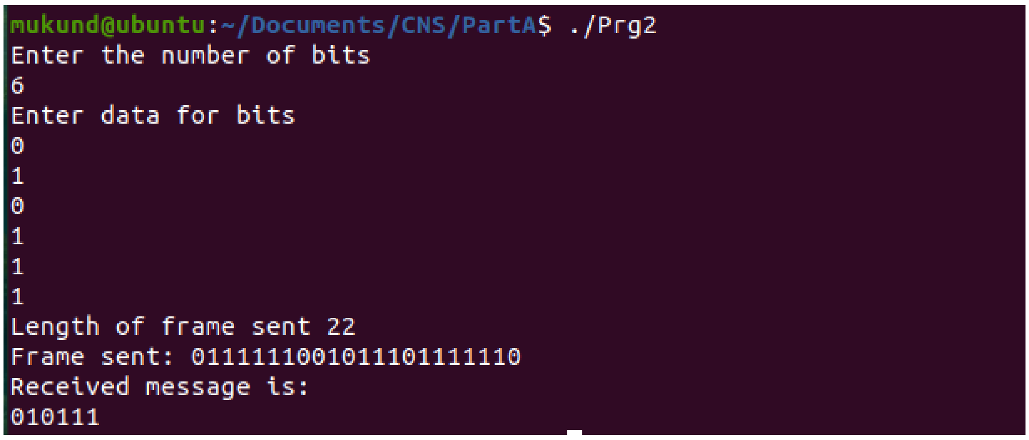
Output

- With Flag



```
mukund@ubuntu:~/Documents/CNS/PartA$ ./Prg2
Enter the number of bits
8
Enter data for bits
0
1
1
1
1
1
1
0
Length of frame sent 25
Frame sent: 0111111001111101001111110
Received message is:
01111110
```

- Without Flag



```
mukund@ubuntu:~/Documents/CNS/PartA$ ./Prg2
Enter the number of bits
6
Enter data for bits
0
1
0
1
1
1
Length of frame sent 22
Frame sent: 0111111001011101111110
Received message is:
010111
```

Problem-2

Design and Implement error detection methods used in Data link layer.

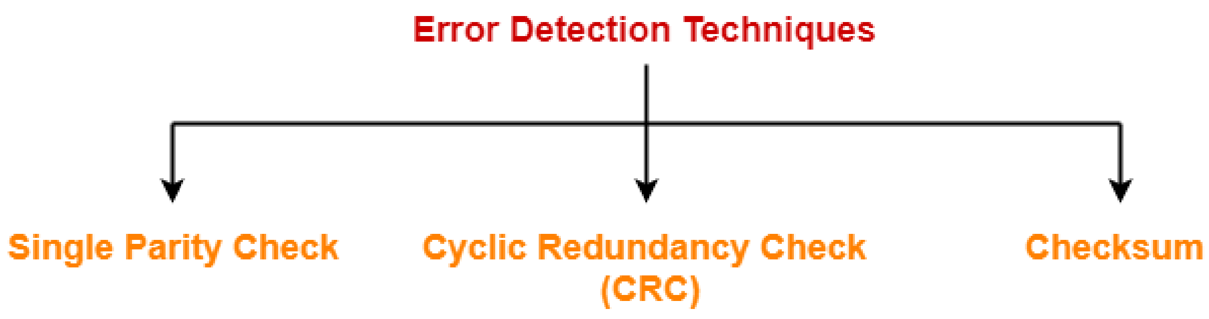
Data communication requires at least two devices working together, one to send and the other to receive. Even such a basic arrangement requires a great deal of coordination for an intelligible exchange to occur. The most important responsibilities of the data link layer are flow control and **error control**

Error Control

Error control is both error detection and error correction. It allows the receiver to inform the sender of any frames lost or damaged in transmission and coordinates the retransmission of those frames by the sender. In the data link layer, the term error control refers primarily to methods of error detection and retransmission. Error control in the data link layer is often implemented simply: Any time an error is detected in an exchange, specified frames are retransmitted. This process is called **automatic repeat request (ARQ)**.

Error detection is a technique that is used to check if any error occurred in the data during the transmission.

Some popular error detection methods are-



- Single Parity Check
- **Cyclic Redundancy Check (CRC)**
- Checksum

Cyclic Redundancy Check (CRC)

- Cyclic Redundancy Check (CRC) is an error detection method.
- It is based on binary division.

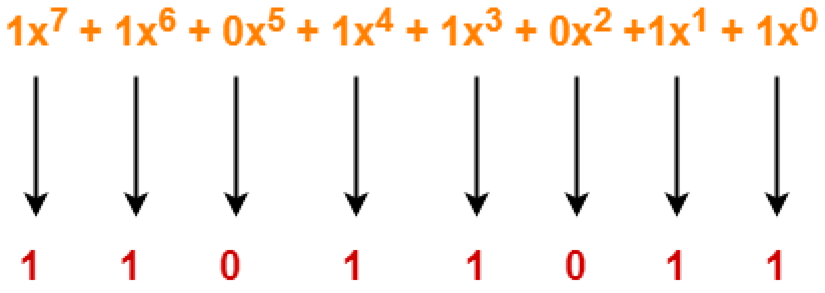
CRC Generator-

- CRC generator is an algebraic polynomial represented as a bit pattern.
- Bit pattern is obtained from the CRC generator using the following rule-
    - The power of each term gives the position of the bit and the coefficient gives the value of the bit.

Example-

Consider the CRC generator is x7 + x6 + x4 + x3 + x + 1. The corresponding binary pattern is obtained as-

Thus, for the given CRC generator, the corresponding binary pattern is **11011011**.



Properties Of CRC Generator-

The algebraic polynomial chosen as a CRC generator should have at least the following properties-

- It should not be divisible by x.
- It should be divisible by x+1.

Steps Involved-

Error detection using CRC technique involves the following steps-

Step-01: Calculation Of CRC At Sender Side-

At sender side,

- A string of n 0’s is appended to the data unit to be transmitted.
- Here, n is one less than the number of bits in CRC generator.
- Binary division is performed of the resultant string with the CRC generator.
- After division, the remainder so obtained is called as **CRC**.
- It may be noted that CRC also consists of n bits.

Step-02: Appending CRC To Data Unit-

At sender side,

- The CRC is obtained after the binary division.
- The string of n 0's appended to the data unit earlier is replaced by the CRC remainder.

Step-03: Transmission To Receiver-

- The newly formed code word (Original data + CRC) is transmitted to the receiver.

Step-04: Checking at Receiver Side-

At receiver side,

- The transmitted code word is received.
- The received code word is divided with the same CRC generator.
  - On division, the remainder so obtained is checked.

The following two cases are possible-

Case-01: Remainder = 0

If the remainder is zero,

- Receiver assumes that no error occurred in the data during the transmission.
- Receiver accepts the data.

Case-02: Remainder ≠ 0

If the remainder is non-zero,

- Receiver assumes that some error occurred in the data during the transmission.
- Receiver rejects the data and asks the sender for retransmission.

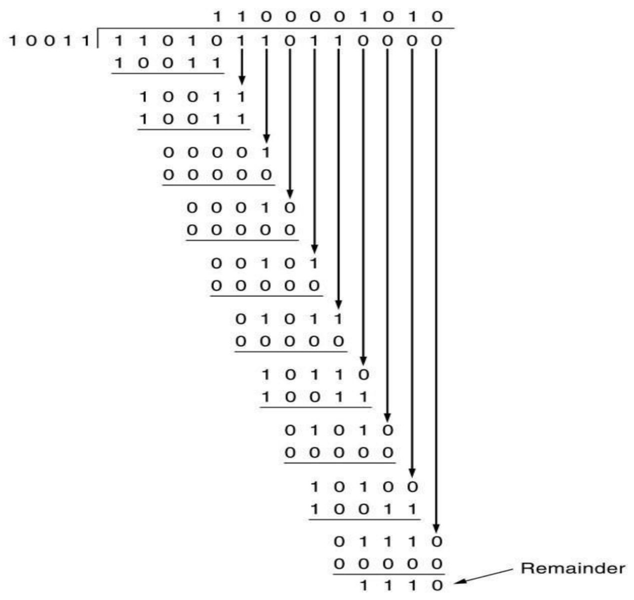
Problem:

A bit stream **1101011011** is transmitted using the standard CRC method. The generator polynomial is **x4+x+1**. What is the actual bit string transmitted?

Solution-

- The generator polynomial G(x) = x4 + x + 1 is encoded as **10011.**
- Clearly, the generator polynomial consists of 5 bits.
- So, a string of 4 zeroes is appended to the bit stream to be transmitted. 11010110110000
  - The resulting bit stream is 1101011011**0000**.

