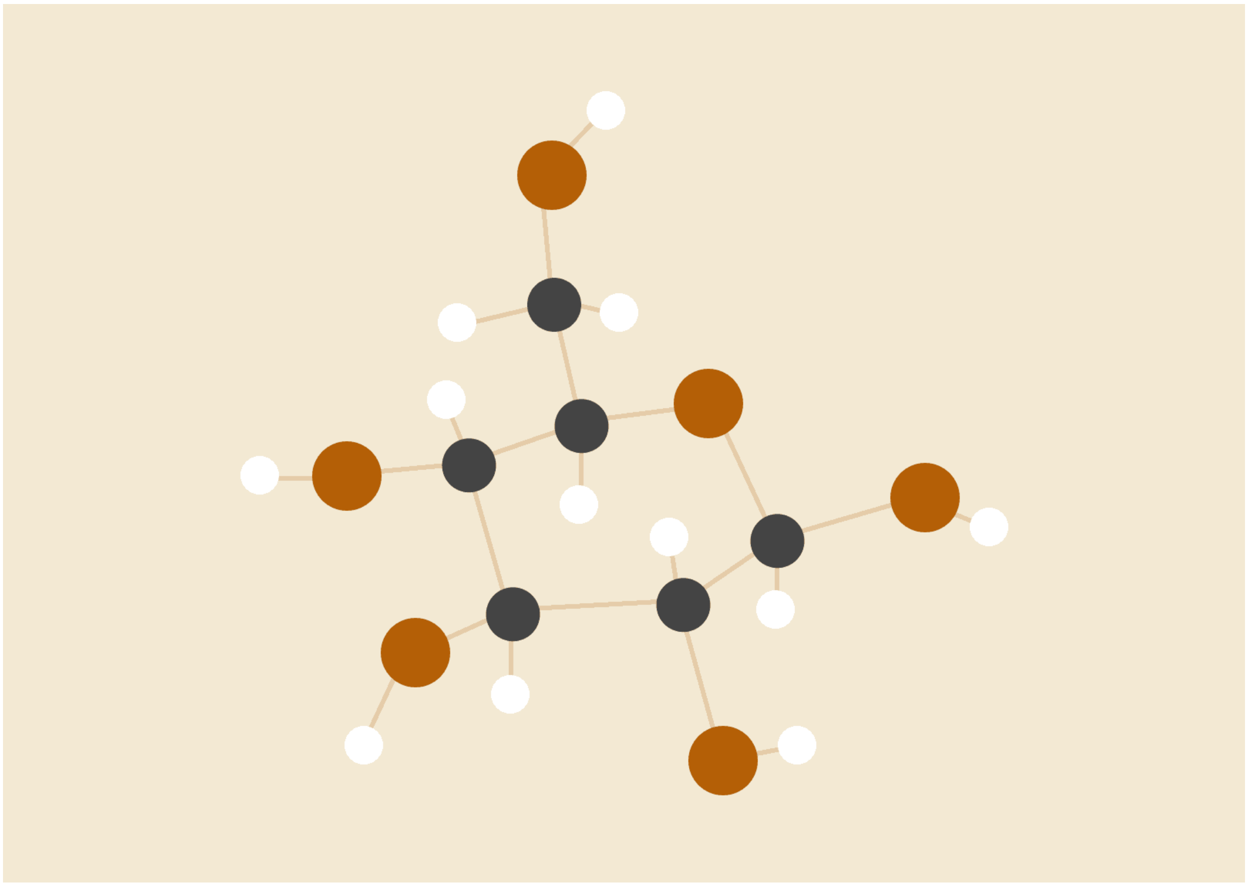
NETWORKING LAB REPORT

CLASS *BCSE III* ***SEM*** *FIFTH* ***YEAR*** *2021*



NAME Neeladri Pal

ROLL 001910501015

GROUP A1

ASSIGNMENT - 4

PROBLEM STATEMENT

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.

