Rashtreeya Sikshana Samithi Trust

R.V. College of Engineering®

(Autonomous Institution affiliated to VTU, Belagavi)

Department of Computer Science & Engineering

Bengaluru -560059

NETWORK PROGRAMMING AND SECURITY

(CS362IA)

VI Semester B.E

VI SEMESTER B.E

LAB INSTRUCTORS MANUAL

2025-2026

Prepared by: Dr. Deepamala N **Verified by: HOD, CSE**

Vision

To achieve leadership in the field of Computer Science & Engineering by strengthening fundamentals and facilitating interdisciplinary sustainable research to meet the ever growing needs of the society.

Mission

- To evolve continually as a centre of excellence in quality education in Computers and allied fields.
- To develop state of the art infrastructure and create environment capable for interdisciplinary research and skill enhancement.
- To collaborate with industries and institutions at national and international levels to enhance research in emerging areas.
- To develop professionals having social concern to become leaders in top-notch industries and/or become entrepreneurs with good ethics.

PROGRAM EDUCATIONAL OBJECTIVES (PEO's):

PEO1: Develop Graduates capable of applying the principles of mathematics, science, core engineering and Computer Science to solve real-world problems in interdisciplinary domains.

PEO2: To develop the ability among graduates to analyze and understand current pedagogical techniques, industry accepted computing practices and state-of-art technology.

PEO3: To develop graduates who will exhibit cultural awareness, teamwork with professional ethics, effective communication skills and appropriately apply knowledge of societal impacts of computing technology.

Schedule of Experiments

Sl. No	Name of Experiment	To be complete
	PART-A	
1	Implement a client and server communication using sockets programming.	Week1
2	Write a program to implement distance vector routing protocol for a simple topology of routers.	Week2
3	Write a program to implement error detection and Correction concept using Checksum and Hamming code.	Week3
4	Implement a simple multicast routing mechanism	Week4
5	Write a program to implement concurrent chat server that allows current logged in users to communicate one with other.	Week5
6	Implementation of concurrent and iterative echo server using both connection and connectionless socket system calls	Week6 &7
7	Implementation of remote command execution using socket system calls.	Week8
8	Write a program to encrypt and decrypt the data using RSA and Exchange the key securely using Diffie-Hellman Key exchange protocol	Week9
	PART-B	
1	Project topic finalization	Week1 and 2
2	Methodology and design	Week 3,4
3	Implementation	Week 5,6
4	Testing	Week 7

5	Demonstration and presentation	Week 8

Rubrics for NPS Lab

Each program is evaluated for 10 marks.

Rubrics for Lab Write-up and Execution rubrics (Max: 6 marks)

Sl. No	Criteria	Measuring methods	Excellent	Good	Poor
1	**Understanding of problem statement.** **(CO1)**	Observations	Student exhibits thorough understanding of requirements and applies suitable algorithm for the problem. **(2 M)**	Student has sufficient understanding of requirements and applies suitable algorithm for the problem. **(<2 M and >=1 M)**	Student does not have a clear understanding of requirements and is unable to apply suitable algorithm for the problem. **(0 M)**
2	**Execution** **(CO4)**	Observations	Student demonstrates the execution of the program with optimized code and shows performance efficiency. Appropriate validations with all test cases are handled. **(2 M)**	Student demonstrates the execution of the program without optimization of the code and shows performance efficiency with only few test cases. **(1 M)**	Student has not executed the program. **(0 M)**
3	**Results and Documentation** **(CO2)**	Observations	Documentation with appropriate comments and output is covered in data sheets and manual. **(2 M)**	Documentation with only few comments and only few output cases is covered in data sheets and manual. **(1 M)**	Documentation with no comments and no output cases is covered in data sheets and manual. **(0 M)**
Rubrics for Viva Voce (Max: 4 marks)					
1	**Conceptual Understanding** **(CO1)**	Viva Voce	Explains thoroughly the algorithm and the data structure used along with related concepts. **(2 M)**	Adequately explains the algorithms and data structures with related concepts **(1 M)**	Unable to explain the algorithm and data structure. **(0 M)**
2	**Use of appropriate Design Techniques** **(CO4)**	Viva Voce	Insightful explanation of appropriate design techniques for the given problem to derive solution. **(1 M)**	Sufficiently explains the use of appropriate design techniques for the given problem to derive solution. **(0.5 M)**	Unable to explain the design techniques for the given problem. **(0 M)**
3	**Communication of Concepts** **(CO3)**	Viva Voce	Communicates the concept used in problem solving well. **(1 M)**	Sufficiently communicates the concepts used in problem solving. **(0.5 M)**	Unable to communicate the concepts used in problem. **(0 M)**

INDEX

Program No.	Date of Submission	Lab Write-up and Execution marks(6 marks)			Viva voce Marks (4 marks)			Total Marks	Signature
		(2M)	(2M)	(2M)	(2M)	(1M)	(1M)		
PART A									
1									
2									
3									
4									
5									
6									
7									
8									
PART B									
1									
2									
3									
4									
Total Marks								120	

INTRODUCTION TO LAB EXPERIMENTS

1. Sockets

Sockets is one the inter process mechanism that allows processes on same or different machines to communicate with each other. It is similar to a file descriptor.

1.2 Applications

A Unix Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.

1.3 Types of socket

There are two types of Internet sockets available to users, Stream socket and Datagram socket.

1. Stream Socket(SOCK_STREAM)

A stream socket uses the Transmission Control Protocol (TCP) for sending messages. TCP provides an ordered and reliable connection between two hosts. This means that for every message sent, TCP guarantees that the message will arrive at the host in the correct order. This is achieved at the transport layer so that the application need not bother about it. Stream sockets are also called as connection oriented sockets.

2. Datagram Socket(SOCK_DGRAM)

A datagram socket uses the User Datagram Protocol (UDP) for sending messages. UDP is a much simpler protocol as it does not provide any of the delivery guarantees that TCP does. Messages, called datagrams, can be sent to another host without requiring any prior communication or a connection having been established. As such, using UDP can lead to lost messages or messages being received out of order. It is assumed that the application can tolerate an occasional lost message or that the application will handle the issue of retransmission. Datagram sockets are also called as connectionless sockets

1.4 Addressing

Every host on the internet is identified by its address called as Internet Protocol (IP) Address and the process within the host is identified by the port numbers.

1.4.1 IP Address

An IP version 4 (IPV4) address is of 32 bits interpreted as four octets. IP version 6 (IPV6) is the recent version of IP protocol and the address is of 128 bits and represented as 8 hextets.

1.4.2 Port Numbers

This address is used to identify the communicating processes. It is 16 bit number and is local for the connection.

1.4.3 Byte order

There are two types of byte ordering - Network Byte order same as Big- Endian and Host Byte Order same as Little – Endian. To have portability between different machines across the internet there are four functions to convert the byte ordering. The conversion to network byte order is done when the data goes out on the wire and convert to host byte order as data come in off the wire.

htons() host to network short

htonl() host to network long

ntohs() network to host short

ntohl() network to host long

1.5 Structures

The structsockaddr holds the socket address information for many types of sockets defined in the header file <sys/socket.h>. The members of this structures are described below

```
structsockaddr
{
unsigned short sa_family; // address family, AF_xxx
charsa_data[14]; // 14 bytes of protocol address
 };
```

sa_family can be a variety of things, AF_INET (IPv4) or AF_INET6 (IPv6). sa_data contains a destination address and port number for the socket.

Usually to deal with structsockaddr, programmers created a parallel structure: structsockaddr_in (“in” for “Internet”) to be used with IPv4.A pointer to a structsockaddr_in can be cast to a pointer to a structsockaddr and vice-versa.

The structure sockaddr_in is described below:

```
structsockaddr_in
{ shortintsin_family; // Address family, AF_INET
unsigned short intsin_port; // Port number
structin_addrsin_addr; // Internet address
unsigned char sin_zero[8]; // Same size as structsockaddr
};
```

This structure makes it easy to reference elements of the socket address. Note that sin_zero (which is included to pad the structure to the length of a structsockaddr) should be set to all zeros with the function memset().sin_family corresponds to sa_family in a structsockaddr and should be set to "AF_INET". The sin_port must be in Network Byte Order.

The structure in_addr is used to refer four byte IP address in Network Byte Order.

```
structin_addr {
 uint32_t s_addr; //  32-bit int (4 bytes)
};
```

1.6 Address Conversion Functions

There are couple of functions that help us in converting the dotted decimal form of IP addresses to the required structin_addr format and vice versa.

1. inet_pton()- converts an IP address in numbers-and-dots notation into structin_addr. The pton stands for "presentation to network".

2. inet_ntop()- converts the address that is in structin_addr format to numbers and dots notation. The ntop stands for "network to presentation".

1.7 Essential system calls for Inter Process communication using sockets

In any process communication, one process acts as a client and other as the server. This section discusses about the system calls used by the client and servers to establish the connection and to transfer the data.

1.7.1 Steps involved in establishing a socket on client side.

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

1. Socket( )- to get the socket descriptor. This system call returns a

Synopsis

```
#include<sys/types.h>
#include<sys/socket.h>
int socket(int domain, int type, int protocol);
```

Description

- domain is PF_INET or PF_INET6,
- type is SOCK_STREAM or SOCK_DGRAM,
- Protocol can be set to 0 to choose the proper protocol for the given type.

Return Value

This system call returns a socket descriptor(sockfd)

2. Connect( )- To establish the connection to server's socket.

Synopsis

```
#include<sys/types.h>
#include<sys/socket.h>
int connect(intsockfd, structsockaddr *serv_addr, intaddrlen);
```

Description

- sockfd is socket descriptor, as returned by the socket() call,
- serv_addr is a structsockaddr containing the destination port and IP address,
- addrlen is the length in bytes of the server address structure.

Return Value

The system call returns -1 on error and 0 on success.

3. send() and recv()

These two functions are for communicating over stream sockets or connected sockets. For unconnected datagram sockets, sendto() and recvfrom() are used.