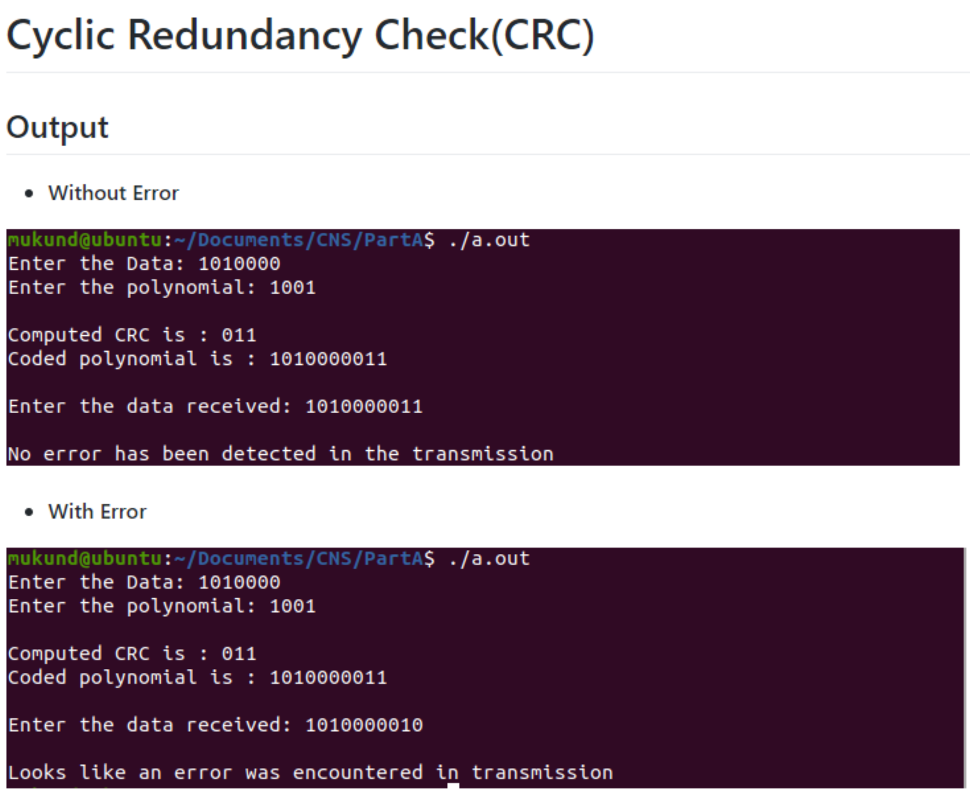
Now, the binary division is performed as-



From here, CRC= 1110. Now,

- The code word to be transmitted is obtained by replacing the last 4 zeroes of 1101011011**0000** with the CRC.
- Thus, the code word transmitted to the receiver = 1101011011**1110**.

Expected Output:



Cyclic Redundancy Check(CRC)

Output

- Without Error

```
mukund@ubuntu:~/Documents/CNS/PartA$ ./a.out
Enter the Data: 1010000
Enter the polynomial: 1001

Computed CRC is : 011
Coded polynomial is : 1010000011

Enter the data received: 1010000011

No error has been detected in the transmission
```

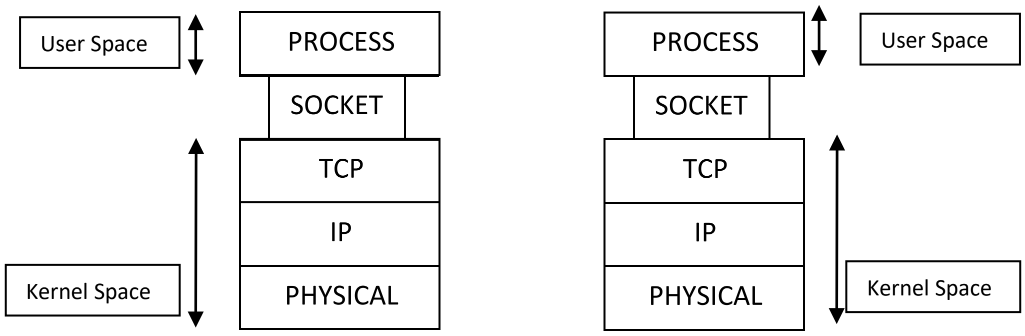
- With Error

```
mukund@ubuntu:~/Documents/CNS/PartA$ ./a.out
Enter the Data: 1010000
Enter the polynomial: 1001

Computed CRC is : 011
Coded polynomial is : 1010000011

Enter the data received: 1010000010

Looks like an error was encountered in transmission
```

Problem-3 (TCP) & 4 (UDP)

Write a C program to implement client-server model using socket programming.

- In layman's term, a Socket is an end point of communication between two systems on a network.
- To be a bit precise, a socket is a combination of IP address and port on one system.
- On each system a socket exists for a process interacting with the socket on other system over the network.
- Socket helps in identifying a process on a host machine through it's protocol, IP address and port number

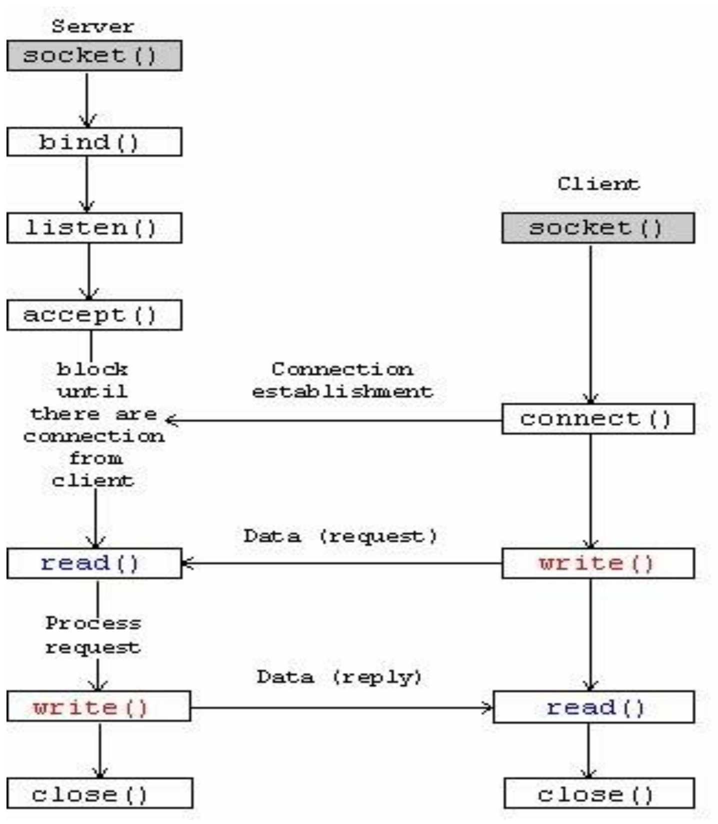


- Primary Socket Calls
  - socket() $\longrightarrow$ create a new socket and return its descriptor
  - bind() $\longrightarrow$ Associate a socket with a port and address
  - listen() $\longrightarrow$ Establish queue for connection requests
  - accept() $\longrightarrow$ Accept a connection request
  - connect() $\longrightarrow$ Initiate a connection to a remote host
  - recv() $\longrightarrow$ Receive data from a socket descriptor
  - send() $\longrightarrow$ Send data to a socket descriptor
  - close() $\longrightarrow$ "one-way" close of a socket descriptor
- Functions which help us in converting from host specific details to network specific details
  - Gethostname $\longrightarrow$ Given a hostname, returns a structure which specifies its DNS name(s) and IP address
  - Getservbyname $\longrightarrow$ Given service name and protocol, returns a structure which specifies its name(s) and its port address

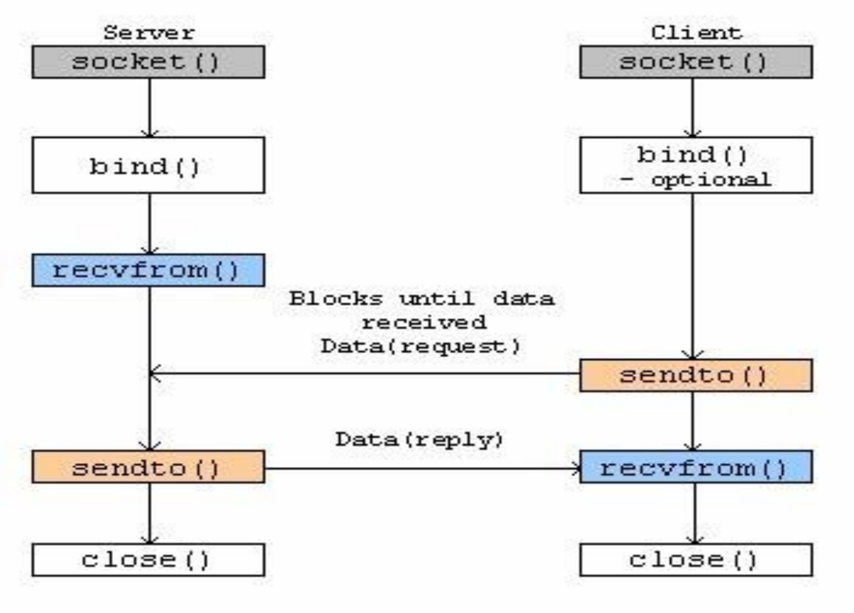
- ntohs/ntohl ⟶ To convert from network byte order to host byte order
- htons/htonl ⟶ To convert from host byte order to network byte order
- inet_itoa/inet_addr ⟶ Convert 32 bit IP address from network byte order to dotted decimal order

