# Laboratory Journal

*Submitted in partial fulfilment of the requirement*

*for the Laboratory*

## ‘Communication Networks Laboratory’

**(Code – EC 314)**

Prepared and Submitted by:

Mr. Neel Thakker

(Admission No.: U19EC033)

B.Tech III (EC), Semester - VI

(2021-22)

Laboratory Teacher: Dr. Raghavendra Pal

**Department of Electronics Engineering**

**Sardar Vallabhbhai National Institute of Technology**

**Surat - 395007, Gujarat, India.**

# Sardar Vallabhbhai National Institute of Technology

Surat, Gujarat, INDIA

**DEPARTMENT OF ELECTRONICS ENGINEERING**

**2021-22**

**Subject- Communication Networks Lab (Code – EC314)**

## Certificate

This is to certify that the Laboratory Journal is prepared & submitted by **B.Tech III (Semester-VI)** student Mr. Neel Thakker bearing **Admission No.** U19EC033 in the partial fulfilment of the requirement for the laboratory **Communication Networks Lab (Code-EC314)** through ONLINE MODE.

We, certify that the work is comprehensive, complete and fit for evaluation.

**Laboratory Teacher:**

| **Name** | **Signature with Date** |
| --- | --- |
| 1. Dr. Raghavendra Pal | |

**Jan-May 2022**

**Communication Networks Lab (Code – EC314)**

**Academic Year (2021-22)**

**LIST OF EXPERIMENTS**

| Sr. No. | List of Experiments | Page No. |
|---|---|---|
| **1.** | Introduction to TCP/IP Networking Commands. | |
| **2.** | Cyclic Redundancy Check (CRC) Method for Error Detection. | |
| **3.** | Hamming Code for Error Detection and Correction. | |
| **4.** | Bit Stuffing. | |
| **5.** | Shortest Path Routing Algorithm. | |
| **6.** | Symmetric Key Ciphering and Deciphering using Classical Ciphers. | |
| **7.** | Asymmetric Key Ciphering and Deciphering using Modern Ciphers. | |
| **8.** | Introduction to Network Simulator (NS3) | |
| **9.** | Elementary Network Model design using NS3 | |
| **10.** | Dynamic Network Model design for Optimum Routing using NS3 | |
| **11.** | Local Area Network Architecture and its Performance Analysis using NS3 | |
| **12.** | Wireless Network Implementation using NS3. | |

**Submitted By**

**Name: Neel Thakker**

**Admission Number: U19EC033**

**Class/Year/Branch: B.Tech III, ECE**

# Experiment 1

**Date:**

**Aim:** Introduction to TCP/IP Networking Commands.

**Tools Required:** Terminal

**Theory:**

Most client and server operating systems that support Transmission Control Protocol/Internet Protocol (TCP/IP) come with a suite of commands and tools that are designed to let you examine TCP/IP configuration information and diagnose and correct problems. Although the exact form of these commands varies between Windows and Unix/Linux, most are surprisingly similar. This tutorial is a reference to the most commonly used TCP/IP commands.

TCP/IP uses the client-server model of communication in which a user or machine (a client) is provided a service, like sending a webpage, by another computer (a server) in the network.

Collectively, the TCP/IP suite of protocols is classified as stateless, which means each client request is considered new because it is unrelated to previous requests. Being stateless frees up network paths so they can be used continuously.

The transport layer itself, however, is stateful. It transmits a single message, and its connection remains in place until all the packets in a message have been received and reassembled at the destination.

**Code:**

*arp* - Address Resolution Protocol

ARP stands for "Address Resolution Protocol" is a protocol for mapping an IP address to a physical MAC address on a local area network. Basically, ARP is a program used by a computer system to find another computer's MAC address based on its IP address.

*getmac* –

Getmac is a Windows command used to display the Media Access Control (MAC) addresses for each network adapter in the computer. These activities will show you how to use the getmac command to display MAC addresses.

*ipconfig* –

The ipconfig (short for IP Configuration) is a basic, yet popular, Windows network command-line utility used to display the TCP/IP network configuration of a computer. If you are familiar with Linux, this tool is similiar to ifconfig. This tool is often used for troubleshooting network connectivity issues. With ipconfig, you can identify the types of network adapaters on your computer, the computer's IP address, the IP addresses of the DNS (Domain Name System) servers being used, and much more.