CDMA

CDMA is a multiple-access method in which the available bandwidth of a link is shared through code between different stations.

- One channel occupies the entire bandwidth of the link.
- Multiple stations can transmit simultaneously.

CDMA is based on Walsh Code generation.
Every station is assigned a unique code which when multiplied with itself generates the number of senders, whereas with other station's code the product comes out to be zero.

A station encodes the bit -1 as 0, 1 as 1 and 0 if it remains silent. These bits are multiplied by their respective Walsh code and the sum of coded data from all senders are sent to all the receivers. The receiver then multiplies the sum with the sender's Walsh code to get the data bit sent by that particular sender.

DESIGN

Sender – Here sender sends whole data bit by bit from a given file multiplied by its Walsh code to the channel.

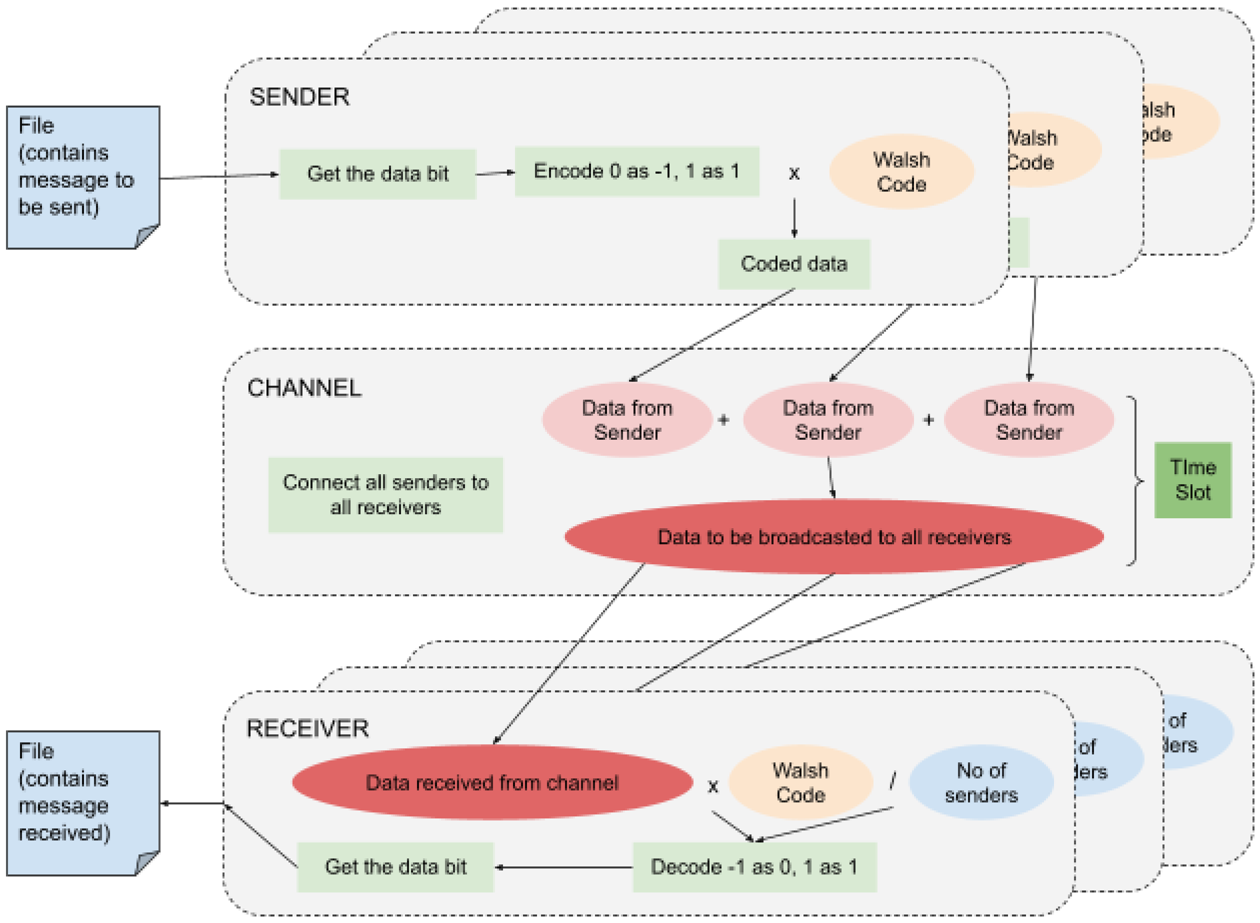
Receiver – Here receiver receives data sent from a particular sender via channel, extracts it by multiplying with the sender's Walsh code and dividing by the number of stations and stores the received data in a specified file.