- Header Files
  - <sys/types.h>
  - <errno.h>
  - <sys/socket.h> ⟶ structsockaddr; system prototypes and constants; system prototypes and constants
  - <netdb.h> ⟶ network info lookup prototypes and structure
  - <netinet/in.h> ⟶ structsockaddr_in; byte ordering macros_in; byte ordering macros
  - <arpa/inet.h>
- Two types of (TCP/IP) sockets
  - Stream sockets (e.g. uses TCP) ⟶ provide reliable byte-stream service
  - Datagram sockets (e.g. uses UDP) ⟶ provide best-effort datagram service and messages up to 65.500 byte
- Client-Server communication
  - Server
  - passively waits for and responds to clients
- Client
  - initiates the communication
  - must know the address and the port of the server
  - active socket

TCP based client server communication



UDP based client server communication



Socket creation in C

intsockid= socket(family, type, protocol);

- sockid $\longrightarrow$ socket descriptor, an integer (like a file-handle)
- family $\longrightarrow$ integer, communication domain, e.g.,
  - AF_INET, IPv4 protocols, Internet addresses (typically used)
  - AF_UNIX, Local communication, File addresses
- Type $\longrightarrow$ Communication type
  - SOCK_STREAM – reliable TCP, 2-way, connection-based service
  - SOCK_DGRAM – unreliable UDP, connectionless, messages of maximum length
- Protocol $\longrightarrow$ specifies protocol IPPROTO_TCP IPPROTO_UDP
- usually set to 0 (i.e., use default protocol)
- upon failure returns -1

Bind Function

- associates and reserves a port for use by the socket
  - **int status = bind(sockid, &addrport, size);**
- sockid: integer, socket descriptor
- addrport: structsockaddr, 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: the size (in bytes) of the addrport structure
- status: upon failure -1 is returned

Listen Function

- Instructs TCP protocol implementation to listen for connections
  - int status = listen(sockid, queueLimit);
- sockid: integer, socket descriptor
- queuelen: integer, number of active participants that can “wait” for a connection
- status: 0 if listening, -1 if error

Connect Function

- The client establishes a connection with the server by calling connect()
    - int status = connect(sockid, &foreignAddr, addrlen)
- sockid: integer, socket to be used in connection
- foreignAddr : structsockaddr: address of the passive participant
- addrlen: integer, sizeof(name)
- status: 0 if successful connect, -1 otherwise

Accept Function

- The server gets a socket for an incoming client connection by calling accept()
    - int s= accept(sockid, &clientAddr, &addrLen);
- s: integer, the new socket (used for data-transfer)
- sockid: integer, the orig. socket (being listened on)
- clientAddr: structsockaddr, address of the active participant filled in upon return
- addrLen: sizeof(clientAddr): value/result parameter must be set appropriately before call adjusted upon return

Exchanging data with stream socket

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

- msg: const void[], message to be transmitted
- msgLen: integer, length of message (in bytes) to transmit
- flags: integer, special options, usually just 0
- count: Number bytes transmitted (-1 if error)

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

- recvBuf: void[], stores received bytes
- bufLen: Number bytes received
- flags: integer, special options, usually just 0

- count: Number bytes received (-1 if error)

Exchanging Data with Datagram Sockets

int count = sendto(sockid, msg, msgLen, flags, &foreignAddr, addrlen);

- msg, msgLen, flags, count: same with send()
- foreignAddr : structsockaddr, address of the destination
- addrLen: sizeof(foreignAddr)

int count = recvfrom(sockid, recvBuf, bufLen, flags,&clientAddr, addrlen);

- recvBuf, bufLen, flags, count: same with recv()
- clientAddr: structsockaddr, address of the client
- addrLen: sizeof(clientAddr)

- Specifying Addresses
  - ❖ Socket API defines a generic data type for addresses:

```
structsockaddr
{
                    unsigned short sa_family; /* Address family (e.g.
                                   AF_INET) */
               char sa_data[14]; /* Family-specific address
information*/
}
```

  - ❖ Particular form of the sockaddr used for TCP/IP addresses:

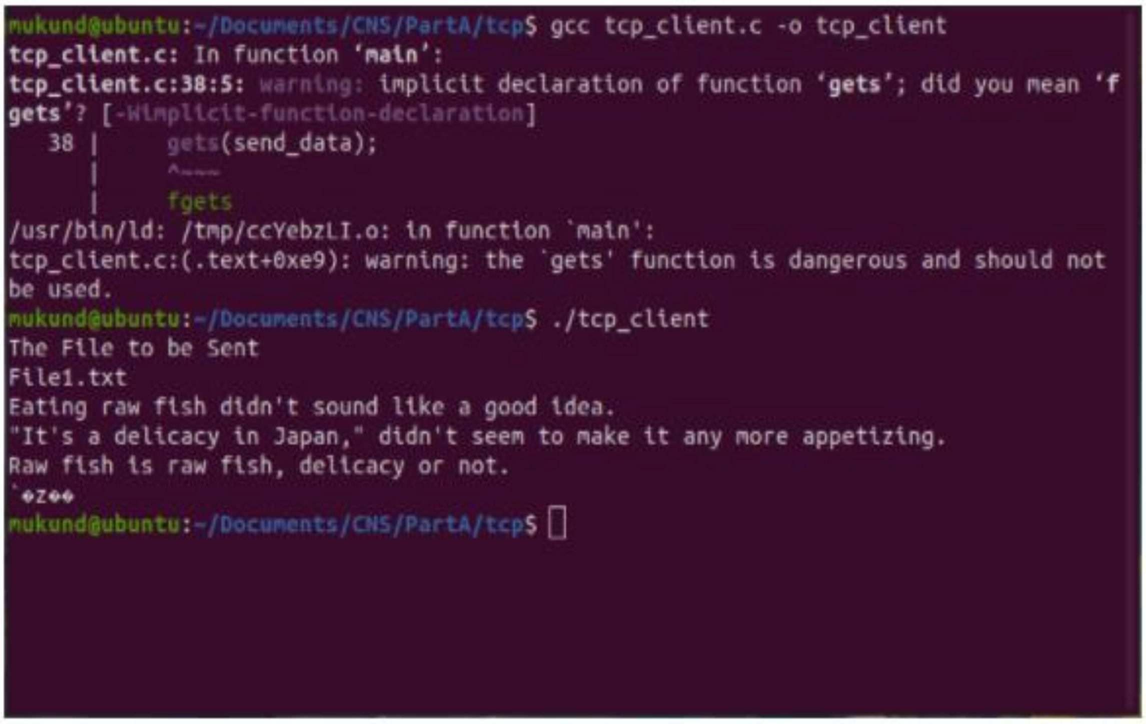
```
structin_addr
 {
            unsigned long s_addr; /* Internet address (32 bits) */
}
```

```
structsockaddr_in
{
            unsigned short sin_family; /* Internet protocol (AF_INET)
            */
            unsigned short sin_port; /* Address port (16 bits) */
            structin_addrsin_addr; /* Internet address (32 bits) */
            char sin_zero[8]
}
```

Expected Output:

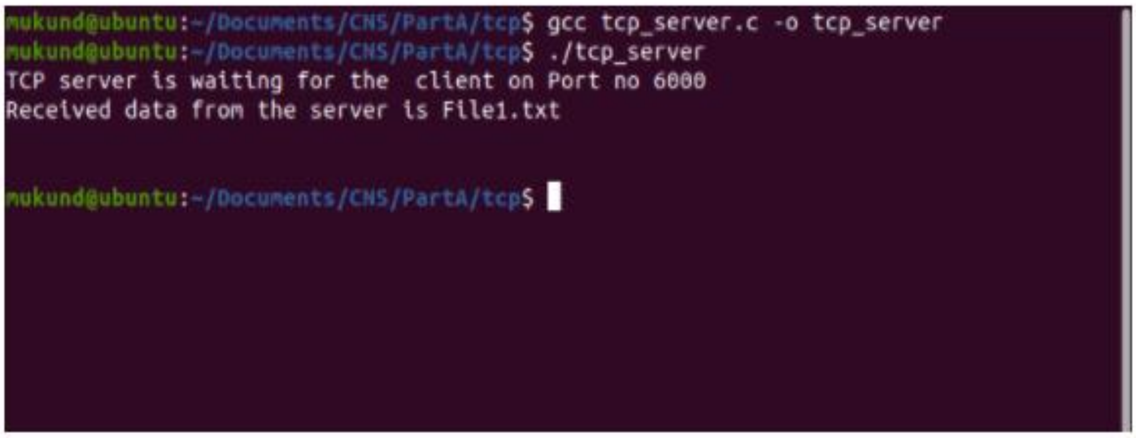
TCP Output

- TCP CLIENT



```
mukund@ubuntu:~/Documents/CNS/PartA/tcp$ gcc tcp_client.c -o tcp_client
tcp_client.c: In function ‘main’:
tcp_client.c:38:5: warning: implicit declaration of function ‘gets’; did you mean ‘f
gets’? [-Wimplicit-function-declaration]
   38 |     gets(send_data);
      |     ^~~~
      |     fgets
/usr/bin/ld: /tmp/ccYebzLI.o: in function `main':
tcp_client.c:(.text+0xe9): warning: the `gets' function is dangerous and should not
be used.
mukund@ubuntu:~/Documents/CNS/PartA/tcp$ ./tcp_client
The File to be Sent
File1.txt
Eating raw fish didn't sound like a good idea.
"It's a delicacy in Japan," didn't seem to make it any more appetizing.
Raw fish is raw fish, delicacy or not.
`◆z◆◆
mukund@ubuntu:~/Documents/CNS/PartA/tcp$
```

