NAGALAND UNIVERSITY

SCHOOL OF ENGINEERING & TECHNOLOGY

LAB MANUAL

NETWORK LAB

SHANCHAMO YANTHAN

ASSISTANT PROFESSOR

DEPARTMENT OF INFORMATION TECHNOLOGY

TABLE OF CONTENTS

LAB EXPERIMENT 1

1. Study of different types of Network cables and practically implement the cross-wired cable and straight through cable using Crimping tool. Make RJ45 Male Connector.
   The different classes of media are shown in Figure 1.

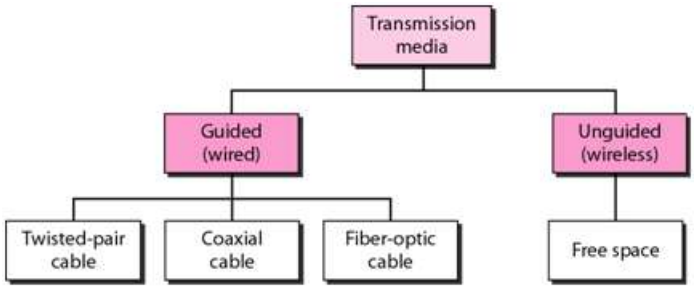

Figure 1.

a. **Twisted-Pair Cable:** A twisted pair consists of two conductors (normally copper), each with its own plastic insulation, twisted together, as shown in Figure 2.

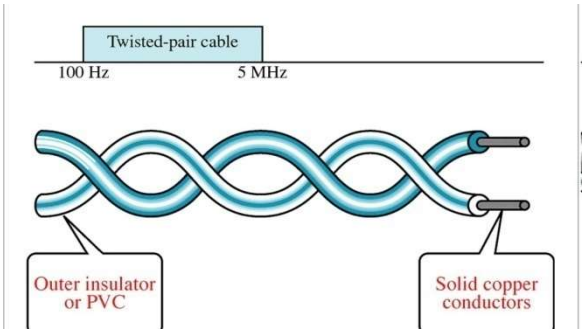

Figure 2.

It can be Unshielded (UTP) or Shield Twisted pair (STP) as shown in Figure 3(a) and 3(b)

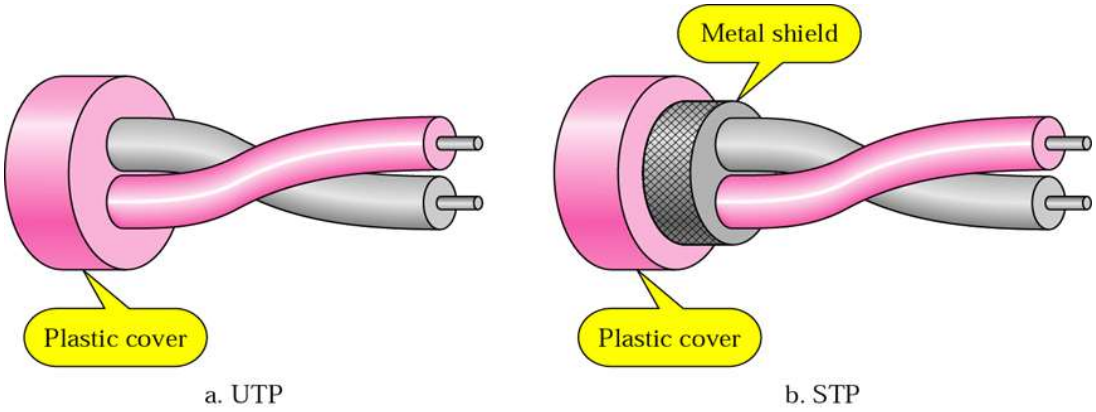

Figure 3.

b. **Coaxial Cable:** Coaxial cable (or *coax)* carries signals of higher frequency ranges than those in twisted pair cable, in part because the two media are constructed quite differently. Instead of having two wires, coax has a central core conductor of solid or stranded wire (usually copper) enclosed in an insulating sheath, which is, in turn, encased in an outer conductor of metal foil, braid, or a combination of the two as shown in Figure 4.