Channel – It serves as the communication link between the sender and receiver. It collects coded data sent from the sender, adds them and sends them to all the receivers in the next time slot.

Data size for coded data is the length of the Walsh code.

At each time slot only one bit (1/-1) is sent from each sender using Walsh code.

GENERAL DESIGN OF CDMA



WALSH CODE GENERATION

$$W_1 : \begin{bmatrix} +1 \end{bmatrix} \qquad W_{2N} : \begin{bmatrix} W_N & W_N \\ W_N & -W_N \end{bmatrix}$$

$$W_1 : \begin{bmatrix} +1 \end{bmatrix}$$

$$W_2 : \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix}$$

$$W_4 : \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}$$

IMPLEMENTATION

Walsh Code Generation →

```
def buildWalshTable(wTable, length, i1,i2, j1,j2, isComplement):
    if length == 2:
        if not isComplement:
            wTable[i1][j1] = 1
            wTable[i1][j2] = 1
            wTable[i2][j1] = 1
            wTable[i2][j2] = -1
        else:
            wTable[i1][j1] = -1
            wTable[i1][j2] = -1
            wTable[i2][j1] = -1
            wTable[i2][j2] = 1

        return

    midi = (i1+i2)//2
    midj = (j1+j2)//2

    buildWalshTable(wTable, length/2, i1, midi, j1, midj, isComplement)
    buildWalshTable(wTable, length/2, i1, midi, midj+1, j2, isComplement)
    buildWalshTable(wTable, length/2, midi+1, i2, j1, midj, isComplement)
    buildWalshTable(wTable, length/2, midi+1, i2, midj+1, j2, not
isComplement)
```

