NAME: Debadrita Roy

CLASS: BCSE-III

GROUP: A1

ASSIGNMENT NUMBER: 4

PROBLEM STATEMENT: Implement CDMA with Walsh Code.

In this assignment you have to implement CDMA for multiple access of a common channel by n stations. Each sender uses a unique code word, given by the Walsh set, to encode its data, send it across the channel, and then perfectly reconstruct the data at n stations.

DEADLINE: 28th September, 2021

DATE OF SUBMISSION: 16th November, 2021

DESIGN

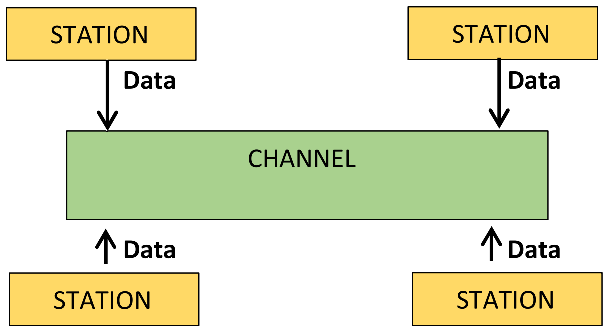
The purpose of the program is to simulate the real-world network environment and design and implement CDMA (Code Division Multiple Access). The required programs are written in Python 3.

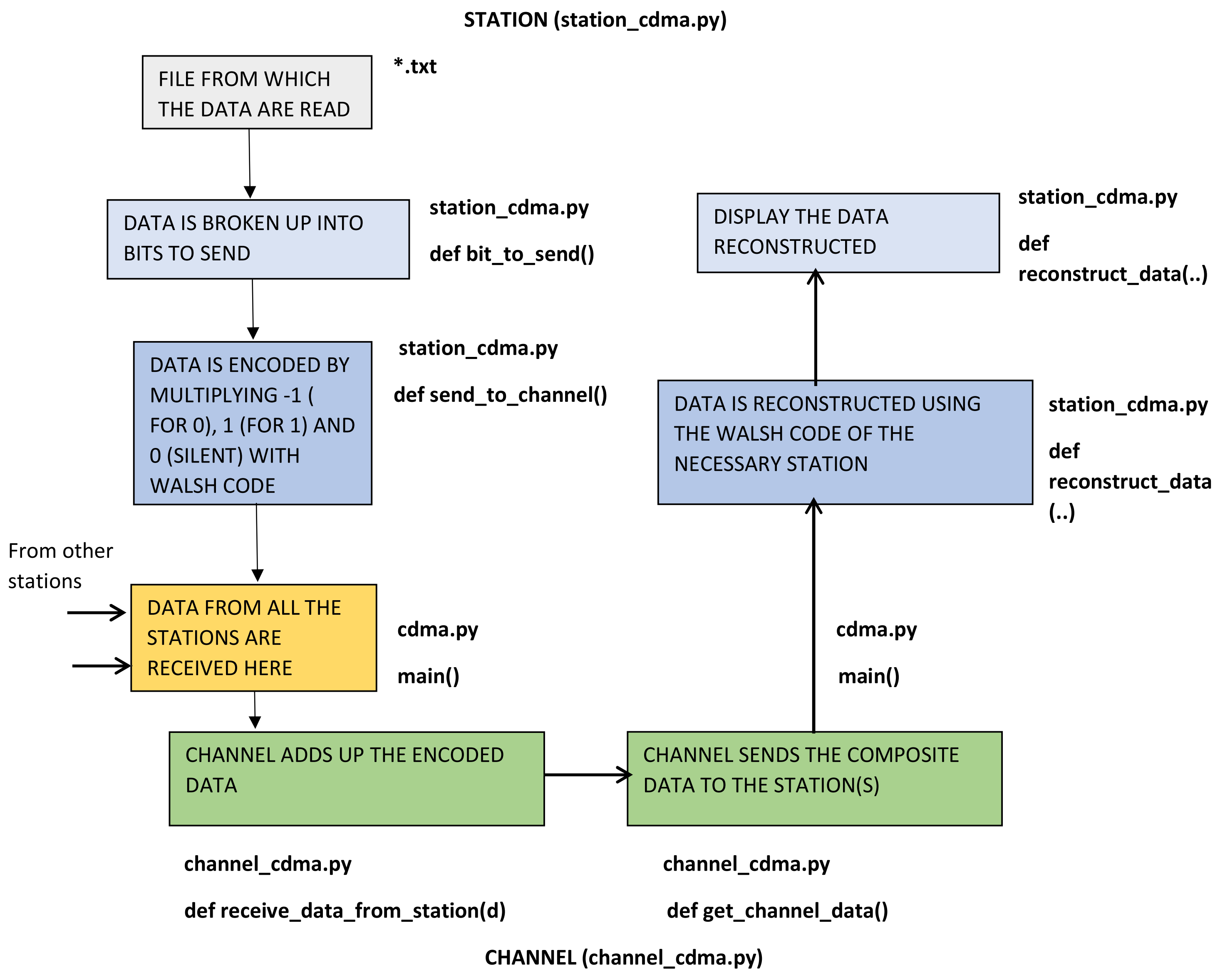
The station program (**station.cdma.py**) simulates a station. Each station is initialized with its walsh code and is responsible for sending data from an input file to the channel program (**channel_cdma.py**) after encoding the data using its walsh code. The station can also reconstruct data from any station in the network by taking the common data in the channel and the walsh code of the station whose data is to be reconstructed as inputs. The function(s) responsible for building the walsh set are written in the module **walshcode.py** which is then imported wherever it is required. The channel program contains functions for managing, receiving and sending the data.

Data Structure:

The channel carries all the transmissions simultaneously. It adds up the encoded data received from each station every time slot and sends them to the stations in the network. The data is encoded by multiplying -1,0 or 1 (depending on whether the station is sending bit 0, is silent, or is sending bit 1 to the channel).