*tracert* - In computing, traceroute and tracert are computer network diagnostic commands for displaying possible routes and measuring transit delays of packets across an Internet Protocol network

*netstat* - The netstat command generates displays that show network status and protocol statistics. You can display the status of TCP and UDP endpoints in table format, routing table information, and interface information.

**Output Results/Graphs**

*arp* -

```
Interface: 192.168.2.3 --- 0x10
  Internet Address      Physical Address      Type
  192.168.2.1           fe-e2-6c-80-ef-64     dynamic
  192.168.2.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.102.18        01-00-5e-7f-66-12     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

*getmac* -

```
C:\Users\DELL>getmac

Physical Address    Transport Name
=================== ==========================================================
80-2B-F9-BA-26-1D   \Device\Tcpip_{D41D7407-C5CF-4BC5-9728-7A81A3F12A41}
6C-2B-59-31-C8-B2   Media disconnected
```

*ipconfig* -

```
C:\Users\DELL>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 10:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::8833:609c:2a5f:a866%16
   IPv4 Address. . . . . . . . . . . : 192.168.2.3
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.2.1
```

*tracert -*

```
C:\Users\DELL>tracert -h 30 google.com

Tracing route to google.com [142.250.192.110]
over a maximum of 30 hops:

  1     5 ms     5 ms     4 ms  MACBOOKAIR-1A4E [192.168.2.1]
  2    16 ms    16 ms     6 ms  172.39.4.1
  3     9 ms     5 ms     5 ms  172.16.1.1
  4    10 ms    17 ms     6 ms  192.168.168.1
  5     9 ms    11 ms     8 ms  static.ill.117.239.204.225/24.bsnl.in [117.239.204.225]
  6   137 ms    99 ms    11 ms  172.24.193.226
  7     *        *        *     Request timed out.
  8    18 ms    19 ms    11 ms  142.250.161.230
  9    19 ms    15 ms     *     216.239.57.17
 10     *       25 ms    22 ms  72.14.237.139
 11    22 ms    23 ms     *     bom12s17-in-f14.1e100.net [142.250.192.110]
 12    22 ms     *       23 ms  bom12s17-in-f14.1e100.net [142.250.192.110]