Send()

Synopsis

```
#include<sys/types.h>
#include<sys/socket.h>
int send(intsockfd, const void *msg, intlen, int flags);
```

Description

- sockfd is the socket descriptor on which data has to be sent (whether it's the one returned by socket() or the one got with accept().)
- msg is a pointer to the data that has to be sent,
- len is the length of that data in bytes and

Flag is set to 0.

Return value

The system call returns number of bytes actually sent out.

Recv()

Synopsis

#include<sys/types.h>

#include<sys/socket.h>

intrecv(intsockfd, void *buf, intlen, int flags);

Description

- sockfd is the socket descriptor to read the data from.
- Buf is the buffer to read the information into,
- len is the length of the buffer and
- Flag is set to 0.

Return value

The system call returns number of bytes actually read into the buffer, or -1 on error.

1.7.2 Steps involved in establishing a socket on server side.

- Create a socket with the **socket()** system call.
- Bind the socket to an address using the **bind()** system call.
- Listen for connections with the **listen()** system call.
- Accept a connection with the **accept()** system call. This call typically blocks the connection until a client connects with the server.
- Send and receive data using the **send()** and recv**()** system calls.

1. bind()-binds the server process to the address

Once the socket is created at the server side, it should get associated with the port number on the local machine. The port number is used by the kernel to match an incoming packet to a certain processe's socket descriptor.

Synopsis

#include<sys/types.h>

#include<sys/socket.h>

int bind(intsockfd, structsockaddr *my_addr, intaddrlen);

Description

- sockfd is the socket file descriptor returned by socket().
- my_addr is a pointer to a structsockaddr that contains information about server's address, namely, port and IP address.

- addrlen is the length in bytes of that address.

Return Value

bind() also returns -1 on error and sets errno to the error's value.

2. listen( )- Block until a connection arrives.

This system call makes the process wait for incoming connections.

Synopsis

```
#include<sys/types.h>
#include<sys/socket.h>
int listen(intsockfd, int backlog);
```

Description

- sockfd is the usual socket file descriptor from the socket() system call.
- backlog is the number of connections allowed on the incoming queue. The incoming connections are wait the queue until they are accepted by accept() call. It is the limit on how many can queue up. Most systems silently limit this number to about 20.

Return Value

listen() returns -1 and sets errno on error.

3. Accept( ) – To accept the incoming connections.

This system call creats a new file descriptor to use for this connection.

Synopsis

```
#include<sys/types.h>
#include<sys/socket.h>
int accept(intsockfd, structsockaddr *addr, socklen_t *addrlen);
```

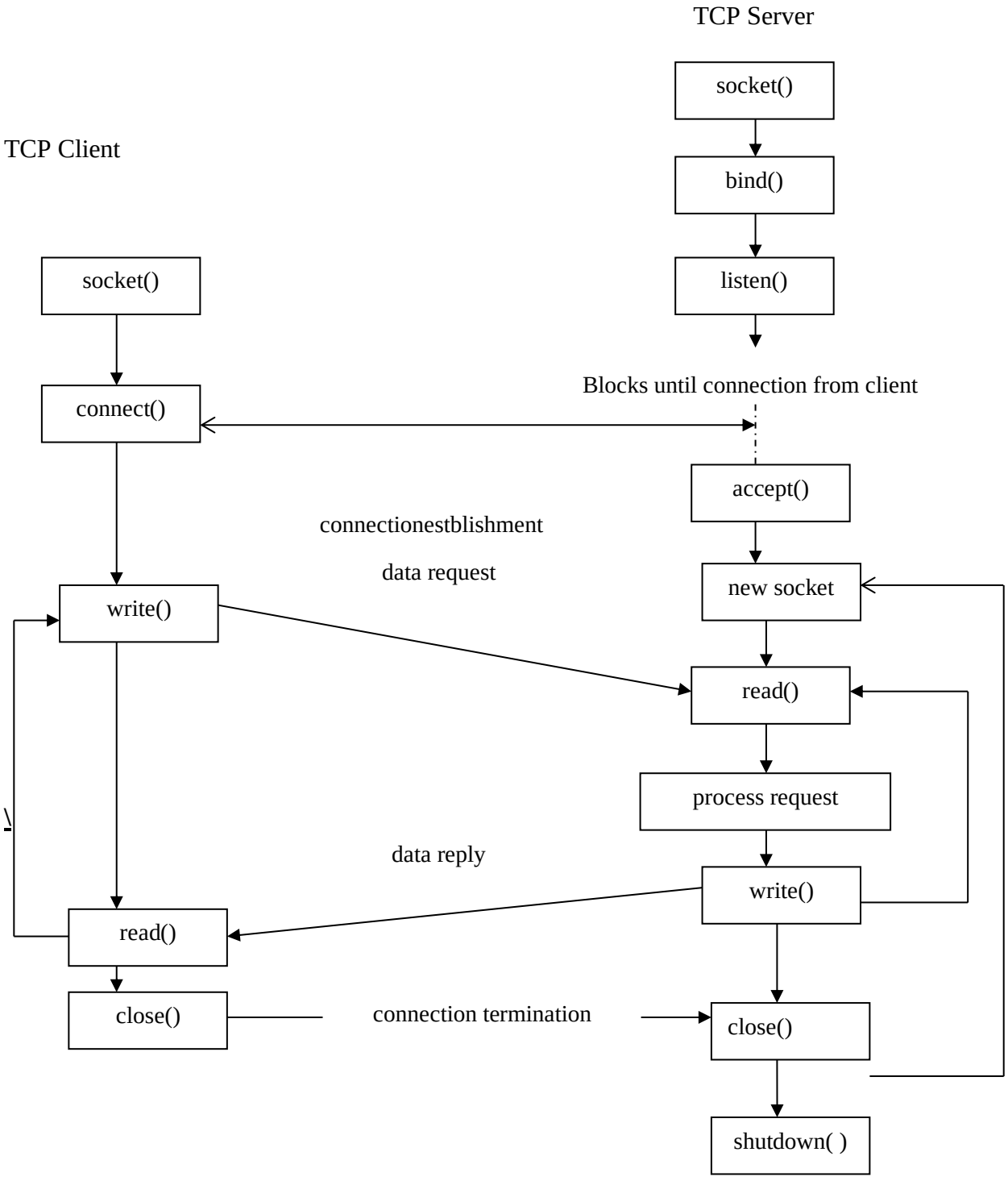
Description

- sockfd is the listen()ing socket descriptor.
- addr will usually be a pointer to a local structure.
- addrlen is a local integer variable that should be set to sizeof(structsockaddr_storage) before its address is passed to accept().

Return Value

On success returns a new socket descriptor and returns -1 and sets errno if an error occurs.

1.8 Flow diagram for client server interaction



2 Distance vector

Routing algorithm is a part of network layer software which is responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagram internally, this decision

must be made a new for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new established route is being set up. The latter case is sometimes called session routing, because a route remains in force for an entire user session.

Routing algorithms can be grouped into two major classes: adaptive and non adaptive. Non adaptive algorithms do not base their routing decisions on measurement or estimates of current traffic and topology. Instead, the choice of route to use to get from *I* to *J* (for all *I* and *J*) is compute in advance, offline, and downloaded to the routers when the network ids booted. This procedure is sometime called static routing.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every *T*sec, when the load changes, or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time).

Two algorithms in particular, distance vector routing and link state routing are the most popular. Distance vector routing algorithms operate by having each router maintain a table (i.e.,vector) giving the best known distance to each destination and which line to get there. These tables are updated by exchanging information with the neighbours.

The distance vector routing algorithm is sometimes called by other names, including the distributed Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the RIP and in early versions of DECnet and Novell's IPX. AppleTalk and Cisco routers use improved distance vector protocols.

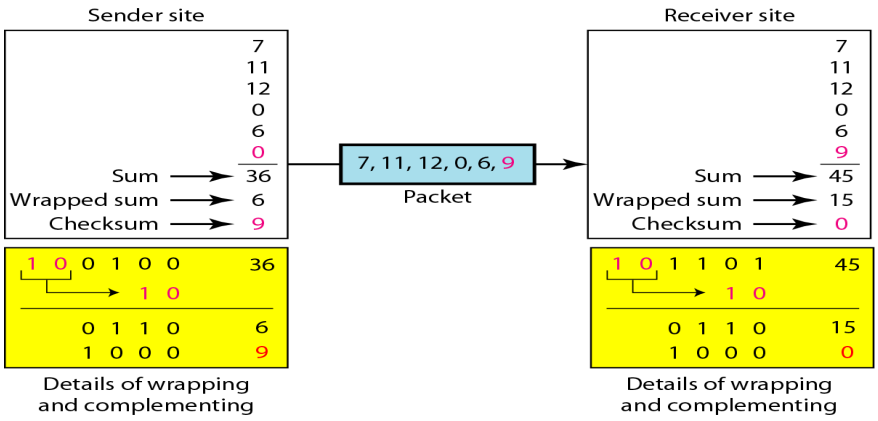
In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in subnet. This entry contains two parts: the preferred out going line touse for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

The router is assumed to know the "distance" to each of its neighbour. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special ECHO packets has the receiver just time stamps and sends back as fast as possible.

3. Internet checksum

A checksum is a count of the number of bits in a transmission unit that is included with the unit so that the receiver can check to see whether the same number of bits arrived.

The checksum is used in the Internet by several protocols although not at the data link layer.



Sender side:

1. The message is divided into 16-bit words.
2. The value of the checksum word is set to 0.
3. All words including the checksum are added using one's complement addition.
4. The sum is complemented and becomes the checksum.
5. The checksum is sent with the data.

Receiver side:

1. The message (including checksum) is divided into 16-bit words.
2. All words are added using one's complement addition.
3. The sum is complemented and becomes the new checksum.
4. If the value of checksum is 0, the message is accepted; otherwise, it is rejected.