Structure Diagram:




STATION (station_cdma.py)
FILE FROM WHICH THE DATA ARE READ
*.txt
DATA IS BROKEN UP INTO BITS TO SEND
station_cdma.py
def bit_to_send()
DATA IS ENCODED BY MULTIPLYING -1 (FOR 0), 1 (FOR 1) AND 0 (SILENT) WITH WALSH CODE
station_cdma.py
def send_to_channel()
From other stations
DATA FROM ALL THE STATIONS ARE RECEIVED HERE
cdma.py
main()
CHANNEL ADDS UP THE ENCODED DATA
channel_cdma.py
def receive_data_from_station(d)
CHANNEL SENDS THE COMPOSITE DATA TO THE STATION(S)
channel_cdma.py
def get_channel_data()
cdma.py
main()
DATA IS RECONSTRUCTED USING THE WALSH CODE OF THE NECESSARY STATION
station_cdma.py
def reconstruct_data (..)
DISPLAY THE DATA RECONSTRUCTED
station_cdma.py
def reconstruct_data(..)
CHANNEL (channel_cdma.py)

Input Format:

The number of stations in the network is taken as input from the user.

Output Format:

The data sent by each station to the channel is displayed on the terminal. The data carried by the channel at every time slot is also displayed. The required data after reconstruction at a station is also displayed on the terminal. Lastly, the total channel processing time and the total transmission time is displayed.

IMPLEMENTATION

Station_cdma.py

The Station class is responsible for sending the data from a file which is provided to the class, encoding it using a unique codeword from the walsh set and sending it to the channel. It also provides functionality for reconstructing the data sent from a given station using the common data sent from the channel and the walsh code of the station.

def __init__(self,num,walsh)

Method Description: The constructor initializes the number of stations in the network, the walsh code of this station, the pointer to the bit in the file which is to be sent next and stores null value in the variable for storing the data to be read.

Code Snippet:

```
def __init__(self,num,walsh):
  self.n=num
  self.wc=walsh
  self.ptr=0
  self.fc=''
```

def set_file_name(self,file)

Method Description: This method stores the file containing the data into the variable fc and initializes the pointer ptr showing the index of the next bit to be sent to the channel.

Code Snippet:

```
def set_file_name(self,file):
  f=open(file,'r')
  self.fc=f.read()
  f.close()
  self.ptr=0
```

def bit_to_send(self)

Method Description: This method returns -1,0 or 1 depending upon the bit to be sent (0, none and 1 respectively). It also increases ptr so that it points to the next bit to be sent.

Code Snippet:

```
def bit_to_send(self):
  if self.ptr<len(self.fc):
    ch=self.fc[self.ptr]
    self.ptr=self.ptr+1
    if ch=='0':
      return -1
    else:
      return 1
  else:
    return 0
```

def send_to_channel(self)

Method Description: This method returns the encoded data to be sent to the channel.

Code Snippet:

```
def send_to_channel(self):
  bit=self.bit_to_send()
  encoded_data=[0 for _ in range(len(self.wc))]
  for i in range(len(self.wc)):
    encoded_data[i]=(bit*(self.wc[i]))
  return encoded_data
```

def reconstruct_data(self)

Method Description: This method reconstructs the data sent by the station having walsh code walsh_st and displays it.

Code Snippet:

```
def reconstruct_data(self,composite_data,walsh_st):
  i=0
  sum=0
  while i< len(composite_data):
    sum=sum+(composite_data[i]*walsh_st[i])
    time.sleep(0.002)
    i=i+1
  # print(sum)
  b=sum/self.n
  # print(b)
  if b==0:
    print('Station having walsh code ',walsh_st,' is silent')
  elif b==-1:
    print('Station having walsh code ',walsh_st,' sent bit 0')
  elif b==1:
    print('Station having walsh code ', walsh_st, ' sent bit 1')
```

channel_cdma.py

The channel file contains functions to manipulate the common data in the channel.

def __init__(self,n)

Method Description: This method initializes the number of stations in the network and the data that is to be stored in the channel at the beginning.

Code Snippet:

```
def __init__(self,n):
  self.n=n
  x=walshcode.getpof2(self.n)
  self.data=[0 for _ in range(x)]
```

def refresh_channel(self)

Method Description: This method initializes the data that is to be stored in the channel at the beginning of each time slot, before all transmissions.

Code Snippet:

```
def refresh_channel(self):
    for i in range(len(self.data)):
        self.data[i]=0;
```

def get_channel_data(self)

Method Description: This method returns the common data in the channel.

Code Snippet:

```
def get_channel_data(self):
    return self.data
```

def receive_data_from_station(self,d)

Method Description: This method adds the data from the next station into the common data.

Code Snippet:

```
def receive_data_from_station(self,d):
    for i in range(len(self.data)):
        self.data[i]+=d[i]
        time.sleep(0.002)
```

cdma.py

This file contains the main() function responsible for running and testing the entire program.

main()

Method Description: It contains the code for taking user input, creating the station objects, creating the channel and providing an interface between the station_cdma.py and the channel_cdma.py programs. It also displays the total channel processing time and the total transmission time taken.