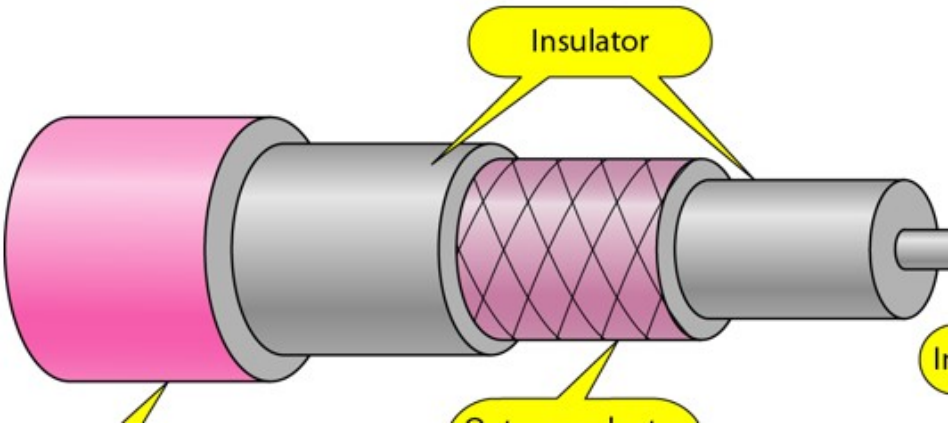

Figure 4.

c. **Fiber-Optic Cable:** A fiber-optic cable is made of glass or plastic and transmits signals in the form of light as shown in Figure 5.

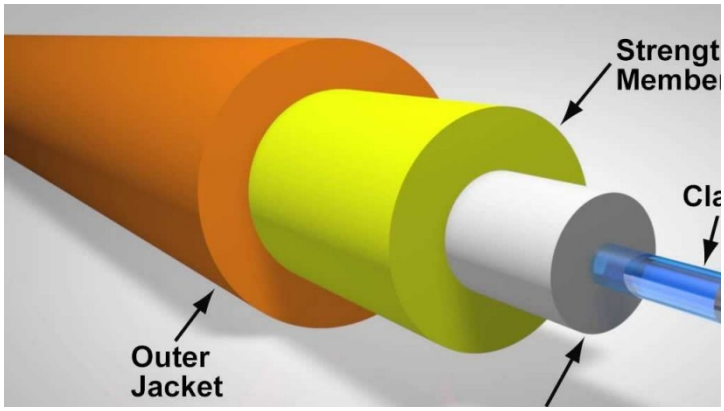

Figure 5.

Steps to make RJ45 connector using crimping tool
(Source https://www.wikihow.com/Crimp-Rj45)

STEP1: **Strip the cable back 1 inch (25 mm) from the end.** Insert the cable into the stripper section of the tool and squeeze it tight. Then, rotate the crimping tool around the cable in a smooth and even motion to create a clean cut. Keep the tool clamped and pull away towards the end of the wire to remove the sheathing.

- The stripping section is a round hole near the handle of the tool.
- The sheathing should come off cleanly, leaving the wires exposed.

STEP2: **Untwist and straighten the wires inside of the cable.** Inside of the cable you'll see a bunch of smaller wires twisted together. Separate the twisted wires and straighten them out so they're easier to sort into the right order.

- Cut off the small plastic wire separator or core so it's out of the way.
- Don't cut off or remove any of the wires or you won't be able to crimp them into the connector.

STEP3: **Arrange the wires into the right order.** Use your fingers to put the wires in the correct order so they can be properly crimped. The proper sequence is as follows from left to right: Orange/White, Orange, Green/White, Blue, Blue/White, Green, Brown/White, Brown as shown in Figure 6.