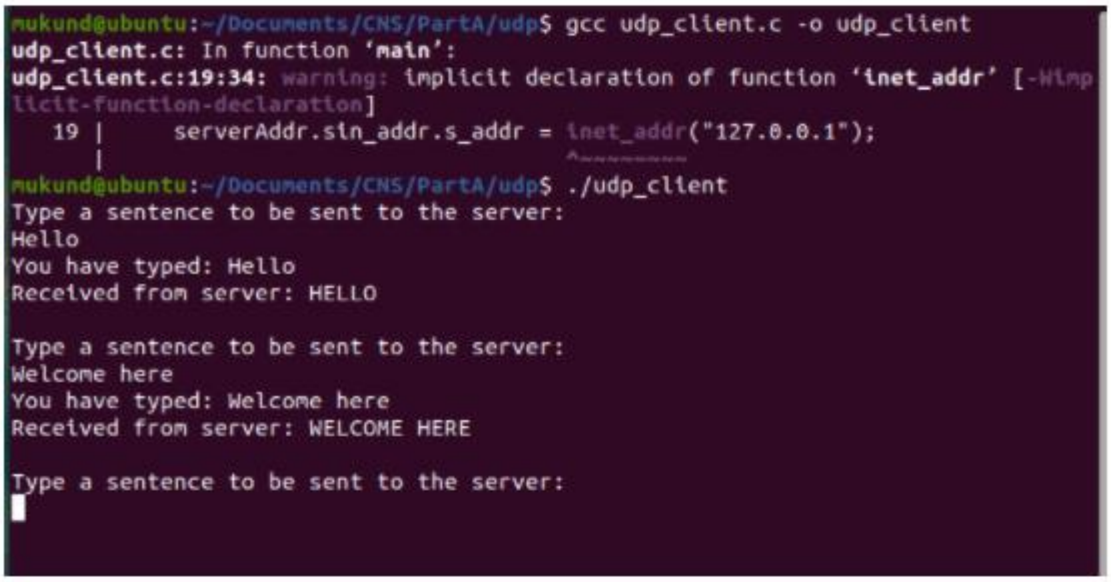
- TCP SERVER



```
mukund@ubuntu:~/Documents/CNS/PartA/tcp$ gcc tcp_server.c -o tcp_server
mukund@ubuntu:~/Documents/CNS/PartA/tcp$ ./tcp_server
TCP server is waiting for the  client on Port no 6000
Received data from the server is File1.txt


mukund@ubuntu:~/Documents/CNS/PartA/tcp$
```

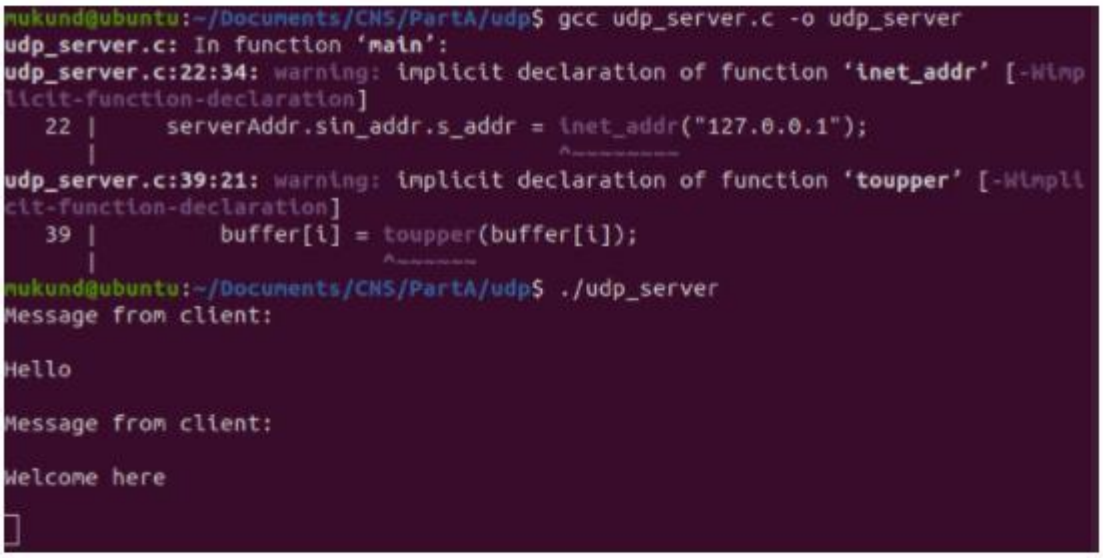
UDP Output

- UDP CLIENT



```
mukund@ubuntu:~/Documents/CNS/PartA/udp$ gcc udp_client.c -o udp_client
udp_client.c: In function ‘main’:
udp_client.c:19:34: warning: implicit declaration of function ‘inet_addr’ [-Wimp
licit-function-declaration]
   19 |     serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
      |                                  ^~~~~~~~~
mukund@ubuntu:~/Documents/CNS/PartA/udp$ ./udp_client
Type a sentence to be sent to the server:
Hello
You have typed: Hello
Received from server: HELLO

Type a sentence to be sent to the server:
Welcome here
You have typed: Welcome here
Received from server: WELCOME HERE

Type a sentence to be sent to the server:
```

- UDP SERVER

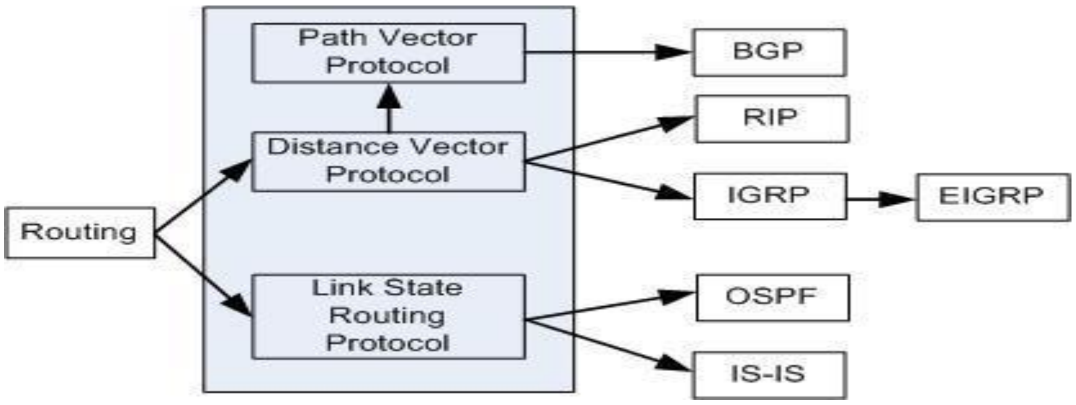


```
mukund@ubuntu:~/Documents/CNS/PartA/udp$ gcc udp_server.c -o udp_server
udp_server.c: In function ‘main’:
udp_server.c:22:34: warning: implicit declaration of function ‘inet_addr’ [-Wimp
licit-function-declaration]
   22 |     serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
      |                                  ^~~~~~~~~
udp_server.c:39:21: warning: implicit declaration of function ‘toupper’ [-Wimpli
cit-function-declaration]
   39 |         buffer[i] = toupper(buffer[i]);
      |                     ^~~~~~~
mukund@ubuntu:~/Documents/CNS/PartA/udp$ ./udp_server
Message from client:

Hello

Message from client:

Welcome here

```

Problem-5

Design and Implement a C solution for Routing Packets.