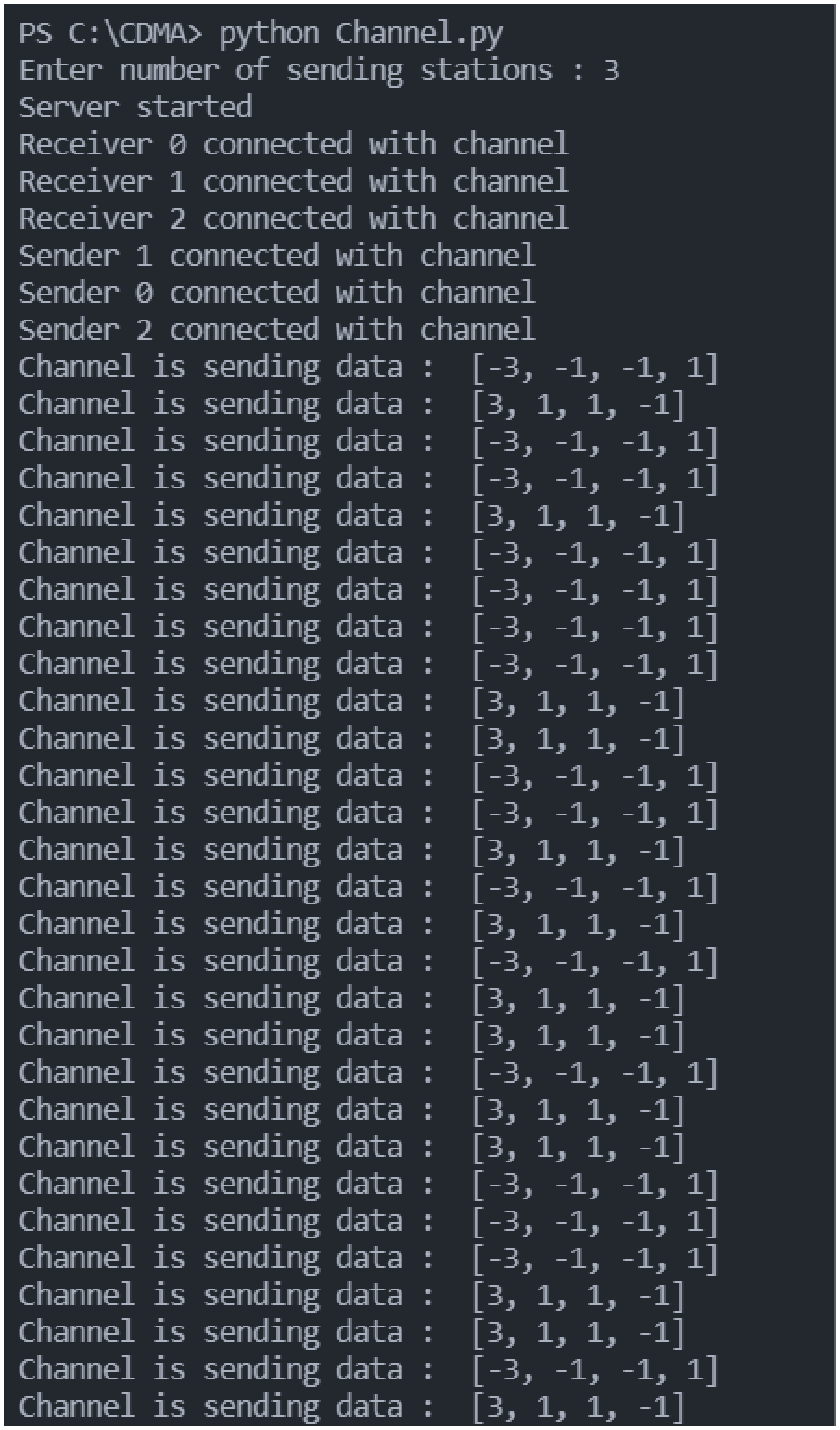
Sender Side Encoding →

```
# Read first byte of data
        byte = file.read(1)

        # loop until whole data is sent
        while byte:
            # Print coded data to be sent
            print('Sender ',self.senderNo,' is sending character : ',byte)

            # send the data bytes
            data = '{0:08b}'.format(ord(byte))

            # Convert character byte into code, 0 -> -1, 1 -> 1
            for i in range(len(data)):
                dataToSend = []
                dataBit = int(data[i])
                if dataBit == 0: dataBit = -1

                # Print data to be sent and walsh code
                print('Sender ',self.senderNo,' is sending data-bit :
',dataBit,' with Walsh-code : ',self.walshCode)

                # Build coded-data from Walsh Code
                for j in self.walshCode:
                    dataToSend.append(j * dataBit)

                # Print coded data to be sent
                print('Sender ',self.senderNo,' is sending coded data :
',dataToSend)

                # Send coded data to channel
                self.connection.send(pickle.dumps(dataToSend))
                self.pktCount += 1

                # Wait for next time slot
                time.sleep(timeSlot)

            byte = file.read(1)
```