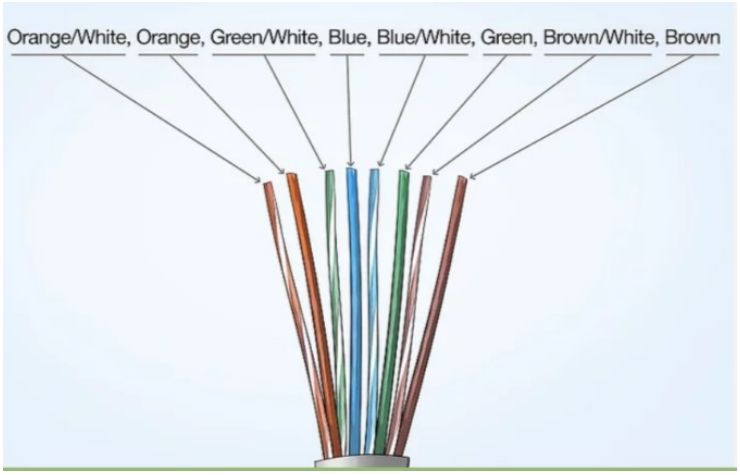

Figure 6.

- There are 8 wires in total that need to be arranged in the right sequence.
- Note that the wires labeled Orange/White or Brown/White indicate the small wires that have 2 colors.

STEP4: **Cut the wires into an even line ½ inch (13 mm) from sheathing.** Hold the wires with your thumb and index finger to keep them in order. Then, use the cutting section of the crimping tool to cut them into an even line as shown in Figure 7.

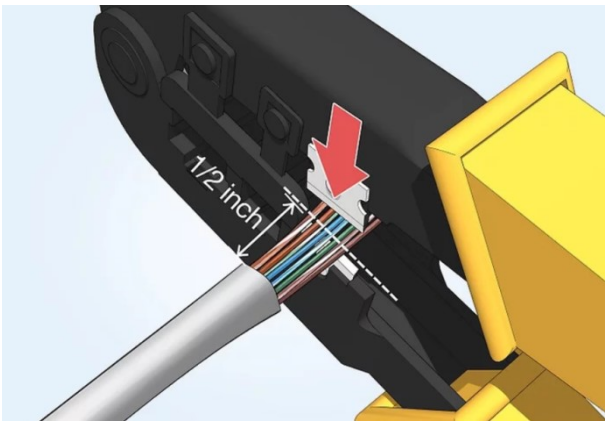

Figure 7.

- The cutting section of the tool will resemble wire cutters.
- The wires must be in an even line to be crimped into the RJ-45 connector properly. If you cut them in an uneven line, move further down the wires and cut them again.

STEP5: I**nsert the wires into the RJ-45 connector.** Hold the RJ-45 connector so the clip is on the underside and the small metal pins are facing up. Insert the cable into the connector so that each of the small wires fits into the small grooves in the connector as shown in Figure 8.

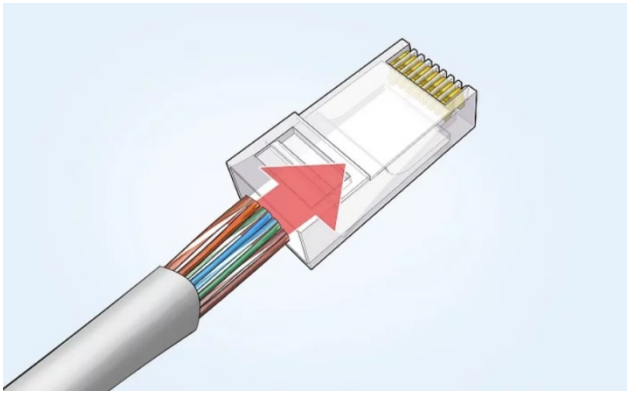

Figure 8.

- The sheathing of the cable should fit just inside of the connector so it's past the base.
- If any of the small wires bend or don't fit into a groove correctly, take the cable out and straighten the wires with your fingers before trying again.
- The wires must be inserted in the correct order and each wire must fit into a groove before you crimp the connector.

STEP6: **Stick the connector into the crimping part of the tool and squeeze twice.** Insert the connector in the crimping section of the tool until it can't fit any further. Squeeze the handles to crimp the connector and secure the wires. Release the handles, then squeeze the tool again to make sure all of the pins are pushed down as shown in Figure 9.