- Routing/Forwarding is the process to place the packet in its route to its destination. Forwarding requires a host or a router to have a routing table.
- When a host has a packet to send or when a router has received a packet to be forwarded, it looks at this table to find the route to the final destination.
- Forwarding Techniques
  - Next-Hop Method Versus Route Method
  - Network-Specific Method Versus Host-Specific Method
- UNICAST ROUTING PROTOCOLS
  - Intra-Domain
  - Inter-Domain
- Intra-Domain
  - Distance Vector Routing
  - Link-State Routing
  - Path Vector Routing



Distance Vector Routing

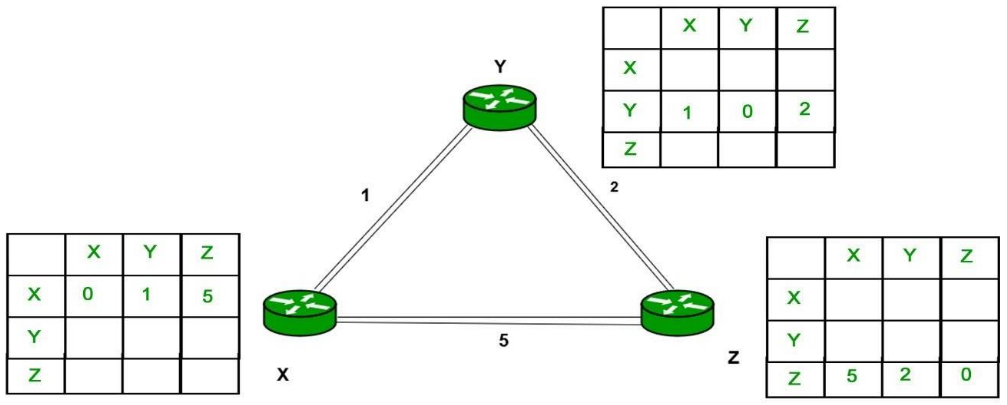
- **Distance-vector routing** protocol in data networks determines the best route for data packets based on **distance**. **Distance-vector routing** protocols measure the **distance** by the number of routers a packet must pass, one **router** counts as one hop.
- In distance vector routing, the least-cost route between any two nodes is the route with minimum distance.

- As the name implies, each node maintains a vector (table) of minimum distances to every node.
- The table at each node also guides the packets to the desired node by showing the next stop in the route
- Routing Table Mainly Consists of
    - TO
    - Cost
    - Next Hop
- Initialization
    - Each node can know only the distance between itself and its immediate neighbours, those directly connected to it.
    - Each node can send a message to the immediate neighbours and find the distance between itself and these neighbours.
- Sharing
    - The whole idea of distance vector routing is the sharing of information between neighbours.
    - Sharing of First two fields (TO and COST)
- Updating: When a node receives a two-column table from a neighbour, it needs to update its routing table. Updating takes three steps:
    - The receiving node needs to add the cost between itself and the sending node to each value in the second column. The logic is clear. If node 'C' claims that its distance to a destination is x miles, and the distance between 'A' and 'C' is y miles, then the distance between 'A' and that destination, via 'C', is x + y miles.
    - The receiving node needs to add the name of the sending node to each row as the third column if the receiving node uses information from any row. The sending node is the next node in the route.
    - The receiving node needs to compare each row of its old table with the corresponding row of the modified version of the received table.

▪ If the next-node entry is different, the receiving node chooses the row with the smaller cost. If there is a tie, the old one is kept.

▪ If the next-node entry is the same, the receiving node chooses the new row. For example, suppose node 'C' has previously advertised a route to node 'X' with distance 3. Suppose that now there is no path between 'C' and 'X'; node C now advertises this route with a distance of infinity. Node ‘A’ must not ignore this value even though its old entry is smaller. The old route does not exist anymore. The new route has infinity.

Example – Consider 3-routers X, Y and Z as shown in figure. Each router have their routing table. Every routing table will contain distance to the destination nodes.

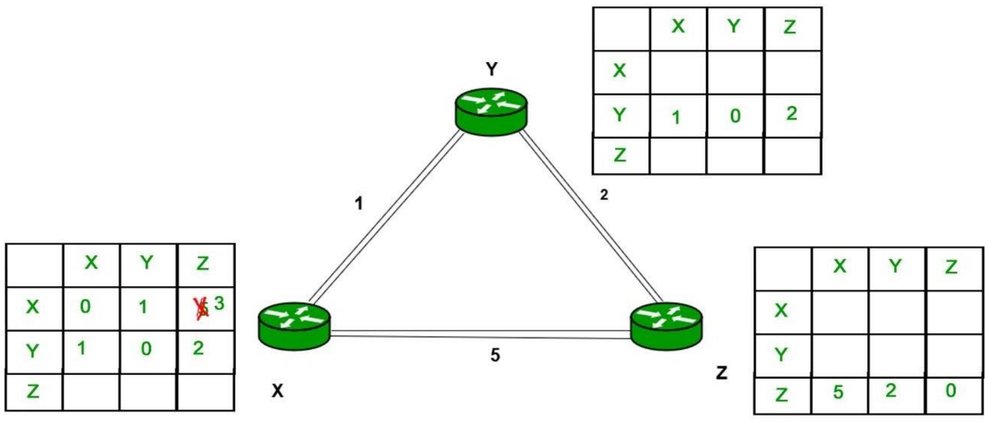


	X	Y	Z
X	0	1	5
Y			
Z			

	X	Y	Z
X			
Y	1	0	2
Z			

	X	Y	Z
X			
Y			
Z	5	2	0

Consider router X , X will share it routing table to neighbours and neighbours will share it routing table to it to X and distance from node X to destination will be calculated

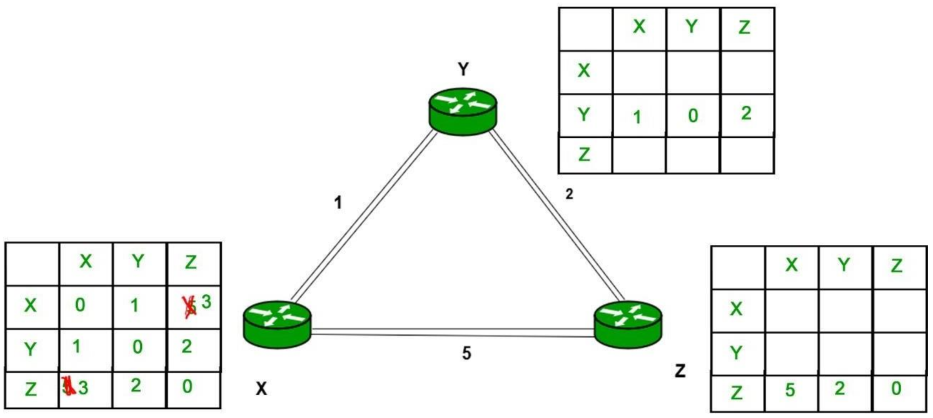


	X	Y	Z
X	0	1	~~5~~ 3
Y	1	0	2
Z			

	X	Y	Z
X			
Y	1	0	2
Z			

	X	Y	Z
X			
Y			
Z	5	2	0

Similarly for Z also –

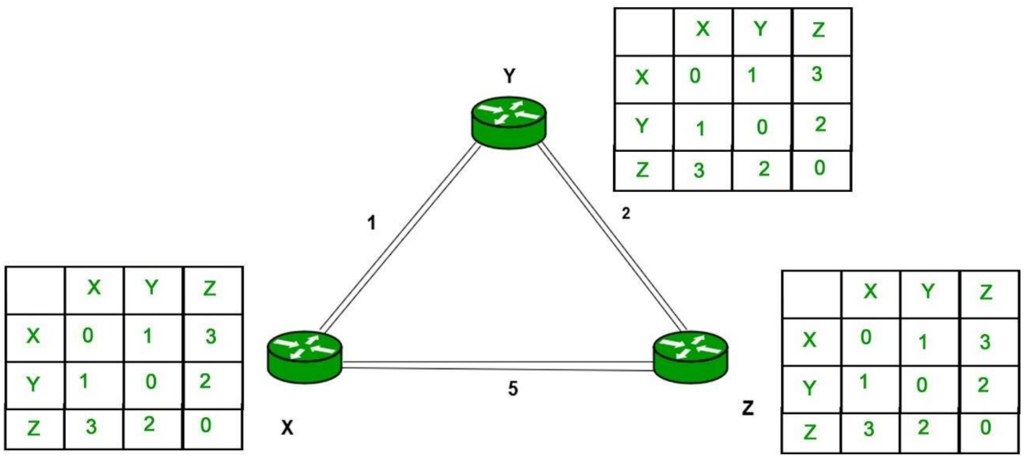


	X	Y	Z
X	0	1	~~5~~ 3
Y	1	0	2
Z	~~5~~ 3	2	0

	X	Y	Z
X			
Y	1	0	2
Z			

	X	Y	Z
X			
Y			
Z	5	2	0

Finally, the routing table for all –



	X	Y	Z
X	0	1	3
Y	1	0	2
Z	3	2	0

	X	Y	Z
X	0	1	3
Y	1	0	2
Z	3	2	0

	X	Y	Z
X	0	1	3
Y	1	0	2
Z	3	2	0

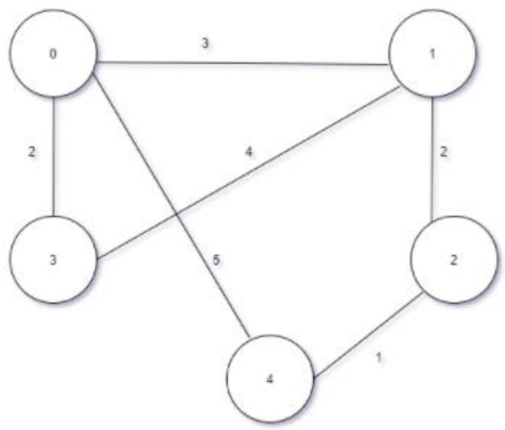
Algorithm

```
struct table
{
        unsigned cost[20];
        unsigned next_hop[20];
}router[10];

Cost_matrix=cost_of_each_link;

/*INITIALIZATION*/
If(cost of link (i,j) is present in cost matrix)
{
        Router[i].cost[j]=cost(i,j)
        Router[i].next_hop[j]=j;
}
Else
{
        Router[i].cost[j]=infinity
}
/*UPDATION*/ for
i from 1 to n-1:
     for each edge (u, v) with weight w in edges:
        if Router[u].cost[v] + w <Router[u].cost[v]:
Router[u].cost[v] := Router[u].cost[v] + w
```