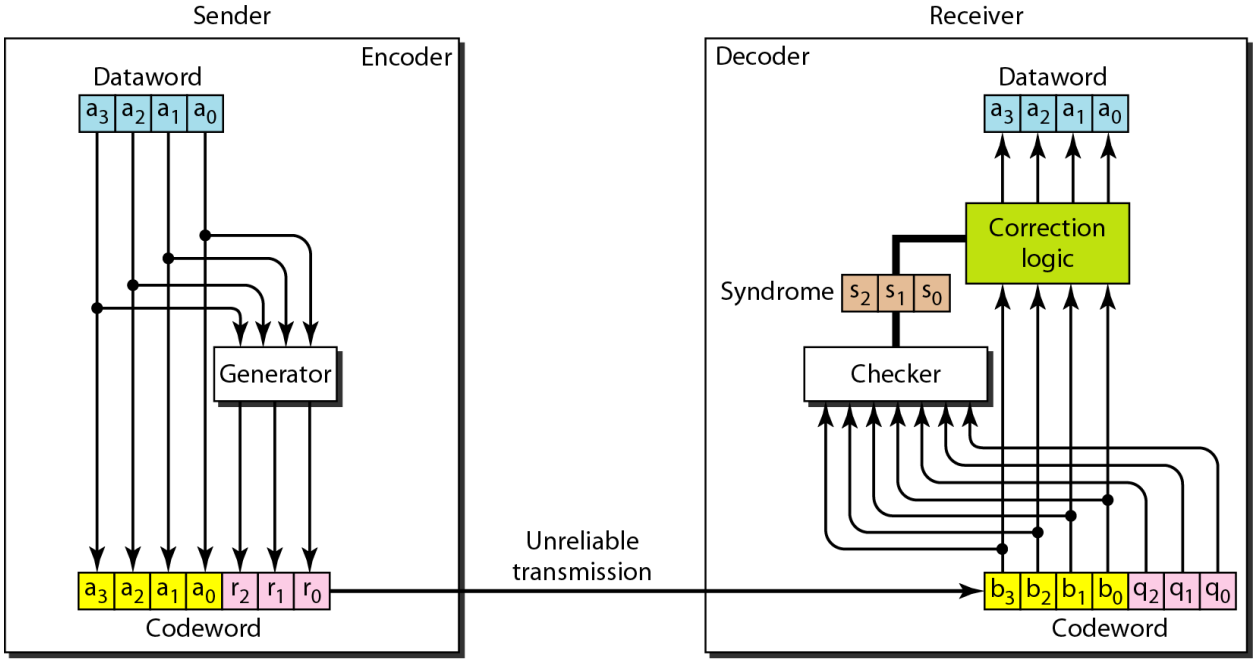
4. Hamming code

Hamming code is a set of error-correction codes that are used to **detect and correct the errors** that occurs during data transmission.

4.1 Redundant bits

Redundant bits are extra binary bits that are generated and added to the information-carrying bits of data transfer to ensure that no bits were lost during the data transfer. The number of redundant bits can be calculated using the formula: $2^r >= m+r+1$, where r represents redundant bits, m represents data bits. For example if m=4 then according to formula the number of redundant bits to be added to data bits is 3.

4.2 Structure of encoder and decoder for hamming code



In the encoding process the data bits are appended with the redundant bits(ro,r1 and r2). The value of redundant bits are determined and transmitted. At the receiver side the codeword is passes through the checker to determine the syndrome bits(s0,s1 and s2). If the bits are zero it indicates there is no error in the received codeword, otherwise the binary value of the syndrome bits represents the position of error in the codeword.

4.3 Determining the redundant bits

To establish the relationship between the redundant bits and the data bits, the position of the redundant bits must be determined. The redundant bits are placed at the positions corresponding to power of 2(2^0,2^1,2^2…). The value of these redundant bits is determined by performing addition modulo 2 of various position bits.

Consider an example: m=4 (1011) hence r=3(r2, r1,r0). The resulting codeword is of 7 bits.

Bit Positions	7	6	5	4	3	2	1
Power of 2 positions				2^2		2^1	2^0

Positions of redundant bits				r2		r1	r0
Position of data bits	d3	d2	d1		d0		
Codeword	d3	d2	d1	r2	d0	r1	r0
For example	1	0	1	r2	1	r1	r0

The value of r1 is determined by performing arithmetic addition of the bits whose binary representation of the position has least significant bit 1(bits at position 3,5, and 7). Similarly r2 is determined with bit position having 1 at second position from least significant bit(3,6,7) and r3 is determined with numbers having bit position having 1 at third position from least significant bit(5,6,7)

r1= d0+d1+d3 mod 2
= 1+1+1
=1

r2= d0+d2+d3 mod 2
= 1+0+1
=0

r3= d1+d2+d3 mod 2
= 1+0+1
=0

The transmitted codeword is 1011 001

Let us assume sixth bit is in error. The received codeword is 1111 001. To detect error, the codeword is sent to the checker, the checker computes the syndrome bits same as redundant bits at the sender side.

s1= d0+d1+d3+r1 mod 2
= 1+1+1+1
=0

s2= d0+d2+d3+r2 mod 2
= 1+1+1+0
=1

s3= d1+d2+d3+r3 mod 2
= 1+1+1+0
=1

Since the syndrome is 110, the error is in sixth bit is in error. If syndrome bits are 000 then the codeword is error free.

5 Congestion control algorithms

The congesting control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination.

In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn

the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer.

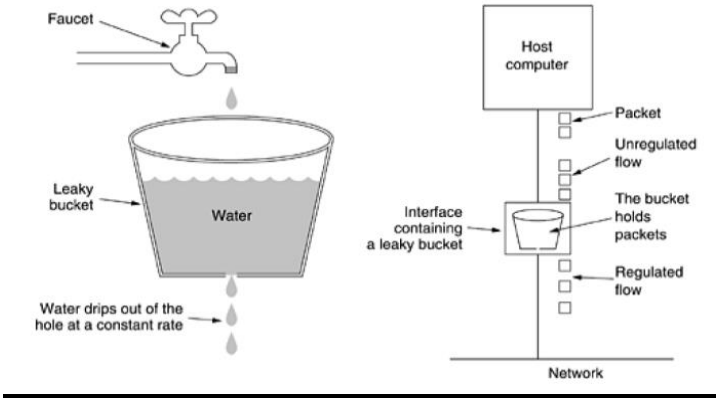
Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called traffic shaping.

5.1 Leaky bucket algorithm.

Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.

The leaky-bucket implementation is used to control the rate at which traffic is sent to the network. A leaky bucket provides a mechanism by which bursty traffic can be shaped to present a steady stream of traffic to the network, as opposed to traffic with erratic bursts of low-volume and high-volume flows.The algorithm can be conceptually understood as follows:



- Consider a bucket with a hole in the bottom.
- If packets arrive, they are placed into the bucket. If the bucket is full, packets are discarded.
- Packets in the bucket are sent at a **constant rate**, equivalent to the size of the hole in the

6. Multicast routing mechanism

Multicast communications refers to one-to-many or many-to many communications. If there is lot of information, that should be transmitted to various hosts over an internet, then Multicast is the solution.

6.1 Multicast Address

In multicasting, "Class D Address" is used. Every IP datagram whose destination address starts with "1110" is an IP Multicast datagram.The remaining 28 bits identify the multicast "group" the datagram is sent to.

6.2 Sending multicast datagram

Multicast traffic is handled at the transport layer with UDP, as TCP provides point−to−point connections and is not feasible for multicast traffic**.**In principle, an application just needs to open a UDP socket and fill with a class D multicast address the destination address where it wants to send data to. However, there are some operations that a sending process must be able to control.

6.3 Multicast Programming

Several extensions to the programming API are needed in order to support multicast. All of them are handled via two system calls:

setsockopt() (used to pass information to the kernel) and

getsockopt() (to retrieve information regarded multicast behavior).

The addition consists on a new set of options (multicast options) that are passed to these system calls, that the kernel must understand. The following are the setsockopt()/getsockopt() function prototypes:

intgetsockopt(int s, int level, intoptname, void* optval, int* optlen);

intsetsockopt(int s, int level, intoptname, const void* optval, intoptlen);

- The first parameter, s, is the socket the system call applies to. For multicasting, it must be a socket of the family AF_INET and its type may be either SOCK_DGRAM or SOCK_RAW. The most common use is with SOCK_DGRAM sockets.
- The second one, level, identifies the layer that is to handle the option, message or query. So, SOL_SOCKET is for the socket layer, IPPROTO_IP for the IP layer, etc... For multicast programming, level will always be IPPROTO_IP.optname identifies the option we are setting/getting. Its value (either supplied by the program or returned by the kernel) is optval. The optnames involved in multicast programming are the following:

	setsockopt()	getsockopt()
IP_MULTICAST_LOOP	yes	yes
IP_MULTICAST_TTL	yes	yes
IP_MULTICAST_IF	yes	yes

IP_ADD_MEMBERSHIP	yes	no
IP_DROP_MEMBERSHIP	yes	no

IP_ADD_MEMBERSHIP.

It is necessary to tell the kernel which multicast groups the receivers are interested in. If no process is interested in a group, packets destined to it that arrive to the host are discarded. In order to inform the kernel of the interests and, thus, become a member of that group, it is required to fill a ip_mreq structure which is passed later to the kernel in the optval field of the setsockopt() system call.

The ip_mreq structure (taken from /usr/include/linux/in.h) has the following members:

```
structip_mreq{ structin_addrimr_multiaddr; /* IP multicast address of group */
structin_addrimr_interface; /* local IP address of interface */
};
```

The first member, imr_multiaddr, holds the group address that receiver want to join. The memberships are also associated with interfaces, not just groups. This is the reason to provide a value for the second member: imr_interface. This way, if it is a multihomed host, then it can join the same group in several interfaces. It is always possible to fill this last member with the wildcard address (INADDR_ANY) and then the kernel will deal with the task of choosing the interface. With this structure filled (say you defined it as: structip_mreqmreq;) a calltosetsockopt() is:

```
setsockopt (socket, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));
```

Notice that a host can join several groups to the same socket, not just one. The limit to this is IP_MAX_MEMBERSHIPS and, as of version 2.0.33, it has the value of 20.

IP_DROP_MEMBERSHIP.