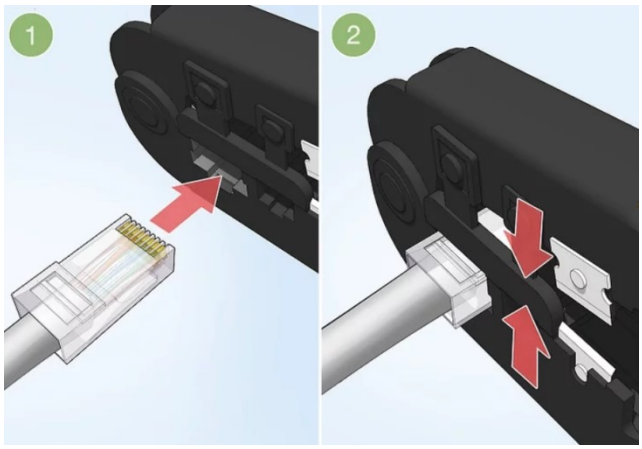

Figure 9.

- The crimping tool pushes small pins in the grooves down onto the wires to hold and connect them to the RJ-45 connector.

STEP7: **Remove the cable from the tool and check that all of the pins are down.** Take the connector out of the tool and look at the pins to see that they're all pushed down in an even line. Lightly tug at the connector to make sure it's attached to the cable.

- If any of the pins aren't pushed down, put the wire back into the crimping tool and crimp it again.

LAB EXPERIMENT 2

2. Study the various Network command-line utilities

a. ipconfig

This is a command-line utility for displaying information on TCP/IP configuration and information pertaining to the DNS and DHCP (Dynamic Host Configuration Protocol).

STEP1: press **Win + R** keys on your keyboard to start the Run program in windows. For Linux, open Terminal and type ifconfig.

STEP2: In the Run textbox, type **cmd** and press enter or click OK.

STEP3: In the Command prompt window, type **ipconfig** and press enter key.

STEP4: to view all configurations, type **ipconfig /all** and press enter.

b. ping

Ping is a basic networking utility that comes with your operating system. You can use it to check whether an IP address can be reached.

STEP1: press **Win + R** and type **cmd** to open the terminal.

STEP2: type **ping *IP_ADDR*** and press enter. IP_ADDR can be any IP address or website URL (refer Figure 10).

```
C:\Users\HP>ping www.google.com

Pinging www.google.com [142.250.192.36] with 32 bytes of data:
Reply from 142.250.192.36: bytes=32 time=54ms TTL=119
Reply from 142.250.192.36: bytes=32 time=55ms TTL=119
Reply from 142.250.192.36: bytes=32 time=56ms TTL=119
Reply from 142.250.192.36: bytes=32 time=55ms TTL=119

Ping statistics for 142.250.192.36:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 54ms, Maximum = 56ms, Average = 55ms
```

Figure 10.

c. netstat

Another useful command-line network utility is netstat. Netstat, short for "network statistics," allows you to display the network connections for TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). Essentially, it lets you check whether the connections exist, and provides statistics to show how the connection is performing. The netstat command will show a list of TCP connections, the IP address of your computer, the IP address of the device the connection goes to (the foreign IP address), the port numbers of both, and the TCP state.

STEP1: press **Win + R** and type **cmd** to open the terminal.

STEP2: type **netstat** and press enter (refer Figure 11).

```
C:\Users\HP>netstat

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    127.0.0.1:49153        SY-PC:49154            ESTABLISHED
  TCP    127.0.0.1:49154        SY-PC:49153            ESTABLISHED
  TCP    127.0.0.1:49694        SY-PC:49695            ESTABLISHED
  TCP    127.0.0.1:49695        SY-PC:49694            ESTABLISHED
  TCP    127.0.0.1:49863        SY-PC:49864            ESTABLISHED
```

Figure 11.

d. tracert

Tracert, also known as traceroute is used to look at the connection between the sender and the destination. tracert provides details on all the "hops" the packet went through to get to the destination, including switches and routers, along with the IP address and DNS information of each. It then breaks down the information of each hop to show the latency between points.

STEP1: press **Win + R** and type **cmd** to open the terminal.