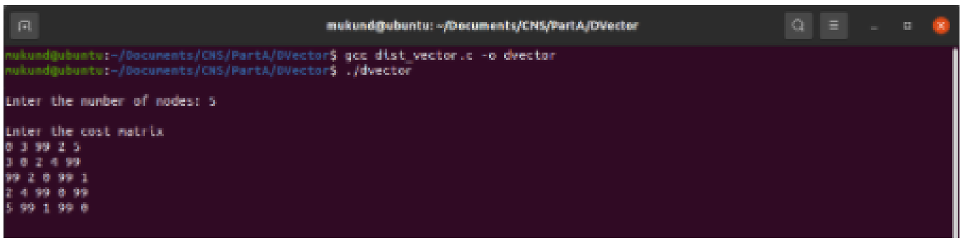
Expected Output:

Output Screenshots:

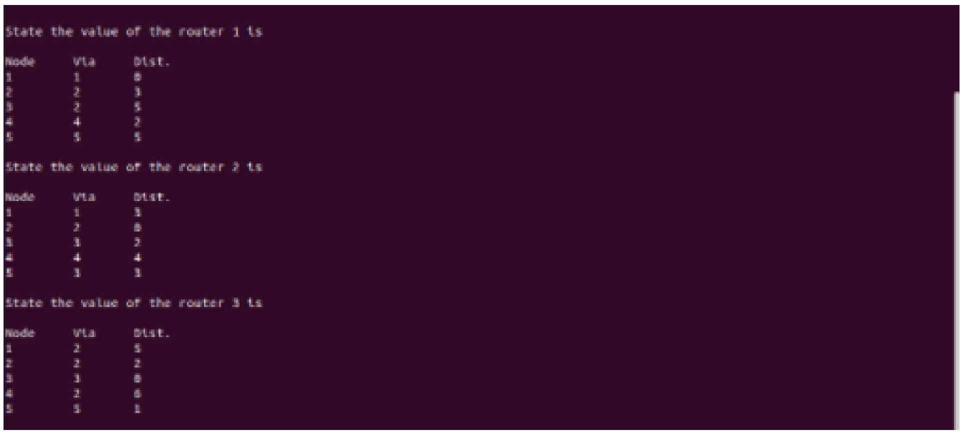
Graph taken:



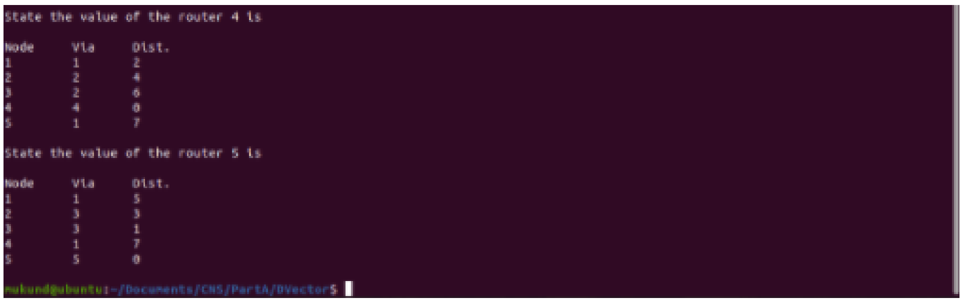
Distance vector routing table using Bellman-Ford algorithm:



```
mukund@ubuntu: ~/Documents/CNS/PartA/DVector
mukund@ubuntu:~/Documents/CNS/PartA/DVector$ gcc dist_vector.c -o dvector
mukund@ubuntu:~/Documents/CNS/PartA/DVector$ ./dvector

Enter the number of nodes: 5

Enter the cost matrix
0 3 99 2 5
3 0 2 4 99
99 2 0 99 1
2 4 99 0 99
5 99 1 99 0
```



```
State the value of the router 1 is

Node      Via      Dist.
1         1        0
2         2        3
3         2        5
4         4        2
5         5        5

State the value of the router 2 is

Node      Via      Dist.
1         1        3
2         2        0
3         3        2
4         4        4
5         3        3

State the value of the router 3 is

Node      Via      Dist.
1         2        5
2         2        2
3         3        0
4         2        6
5         5        1
```



```
State the value of the router 4 is

Node      Via      Dist.
1         1        2
2         2        4
3         2        6
4         4        0
5         1        7

State the value of the router 5 is

Node      Via      Dist.
1         1        5
2         3        3
3         3        1
4         1        7
5         5        0

mukund@ubuntu:~/Documents/CNS/PartA/DVector$ 
```

Problem-6

Design and Implement a C solution for congestion control in Transport Layer.

- A state occurring in network layer when the message traffic is so heavy that it slows down network response time.
- Congestion in a network may occur if the load on the network-the number of packets sent to the network-is greater than the capacity of the network-the number of packets a network can handle.
- Congestion control refers to the mechanisms and techniques to control the congestion and keep the load below the capacity.
- TCP, unlike UDP, takes into account congestion in the network.
- **Effects of Congestion**

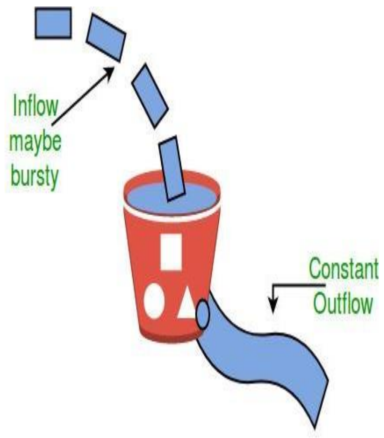
  As delay increases, performance decreases.

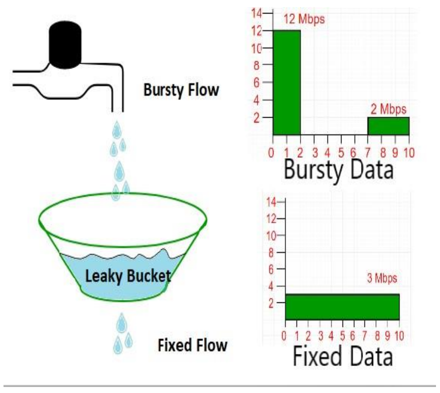
  If delay increases, retransmission occurs, making situation worse
- **Congestion control algorithms**

  Leaky Bucket Algorithm

  Token bucket Algorithm
- **Leaky Bucket Algorithm**

  Let us consider an example to understand
  Imagine a bucket with a small hole in the bottom. No matter at what rate water enters the bucket, the outflow is at constant rate. When the bucket is full of water, additional water entering spills over the sides



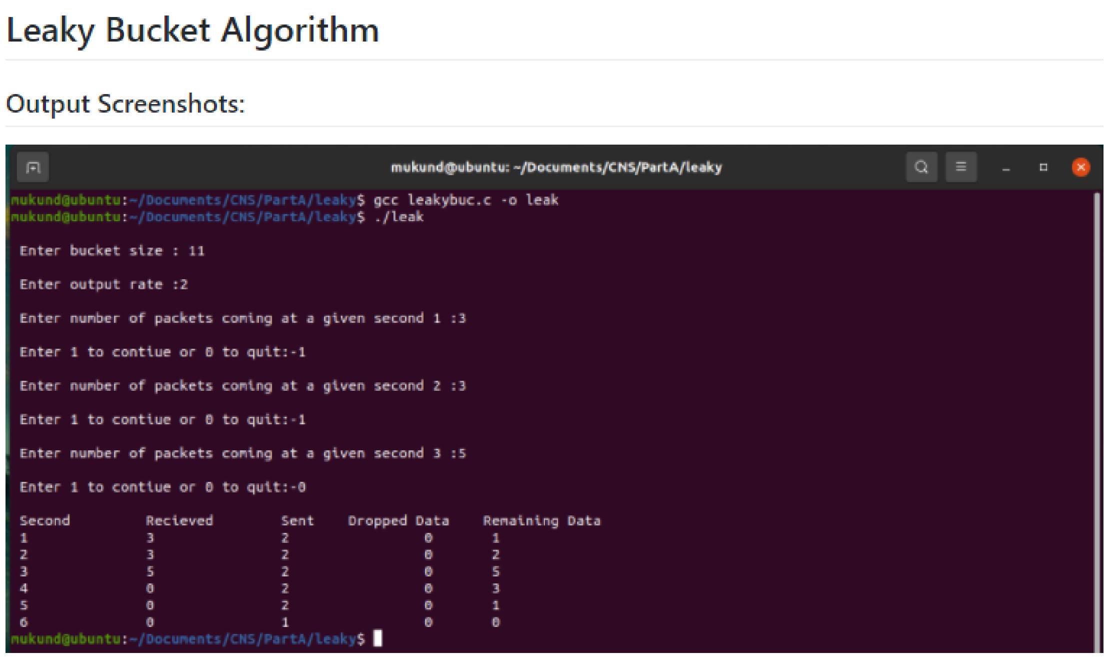


Similarly, each network interface contains a leaky bucket and the following steps are involved in leaky bucket algorithm:

1. When host wants to send packet, packet is thrown into the bucket.
2. The bucket leaks at a constant rate, meaning the network interface transmits packets at a constant rate.
3. Bursty traffic is converted to a uniform traffic by the leaky bucket.
4. In practice the bucket is a finite queue that outputs at a finite rate.