The process is quite similar to joining a group:

```
structip_mreqmreq;
setsockopt (socket, IPPROTO_IP, IP_DROP_MEMBERSHIP, &mreq, sizeof(mreq));
```

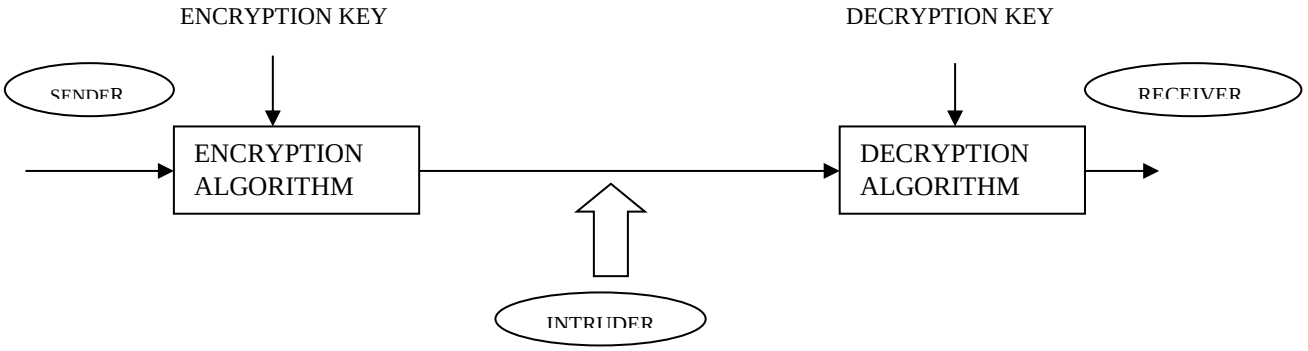
wheremreq is the same structure with the same data used when joining the group.

If the imr_interface member is filled with INADDR_ANY, the first matching group is dropped. When a socket is closed, all memberships associated with it are dropped by the kernel. The same occurs if

the process that opened the socket is killed. Both ADD_MEMBERSHIP and DROP_MEMBERSHIP are nonblocking operations. They should return immediately indicating either success or failure.

7. Encryption and Decryption

The message to be sent through an unreliable medium is known as plaintext. For security purpose it is encrypted before sending over the medium. The encrypted message is known as ciphertext, which is received at the other end of the medium and decrypted to get back the original plaintext message. There are various cryptography algorithms and can be divided into two broad categorize - Symmetric key cryptography and Public key cryptography. The following figure shows a simple cryptography model



7.1 Public key Cryptography

In public key cryptography, there are two keys: a private key and a public key. The public key is announced to the public, where as the private key is kept by the receiver. The sender uses the public key of the receiver for encryption and the receiver uses his private key for decryption.

7.2 RSA

The most popular public-key algorithm is the RSA (named after their inventors Rivest, Shamir and Adleman)

Key features of the RSA algorithm are given below:

1. Public key algorithm that performs encryption as well as decryption based on number theory
2. Variable key length; long for enhanced security and short for efficiency (typical 512 bytes)
3. Variable block size, smaller than the key length
4. The private key is a pair of numbers (d, n) and the public key is also a pair of numbers (e, n)

8. Key exchange algorithms

To preserve data confidentiality during transmission, secure file transfer protocols like FTPS, HTTPS, and SFTP have to encrypt the data through what is known as symmetric encryption. This kind of encryption requires the two communicating parties to have a shared key in order for them to encrypt and decrypt messages. In the real world, the two communicating parties would likely be geographically separated by long distances. The key can't just be sent through ordinary methods because anyone who gets hold of it would then be able to decrypt all the files that the two parties would be sending to one another. A key exchange method should be easy to use, secure, and highly scalable.

8.1 Diffiehallman key exchange algorithm

Diffie-Hellman is a way of establishing a shared secret between two endpoints (parties). Consider following example to understand the algorithm. Let Alice and Bob wants to communicate with each other without John knowing their communication. To start, Alice and Bob decide publicly (John will also get a copy) on two prime numbers, ***g*** and ***n***. Generally ***g*** is a small prime number and ***n*** is quite large, usually 2000 or more commonly 4000 bits long. So now Alice, Bob and John all know these numbers.

Alice now decides secretly on another number, ***a***. and Bob decides secretly on a number, ***b***. Neither Alice nor Bob send these numbers, they are kept to themselves. Alice performs a calculation, ***g^a mod n***, let this be A, since it comes from ***a***. Bob then performs ***g^b mod n*** let this be ***B***.

Alice sends Bob, ***A***, and Bob sends Alice, ***B***. Note John now has 4 numbers, ***A, B, g*** and ***n*** but not ***a*** or ***b***. Alice takes Bob's ***B*** and performs ***B^a mod n***. Similarly, Bob takes Alice's ***A*** and performs ***A^b mod n***. This results in the same number i.e. ***B^a mod n = A^b mod n***. They now have a shared number. John can't figure out what these numbers are from the numbers he's got.

1. Implement a client and server communication using sockets programming.

Algorithm (Client Side)

1. Start.
2. Create a socket using socket() system call.
3. Connect the socket to the address of the server using connect() system call.
4. Send the filename of required file using send() system call.
5. Read the contents of the file sent by server by recv() system call.
6. Stop.

Algorithm (Server Side)

1. Start.

2. Create a socket using socket() system call.
3. Bind the socket to an address using bind() system call.
4. Listen to the connection using listen() system call.
5. accept connection using accept()
6. Receive filename and transfer contents of file with client.
7. Stop.

PROGRAM

```
/*Server*/
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/stat.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#include<fcntl.h>
#include <arpa/inet.h>
int main()
{
intcont,create_socket,new_socket,addrlen,fd;
intbufsize = 1024;
char *buffer = malloc(bufsize);
charfname[256];
structsockaddr_in address;
if ((create_socket = socket(AF_INET,SOCK_STREAM,0)) > 0)
printf("The socket was created\n");
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(15001);
if (bind(create_socket,(structsockaddr *)&address,sizeof(address)) == 0)
printf("Binding Socket\n");
listen(create_socket,3);
addrlen = sizeof(structsockaddr_in);
new_socket = accept(create_socket,(structsockaddr *)&address,&addrlen);
if (new_socket> 0)
    {
printf("The Client %s is Connected...\n",
inet_ntoa(address.sin_addr) );
```

```
    }
recv(new_socket,fname, 255,0);
printf("A request for filename %s Received..\n", fname);
if ((fd=open(fname, O_RDONLY))<0)
    {perror("File Open Failed"); exit(0);}
while((cont=read(fd, buffer, bufsize))>0) {
send(new_socket,buffer,cont,0);
    }
printf("Request Completed\n");
close(new_socket);
return close(create_socket);
}
/*Client*/
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
int main(intargc,char *argv[])
{
intcreate_socket;
intbufsize = 1024, cont;
char *buffer = malloc(bufsize);
charfname[256];
structsockaddr_in address;
if ((create_socket = socket(AF_INET,SOCK_STREAM,0)) > 0)
printf("The Socket was created\n");
address.sin_family = AF_INET;
address.sin_port = htons(15001);
inet_pton(AF_INET,argv[1],&address.sin_addr);
if (connect(create_socket,(structsockaddr *) &address, sizeof(address)) == 0)
printf("The connection was accepted with the server %s...\n",argv[1]);
printf("Enter The Filename to Request : "); scanf("%s",fname);
send(create_socket, fname, sizeof(fname), 0);
printf("Request Accepted... Receiving File...\n\n");
printf("The contents of file are...\n\n");
while((cont=recv(create_socket, buffer, bufsize, 0))>0) {
write(1, buffer, cont);
  }
```

```
printf("\nEOF\n");
return close(create_socket);
}
```

OUTPUT

```
SERVER
exam@dell:~$ gcc -o server  server.c
exam@dell:~$ ./server
The socket was created
Binding Socket
The Client 127.0.0.1 is Connected...
A request for filename test.txt Received..
Request Completed
CLIENT
exam@dell:~$ gcc -o client client.c
exam@dell:~$ ./client 127.0.0.1
The Socket was created
The connection was accepted with the server 127.0.0.1...
Enter The Filename to Request : test.txt
Request Accepted... Receiving File...
The contents of file are...
hello
EOF
```

2.Write a program to implement routing protocol for a simple topology of routers.

Algorithm

Input: Graph and a source vertex src.

Output: Shortest distance to all vertices from *src*. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

1. Initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.

2. Ccalculate shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.

   …..**a)** Do following for each edge u-v

   ……If dist[v] >dist[u] + weight of edge uv, then update dist[v]

   …….dist[v] = dist[u] + weight of edge uv

3. This step reports if there is a negative weight cycle in graph.

   Do following for each edge u-v

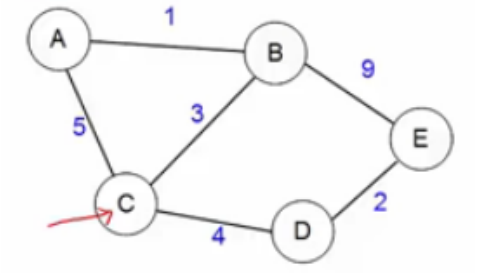
   ……If dist[v] >dist[u] + weight of edge uv, then “Graph contains negative weight cycle”

PROGRAM

```
#include<stdio.h>
int A[10][10], n, d[10], p[10];
voidBellmanFord(int s){
      inti,u,v;
      for(i=1;i<n;i++){
            for(u=0;u<n;u++){
                  for(v=0;v<n;v++){
                        if(d[v] > d[u] + A[u][v]){
                              d[v] = d[u] + A[u][v];
                              p[v] = u;
                        }
                  }
            }
      }
      for(u=0;u<n;u++){
            for(v=0;v<n;v++){
                  if(d[v] > d[u] + A[u][v]){
                        printf("Negative Edge");
                  }
            }
      }
}

int main(){
      printf("Enter the no. of vertices : ");
      scanf("%d",&n);
      printf("Enter the adjacency matrix\n");
      inti,j;
      for(i=0;i<n;i++)
            for(j=0;j<n;j++)
                  scanf("%d",&A[i][j]);

      int source;
      for(source=0;source<n;source++){


            for(i=0;i<n;i++){
                  d[i] = 999;
```

```
                p[i] = -1;
            }
            d[source] = 0;

            BellmanFord(source);

            printf("Router %d\n",source);

            for(i=0;i<n;i++){
                if(i != source){
                    j = i;
                    while(p[j] != -1){
                        printf("%d <- ",j);
                        j = p[j];
                    }
                }
                printf("%d\tCost %d\n\n",source,d[i]);
            }
        }

        return 0;
}
```