STEP2: type **netstat www.google.com** and press enter (refer Figure 12).

```
C:\Users\HP>tracert www.google.com

Tracing route to www.google.com [142.250.192.36]
over a maximum of 30 hops:

  1    <1 ms     1 ms    <1 ms  192.168.1.1
  2     1 ms     1 ms     1 ms  223.27.120.58
  3     1 ms    <1 ms     1 ms  223.27.120.57
  4     1 ms     1 ms     1 ms  192.168.15.5
  5    55 ms    54 ms    54 ms  103.27.170.10
  6    56 ms    55 ms    55 ms  108.170.248.161
  7    55 ms    55 ms    55 ms  142.250.210.183
```

Figure 12.

e. nslookup

Nslookup, which stands for "name server lookup" is used to query the domain name system (DNS) for domain name or IP address mapping, or to obtain other kinds of DNS records.

```
C:\Users\HP>nslookup www.google.com
Server:  dns.google
Address:  8.8.8.8

Non-authoritative answer:
Name:    www.google.com
Addresses:  2404:6800:4009:828::2004
          142.250.192.36
```

Figure 13.

You can use nslookup to troubleshoot issues related to DNS. For example, if there's an issue with name resolution for DNS, you can use nslookup to check the IP address linked to a domain name, or to look at which domain name is linked to which IP address. This way you can check whether the addresses are resolved correctly.

LAB EXPERIMENT 3

3. Write a python program to implement parity check algorithm for error detection

STEP1: for each character in the input stream, count the number of 1's

STEP2: if number of 1's is odd add 1 to the original input stream

STEP3: if number of 1's is even add 0 to the original input stream

STEP4: display the output stream

```
def evenParity(input):
    output=input
    count=0
    for ch in input:
        if ch=='1':
            count+=1
    if count%2==1:
        output+="1"
    else:
        output += "0"
    return output
if __name__=="__main__":
    input=input("Enter a Binary Bit sequence: ")
    print("The output for Even parity is: ",evenParity(input))
```

Figure 15.

LAB EXPERIMENT 4

4. Write a python program to implement checksum algorithm for error detection

```
# Function to find the Checksum of Sent Message
def findChecksum(SentMessage, k):
    # Dividing sent message in packets of k bits.
    c1 = SentMessage[0:k]
    c2 = SentMessage[k:2 * k]
    c3 = SentMessage[2 * k:3 * k]
    c4 = SentMessage[3 * k:4 * k]
    print("==============",c1,int(c1, 2))
    print("==============", c2, int(c2, 2))
    print("==============", c3, int(c3, 2))
    print("==============", c4, int(c4, 2))
    print("==============", bin(632))
    # Calculating the binary sum of packets
    Sum = bin(int(c1, 2) + int(c2, 2) + int(c3, 2) + int(c4, 2))[2:]

    # Adding the overflow bits
    if (len(Sum) > k):
        x = len(Sum) - k
        Sum = bin(int(Sum[0:x], 2) + int(Sum[x:], 2))[2:]
    if (len(Sum) < k):
        Sum = '0' * (k - len(Sum)) + Sum

    print("Sum is ",Sum)
    # Calculating the complement of sum
    Checksum = ''
```

Figure 16(a).

```
    for i in Sum:
        if (i == '1'):
            Checksum += '0'
        else:
            Checksum += '1'
    return Checksum


# Function to find the Complement of binary addition of
# k bit packets of the Received Message + Checksum
def checkReceiverChecksum(ReceivedMessage, k, Checksum):
    # Dividing sent message in packets of k bits.
    c1 = ReceivedMessage[0:k]
    c2 = ReceivedMessage[k:2 * k]
    c3 = ReceivedMessage[2 * k:3 * k]
    c4 = ReceivedMessage[3 * k:4 * k]

    # Calculating the binary sum of packets + checksum
    ReceiverSum = bin(int(c1, 2) + int(c2, 2) +
                      int(c3, 2) + int(c4, 2) + int(Checksum, 2))[2:]

    # Adding the overflow bits
    if (len(ReceiverSum) > k):
        x = len(ReceiverSum) - k
        ReceiverSum = bin(int(ReceiverSum[0:x], 2) + int(ReceiverSum[x:], 2))[2:]

    # Calculating the complement of sum
    ReceiverChecksum = ''
```