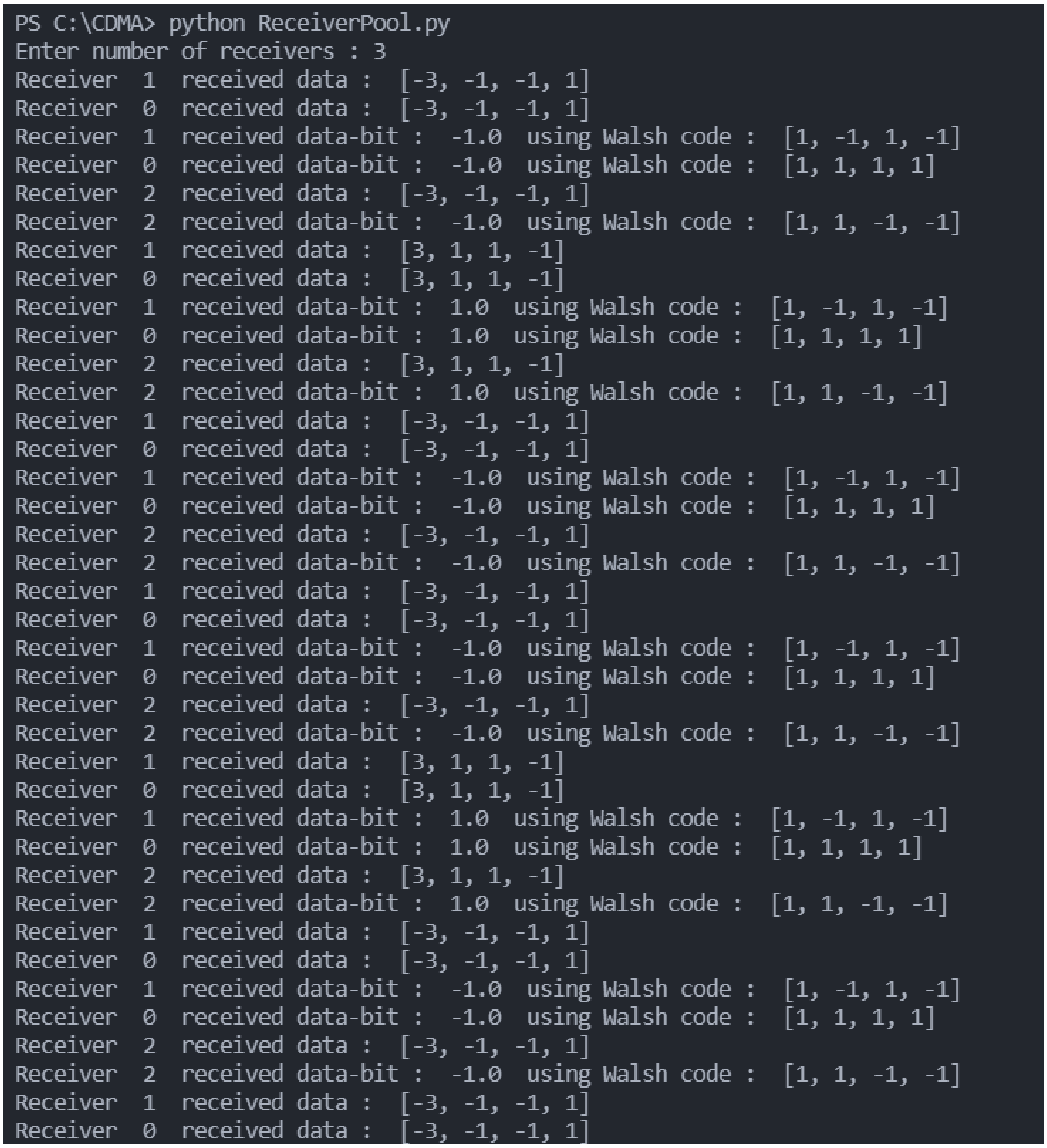
Receiver Side Decoding →

```
# Function for receiving data
    def receive(self):

        # Wait for data and receive
        data = pickle.loads(self.connection.recv(1024))
        self.startTime = time.time()

        wholeData = ''
        totalData = []
        while data!='end':
            # Print received data
            print('Receiver ',self.receiverNo,' received data : ',data)

            # extract data
            summation = 0
            for i in range(len(data)):
                summation += data[i] * self.walshCode[i]

            # extract databit
            summation /= len(self.walshCode)

            # Print coded data to be sent
            print('Receiver ',self.receiverNo,' received data-bit :
',summation,' using Walsh code : ',self.walshCode)

            if summation == 1:
                bit = 1
            elif summation == -1:
                bit = 0
            else: bit = -1

            self.pktCount += 1

            if bit != -1:
                # append the bit to build the character
                totalData.append(str(bit))

            if len(totalData) == 8:
                character = self.getCharacter(totalData)
                # Print received character
                print('Receiver ',self.receiverNo,' received character :
',character)
                wholeData += character
                totalData = []

             # Wait for data and receive
            data = pickle.loads(self.connection.recv(1024))

        if wholeData != '':
            # print(total_data)
            file = open(self.file_name,'a')
            file.write(wholeData)
            file.write('\n')
            file.close()
```