Trace complete.
```

*netstat -*

```
Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    192.168.2.3:50230      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50231      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50232      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50233      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50234      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50235      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50236      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50237      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50238      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50239      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50240      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50241      49.44.166.187:http     CLOSE_WAIT
  TCP    192.168.2.3:50270      sd-in-f188:5228        ESTABLISHED
  TCP    192.168.2.3:50279      whatsapp-cdn-shv-02-bom1:https  ESTABLISHED
  TCP    192.168.2.3:50420      117.18.237.29:http     CLOSE_WAIT
  TCP    192.168.2.3:50467      20.197.71.89:https     ESTABLISHED
  TCP    192.168.2.3:50510      undefined:http         TIME_WAIT
  TCP    192.168.2.3:50516      relay-aa6a8e8a:http    TIME_WAIT
  TCP    192.168.2.3:50517      relay-fa663288:http    ESTABLISHED
  TCP    192.168.2.3:50518      bom07s36-in-f14:http   ESTABLISHED
  TCP    192.168.2.3:50521      173.194.154.170:http   ESTABLISHED
  TCP    192.168.2.3:50524      bom05s25-in-f10:http   ESTABLISHED
  TCP    192.168.2.3:50527      bom07s47-in-f6:http    ESTABLISHED
  TCP    192.168.2.3:50530      bom05s21-in-f9:http    ESTABLISHED
  TCP    192.168.2.3:50531      cache:http             TIME_WAIT
  TCP    192.168.2.3:50533      cache:http             TIME_WAIT
  TCP    192.168.2.3:50534      maa05s36-in-f6:http    ESTABLISHED
```

**Conclusions:**

In this experiment, we have run few tcp/ip protocol command and understood their working.

# Experiment 2

**Date:**

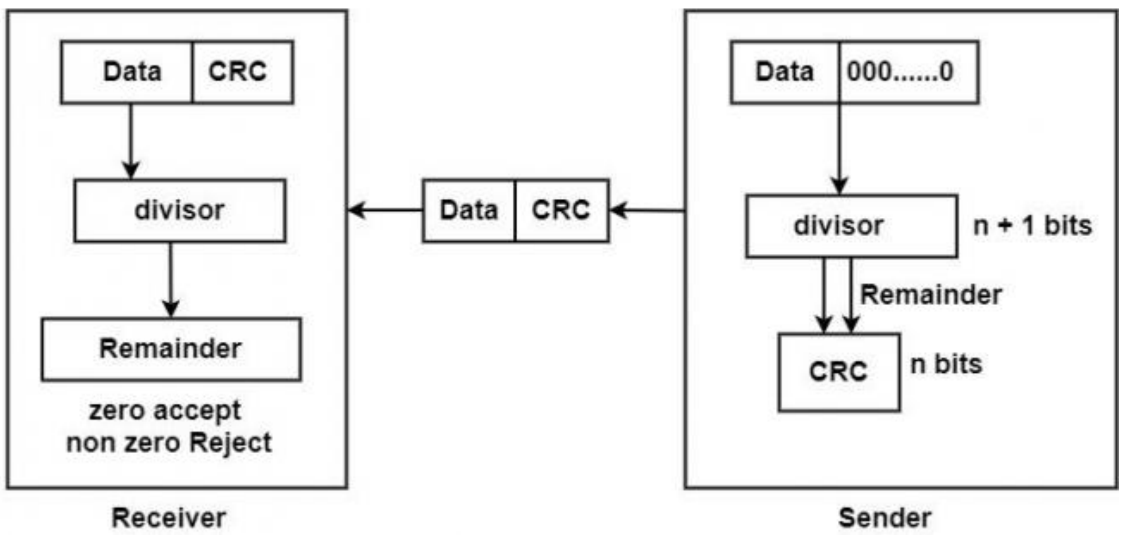
**Aim:** Cyclic Redundancy Check (CRC) Method for Error Detection.

**Tools Required:** MATLAB

**Theory:**

The Cyclic Redundancy Checks (CRC) is the most powerful method for Error-Detection and Correction. It is given as a k-bit message and the transmitter creates an (n – k) bit sequence called frame check sequence. The out coming frame, including n bits, is precisely divisible by some fixed number. Modulo 2 Arithmetic is used in this binary addition with no carries, just like the XOR operation.

Redundancy means duplicacy. The redundancy bits used by CRC are changed by splitting the data unit by a fixed divisor. The remainder is CRC.




**Code:**

```
clc;
clear all;
close all;

dataword = [1 0 1 0 0 0 1]

divisor = [1 0 0 1]

k = length(divisor);

dataword1 = [dataword zeros(1,k-1)];
```

```
p = length(dataword1);
r = 0;
tmp = dataword1(1:k);

for i = 1:(p - k + 1)
    if tmp(1) == 1
        xored = bitxor(tmp,divisor);
    else
        xored = bitxor(tmp,zeros(1,k));
    end

    tmp3 = xored(2:end);

    r = k + i;
    if r <= p
        tmp = [tmp3 dataword1(r)];
    end
end

crc = tmp3

codeword = [dataword crc]

tmp = codeword(1:k);

for i = 1:(p - k + 1)
    if tmp(1) == 1
        xored = bitxor(tmp,divisor);
    else
        xored = bitxor(tmp,zeros(1, k));
    end

    tmp3 = xored(2:end);

    r = k + i;
    if p >=r
        tmp = [tmp3 codeword(r)];
    end
end

syndrome = tmp3
```

## Output Results/Graphs

When no error in transmission:

```
dataword =

     1     0     1     0     0     0     1


divisor =

     1     0     0     1


crc =

     0     1     0


codeword =

     1     0     1     0     0     0     1     0     1     0


syndrome =

     0     0     0
```

When error in transmission

```
dataword =

     1     0     1     0     0     0     1


divisor =

     1     0     0     1


crc =

     0     1     0


codeword =

     1     0     1     0     1     0     1     0     1     0


syndrome =

     1     0     0
```

## Conclusions:

In this experiment, we have implemented cyclic redundancy code in MATLAB and observed value of syndrome when message has error and else.

# Experiment 3

**Date:**

**Aim:** Hamming Code for Error Detection and Correction.

**Tools Required:** MATLAB

**Theory:**

Hamming code is a block code that is capable of detecting up to two simultaneous bit errors and correcting single-bit errors. It was developed by R.W. Hamming for error correction.

In this coding method, the source encodes the message by inserting redundant bits within the message. These redundant bits are extra bits that are generated and inserted at specific positions in the message itself to enable error detection and correction. When the destination receives this message, it performs recalculations to detect errors and find the bit position that has error.

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 following formula:

$$2^r \geq m + r + 1$$

Parity bits:
A parity bit is a bit appended to a data of binary bits to ensure that the total number of 1's in the data is even or odd. Parity bits are used for error detection. There are two types of parity bits:

1. Even Parity Bit

   In the case of even parity, for a given set of bits, the number of 1's are counted. If that count is odd, the parity bit value is set to 1, making the total count of occurrences of 1's an even number. If the total number of 1's in a given set of bits is already even, the parity bit's value is 0.

2. Odd Parity Bit

   In the case of odd parity, for a given set of bits, the number of 1's are counted. If that count is even, the parity bit value is set to 1, making the total count of occurrences of 1's an odd number. If the total number of 1's in a given set of bits is already odd, the parity bit's value is 0.

**Code:**

```
clc;
clear all;
close all;

data = [1 0 1 1 0 0 1]

m = length(data);
i = 1;
while 1
    r = i;
    if 2^r >= m + r + 1
        break
    end

    i = I + 1;
end

data_rev = fliplr(data);

transmitted_rev = zeros(1, m + r);

k = 0;
p = 1;
for j = 1:m + r
    if j == 2.^k
        transmitted_rev(j) = 0;
        k = k + 1;
    else
        transmitted_rev(j) = data_rev(p);
        p = p+1;
    end
end

r1 = transmitted_rev(1:2:11);
r2 = [transmitted_rev(2:3) transmitted_rev(6:7) transmitted_rev(10:11)];
r3 = transmitted_rev(4:7);
r4 = transmitted_rev(8:11);

r1 = ~rem(length(find(r1 == 1)), 2) == 0;
r2 = ~rem(length(find(r2 == 1)), 2) == 0;
r3 = ~rem(length(find(r3 == 1)), 2) == 0;
r4 = ~rem(length(find(r4 == 1)), 2) == 0;

r = [r1 r2 r3 r4];

for i = 1:length(r)
    transmitted_rev(2.^(i-1)) = r(i);
end

transmitted = fliplr(transmitted_rev)

% if no error in message
received_message = [1 0 1 0 1 0 0 1 1 1 0];

arr_rev = fliplr(received_message);
```

```
m1 = arr_rev(1:2:11);
m2 = [arr_rev(2:3) arr_rev(6:7) arr_rev(10:11)];
m3 = arr_rev(4:7);
m4 = arr_rev(8:11);

m1 = ~rem(length(find(m1 == 1)), 2) == 0;
m2 = ~rem(length(find(m2 == 1)), 2) == 0;
m3 = ~rem(length(find(m3 == 1)), 2) == 0;
m4 = ~rem(length(find(m4 == 1)), 2) == 0;

m = [m1 m2 m3 m4];

disp('message recevied without error')
disp(m)

% error in 6th bit

received_message_with_error = [1 0 1 0 1 1 0 1 1 1 0]

arr_rev = fliplr(received_message_with_error);

m1 = arr_rev(1:2:11);
m2 = [arr_rev(2:3) arr_rev(6:7) arr_rev(10:11)];
m3 = arr_rev(4:7);
m4 = arr_rev(8:11);

m1 = ~rem(length(find(m1 == 1)), 2) == 0;
m2 = ~rem(length(find(m2 == 1)), 2) == 0;
m3 = ~rem(length(find(m3 == 1)), 2) == 0;
m4 = ~rem(length(find(m4 == 1)), 2) == 0;

m = [m1 m2 m3 m4];

disp('message recevied with error and its position')
disp(m)

p = bi2de(m)

if p ~= 0
    received_message_with_error(p) = double(~received_message_with_error(p));
end

disp('corrected message')