Code Snippet:

```
if __name__=="__main__":
    n=int(input("Enter the number of stations: "))
    p=walshcode.getpof2(n)
    w=[[0 for i in range(p)] for j in range(p)]
    walshcode.buildWalshTable(w,p,0,p-1,0,p-1)
    # print(w)
    stations_list=[]
    cpt=0
    trt=0
    for i in range(n):
        st=station_cdma.Station(n,w[i])
        st.set_file_name('cdmafile'+str(1+(i%4))+'.txt')
        stations_list.append(st)
    ch=channel_cdma.Channel(n)
    for i in range(20):  # 20 time slots tested
        ch.refresh_channel()  # so that the channel contains 0's at the beginning of each time
slot
        st_time=time.time()
        for st in stations_list:
            d=st.send_to_channel()
            # print('Station ',(stations_list.index(st)+1),' sends ',d)
            time.sleep(0.005)
            ch.receive_data_from_station(d)
        cpt+=(time.time()-st_time)
        # print('Data in channel now: ',ch.get_channel_data())
        # print('Data from station 1 is to be reconstructed')
        stations_list[n-1].reconstruct_data(ch.get_channel_data(),w[0])
        trt+=(time.time()-st_time)
        # print('------------next time slot-----------------')
    print('Total channel processing time:',cpt,' s')
    print('Total transmission time:',trt,' s')
```

walshcode.py

This module contains functions for building the walsh set and calculating the smallest power of 2 greater than or equal to the number of stations in the network. It is imported by cdma.py and channel_cdma.py programs.

def getpof2(num)

Method Description: This method returns the smallest power of 2 greater than or equal to the given number (num).

Code Snippet:

```
def getpof2(num):
    p=1
    while p < num:
        p*=2
    return p
```

def buildWalshTable(w,l,i1,i2,j1,j2,comp=False)

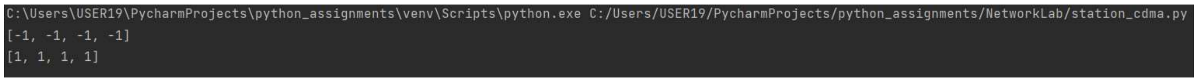
Method Description: This method builds the walsh table for a given length l and stores it in w.

Code Snippet:

```
def buildWalshTable(w,l, i1, i2, j1, j2, comp=False):
    if l == 2:
        if not comp:
            w[i1][j1] = 1
            w[i1][j2] = 1
            w[i2][j1] = 1
            w[i2][j2] = -1
        else:
            w[i1][j1] = -1
        w[i1][j2] = -1
        w[i2][j1] = -1
        w[i2][j2] = 1
      return
    midi = (i1 + i2) // 2
    midj = (j1 + j2) // 2
    buildWalshTable(w,l/2, i1, midi, j1, midj,comp)
    buildWalshTable(w,l/2, i1, midi, midj + 1, j2,comp)
    buildWalshTable(w,l/2, midi + 1, i2, j1, midj,comp)
    buildWalshTable(w,l/2, midi + 1, i2, midj + 1, j2,not comp)
    return
```

TEST CASES

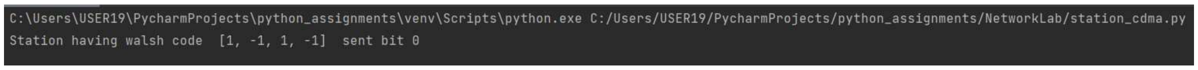
Sample Test 1: To check the working of the encoder part of the station



```
C:\Users\USER19\PycharmProjects\python_assignments\venv\Scripts\python.exe C:/Users/USER19/PycharmProjects/python_assignments/NetworkLab/station_cdma.py
[-1, -1, -1, -1]
[1, 1, 1, 1]
```

The walsh code for given station was [1,1,1,1] and the two bits to be sent were 0 and 1, respectively.

Sample Test 2: To check the working of the reconstruction part of the station

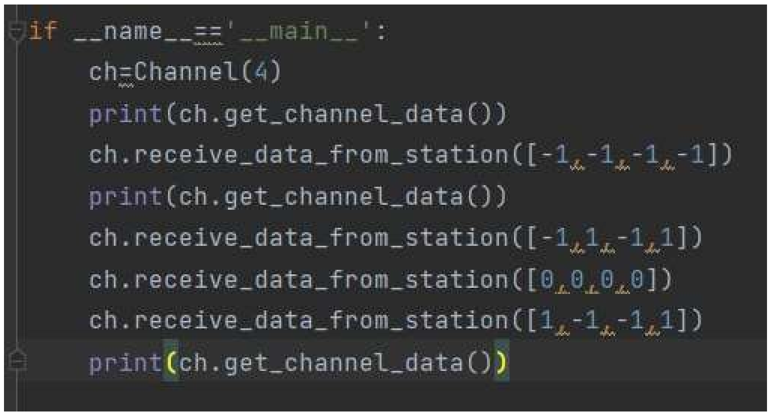


```
C:\Users\USER19\PycharmProjects\python_assignments\venv\Scripts\python.exe C:/Users/USER19/PycharmProjects/python_assignments/NetworkLab/station_cdma.py
Station having walsh code  [1, -1, 1, -1]  sent bit 0
```

The common data in the channel was [-1,-1,-3,1] and the walsh code of the particular station was [1,-1,1,-1].

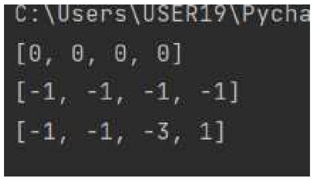
Sample Test 3: To check the working of the channel

TEST



```python
if __name__=='__main__':
    ch=Channel(4)
    print(ch.get_channel_data())
    ch.receive_data_from_station([-1,-1,-1,-1])
    print(ch.get_channel_data())
    ch.receive_data_from_station([-1,1,-1,1])
    ch.receive_data_from_station([0,0,0,0])
    ch.receive_data_from_station([1,-1,-1,1])
    print(ch.get_channel_data())
```

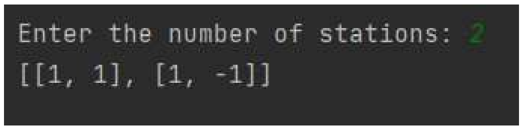
OUTPUT



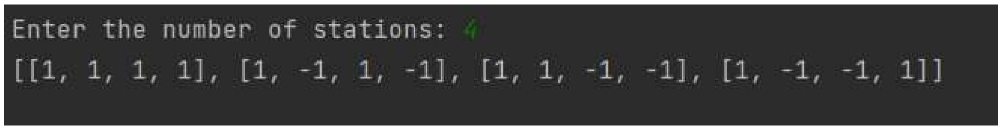
```
C:\Users\USER19\Pycha
[0, 0, 0, 0]
[-1, -1, -1, -1]
[-1, -1, -3, 1]
```

As we can see, the channel is working properly.