OUTPUT




```
exam@dell:~$ gccdvr.c
exam@dell:~$ ./a.out
Enter the no. of vertices : 5
Enter the adjacency matrix
0 1 5 999 999
1 0 3 999 9
5 3 0 4 999
999 999 4 0 2
999 9 999 2 0
Router 0
0     Cost 0

1 <- 0      Cost 1

2 <- 1 <- 0      Cost 4

3 <- 2 <- 1 <- 0 Cost 8

4 <- 1 <- 0      Cost 10

Router 1
0 <- 1      Cost 1

1     Cost 0

2 <- 1      Cost 3

3 <- 2 <- 1      Cost 7

4 <- 1      Cost 9
```

```
Router 2
0 <- 1 <- 2      Cost 4

1 <- 2     Cost 3

2     Cost 0

3 <- 2     Cost 4

4 <- 3 <- 2      Cost 6

Router 3
0 <- 1 <- 2 <- 3 Cost 8

1 <- 2 <- 3      Cost 7

2 <- 3     Cost 4

3     Cost 0

4 <- 3     Cost 2

Router 4
0 <- 1 <- 4      Cost 10

1 <- 4     Cost 9

2 <- 3 <- 4      Cost 6

3 <- 4     Cost 2

4     Cost 0
```

3 a).Write a program to implement error detection algorithm.

Note- Use IP header fields obtained using wireshark packet analyzer tool

Algorithm at sender side

Set the value of the checksum field to 0.

Divide the header into 16-bit words •

Add all segments using one's complement arithmetic

The final result is complemented to obtain the checksum.

Algorithm at receiver side

Divide header into 16-bit words, adds them, and complement's the results

All zero's => accept datagram, else reject.

PROGRAM

```
#include<stdio.h>
unsigned fields[10];
unsigned short checksum()
{
      inti;
      int sum=0;
      printf("Enter IP header information in 16 bit words\n");

      for(i=0;i<9;i++)
      {
            printf("Field %d\n",i+1);
            scanf("%x",&fields[i]);
            sum=sum+(unsigned short)fields[i];
            while (sum>>16)
            sum = (sum & 0xFFFF) + (sum >> 16);
      }
      sum=~sum;
      return (unsigned short)sum;
}

int main()
{
unsigned short result1, result2;
//Sender
result1=checksum();
printf("\n COmputed Checksum at  sender %x\n", result1);
//Receiver
result2=checksum();
printf("\n COmputed Checksum at  receiver %x\n", result2);
if(result1==result2)
printf("No error");
else
printf("Error in data received");

}
```

OUTPUT

Values obtained from wireshark:

```
4500 003c 1c46 4000 4006 b1e6 ac10 0a63 ac10 0a0c
```

- 45 corresponds to the first two fields in the header ie4 corresponds to the IP version and 5 corresponds to the header length. Since header length is described in 4 byte words so actual header length comes out to be 5*4=20 bytes.
- 00 corresponds to TOS or the type of service. This value of TOS indicated normal operation.

- 003c corresponds to total length field of IP header. So in this case the total length of IP packet is 60.
- 1c46 corresponds to the identification field.
- 4000 can be divided into two bytes. These two bytes (divided into 3 bits and 13 bits respectively) correspond to the flags and fragment offset of IP header fields.
- 4006 can be divided into 40 and 06. The first byte 40 corresponds to the TTL field and the byte 06 corresponds to the protocol field of the IP header. 06 indicates that the protocol is TCP.
- be16 corresponds to the checksum which is set at the source end (which sent the packet).
- The next set of bytes ac10 and 0a0c correspond to the source IP address and the destination IP address in the IP header.

```
exam@dell:~$ gcccchecksum.c
exam@dell:~$ ./a.out
Enter IP header information in 16 bit words
Field 1 4500 Field 2 003c Field 3 1c46 Field 4 4000 Field 5 4006 Field 6
ac10 Field 7 0a63 Field 8 ac10 Field 9 0a0c
COmputed Checksum at  sender b1e6
Enter IP header information in 16 bit words
Field 1 4500 Field 2 003c Field 3 1c46 Field 4 4000 Field 5 4006 Field 6
ac10 Field 7 0a63 Field 8 ac10 Field 9 0a0c
COmputed Checksum at  receiver b1e6
No error
```

3 b).Write a program to illustrate error correction concept.

Algorithm sender side.

1. Read the m bit data word.
2. Determine number of redundant bits required using the formula $2^m>=m+r+1$.
3. Establish the relationship between redundant bits and data bits.
4. Transmit the code word.

Algorithm receiver side.

1. Receive the codeword.
2. Determine the syndrome bits.
3. Check for error or no error.
4. If error, correct the respective bit.

PROGRAM

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

```
int main()
{
      int a[4],b[4],r[3],s[3],i,q[3],c[7];
      printf("\nenter 4 bit data word:\n");
      for(i=3;i>=0;i--)
      {
            scanf("%d",&a[i]);
      }
      r[0]=(a[3]+a[1]+a[0])%2;
      r[1]=(a[0]+a[2]+a[3])%2;
      r[2]=(a[1]+a[2]+a[3])%2;

      printf("\n\nthe 7bit hamming code word: \n");
      for(i=3;i>=0;i--)
      {
            printf("%d\t",a[i]);
      }
      for(i=2;i>=0;i--)
      {
            printf("%d\t",r[i]);
      }
      printf("\n");
      printf("\nenter the 7bit recievedcodeword: ");

      for(i=7;i>0;i--)
      scanf ("%d",&c[i]);

b[3]=c[7];b[2]=c[6];b[1]=c[5];b[0]=c[4];
r[2]=c[3];r[1]=c[2];r[0]=c[1];

//calculating syndrome bits
s[0]=(b[0]+b[1]+b[3]+r[0])%2;
s[1]=(b[0]+b[2]+b[3]+r[1])%2;
s[2]=(b[1]+b[2]+b[3]+r[2])%2;
printf("\nsyndrome is: \n");

for(i=2;i>=0;i--)
{
printf("%d",s[i]);
}
if((s[2]==0) && (s[1]==0) && (s[0]==0))
printf("\n RECIEVED WORD IS ERROR FREE\n");
if((s[2]==1)&&(s[1]==1)&&(s[0]==1))
{
      printf("\nError in received codeword, position-  7th bit from
right\n");
      if(c[7]==0)
      c[7]=1;
      else
      c[7]=0;
      printf("\n Corrected codeword is\n");
      for(i=7;i>0;i--)
      printf("%d \t", c[i]);
```

```
}
if((s[2]==1)&&(s[1]==1)&&(s[0]==0))
{
      printf("\nError in received codeword, Position-  6th bit from
right\n");
      if(c[6]==0)
      c[6]=1;
      else
      c[6]=0;
      printf("\n Corrected codeword is\n");
      for(i=7;i>0;i--)
      printf("%d \t", c[i]);
}
if((s[2]==1)&&(s[1]==0)&&(s[0]==1))
{
      printf("\nError in received codeword, Position-  5th bit from
right\n");
      if(c[5]==0)
      c[5]=1;
      else
      c[5]=0;
      printf("\n Corrected codeword is\n");
      for(i=7;i>0;i--)
      printf("%d \t", c[i]);
}
if((s[2]==1)&&(s[1]==0)&&(s[0]==0))
{
      printf("\nError in received codeword, Position-  4th bit from
right\n");
      if(c[4]==0)
      c[4]=1;
      else
      c[4]=0;
      printf("\n Corrected codeword is\n");
      for(i=7;i>0;i--)
      printf("%d \t", c[i]);
}
if((s[2]==0)&&(s[1]==1)&&(s[0]==1))
{
      printf("\nError in received codeword, Position-  3rd bit from
right\n");
      if(c[3]==0)
      c[3]=1;
      else
      c[3]=0;
      printf("\n Corrected codeword is\n");
      for(i=7;i>0;i--)
      printf("%d \t", c[i]);
}
if((s[2]==0)&&(s[1]==1)&&(s[0]==0))
{
      printf("\nError in received codeword, Position-  2nd bit from
right\n");
      if(c[2]==0)
      c[2]=1;
```

```
      else
      c[2]=0;
      printf("\n Corrected codeword is\n");
      for(i=7;i>0;i--)
      printf("%d \t", c[i]);
}
if((s[2]==0)&&(s[1]==0)&&(s[0]==1))
{
      printf("\nError in received codeword, Position-  1st bit from
right\n");
      if(c[1]==0)
      c[1]  =1;
      else
      c[1]=0;
      printf("\n Corrected codeword is\n");
      for(i=7;i>0;i--)
      printf("%d \t", c[i]);
}
return(1);
}//End of Hamming code program*/
```

OUTPUT

RUN 1

```
exam@dell:~$ gcchamming.c
exam@dell:~$ ./a.out
enter 4 bit data word:
1 0 1 1
The 7bit hamming code word:
1     0     1     1     0     0     1
Enter the 7bit recievedcodeword:
1 0 1 1 0 0 1
syndrome is:
000
RECIEVED WORD IS ERROR FREE
```

RUN 2

```
exam@dell:~$ ./a.out
enter 4 bit data word:
1 0 1 1
the 7bit hamming code word:
1     0     1     1     0     0     1
enter the 7bit recievedcodeword: 1 1 1 1 0 0 1
syndrome is:
110
recieved word is error, position-  6th bit from right
 Corrected codeword is
1     0     1     1     0     0     1
```

4. Implement a simple multicast routing mechanism.

Algorithm server side:

1. Create an AF_INET, SOCK_DGRAM type socket.

2. Initialize a sockaddr_in structure with the destination group IP address and port number.
3. Set the IP_MULTICAST_LOOP socket option according to whether the sending system should receive a copy of the multicast datagrams that are transmitted.
4. Set the IP_MULTICAST_IF socket option to define the local interface over which you want to send the multicast datagrams.
5. Send the datagram.