disp(received_message_with_error)
```

**Output Results/Graphs**

```
data =

     1     0     1     1     0     0     1


transmitted =

     1     0     1     0     1     0     0     1     1     1     0

message recevied without error
   0   0   0   0


received_message_with_error =

     1     0     1     0     1     1     0     1     1     1     0

message recevied with error and its position
   0   1   1   0


p =

     6

corrected message
     1     0     1     0     1     0     0     1     1     1     0
```

**Conclusions:**

In this experiment, we have implemented hamming code for error detection and correction in MATLAB.

# Experiment 4

**Date:**

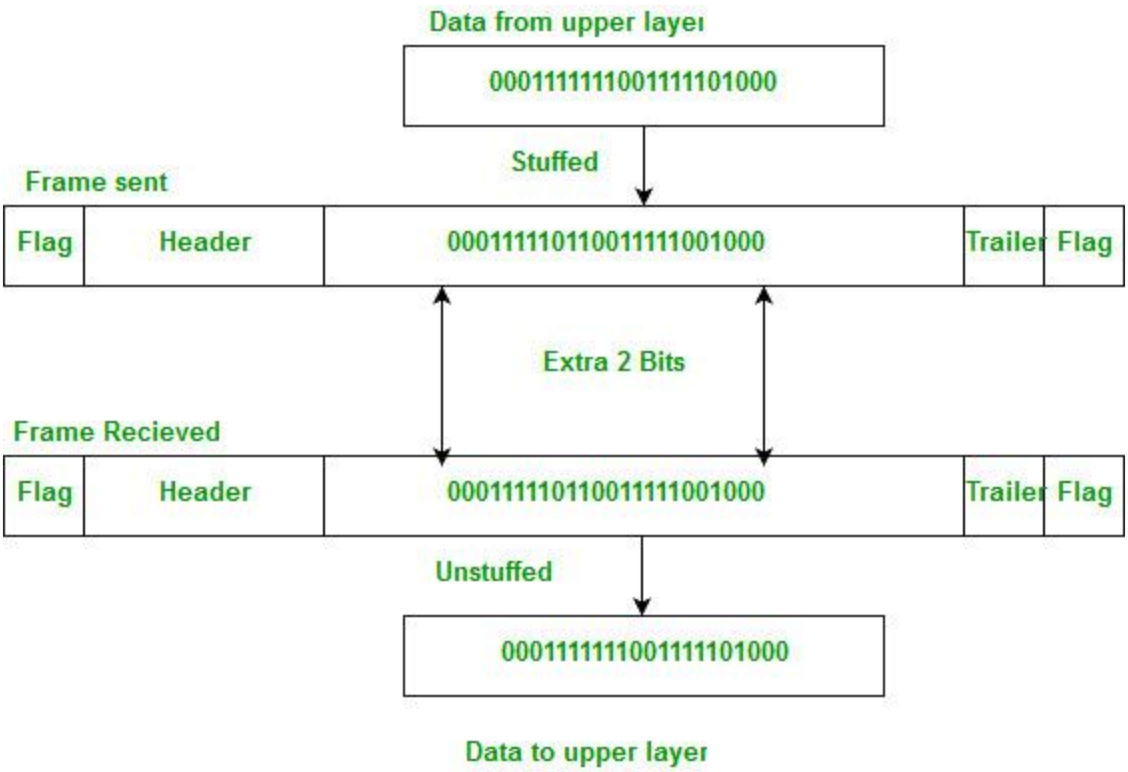
**Aim:** Bit Stuffing.

**Tools Required:** MATLAB

**Theory:**

Data link layer is responsible for something called Framing, which is the division of stream of bits from network layer into manageable units (called frames). Frames could be of fixed size or variable size. In variable-size framing, we need a way to define the end of the frame and the beginning of the next frame.

Bit stuffing is the insertion of non-information bits into data. Note that stuffed bits should not be confused with overhead bits. Overhead bits are non-data bits that are necessary for transmission (usually as part of headers, checksums etc.).




Applications of Bit Stuffing –

1. synchronize several channels before multiplexing
2. rate-match two single channels to each other
3. run length limited coding

**Code:**

```
clc;
clear all;
close all;

% msg = randi([0,1],1,100);
msg = [0 1 1 1 1 1 1 0 0 1 1 1 1 1]
l = length(msg);
stuffcount = 0;
count = 0;