In the figure, we assume that the network has committed a bandwidth of 3 Mbps for a host. The use of the leaky bucket shapes the input traffic to make it conform to this commitment. In Figure the host sends a burst of data at a rate of 12 Mbps for 2 s, for a total of 24 Mbits of data. The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in 10 s. The leaky bucket smooths the traffic by sending out data at a rate of 3 Mbps during the same 10 s.
Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the leaky bucket may prevent congestion.

Expected Output:



Leaky Bucket Algorithm

Output Screenshots:

```
mukund@ubuntu: ~/Documents/CNS/PartA/leaky
mukund@ubuntu:~/Documents/CNS/PartA/leaky$ gcc leakybuc.c -o leak
mukund@ubuntu:~/Documents/CNS/PartA/leaky$ ./leak

 Enter bucket size : 11

 Enter output rate :2

 Enter number of packets coming at a given second 1 :3

 Enter 1 to contiue or 0 to quit:-1

 Enter number of packets coming at a given second 2 :3

 Enter 1 to contiue or 0 to quit:-1

 Enter number of packets coming at a given second 3 :5

 Enter 1 to contiue or 0 to quit:-0

 Second         Recieved        Sent    Dropped Data    Remaining Data
 1              3               2               0        1
 2              3               2               0        2
 3              5               2               0        5
 4              0               2               0        3
 5              0               2               0        1
 6              0               1               0        0
mukund@ubuntu:~/Documents/CNS/PartA/leaky$ 
```

Part B

Network Simulating Using NS-3

Introduction

The *ns-3* simulator is a discrete-event network simulator targeted primarily for research and educational use. The **ns-3 project**, started in 2006, is an open-source project developing *ns-3*. The purpose of this tutorial is to introduce new *ns-3* users to the system in a structured way. It is sometimes difficult for new users to glean essential information from detailed manuals and to convert this information into working simulations. In this tutorial, we will build several example simulations, introducing and explaining key concepts and features as we go.

As the tutorial unfolds, we will introduce the full *ns-3* documentation and provide pointers to source code for those interested in delving deeper into the workings of the system.

A few key points are worth noting at the onset:

- *ns-3* is open-source, and the project strives to maintain an open environment for researchers to contribute and share their software.
- *ns-3* is not a backwards-compatible extension of **ns-2**; it is a new simulator. The two simulators are both written in C++, but *ns-3* is a new simulator that does not support the *ns-2* APIs.

About NS-3

ns-3 has been developed to provide an open, extensible network simulation platform, for networking research and education. In brief, *ns-3* provides models of how packet data networks work and perform and provides a simulation engine for users to conduct simulation experiments. Some of the reasons to use *ns-3* include to perform studies that are more difficult or not possible to perform with real systems, to study system behaviour in a highly controlled, reproducible environment, and to learn about how networks work. Users will note that the available model set in *ns-3* focuses on modelling how Internet protocols and networks work, but *ns-3* is not limited to Internet systems; several users are using *ns-3* to model non-Internet-based systems.

Many simulation tools exist for network simulation studies. Below are a few distinguishing features of *ns-3* in contrast to other tools.

- *ns-3* is designed as a set of libraries that can be combined and also with other external software libraries. While some simulation platforms provide users with a single, integrated graphical user interface environment in which all tasks are carried out, *ns-3* is more modular in this regard. Several external animators and data analysis and visualization tools can be used with *ns-3*. However, users should expect to work at the command line and with C++ and/or Python software development tools.

- *ns-3* is primarily used on Linux or macOS systems, although support exists for BSD systems and for Windows frameworks that can build Linux code, such as Windows Subsystem for Linux, or Cygwin. Native Windows Visual Studio is not presently supported although a developer is working on future support. Windows users may also use a Linux virtual machine.
- *ns-3* is not an officially supported software product of any company. Support for *ns-3* is done on a best-effort basis on the ns-3-users forum

NS-3 Installation Steps

This option is for the new user who wishes to download and experiment with the most recently released and packaged version of *ns-3*. *ns-3* publishes its releases as compressed source archives, sometimes referred to as a tarball. A tarball is a particular format of software archive where multiple files are bundled together, and the archive is usually compressed. The process for downloading *ns-3* via tarball is simple; you just must pick a release, download it and uncompress it.

Let's assume that you, as a user, wish to build *ns-3* in a local directory called workspace. If you adopt the workspace directory approach, you can get a copy of a release by typing the following into your Linux shell (substitute the appropriate version numbers, of course)

```
$ cd
$ mkdir workspace
$ cd workspace
$ wget https://www.nsnam.org/release/ns-allinone-3.29.tar.bz2
$ tar xjf ns-allinone-3.29.tar.bz2
```

Notice the use above of the wget utility, which is a command-line tool to fetch objects from the web; if you do not have this installed, you can use a browser for this step.

Following these steps, if you change into the directory ns-allinone-3.29, you should see several files and directories

```
$ cd ns-allinone-3.29
$ ls
bake      constants.py   ns-3.29                          README
build.py  netanim-3.108  pybindgen-0.17.0.post58+ngcf00cc0  util.py
```

You are now ready to build the base *ns-3* distribution and may skip ahead to the section on building *ns-3*.

Just as we have seen point-to-point topology helper objects when constructing point-to-point topologies, we will see equivalent CSMA topology helpers in this section. The appearance and operation of these helpers should look quite familiar to you.

```
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
```

Program

In this section we are going to expand our mastery of *ns-3* network devices and channels to cover an example of a bus network. *ns-3* provides a net device and channel we call CSMA (Carrier Sense Multiple Access).

The *ns-3* CSMA device models a simple network in the spirit of Ethernet. A real Ethernet uses CSMA/CD (Carrier Sense Multiple Access with Collision Detection) scheme with exponentially increasing backoff to contend for the shared transmission medium. The *ns-3* CSMA device and channel models only a subset of this.

The actual code begins by loading module include files just as was done in the first.cc example.

```
#include "ns3/core-module.h"
```

In this case, you can see that we are going to extend our point-to-point example (the link between the nodes n0 and n1 below) by hanging a bus network off the right side. Notice that this is the default network topology since you can vary the number of nodes created on the LAN. If you set nCsma to one, there will be a total of two nodes on the LAN (CSMA channel) — one required node and one “extra” node. By default there are three “extra” nodes as seen below:

```
// Default Network Topology
//
//       10.1.1.0
// n0 -------------- n1   n2   n3   n4
//    point-to-point  |    |    |    |
//                    ================
//                      LAN 10.1.2.0
```

Then the ns-3 namespace is used and a logging component is defined.

```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");
```

The next step is to create two nodes that we will connect via the point-to-point link. The NodeContainer is used to do this just as was done in first.cc.

```
NodeContainer p2pNodes;
p2pNodes.Create (2);
```

Next, we declare another NodeContainer to hold the nodes that will be part of the bus (CSMA) network. First, we just instantiate the container object itself.

```
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
```

The next line of code Gets the first node (as in having an index of one) from the point-to-point node container and adds it to the container of nodes that will get CSMA devices. The node in question is going to end up with a point-to-point device *and* a CSMA device. We then create a number of “extra” nodes that compose the remainder of the CSMA network. Since we already have one node in the CSMA network – the one that will have both a point-to-point and CSMA net device, the number of “extra” nodes means the number nodes you desire in the CSMA section minus one.

The next bit of code should be quite familiar by now. We instantiate a PointToPointHelper and set the associated default Attributes so that we create a five megabit per second transmitter on devices created using the helper and a two millisecond delay on channels created by the helper.