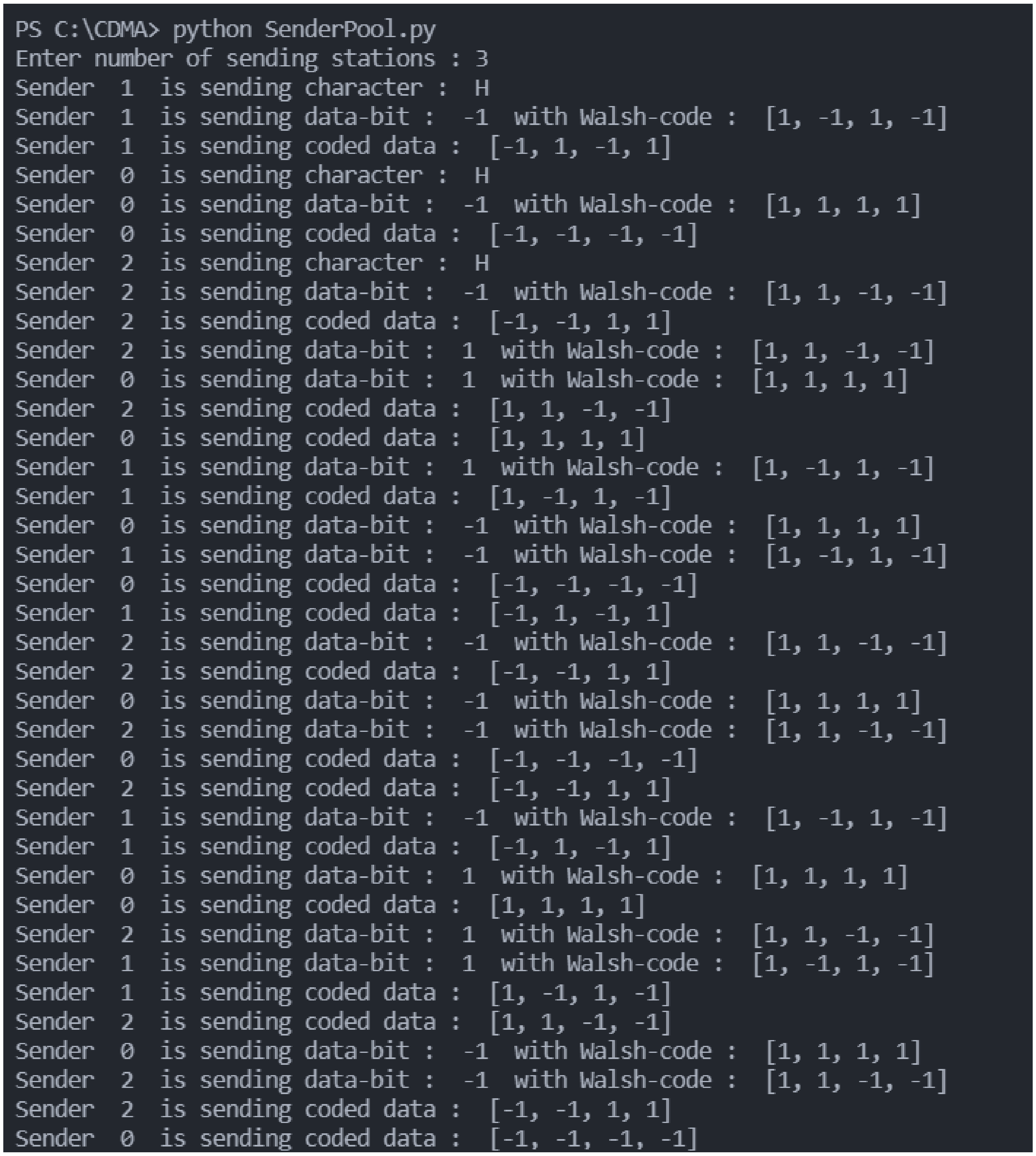
OUTPUT



```
PS C:\CDMA> python Channel.py
Enter number of sending stations : 3
Server started
Receiver 0 connected with channel
Receiver 1 connected with channel
Receiver 2 connected with channel
Sender 1 connected with channel
Sender 0 connected with channel
Sender 2 connected with channel
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [3, 1, 1, -1]
Channel is sending data :  [-3, -1, -1, 1]
Channel is sending data :  [3, 1, 1, -1]
```



```
PS C:\CDMA> python ReceiverPool.py
Enter number of receivers : 3
Receiver  1  received data :  [-3, -1, -1, 1]
Receiver  0  received data :  [-3, -1, -1, 1]
Receiver  1  received data-bit :  -1.0  using Walsh code :  [1, -1, 1, -1]
Receiver  0  received data-bit :  -1.0  using Walsh code :  [1, 1, 1, 1]
Receiver  2  received data :  [-3, -1, -1, 1]
Receiver  2  received data-bit :  -1.0  using Walsh code :  [1, 1, -1, -1]
Receiver  1  received data :  [3, 1, 1, -1]
Receiver  0  received data :  [3, 1, 1, -1]
Receiver  1  received data-bit :  1.0  using Walsh code :  [1, -1, 1, -1]
Receiver  0  received data-bit :  1.0  using Walsh code :  [1, 1, 1, 1]
Receiver  2  received data :  [3, 1, 1, -1]
Receiver  2  received data-bit :  1.0  using Walsh code :  [1, 1, -1, -1]
Receiver  1  received data :  [-3, -1, -1, 1]
Receiver  0  received data :  [-3, -1, -1, 1]
Receiver  1  received data-bit :  -1.0  using Walsh code :  [1, -1, 1, -1]
Receiver  0  received data-bit :  -1.0  using Walsh code :  [1, 1, 1, 1]
Receiver  2  received data :  [-3, -1, -1, 1]
Receiver  2  received data-bit :  -1.0  using Walsh code :  [1, 1, -1, -1]
Receiver  1  received data :  [-3, -1, -1, 1]