Sample Test 4: To check whether the function responsible for building the walsh table is working properly or not



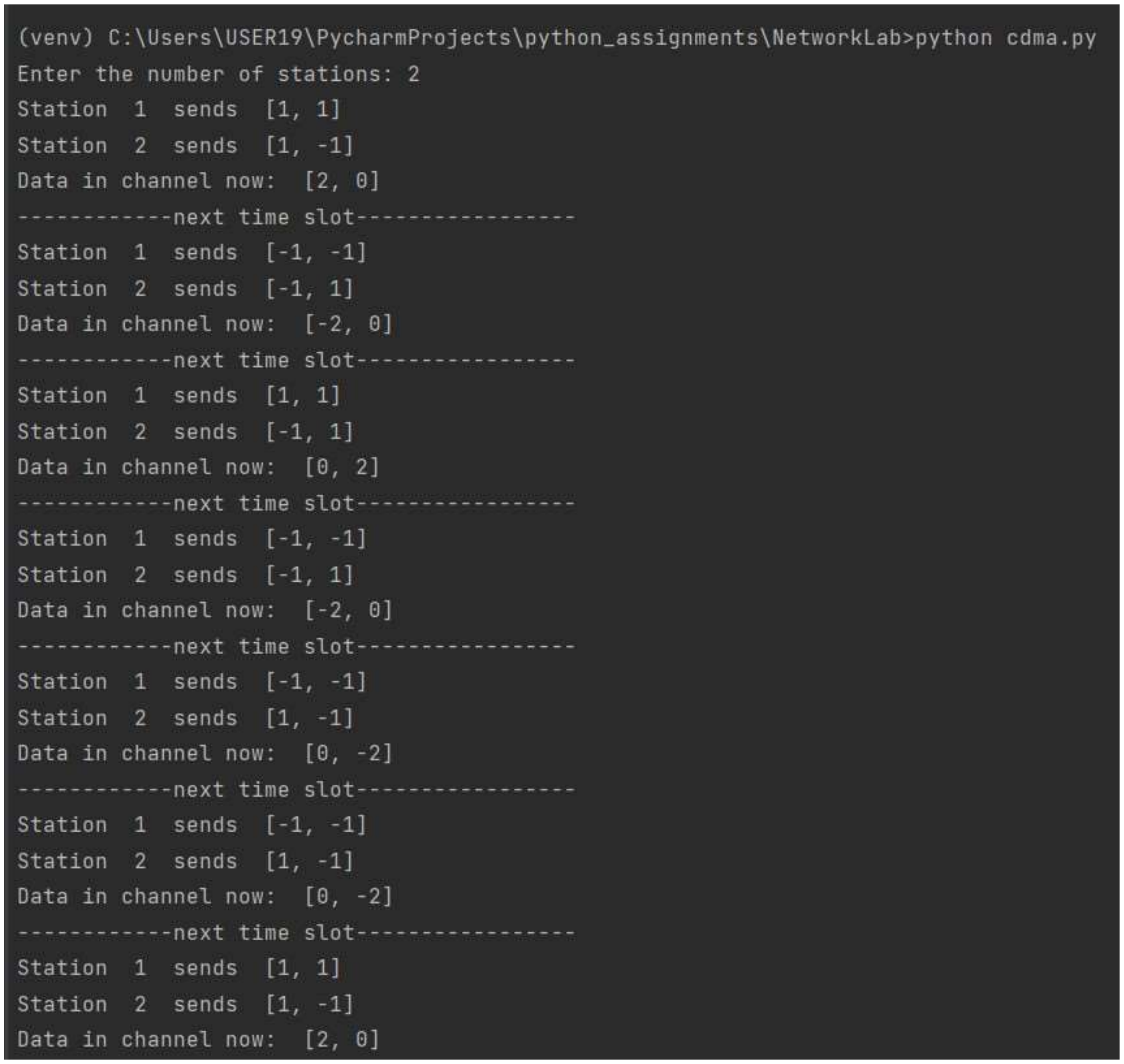
```
Enter the number of stations: 2
[[1, 1], [1, -1]]
```