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
```

We then instantiate a NetDeviceContainer to keep track of the point-to-point net devices and we Install devices on the point-to-point nodes.

We mentioned above that you were going to see a helper for CSMA devices and channels, and the next lines introduce them. The CsmaHelper works just like a PointToPointHelper, but it creates and connects CSMA devices and channels. In the case of a CSMA device and channel pair, notice that the data rate is specified by a *channel* Attribute instead of a device Attribute. This is because a real CSMA network does not allow one to mix, for example, 10Base-T and 100Base-T devices on a given channel. We first set the data rate to 100 megabits per second, and then set the speed-of-light delay of the channel to 6560 nano-seconds (arbitrarily chosen as 1 nanosecond per foot over a 2000 meter segment). Notice that you can set an Attribute using its native data type.

```
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
```

Just as we created a NetDeviceContainer to hold the devices created by the PointToPointHelper we create a NetDeviceContainer to hold the devices created by our CsmaHelper. We call the Install method of the CsmaHelper to install the devices into the nodes of the csmaNodes NodeContainer.

We now have our nodes, devices and channels created, but we have no protocol stacks present. Just as in the first.cc script, we will use the InternetStackHelper to install these stacks.

```
InternetStackHelper stack;
stack.Install (p2pNodes.Get (0));
stack.Install (csmaNodes);
```

Recall that we took one of the nodes from the p2pNodes container and added it to the csmaNodes container. Thus we only need to install the stacks on the remaining p2pNodes node, and all of the nodes in the csmaNodes container to cover all of the nodes in the simulation.

Just as in the first.cc example script, we are going to use the Ipv4AddressHelper to assign IP addresses to our device interfaces. First we use the network 10.1.1.0 to create the two addresses needed for our two point-to-point devices.

```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);
```

Recall that we save the created interfaces in a container to make it easy to pull out addressing information later for use in setting up the applications.

We now need to assign IP addresses to our CSMA device interfaces. The operation works just as it did for the point-to-point case, except we now are performing the operation on a container that has a variable number of CSMA devices — remember we made the number of CSMA devices changeable by command line argument. The CSMA devices will be associated with IP addresses from network number 10.1.2.0 in this case, as seen below.

```
address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);
```

Now we have a topology built, but we need applications. This section is going to be fundamentally similar to the applications section of first.cc but we are going to instantiate the server on one of the nodes that has a CSMA device and the client on the node having only a point-to-point device.

First, we set up the echo server. We create a UdpEchoServerHelper and provide a required Attribute value to the constructor which is the server port number. Recall that this port can be changed later using the SetAttribute method if desired, but we require it to be provided to the constructor.

```
UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

Recall that the csmaNodes NodeContainer contains one of the nodes created for the point-to-point network and nCsma “extra” nodes. What we want to get at is the last of the “extra” nodes. The zeroth entry of the csmaNodes container will be the point-to-point node. The easy way to think of this, then, is if we create one “extra” CSMA node, then it will be at index one of the csmaNodes container. By induction if we create nCsma “extra” nodes the last one will be at index nCsma. You see this exhibited in the Get of the first line of code.

The client application is set up exactly as we did in the first.cc example script. Again, we provide required Attributes to the UdpEchoClientHelper in the constructor (in this case the remote address and port). We tell the client to send packets to the server we just installed on the last of the “extra” CSMA nodes. We install the client on the leftmost point-to-point node seen in the topology illustration.

```
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

Since we have built an internetwork here, we need some form of internetwork routing. *ns-3* provides what we call global routing to help you out. Global routing takes advantage of the fact that the entire internetwork is accessible in the simulation and runs through the all of the nodes created for the simulation — it does the hard work of setting up routing for you without having to configure routers.

Basically, what happens is that each node behaves as if it were an OSPF router that communicates instantly and magically with all other routers behind the scenes. Each node generates link advertisements and communicates them directly to a global route manager which uses this global information to construct the routing tables for each node. Setting up this form of routing is a one-liner:

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Next we enable pcap tracing. The first line of code to enable pcap tracing in the point-to-point helper should be familiar to you by now. The second line enables pcap tracing in the CSMA helper and there is an extra parameter you haven't encountered yet.

```
pointToPoint.EnablePcapAll ("second");
csma.EnablePcap ("second", csmaDevices.Get (1), true);
```

The CSMA network is a multi-point-to-point network. This means that there can (and are in this case) multiple endpoints on a shared medium. Each of these endpoints has a net device associated with it. There are two basic alternatives to gathering trace information from such a network. One way is to create a trace file for each net device and store only the packets that are emitted or consumed by that net device. Another way is to pick one of the devices and place it in promiscuous mode. That single device then "sniffs" the network for all packets and stores them in a single pcap file. This is how tcpdump, for example, works. That final parameter tells the CSMA helper whether or not to arrange to capture packets in promiscuous mode.

In this example, we are going to select one of the devices on the CSMA network and ask it to perform a promiscuous sniff of the network, thereby emulating what tcpdump would do. If you were on a Linux machine you might do something like tcpdump -i eth0 to get the trace. In this case, we specify the device using csmaDevices.Get(1), which selects the first device in the container. Setting the final parameter to true enables promiscuous captures.

The last section of code just runs and cleans up the simulation just like the first.cc example.

```
 Simulator::Run ();
 Simulator::Destroy ();
 return 0;
}
```

To run this example, copy the second.cc example script into the scratch directory and use waf to build just as you did with the first.cc example. If you are in the top-level directory of the repository, you just type,

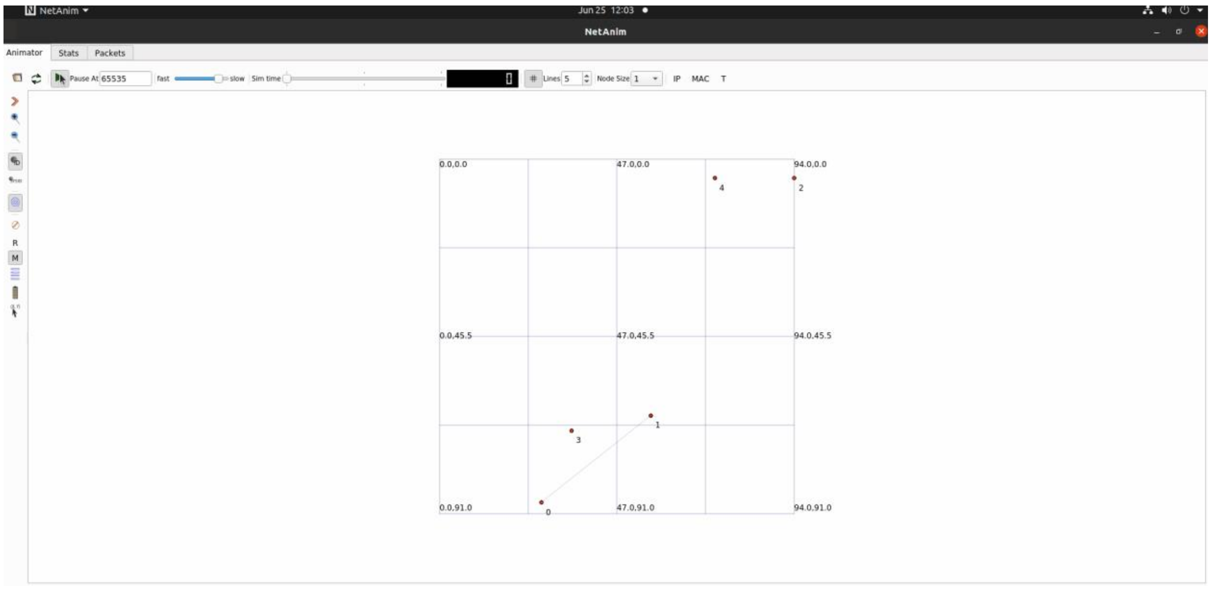
```
$ cp examples/tutorial/second.cc scratch/mysecond.cc
$ ./waf
```

Warning: We use the file second.cc as one of our regression tests to verify that it works exactly as we think it should to make your tutorial experience a positive one. This means that an executable named second already exists in the project. To avoid any confusion about what you are executing, please do the renaming to mysecond.cc suggested above.

If you are following the tutorial religiously (you are, aren't you) you will still have the NS_LOG variable set, so go ahead and clear that variable and run the program.

```
$ export NS_LOG=
$ ./waf --run scratch/mysecond
```

Expected Output:



References:

- Behrouz A. ForouzanData Communications and Networking, 4th Edition, Tata McGraw- Hill, 2006.
- Alberto Leon-garcia and IndraWidjaja Communication Networks, Second Edition, TataMcGraw Hill,2004
- William Stallings: Data and Computer Communication, 8th Edition, Pearson Education, 2007.
- Larry L. Peterson and Bruce S. David Computer Networks - A Systems Approach, 4th Edition, Elsevier, 2007.
- Wayne TomasiIntroduction to Data Communications and Networking, Pearson Education, 2005.
- https://nptel.ac.in/courses/106105082/
- https://nptel.ac.in/courses/106105183/
- https://www.nsnam.org/docs/tutorial/html/index.html
- https://www.nsnam.org/docs/release/3.9/tutorial/tutorial_31.html
- https://www.nsnam.org/docs/tutorial/html/building-topologies.html#building-a-bus-network-topology