Figure 16(b).

```
    for i in ReceiverSum:
        if (i == '1'):
            ReceiverChecksum += '0'
        else:
            ReceiverChecksum += '1'
    return ReceiverChecksum


# Driver Code
SentMessage = "10010101011000111001010011101100"
k = 8
ReceivedMessage = "10010101011000111001010011101100"

# Calling the findChecksum() funtion
Checksum = findChecksum(SentMessage, k)
print("Checksum is ",Checksum)
# Calling thr checkReceiverChecksum() function
ReceiverChecksum = checkReceiverChecksum(ReceivedMessage, k, Checksum)

# Printing Checksum
print("SENDER SIDE CHECKSUM: ", Checksum)
print("RECEIVER SIDE CHECKSUM: ", ReceiverChecksum)

# If sum = 0, No error is detected
if (int(ReceiverChecksum, 2) == 0):
    print("Receiver Checksum is equal to 0. Therefore,")
    print("STATUS: ACCEPTED")
```

Figure 16(c).

LAB EXPERIMENT 5

5. Write a python program to implement CRC algorithm for error detection

CRC ENCODE

```python
def xor(a, b):
    # initialize result
    result = []
    # Traverse all bits, if bits are same, then XOR is 0, else 1
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')
    return ''.join(result)

def mod2div(divident, divisor):
    # Number of bits to be XORed at a time.
    pick = len(divisor)
    # Slicing the divident to appropriate length for particular step
    tmp = divident[0: pick]
    while pick < len(divident):
        if tmp[0] == '1':
            # replace the divident by the result of XOR and pull 1 bit down
            tmp = xor(divisor, tmp) + divident[pick]
        else:  # If leftmost bit is '0'
            # If the leftmost bit of the dividend (or the part used in each
            # step) is 0, the step cannot use the regular divisor; we need to
            # use an all-0s divisor.
            tmp = xor('0' * pick, tmp) + divident[pick]
        # increment pick to move further
        pick += 1
```

Figure 17(a).

```
    # For the last n bits, we have to carry it out normally as increased
    # value of pick will cause Index Out of Bounds.
    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0' * pick, tmp)
    checkword = tmp
    return checkword

# Function used at the sender side to encode data by appending remainder
# of modular division at the end of data.
def encodeData(data, key):
    l_key = len(key)
    # Appends n-1 zeroes at end of data
    appended_data = data + '0' * (l_key - 1)
    remainder = mod2div(appended_data, key)
    # Append remainder in the original data
    codeword = data + remainder
    return codeword


input_string = input("Enter data you want to send->")
# s.sendall(input_string)
data = (''.join(format(ord(x), 'b') for x in input_string))
print(data)
key = "1001"
ans = encodeData(data, key)
print(ans)
```

Figure 17(b).

CRC DECODE

```
def xor(a, b):
    # initialize result
    result = []
    # Traverse all bits, if bits are same, then XOR is 0, else 1
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')
    return ''.join(result)

def mod2div(divident, divisor):
    # Number of bits to be XORed at a time.
    pick = len(divisor)
    # Slicing the divident to appropriate length for particular step
    tmp = divident[0: pick]
    while pick < len(divident):
        if tmp[0] == '1':
            # replace the divident by the result of XOR and pull 1 bit down
            tmp = xor(divisor, tmp) + divident[pick]
        else:  # If leftmost bit is '0'
            # If the leftmost bit of the dividend (or the part used in each
            # step) is 0, the step cannot use the regular divisor; we need to
            # use an all-0s divisor.
            tmp = xor('0' * pick, tmp) + divident[pick]
        # increment pick to move further
        pick += 1
```

Figure 17(c).

```
    # For the last n bits, we have to carry it out normally as increased
    # value of pick will cause Index Out of Bounds.
    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0' * pick, tmp)
    checkword = tmp
    return checkword

# Function used at the receiver side to decode data

def decodeData(data, key):
    l_key = len(key)
    # Appends n-1 zeroes at end of data
    appended_data = data + '0' * (l_key - 1)
    remainder = mod2div(appended_data, key)
    return remainder

received_data = input("Enter data Received->")
print(received_data)
key = "1001"
ans = decodeData(received_data, key)
print("Remainder is ",ans)
if int(ans) == 0:
    print("Accept")
else :
    print("Error")
```

Figure 17(d).