Algorithm Client side:

1. Create an AF_INET, SOCK_DGRAM type socket.
2. Set the SO_REUSEADDR option to allow multiple applications to receive datagrams that are destined to the same local port number.
3. Use the bind() verb to specify the local port number. Specify the IP address as INADDR_ANY in order to receive datagrams that are addressed to a multicast group.
4. Use the IP_ADD_MEMBERSHIP socket option to join the multicast group that receives the datagrams. When joining a group, specify the class D group address along with the IP address of a local interface. The system must call the IP_ADD_MEMBERSHIP socket option for each local interface receiving the multicast datagrams.
5. Receive the datagram.

PROGRAM

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

#define HELLO_PORT 12345
#define HELLO_GROUP "225.0.0.37"
#define MSGBUFSIZE 25

main(intargc, char *argv[])
{
structsockaddr_inaddr;
intfd, nbytes,addrlen;
structip_mreqmreq;
```

```
charmsgbuf[MSGBUFSIZE];

u_int yes=1;            /*** MODIFICATION TO ORIGINAL */

     /* create what looks like an ordinary UDP socket */
if ((fd=socket(AF_INET,SOCK_DGRAM,0)) < 0) {
      perror("socket");
      exit(1);
     }


/**** MODIFICATION TO ORIGINAL */
    /* allow multiple sockets to use the same PORT number */
if (setsockopt(fd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(yes)) < 0) {
perror("Reusing ADDR failed");
exit(1);
       }
/*** END OF MODIFICATION TO ORIGINAL */

     /* set up destination address */
memset(&addr,0,sizeof(addr));
addr.sin_family=AF_INET;
addr.sin_addr.s_addr=htonl(INADDR_ANY); /* N.B.: differs from sender */
addr.sin_port=htons(HELLO_PORT);

     /* bind to receive address */
if (bind(fd,(structsockaddr *) &addr,sizeof(addr)) < 0) {
      perror("bind");
      exit(1);
     }

     /* use setsockopt() to request that the kernel join a multicast
group */
mreq.imr_multiaddr.s_addr=inet_addr(HELLO_GROUP);
mreq.imr_interface.s_addr=htonl(INADDR_ANY);
if (setsockopt(fd,IPPROTO_IP,IP_ADD_MEMBERSHIP,&mreq,sizeof(mreq)) < 0) {
      perror("setsockopt");
      exit(1);
     }

     /* now just enter a read-print loop */
while (1) {
      addrlen=sizeof(addr);
      if ((nbytes=recvfrom(fd,msgbuf,MSGBUFSIZE,0,
                        (structsockaddr *) &addr,&addrlen)) < 0) {
      perror("recvfrom");

        }
      puts(msgbuf);
     }
}

/*Sender*/
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include <string.h>
#include <stdio.h>
#include<stdlib.h>

#define HELLO_PORT 12345
#define HELLO_GROUP "225.0.0.37"

main(intargc, char *argv[])
{
structsockaddr_inaddr;
intfd, cnt;
structip_mreqmreq;
char *message="RVCE-CSE";

     /* create what looks like an ordinary UDP socket */
if ((fd=socket(AF_INET,SOCK_DGRAM,0)) < 0) {
      perror("socket");
      exit(1);
     }

     /* set up destination address */
memset(&addr,0,sizeof(addr));
addr.sin_family=AF_INET;
addr.sin_addr.s_addr=inet_addr(HELLO_GROUP);
addr.sin_port=htons(HELLO_PORT);

     /* now just sendto() our destination! */
while (1) {
      if (sendto(fd,message,sizeof(message),0,(structsockaddr *) &addr,
            sizeof(addr)) < 0) {
      perror("sendto");
      exit(1);
        }
      sleep(1);
     }
}

Output
SENDER:
exam@dell:~$ gcc -o sender sender.c
exam@dell:~$ ./sender   // Starts sending messages


LISTENER1
exam@dell:~$ gcc -o listen listener.c
exam@dell:~$ ./listen  // Listens to messages by joining multicast group
RVCE-CSE
RVCE-CSE
RVCE-CSE
RVCE-CSE

LISTENER2
```

```
exam@dell:~$ ./listen  // Listens to messages by joining multicast group
RVCE-CSE
RVCE-CSE
RVCE-CSE
RVCE-CSE
```

5. Write a program to implement concurrent chat server that allows current logged in users to communicate one with other.

```
/*Client*/
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
```

```
void str_cli(FILE *fp, int sockfd)
{
   int bufsize = 1024, cont;
   char *buffer = malloc(bufsize);
fputs("Me:",stdout);
while(fgets(buffer,bufsize,fp)!=NULL)
{

        send(sockfd, buffer, sizeof(buffer), 0);

  if((cont=recv(sockfd, buffer, bufsize, 0))>0) {
        fputs("Server:",stdout);
    fputs(buffer,stdout);
        //bzero(buffer,10240);
  }
        fputs("Me:",stdout);
}
  printf("\nEOF\n");
}
int main(int argc,char *argv[])
{
  int create_socket;

  //char fname[256];
  struct sockaddr_in address;
  if ((create_socket = socket(AF_INET,SOCK_STREAM,0)) > 0)
  printf("The Socket was created\n");
  address.sin_family = AF_INET;
  address.sin_port = htons(16001);
  inet_pton(AF_INET,argv[1],&address.sin_addr);
  if (connect(create_socket,(struct sockaddr *) &address, sizeof(address)) ==
0)
        printf("The connection was accepted with the server %s...\n",argv[1]);
   else
        printf("error in connect \n");
  //printf("Enter The Filename to Request : "); scanf("%s",fname);
  //send(create_socket, fname, sizeof(fname), 0);
  //printf("Request Accepted... Receiving File...\n\n");
  //printf("The contents of file are...\n\n");

  str_cli(stdin,create_socket);


  return close(create_socket);
}
```

/*server*/

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

void str_echo(int connfd)
```

```
{
      int n;
      int bufsize = 10240;
      char *buffer = malloc(bufsize);
       //printf("inside the function");
       while((n=recv(connfd, buffer, bufsize, 0))>0) {
              fputs("client:",stdout);
              fputs(buffer,stdout);
              fputs("Me:",stdout);
  if(fgets(buffer,bufsize,stdin)!=NULL)
       {
       send(connfd, buffer, sizeof(buffer), 0);

       }
bzero(buffer,10240);

}}
int main()
{
  int cont,listenfd,connfd,addrlen,addrlen2,fd,pid,addrlen3;

  //char fname[256];
  struct sockaddr_in address,cli_address;
  if ((listenfd = socket(AF_INET,SOCK_STREAM,0)) > 0) //sockfd
    printf("The socket was created\n");
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(16001);
printf("The address before bind %s  ...\n",inet_ntoa(address.sin_addr) );
    if (bind(listenfd,(struct sockaddr *)&address,sizeof(address)) == 0)
    printf("Binding Socket\n");
       printf("The address after bind %s  ...\n",inet_ntoa(address.sin_addr)
);

    listen(listenfd,3);
     printf("server is listening\n");
       //server local address
     getsockname(listenfd,(struct sockaddr *)&address,&addrlen3);
     printf("The server's local address %s ...and port
%d\n",inet_ntoa(address.sin_addr),htons(address.sin_port));
for(;;){
    addrlen = sizeof(struct sockaddr_in);
    connfd = accept(listenfd,(struct sockaddr *)&cli_address,&addrlen);
    //printf("The address %s  ...\n",inet_ntoa(address.sin_addr) );
       addrlen2 = sizeof(struct sockaddr_in);
       int i = getpeername(connfd,(struct sockaddr *)&cli_address,&addrlen);

            printf("The Client  %s is Connected...on port
%d\n",inet_ntoa(cli_address.sin_addr),htons(cli_address.sin_port));

     if((pid=fork())==0)
     {
        printf("inside child\n");
         close(listenfd);

          str_echo(connfd);
          exit(0);
       }
```

```
    close(connfd);}
    return 0 ;
}
```

6. a) Implementation of concurrent and iterative echo server using connection oriented socket system calls

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

void str_cli(FILE *fp, int sockfd)
{
   int bufsize = 1024, cont;
   char *buffer = malloc(bufsize);

while(fgets(buffer,bufsize,fp)!=NULL)
{
        send(sockfd, buffer, sizeof(buffer), 0);

  if((cont=recv(sockfd, buffer, bufsize, 0))>0) {
    fputs(buffer,stdout);
  }}
  printf("\nEOF\n");
}
int main(int argc,char *argv[])
{
  int create_socket;

  struct sockaddr_in address;
  if ((create_socket = socket(AF_INET,SOCK_STREAM,0)) > 0)
  printf("The Socket was created\n");
  address.sin_family = AF_INET;
  address.sin_port = htons(15001);
  inet_pton(AF_INET,argv[1],&address.sin_addr);
  if (connect(create_socket,(struct sockaddr *) &address, sizeof(address)) ==
0)
        printf("The connection was accepted with the server %s...\n",argv[1]);
   else
        printf("error in connect \n");

  str_cli(stdin,create_socket);


  return close(create_socket);
}
```

/*server*/

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

```
#include<sys/stat.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#include<fcntl.h>
#include <arpa/inet.h>

void str_echo(int connfd)
{
      int n;
      int bufsize = 1024;
      char *buffer = malloc(bufsize);
       //printf("inside the function");
  again: while((n=recv(connfd, buffer, bufsize, 0))>0)
            send(connfd,buffer,n,0);
             //printf("%d n",n);
        if(n<0)
            goto again;