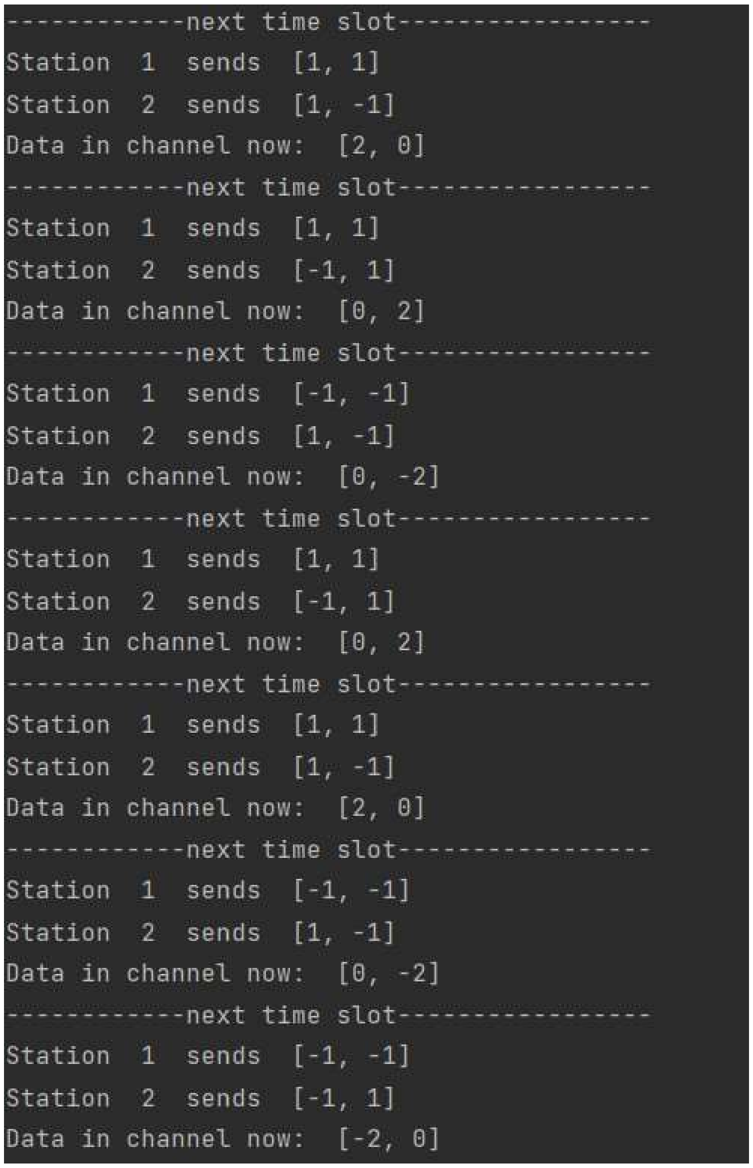
```
Enter the number of stations: 4
[[1, 1, 1, 1], [1, -1, 1, -1], [1, 1, -1, -1], [1, -1, -1, 1]]
```

From the above inputs, the walsh table is shown to be calculated correctly.

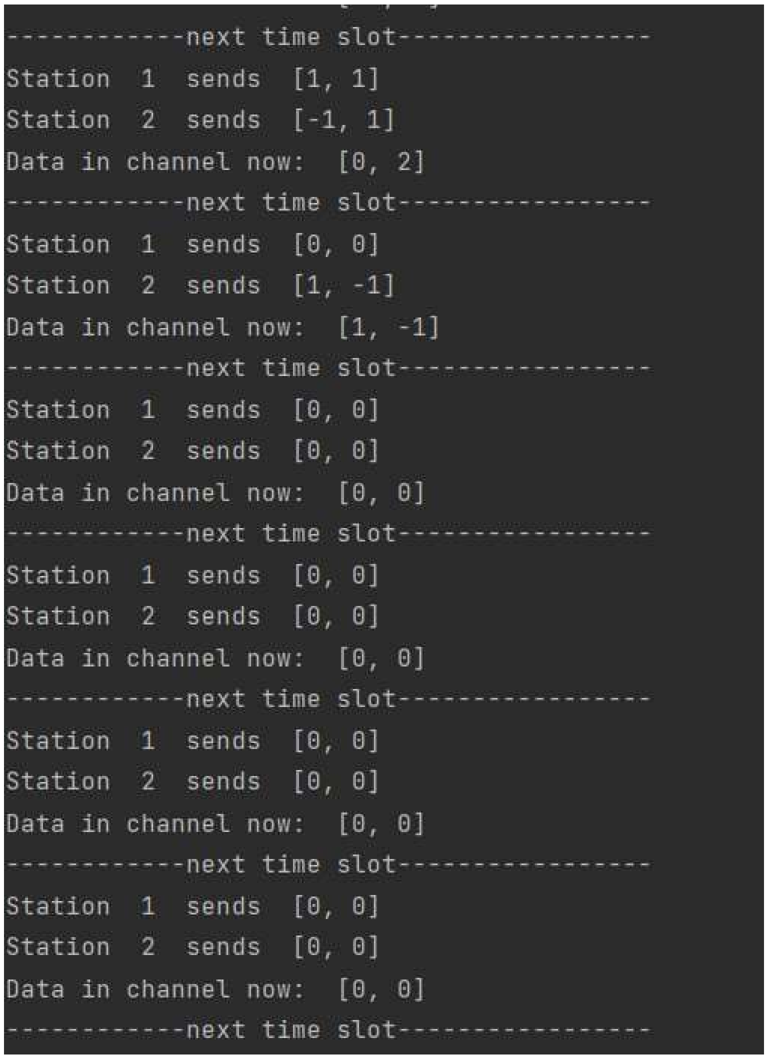
Sample Test 5: To check the working of the sending part of the stations and the making of the common data in the channel for input files containing data



```
(venv) C:\Users\USER19\PycharmProjects\python_assignments\NetworkLab>python cdma.py
Enter the number of stations: 2
Station  1  sends  [1, 1]
Station  2  sends  [1, -1]
Data in channel now:  [2, 0]
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [-1, 1]
Data in channel now:  [-2, 0]
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [-1, 1]
Data in channel now:  [0, 2]
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [-1, 1]
Data in channel now:  [-2, 0]
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [1, -1]
Data in channel now:  [0, -2]
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [1, -1]
Data in channel now:  [0, -2]
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [1, -1]
Data in channel now:  [2, 0]
```