Receiver  0  received data :  [-3, -1, -1, 1]
Receiver  1  received data-bit :  -1.0  using Walsh code :  [1, -1, 1, -1]
Receiver  0  received data-bit :  -1.0  using Walsh code :  [1, 1, 1, 1]
Receiver  2  received data :  [-3, -1, -1, 1]
Receiver  2  received data-bit :  -1.0  using Walsh code :  [1, 1, -1, -1]
Receiver  1  received data :  [3, 1, 1, -1]
Receiver  0  received data :  [3, 1, 1, -1]
Receiver  1  received data-bit :  1.0  using Walsh code :  [1, -1, 1, -1]
Receiver  0  received data-bit :  1.0  using Walsh code :  [1, 1, 1, 1]
Receiver  2  received data :  [3, 1, 1, -1]
Receiver  2  received data-bit :  1.0  using Walsh code :  [1, 1, -1, -1]
Receiver  1  received data :  [-3, -1, -1, 1]
Receiver  0  received data :  [-3, -1, -1, 1]
Receiver  1  received data-bit :  -1.0  using Walsh code :  [1, -1, 1, -1]
Receiver  0  received data-bit :  -1.0  using Walsh code :  [1, 1, 1, 1]
Receiver  2  received data :  [-3, -1, -1, 1]
Receiver  2  received data-bit :  -1.0  using Walsh code :  [1, 1, -1, -1]
Receiver  1  received data :  [-3, -1, -1, 1]
Receiver  0  received data :  [-3, -1, -1, 1]
```



```
PS C:\CDMA> python SenderPool.py
Enter number of sending stations : 3
Sender  1  is sending character :  H
Sender  1  is sending data-bit :  -1  with Walsh-code :  [1, -1, 1, -1]
Sender  1  is sending coded data :  [-1, 1, -1, 1]
Sender  0  is sending character :  H
Sender  0  is sending data-bit :  -1  with Walsh-code :  [1, 1, 1, 1]
Sender  0  is sending coded data :  [-1, -1, -1, -1]
Sender  2  is sending character :  H
Sender  2  is sending data-bit :  -1  with Walsh-code :  [1, 1, -1, -1]
Sender  2  is sending coded data :  [-1, -1, 1, 1]