LAB EXPERIMENT 6

6. Write a python program to implement bit stuffing and de-stuffing algorithms

```python
def bitstuff(str):
    result=""
    counter=0
    prev=""
    for c in str:
        result+=c
        if c=="0":
            counter=0
            prev=c
        else:
            if prev=="0":
                counter+=1
        if counter==5:
            prev=""
            counter = 0
            result += "0"
    return result
input=input("Enter the input String:")
print(input)
print("The input after stuffing: ",bitstuff(input))
```

Figure 18(a) – Bit Stuffing.

```python
def bitdestuff(str):
    result=""
    counter=0
    prev=""
    for c in str:
        if c=="1":
            if prev=="0":
                counter+=1
        else:
            if counter==5:
                counter=0
                prev=""
                c=""
            else:
                prev=c
        result += c
    return result
input=input("Enter the Received String:")
print(input)
print("The input after stuffing: ",bitdestuff(input))
```

Figure 18(b) Bit De-stuffing.

LAB EXPERIMENT 7

7. Write a simple socket program in python to connect to www.google.com

STEP1: import socket and sys packages

STEP2: use socket() function to create a socket

STEP3: obtain the IP address using gethostbyname()

STEP4: use connect() function to connect to the IP address obtained

```python
import socket
import sys
try:
    s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    #s=socket()
except socket.error as err:
    print("socket creation failed with error ",str(err))
port=80
try:
    host_ip=socket.gethostbyname('www.google.com')
    #host_ip="216.58.199.174"
    print(str(host_ip))
except socket.gaierror as err:
    print("there was an error resolving the host ",str(err))
    sys.exit()
try:
    s.connect((host_ip, port))
except socket.error as err:
    print("there was an error connecting to the host ", str(err))
    sys.exit()
print("the socket has successfully connected to ",socket.gethostname(),socket.gethostbyaddr(host_ip))
```

Figure 19.

LAB EXPERIMENT 8

8. Write a Server Chat program that will accept TCP connection from a Chat Client and send or receive messages. The Chat program must be able to send or receive messages to/from the Server Program.

SERVER CHAT PROGRAM

```
import socket
host = "127.0.0.1"
port=5556
try:
    s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
except socket.error as err:
    print("socket creation failed with error ", str(err))
    exit(0)
try:
    print("Binding the port.. "+str(port))
    s.bind((host,port))
    s.listen(5)#5 is the number of connections
except socket.error as msg:
    print("socket binding error "+str(msg)+"\n"+"Retrying.....")
conn,addr=s.accept()#it gives the connection object and the list of IP address and port
print("Connection has been established| IP "+addr[0]+" | PORT "+str(addr[1]))
while True:
    cmd = input("SERVER | Enter the Message: ")
    if cmd == 'quit':
        conn.close()
        s.close()
        exit(0)
    if len(str.encode(cmd)) > 0:
        conn.send(str.encode("SERVER: "+cmd))
        print("\nSERVER: Waiting for the CLIENT to respond...")
        client_response = str(conn.recv(1024), "utf-8")
        print(client_response, end="\n")  # end makes its go to the next line
#print("Socket created successfully!")
```

Figure 20(a).

CLIENT CHAT PROGRAM

```
import socket
import sys
host = "127.0.0.1"
port=5556
try:
    s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
except socket.error as err:
    print("socket creation failed with error ", str(err))
    exit(0)
s.connect((host,port))
while True:
    data=s.recv(1024)
    if len(data) > 0:
        print(data.decode('utf-8'),end="\n")
    cmd = input("CLIENT | Enter the Message: ")
    if cmd == 'quit':
        s.close()
        sys.exit(0)
    if len(str.encode(cmd)) > 0:
        s.send(str.encode("CLIENT: "+cmd))
        print("\nCLIENT: Waiting for the SERVER to respond...")
#print("Successfully created socket")
```

Figure 20(b).