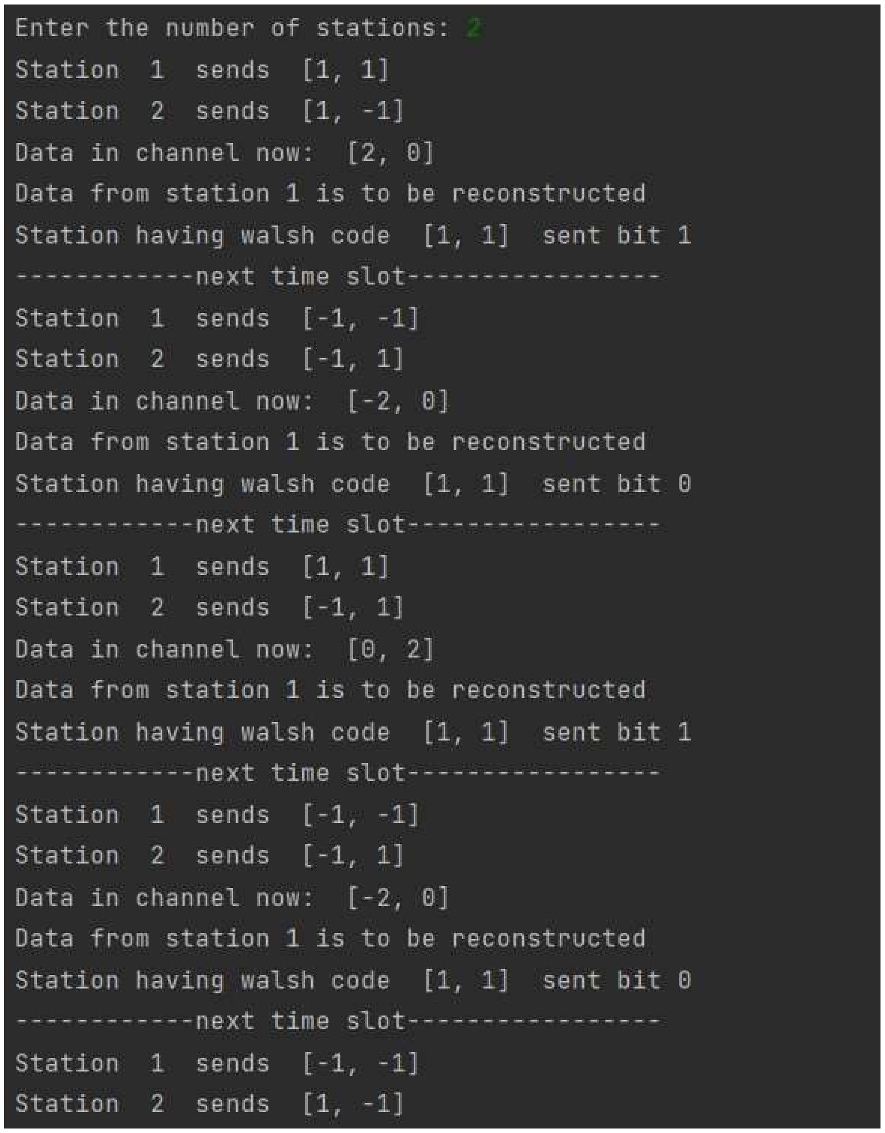
```
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [1, -1]
Data in channel now:  [2, 0]
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [-1, 1]
Data in channel now:  [0, 2]
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [1, -1]
Data in channel now:  [0, -2]
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [-1, 1]
Data in channel now:  [0, 2]
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [1, -1]
Data in channel now:  [2, 0]
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [1, -1]
Data in channel now:  [0, -2]
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [-1, 1]
Data in channel now:  [-2, 0]
```



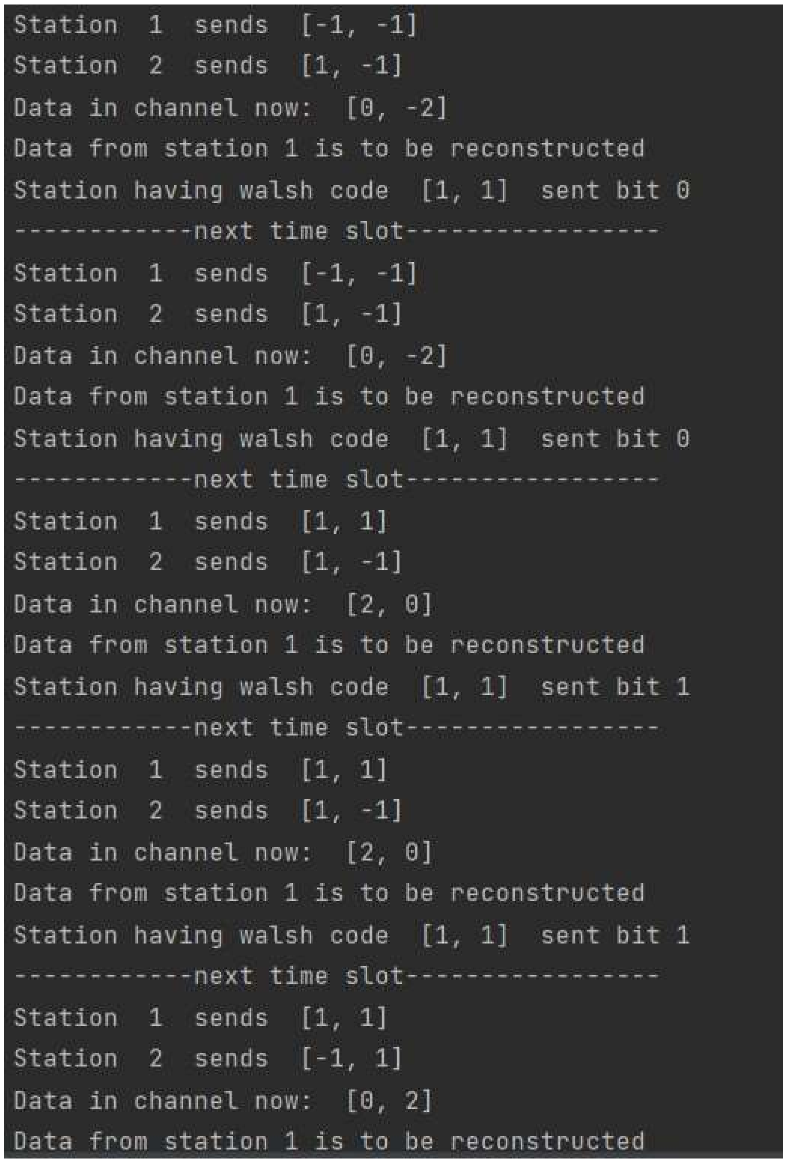
```
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [-1, 1]
Data in channel now:  [0, 2]
------------next time slot-----------------
Station  1  sends  [0, 0]
Station  2  sends  [1, -1]
Data in channel now:  [1, -1]
------------next time slot-----------------
Station  1  sends  [0, 0]
Station  2  sends  [0, 0]
Data in channel now:  [0, 0]
------------next time slot-----------------
Station  1  sends  [0, 0]
Station  2  sends  [0, 0]
Data in channel now:  [0, 0]
------------next time slot-----------------
Station  1  sends  [0, 0]
Station  2  sends  [0, 0]
Data in channel now:  [0, 0]
------------next time slot-----------------
Station  1  sends  [0, 0]
Station  2  sends  [0, 0]
Data in channel now:  [0, 0]
------------next time slot-----------------
```