}
int main()
{
  int cont,listenfd,connfd,addrlen,addrlen2,fd,pid,addrlen3;


  struct sockaddr_in address,cli_address;
  if ((listenfd = socket(AF_INET,SOCK_STREAM,0)) > 0) //sockfd
    printf("The socket was created\n");
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(15001);
printf("The address before bind %s  ...\n",inet_ntoa(address.sin_addr) );
    if (bind(listenfd,(struct sockaddr *)&address,sizeof(address)) == 0)
    printf("Binding Socket\n");
       printf("The address after bind %s  ...\n",inet_ntoa(address.sin_addr)
);

    listen(listenfd,3);
     printf("server is listening\n");
       //server local address
     getsockname(listenfd,(struct sockaddr *)&address,&addrlen3);
     printf("The server's local address %s ...and port
%d\n",inet_ntoa(address.sin_addr),htons(address.sin_port));
for(;;){
    addrlen = sizeof(struct sockaddr_in);
    connfd = accept(listenfd,(struct sockaddr *)&cli_address,&addrlen);
    //printf("The address %s  ...\n",inet_ntoa(address.sin_addr) );
       addrlen2 = sizeof(struct sockaddr_in);
       int i = getpeername(connfd,(struct sockaddr *)&cli_address,&addrlen);

            printf("The Client  %s is Connected...on port
%d\n",inet_ntoa(cli_address.sin_addr),htons(cli_address.sin_port));

     if((pid=fork())==0)                    //don't call fork for having the
                                       iterative server version
     {
        printf("inside child\n");
         close(listenfd);
```

```
           str_echo(connfd);
           exit(0);
        }

    close(connfd);}
    return 0 ;
}
```

b) Implementation of concurrent and iterative echo server using connectionless socket system calls

```
/*Client*/
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#include<arpa/inet.h>

void str_cli(FILE *fp, int sockfd,struct sockaddr* serv_address, int servlen)
{
   int bufsize = 1024, cont;
   char *buffer = malloc(bufsize);
        int addrlen = sizeof(struct sockaddr_in);
        struct sockaddr_in *preply_addr;
        int len = sizeof(struct sockaddr_in);
        while(fgets(buffer,bufsize,fp)!=NULL){

        sendto(sockfd, buffer, sizeof(buffer),0,serv_address,servlen);

        if((cont=recvfrom(sockfd, buffer, bufsize, 0,(struct
sockaddr*)preply_addr,&len)>0))
        {
                printf("The address %s  ...\n",inet_ntoa(preply_addr-
>sin_addr) );
        fputs(buffer,stdout);
        }}

  printf("\nEOF\n");
}
int main(int argc,char *argv[])
{
  int sockfd;

  //char fname[256];
  struct sockaddr_in serv_address;
  if ((sockfd = socket(AF_INET,SOCK_DGRAM,0)) > 0)
  printf("The Socket was created\n");
  serv_address.sin_family = AF_INET;
  serv_address.sin_port = htons(16001);

  inet_pton(AF_INET,argv[1],&serv_address.sin_addr);
```

```
  str_cli(stdin,sockfd,(struct sockaddr
*)&serv_address,sizeof(serv_address));


  exit(0);
}
```

/*server*/

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/stat.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#include<fcntl.h>
#include <arpa/inet.h>

void str_echo(int sockfd,struct sockaddr* cli_address, int clilen)
{
      int n;
      int bufsize = 1024;
      char *buffer = malloc(bufsize);
       int addrlen;

       for(;;){
       addrlen = clilen;
       n=recvfrom(sockfd,buffer,bufsize,0,cli_address,&addrlen);
       //printf("%s",buffer);
       sendto(sockfd,buffer,n,0,cli_address,addrlen);}
            //printf("%d n",n);


}
int main()
{
  int sockfd;
  struct sockaddr_in serv_address,cli_address;

  if ((sockfd = socket(AF_INET,SOCK_DGRAM,0)) > 0) //sockfd
    printf("The socket was created\n");

    serv_address.sin_family = AF_INET;
    serv_address.sin_addr.s_addr = INADDR_ANY;
    serv_address.sin_port = htons(16001);
    printf("The address before bind %s
...\n",inet_ntoa(serv_address.sin_addr) );
    if (bind(sockfd,(struct sockaddr *)&serv_address,sizeof(serv_address)) ==
0)
    printf("Binding Socket\n");
   str_echo(sockfd,(struct sockaddr *)&cli_address,sizeof(cli_address));

    return 0 ;
}
```

7. Implementation of remote command execution using socket system calls.

/*server*/

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<errno.h>
int main()
{
int sd,acpt,len,bytes,port;
char send[50],receiv[50];
struct sockaddr_in serv,cli;
 if((sd=socket(AF_INET,SOCK_STREAM,0))<0)
        {
              printf("Error in socket\n");
                     exit(0);
        }
 bzero(&serv,sizeof(serv));

 serv.sin_family=AF_INET;
serv.sin_port=htons(15002);
serv.sin_addr.s_addr=htonl(INADDR_ANY);
 if(bind(sd,(struct sockaddr *)&serv,sizeof(serv))<0)
 { printf("Error in bind\n"); exit(0); }
 if(listen(sd,3)<0)
 { printf("Error in listen\n"); exit(0); }
 if((acpt=accept(sd,(struct sockaddr*)NULL,NULL))<0)
{ printf("\n\t Error in accept"); exit(0); }
 while(1) { bytes=recv(acpt,receiv,50,0); receiv[bytes]='\0';
if(strcmp(receiv ,"end")==0)
{ close(acpt); close(sd); exit(0); }
 else { printf("Command received : %s",receiv); system(receiv); printf("\n");
} }
}
```

/*client*/

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<errno.h>
int
main ()
{

  int sd, acpt, len, bytes, port;
  char send1[50], receiv[50];
  struct sockaddr_in serv, cli;
  if ((sd = socket (AF_INET, SOCK_STREAM, 0)) < 0)
```

```
    {
      printf ("Error in socket\n");
      exit (0);
    }
  bzero (&serv, sizeof (serv));

  serv.sin_family = AF_INET;
  serv.sin_port = htons (15002);
  serv.sin_addr.s_addr = htonl (INADDR_ANY);
  if (connect (sd, (struct sockaddr *) &serv, sizeof (serv)) < 0)
    {
      printf ("Error in connection\n");
      exit (0);
    }
  while (1)
    {
      printf ("Enter the command:");
      gets (send1);
      if (strcmp (send1, "end") != 0)
        {
          send (sd, send1, 50, 0);
        }
      else
        {
          send (sd, send1, 50, 0);
          close (sd);
          break;
        }
    }
}
```

8. a) **Write a program to encrypt and decrypt the data using RSA and**

b) Write a program to exchange the key securely using Diffie-Hellman Key exchange protocol.

RSA Algorithm

Generating Public Key

1. Select two prime numbers p and q.
2. Compute n=p*q.
3. Choose e, such that it is an integer and not the factor of n.
4. Public key –(n,e)

Generating Private Key

1. Compute z=(p-1)*(q-1)
2. Determine the private key d= (k*z+1)/e, for some integer k.
3. Private key is d.

Diffie Hallman algorithm

1. Consider two prime numbers g and p.
2. Pick a secret number (a) and compute A= ga mod p.
3. Pick a secret number b and compute B= gb mod p
4. Encrypt message with Ba modpand send
5. Decrypt the received message with Ab mod p.

a) PROGRAM

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <string.h>

using namespace std;

longintgcd(long int a, long int b)
{
if(a == 0)
return b;
if(b == 0)
return a;
returngcd(b, a%b);
}

longintisprime(long int a)
{
inti;
for(i = 2; i< a; i++){
if((a % i) == 0)
return 0;
    }
return 1;
}

longint encrypt(char ch, long int n, long int e)
{
inti;
longint temp = ch;
for(i = 1; i< e; i++)
temp = (temp * ch) % n;
return temp;
}

char decrypt(long intch, long int n, long int d)
{
inti;
longint temp = ch;
for(i = 1; i< d; i++)
ch =(temp * ch) % n;
returnch;
}

int main()
{
longinti, len;
longint p, q, n, phi, e, d, cipher[50];
char text[50];

cout<< "Enter the text to be encrypted: ";
cin.getline(text, sizeof(text));
```

```
len = strlen(text);
do {
        p = rand() % 30;
    } while (!isprime(p));
do {
        q = rand() % 30;
    } while (!isprime(q));
    n = p * q;
phi = (p - 1) * (q - 1);
do {
        e = rand() % phi;
    } while (gcd(phi, e) != 1);
do {
        d = rand() % phi;
    } while (((d * e) % phi) != 1);
cout<< "Two prime numbers (p and q) are: " << p << " and " << q <<endl;
cout<< "n(p * q) = " << p << " * " << q << " = " << p*q <<endl;
cout<< "(p - 1) * (q - 1) = "<< phi <<endl;
cout<< "Public key (n,  e): (" << n << ", " << e << ")\n";
cout<< "Private key (n, d): (" << n << ", " << d << ")\n";
for (i = 0; i<len; i++)
cipher[i] = encrypt(text[i], n, e);
cout<< "Encrypted message: ";
for (i = 0; i<len; i++)
cout<< cipher[i];
for (i = 0; i<len; i++)
text[i] = decrypt(cipher[i], n, d);
cout<<endl;
cout<< "Decrypted message: ";
for (i = 0; i<len; i++)
cout<< text[i];
cout<<endl;
return 0;
}
```

OUTPUT

```
exam@dell:~$ g++ rsa.cpp
exam@dell:~$ ./a.out
Enter the text to be encrypted: rvcecse
Two prime numbers (p and q) are: 13 and 23
n(p * q) = 13 * 23 = 299
(p - 1) * (q - 1) = 264
Public key (n,  e): (299, 103)
Private key (n, d): (299, 223)
Encrypted message: 11419683758318475
Decrypted message: rvcecse
```

b) PROGRAM

```
#include <stdio.h>

// Function to compute a^m mod n
int compute(int a, int m, int n)
{
      int r;
      int y = 1;

      while (m > 0)
      {
            r = m % 2;

            // fast exponention
            if (r == 1)
                  y = (y*a) % n;
            a = a*a % n;

            m = m / 2;
      }

      return y;
}

// C program to demonstrate Diffie-Hellman algorithm
int main()
{
      int p = 23;           // modulus
      int g = 5;       // base

      int a, b;  // a - Alice's Secret Key, b - Bob's Secret Key.
      int A, B;  // A - Alice's Public Key, B - Bob's Public Key