% stuffing
m = msg;
for i = 1:l
    if msg(i) == 1
        count = count + 1;
        if count == 5
            m = [m(1:i+stuffcount) 0 m(i+stuffcount+1:end)];
            count = 0;
            stuffcount=stuffcount+1;
        end
    else
        count = 0;
    end

end

p = l + stuffcount;

disp('stuffed bits')
disp(m)

idx = zeros(1,stuffcount);
% unstuffing
i = 0;
j = 1;
count = 0;
for i = 1:p
    if m(i) == 1
        count = count + 1;
        if count == 5
            idx(j) = i+1;
            count = 0;
            j = j+1;
        end
    else
        count = 0;
    end

end

m(idx) = [];

disp('unstuffed bits')
disp(m)
```

## Output Results/Graphs

```
msg =

     0     1     1     1     1     1     1     0     0     1     1     1     1     1

stuffed bits
     0     1     1     1     1     1     0     1     0     0     1     1     1     1     1     0

unstuffed bits
     0     1     1     1     1     1     1     0     0     1     1     1     1     1
```

## Conclusions:

In this experiment, we have implemented Bit stuffing and unstuffing in MATLAB.

# Experiment 5

**Date:**

**Aim:** Shortest Path Routing Algorithm.

**Tools Required:** MATLAB

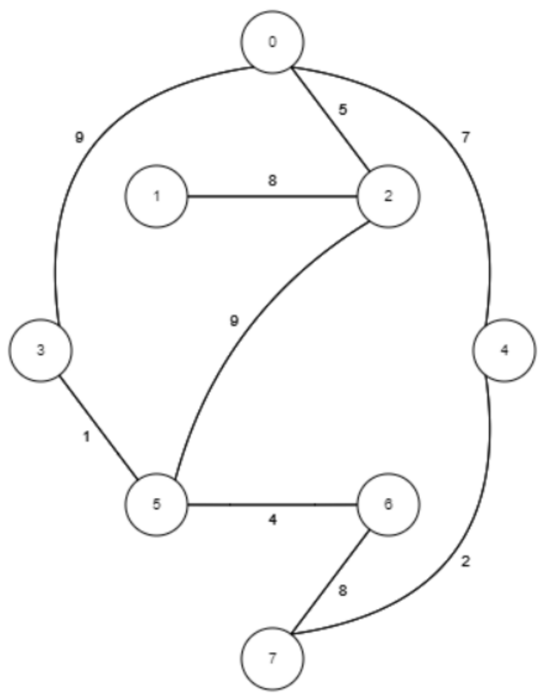
**Theory:**

With Dijkstra's Algorithm, you can find the shortest path between nodes in a graph. Particularly, you can find the shortest path from a node (called the "source node") to all other nodes in the graph, producing a shortest-path tree.

This algorithm is used in GPS devices to find the shortest path between the current location and the destination. It has broad applications in industry, especially in domains that require modelling networks.

Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyses the graph to find the shortest path between that node and all the other nodes in the graph. The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path. Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path. The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.

**Code:**



```
clc;
clear all;
close all;

arr = inf*ones(8);
for i = 1:length(arr)
    arr(i, i) = 0;
end

%defining graph
[arr(1, 3), arr(3, 1)] = deal(5);
[arr(1, 4), arr(4, 1)] = deal(9);
[arr(1, 5), arr(5, 1)] = deal(7);
[arr(2, 3), arr(3, 2)] = deal(8);
[arr(3, 6), arr(6, 3)] = deal(9);
[arr(4, 6), arr(6, 4)] = deal(1);
[arr(5, 8), arr(8, 5)] = deal(2);
[arr(6, 7), arr(7, 6)] = deal(4);
[arr(7, 8), arr(8, 7)] = deal(8);

initial_node = 1;
N = length(arr);
d(1:N) = inf;
vis(1:N) = 0;
d(initial_node) = 0;

while sum(vis) < N
    % find unvisited nodes
    set(1:N) = inf;
    for i = 1:N
        if vis(i) == 0
            set(i) = d(i);
        end
    end

    [currentDistance, position] = min(set);
    vis(position) = 1;

    for i = 1:N
        newDistance = currentDistance + arr(position, i);
        if newDistance < d(i)
            d(i) = newDistance;
        end
    end
end

x = 1:8;
T = table(x', d', 'VariableNames', {'Node', 'Distance'});
disp('Minimum distance from source node to other node in network is show
below:')
disp(T)
```

**Output Results/Graphs**