Clearly, the parts are working correctly.

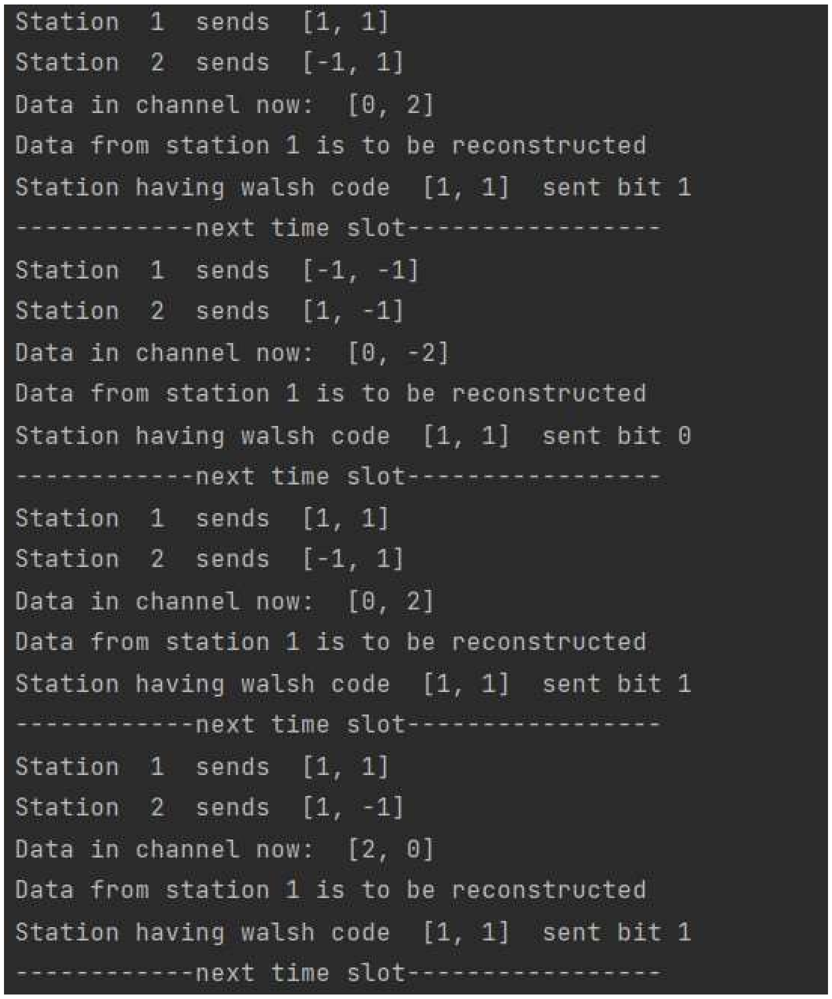
Sample Test 6: To check the correctness of the reconstruction part and the overall program



```
Enter the number of stations: 2
Station  1  sends  [1, 1]
Station  2  sends  [1, -1]
Data in channel now:  [2, 0]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 1
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [-1, 1]
Data in channel now:  [-2, 0]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 0
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [-1, 1]
Data in channel now:  [0, 2]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 1
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [-1, 1]
Data in channel now:  [-2, 0]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 0
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [1, -1]
```



```
Station  1  sends  [-1, -1]
Station  2  sends  [1, -1]
Data in channel now:  [0, -2]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 0
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [1, -1]
Data in channel now:  [0, -2]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 0
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [1, -1]
Data in channel now:  [2, 0]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 1
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [1, -1]
Data in channel now:  [2, 0]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 1
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [-1, 1]
Data in channel now:  [0, 2]
Data from station 1 is to be reconstructed
```



```
Station  1  sends  [1, 1]
Station  2  sends  [-1, 1]
Data in channel now:  [0, 2]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 1
------------next time slot-----------------
Station  1  sends  [-1, -1]
Station  2  sends  [1, -1]
Data in channel now:  [0, -2]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 0
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [-1, 1]
Data in channel now:  [0, 2]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 1
------------next time slot-----------------
Station  1  sends  [1, 1]
Station  2  sends  [1, -1]
Data in channel now:  [2, 0]
Data from station 1 is to be reconstructed
Station having walsh code  [1, 1]  sent bit 1
------------next time slot-----------------
```

Thus, we have verified the correctness of the working of the different parts of our programs.

RESULTS

We set up a number of stations with files containing some random bits. The stations send the encoded data bits to the channel and the channel after processing the data from the different stations, sends the composite data to a station designated as the receiver.