      // choose secret integer for Alice's Pivate Key (only known to
Alice)
srand(time(0))   ;
      a = rand();           // or use rand()

      // Calculate Alice's Public Key (Alice will send A to Bob)
      A = compute(g, a, p);

      // choose secret integer for Bob's Pivate Key (only known to Bob)
      srand(time(0))   ;
      b = rand();           // or use rand()

      // Calculate Bob's Public Key (Bob will send B to Alice)
      B = compute(g, b, p);

      // Alice and Bob Exchanges their Public Key A & B with each other

      // Find Secret key
      intkeyA = compute(B, a, p);
      intkeyB = compute(A, b, p);
```

```
      printf("\nAlice's Secret Key is %d\nBob's Secret Key is %d\n\n",
keyA, keyB);

      return 0;
}
```

OUTPUT

```
RUN-1
exam@dell:~$ ./a.out
Alice's Secret Key is 4
Bob's Secret Key is 4
RUN-2
exam@dell:~$ ./a.out
Alice's Secret Key is 11
Bob's Secret Key is 11
RUN-3
exam@dell:~$ ./a.out
Alice's Secret Key is 10
Bob's Secret Key is 10
```

PARTB Rubrics:

Marks Obtained:

Criteria	Excellent	Good	Poor	Marks Obtained
NPS topic Choice **05 marks**	Selects the most suitable and latest NPS topic with proper reasoning. **(5 M)**	Topic choice is somewhat appropriate but could be improved. **(2 - 4 M)**	Topic choice is incorrect or lacks justification. **(0 - 1 M)**	
Network and security concept used **05 marks**	Uses optimal NPS with clear reasoning for selection. **(5 M)**	Network concepts used are appropriate but may not be optimal. **(2 - 4 M)**	Uses inefficient or inappropriate concepts **(0 - 1 M)**	
Methodology and design **05 marks**	Provides accurate methodology and design with proper justification. **(5 M)**	Methodology and design has minor errors. **(2 - 4 M)**	Design is missing or incorrect. **(0 - 1 M)**	
Implementation and testing **03 marks**	Code is correct, optimized, and efficiently implements the topic **(3 M)**	Code is mostly correct but may contain inefficiencies or minor bugs. **(1 – 2 M)**	Code is incorrect, inefficient, or does not execute properly. **(0 M)**	
Demonstration and presentation	Well presented with emphasis on relevance. **(2 M)**	Moderately presented with justification. **(1 M)**	Not satisfactory. **(0 M)**	

Criteria	Excellent	Good	Poor	Marks Obtained
02 marks				

Total marks out of 20:

VIVA questions

1. **Explain about DNS?**

DNS - Domain Name System. DNS is the Naming System for the resources over Internet; includes Physical nodes and Applications. DNS –It is the easy way to locate to a resource easily over a network and serves to be an essential component necessary for the working of Internet.

2. **What is a Network?**

A network means a set of devices that connected by physical media links. A network is recursively a connection of more than two nodes by a physical link or two or more networks connected by one or more nodes.

3. **What is Bandwidth?**

Each and Every Signal often has a limit of an upper range and lower range of frequency of signal it can carry. So this range of limit of network between its upper frequency and lower frequency is termed as Bandwidth.

4. **List the criteria to check the network reliability?**

A network Reliability is measured by the following factors.

1. a) Downtime is the time it takes to recover.
2. b) Failure Frequency is the frequency when it fails to work the way it is intended**.**
3. **Define a Link?**

At the basic level, a network includes two or more computers directly connected by some physical medium such as co-axial cable or optical fiber. And that physical medium is called a Link.

6. **What is the DNS forwarder?**

DNS servers usually communicate with outside DNS servers of the local network. A forwarder is an entry that is used when a DNS server receives DNS queries that it cant resolve locally. And then it forwards those requests to external DNS servers for resolution

7. **Define node?**

A network consist of two or more computers which will be directly connected by some physical medium such as coaxial cable or optical fiber. And that physical medium is called as Links and the computer it connects is termed as Nodes.

8. **What is a gateway or Router**?

A router or gateway is a node that is connected to two or more networks. In general it forwards message from one network to another.

9. **What is point-point link?**

If the physical links are limited to a pair of nodes then it is said to be point-point link.

10. **What is DHCP scope?**

A scope is a range, or pool of IP addresses that can be leased to DHCP clients on a given subnet.

11. **What is FQDN?**

An FQDN - fully qualified domain name contains both the hostname and a domain name. FQDN is uniquely identifies a host within a DNS hierarchy.

12. **What is MAC address? Does it have some link or something in common to Mac OS of Apple?**

MAC - Media Access Control. MAC is the address of the device identified at Media Access Control Layer of Network Architecture. Similar to IP address MAC address is unique address, i.e., no two device can have same MAC address. MAC address is stored at the ROM Read Only Memory of the device.

MAC Address and Mac OS are two different things and it should not be confused with each other. Mac OS is a POSIX standard Operating System Developed upon FreeBSD used by Apple devices.

What is POP3?

POP3 stands for Post Office Protocol Version3 (Current Version). POP is a protocol which listens on port 110 and is responsible for accessing the mail service on a client machine. POP3 works in two modes such as Delete Mode and Keep Mode.

1. **a)Delete Mode:** A mail is deleted from the mailbox after successful retrieval.
2. **b)b) Keep Mode:** The Mail remains Intact in the mailbox after successful retrieval.
3. **How will check IP address on 98**

16. **What is IP?**

IP is a unique 32 bits software address of a node in a network.

17. **What is private IP?**

Three ranges of IP addresses have been reserved for private address and they are not valid for use on the Internet. If you want to access internet with these address you must have to use proxy server or NAT server (on normal cases the role of proxy server is played by your ISP.).If you do decide to implement a private IP address range, you can use IP addresses from any of the following classes:

Class A: 10.0.0.0 10.255.255.255
Class B: 172.16.0.0 172.31.255.255
Class C: 192.168.0.0 192.168.255.255

What is public IP address?

A public IP address is an address leased from an ISP that allows or enables direct Internet communication.

What is the benefit of sub netting?

What is virtual path?

Along any transmission path from a given source to a given destination, a group of virtual circuits can be grouped together into what is called path.

What is virtual channel?

Virtual channel is normally a connection from one source to one destination, although multicast connections are also permitted. The other name for virtual channel is virtual circuit.

What are the benefits of OSI Reference Model?

It provides a framework for discussing network operations and design.

What is the difference between routable and non- routable protocols?

Routable protocols can work with a router and can be used to build large networks. Non-Routable protocols are also there which is designed to work on small, local networks and cannot be used with a router

What is the difference between TFTP and FTP application layer protocols?

The Trivial File Transfer Protocol (TFTP) allows a local host to obtain files from a remote host but does not provide reliability or security. It uses the fundamental packet delivery services offered by UDP.

The File Transfer Protocol (FTP) is the standard mechanism provided by TCP / IP for copying a file from one host to another. It uses the services offered by TCP and so is reliable and secure. It establishes two connections (virtual circuits) between the hosts, one for data transfer and another for control information.

What is the minimum and maximum length of the header in the TCP segment and IP datagram?

The header should have a minimum length of 20 bytes and can have a maximum length of 60 bytes.

What is difference between ARP and RARP?

The address resolution protocol (ARP) is used to associate the 32 bit IP address with the 48 bit physical address, used by a host or a router to find the physical address of another host on its network by sending a ARP query packet that includes the IP address of the receiver.

The reverse address resolution protocol (RARP) allows a host to discover its Internet address when it knows only its physical address.

Define about ICMP?

ICMP is Internet Control Message Protocol, a network layer protocol of the TCP/IP suite used by hosts and gateways to send notification of datagram problems back to the sender. It uses the echo test / reply to test whether a destination is reachable and responding. It also handles both control and error messages.

1. _____ facilitates allocation of IP addresses in variable-sized blocks.

2. If an organization needs 300 IP addresses, it is allotted a block of _____ addresses on a _____-byte boundary.

3. A block of addresses is granted to an organization. One of these addresses is 202.16.37.40/28. The first address of the block is _____.

4. A block of addresses is granted to an organization. One of these addresses is 202.16.37.40/28. The last address of the block is _____.

5. What is the basic function of a NAT box ?

6. What is the basic idea behind Network Address Translation ?

7. Which of the two implementations, ARP or RARP, requires an additional resource / system ?Why ?

8. A set of LANs, interconnected through routers, use a common DHCP server on one of the LANs. In this scenario, a _____ agent is required on each of the other LANs to access the DHCP server.

9. Write an IPv4 address using a combination of colons and the dotted decimal number.

10. Write the expanded form of the IPv6 address : 8000:123:4567:89AB:CDEF.

11. The _____ extension header in IPv6 facilitates verification of the sender's identity.

12. The _____ extension header in IPv6 provides information about the encrypted contents.

13. _____ are exchanged between a pair of peer transport entities.

14. Define *transport entity*.

15. The generic term used to refer to the end point “port” is _____.

16. The initial connection protocol uses a special _____ server that acts as proxy for servers which are not heavily used.

17. In the transport layer, the connection establishment is a _____ handshake process.

18. _____ connection release may result in data loss.

19. Why is it necessary for a sender using TCP to buffer the TPDUs it transmits ?

20. Why is it necessary for a sender using TCP to buffer the TPDUs it transmits ?

21. In RPC, the client and server programs are bound with a library procedure called _____.

22. In RPC, the process of the client packing the parameters into a message is known as _____.

23. The field ‘Payload type’ in the RTP header is used for _____.

24. Why is ‘Sequence number’ used in the RTP header ?

25. Nagle’s algorithm solves the problem of _____.

26. Clark’s solution solves the problem of _____.

27. The persistence timer used by TCP is designed to _____.

28. The persistence timer used by TCP is designed to _____.

29. In the hierarchical design of domain name space, the inverted tree structure can have levels from level _____ to level _____.

30. A full domain name comprises a sequence of _____ separated by _____.