```
minimum distance from source node to other node in network:
node
     1     2     3     4     5     6     7     8

distance
     0    13     5     9     7    10    14     9
```

**Conclusions:**

In this experiment, we have implemented dijkstra's shortest path finding algorithm in MATLAB for specific network and found out cost to go from source node to other nodes.

# Experiment 6

**Date:**

**Aim:** Symmetric Key Ciphering and Deciphering using Classical Ciphers

**Tools Required:** MATLAB

**Theory:**

A symmetric cipher is one that uses the same key for encryption and decryption.

Ciphers or algorithms can be either symmetric or asymmetric. Symmetric ones use the same key (called a secret key or private key) for transforming the original message, called plaintext, into cipher text and vice versa. Symmetric ciphers are generally faster than their asymmetric counterparts,

If the decryption key and the encryption key are the same, it is called symmetric key system. The analogy is that of a mechanical lock. You lock some secret in a box and send it to your remote friend through a servant. The key itself is not sent through the servant because there is a risk that he may open the lock and take the content. But, your friend will be having another copy of the same key as yours and he can open it. This is unlike a Public key system where the locking and unlocking keys are different and seemingly unrelated. Good thing about the symmetric key is that it is present with only the sender and the receiver and others have no clue about it.

**Code:**

```
clc;
clear all;
close all;

msg = 'Neel Thakker';
key = 6;

t = double(msg);
encrypted = t + key;
encrypted = char(encrypted);

fprintf('\nMessage:  ')
fprintf(msg)

fprintf('\ndata transmitted:  ')
fprintf(encrypted)

received_data = double(encrypted);
decrypted = received_data - key;
decrypted = char(decrypted);

fprintf('\ndecrypted data:  ')
fprintf(decrypted)
fprintf('\n')
```

**Output Results/Graphs**

```
Message:  Neel Thakker
data transmitted:  Tkkr&Zngqqkx
decrypted data:  Neel Thakker
```

**Conclusions:**

In this experiment, we have implanted symmetric key ciphering Using MATLAB and encrypted and decrypted message using a very classical logic.

# Experiment 7

**Date:**

**Aim:** Asymmetric Key Ciphering and Deciphering using Modern Ciphers.

**Tools Required:** MATLAB

**Theory:**

Asymmetric cryptography, also known as public-key cryptography, is a process that uses a pair of related keys -- one public key and one private key -- to encrypt and decrypt a message and protect it from unauthorized access or use.

A public key is a cryptographic key that can be used by any person to encrypt a message so that it can only be decrypted by the intended recipient with their private key. A private key -- also known as a secret key -- is shared only with key's initiator.
When someone wants to send an encrypted message, they can pull the intended recipient's public key from a public directory and use it to encrypt the message before sending it. The recipient of the message can then decrypt the message using their related private key.

If the sender encrypts the message using their private key, the message can be decrypted only using that sender's public key, thus authenticating the sender. These encryption and decryption processes happen automatically; users do not need to physically lock and unlock the message.

Asymmetric encryption uses a mathematically related pair of keys for encryption and decryption: a public key and a private key. If the public key is used for encryption, then the related private key is used for decryption. If the private key is used for encryption, then the related public key is used for decryption.

Asymmetric cryptography is typically used to authenticate data using digital signatures. A digital signature is a mathematical technique used to validate the authenticity and integrity of a message, software or digital document. It is the digital equivalent of a handwritten signature or stamped seal.

**Code:**

```
clc;
clear all;
close all;

% RSA Algorithm
p = 3;
q = 7;

n = p*q;
phi = (p-1)*(q-1);

k = 2;
% e = 2111;
e = 2;
```

```
while e < phi
   if gcd(e, phi) == 1 && mod(k*phi + 1, e) == 0
       break;
   end

   e = e + 1;
end

d = (1 + (k*phi))/e;
% d = 1;
% while mod(e*d, phi) ~= 1
%     d = d+1;
% end

fprintf('\ne: %d, d: %d, n: %d, phi: %d\n', e, d, n, phi)

% msg = 'N';
% data = double(msg);
data = 12;
len = length(data);
encrypted_data = zeros(1, len);
decrypted_data = zeros(1, len);

i = 1;
while i <= len
    encrypted_data(i) = mod(data(i)^e, n);
%     disp(encrypted_data(i)^d)
    decrypted_data(i) = mod(encrypted_data(i)^d, n);
    i = i + 1;
end

disp('Message')
disp(data)

disp('Encrypted message')
disp(encrypted_data)

disp('Decrypted message')
disp(decrypted_data)
```

### Output Results/Graphs

```
Command Window
  e: 5, d: 5, n: 21, phi: 12
  Message
      12

  Encrypted message
       3

  Decrypted message
      12
```

### Conclusions:

In this experiment we have implemented RSA asymmetric key ciphering and deciphering using MATLAB.