Performance metrics for evaluation: We vary the number of stations in the network and measure the total channel processing time (time for adding up the encoded data bits from each station), the total transmission time (time spent from the sending of the encoded data to the channel to the time taken to reconstruct the data from a given station at the designated receiver station for every time slot added up). Delays were introduced at various places to simulate the real-world transmission and processing delays.

Observation Tables:

Each observation percentage is correct to two decimal places.

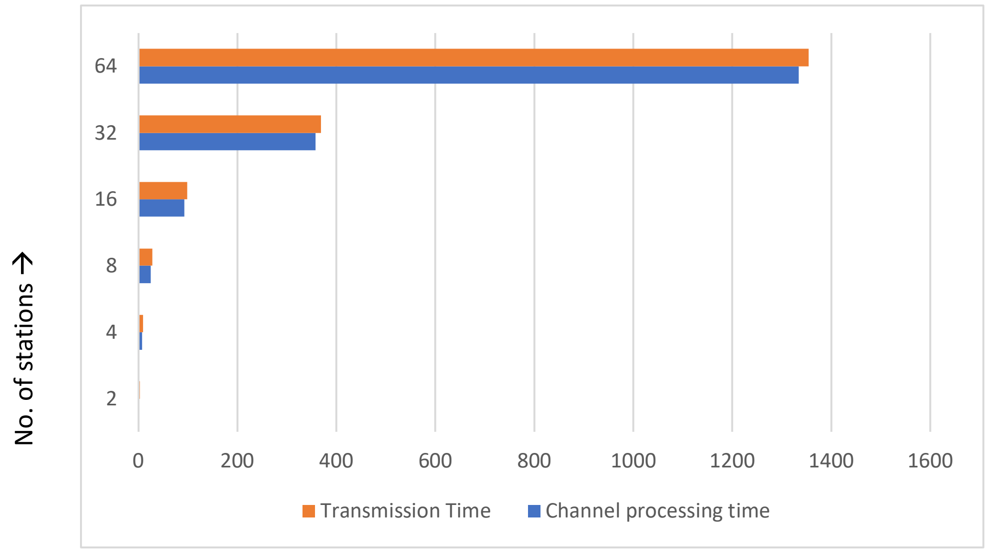
CHANNEL PROCESSING TIME

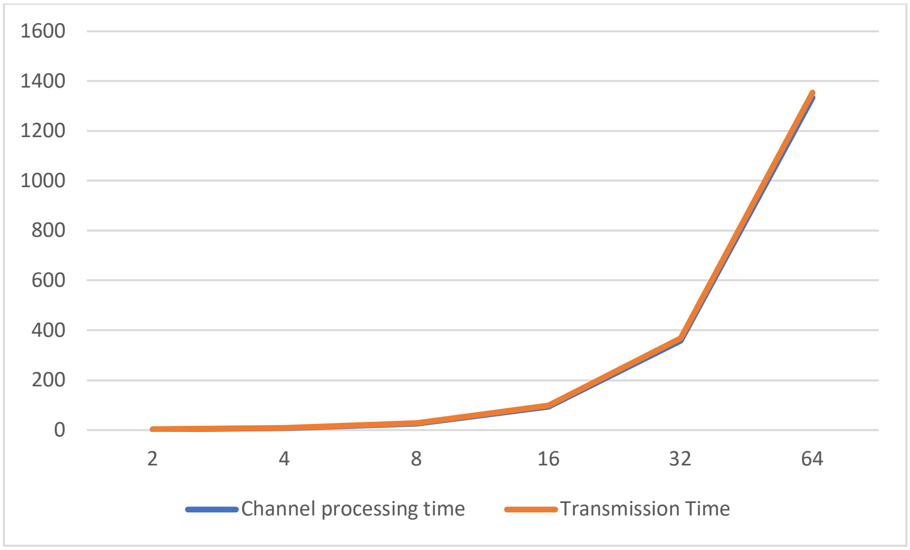
NUMBER OF STATIONS	CHANNEL PROCESSING TIME (IN SECONDS)
2	2.13
4	7.45
8	24.95
16	92.97
32	358.13
64	1334.08

TOTAL TRANSMISSION TIME

NUMBER OF STATIONS	TOTAL TRANSMISSION TIME (IN SECONDS)
2	2.85
4	8.94
8	27.76
16	98.42
32	369.05
64	1354.42

Graphs:





The above graphs show how the channel processing time and the total transmission time vary with increase in the number of stations.

ANALYSIS

From the test cases and observations, we found that the channel processing time and the total transmission time increases with the increase in the number of stations in the network. Also, as the channel carries the data of all the transmissions simultaneously, a large bandwidth is required compared to others. However, multiple stations can access the channel simultaneously. So, CDMA is good for when there are less number of stations in the network and also when we do not need to worry about transmission time or channel processing time but we require simultaneous reception of the data from different stations to more than one station. Other techniques may transfer data from multiple senders to multiple receivers but CDMA is unique in that it sends the data from all the stations simultaneously. The transmission time is more or less same for every station receiving the data from the channel. So, CDMA can be used when we require simultaneous transmission of data from multiple senders to one or more receivers.

Possible Improvements: With the program as it is now, there is no functionality for error detection or correction. Also, the channel is assumed to be noiseless. In the real-world, however, channels are noisy and so the data can get corrupted or lost. Techniques to handle these scenarios may be incorporated in the program.

COMMENTS

The lab assignment was interesting as it allowed us to implement CDMA and simulate it in a real-world scenario. We could check the observations we read about in theory and see how the different parts of the programs work both independently and as a whole. Writing the programs was not particularly difficult after figuring out the design and the algorithms needed, however it was not very easy either. I liked the CDMA protocol for multiple access of a channel in the theory class so I was interested in implementing it in a programming environment. Building up the walsh set was also a tricky part.