Sender  2  is sending data-bit :  1  with Walsh-code :  [1, 1, -1, -1]
Sender  0  is sending data-bit :  1  with Walsh-code :  [1, 1, 1, 1]
Sender  2  is sending coded data :  [1, 1, -1, -1]
Sender  0  is sending coded data :  [1, 1, 1, 1]
Sender  1  is sending data-bit :  1  with Walsh-code :  [1, -1, 1, -1]
Sender  1  is sending coded data :  [1, -1, 1, -1]
Sender  0  is sending data-bit :  -1  with Walsh-code :  [1, 1, 1, 1]
Sender  1  is sending data-bit :  -1  with Walsh-code :  [1, -1, 1, -1]
Sender  0  is sending coded data :  [-1, -1, -1, -1]
Sender  1  is sending coded data :  [-1, 1, -1, 1]
Sender  2  is sending data-bit :  -1  with Walsh-code :  [1, 1, -1, -1]
Sender  2  is sending coded data :  [-1, -1, 1, 1]
Sender  0  is sending data-bit :  -1  with Walsh-code :  [1, 1, 1, 1]
Sender  2  is sending data-bit :  -1  with Walsh-code :  [1, 1, -1, -1]
Sender  0  is sending coded data :  [-1, -1, -1, -1]
Sender  2  is sending coded data :  [-1, -1, 1, 1]
Sender  1  is sending data-bit :  -1  with Walsh-code :  [1, -1, 1, -1]
Sender  1  is sending coded data :  [-1, 1, -1, 1]
Sender  0  is sending data-bit :  1  with Walsh-code :  [1, 1, 1, 1]
Sender  0  is sending coded data :  [1, 1, 1, 1]
Sender  2  is sending data-bit :  1  with Walsh-code :  [1, 1, -1, -1]
Sender  1  is sending data-bit :  1  with Walsh-code :  [1, -1, 1, -1]
Sender  1  is sending coded data :  [1, -1, 1, -1]
Sender  2  is sending coded data :  [1, 1, -1, -1]
Sender  0  is sending data-bit :  -1  with Walsh-code :  [1, 1, 1, 1]
Sender  2  is sending data-bit :  -1  with Walsh-code :  [1, 1, -1, -1]
Sender  2  is sending coded data :  [-1, -1, 1, 1]
Sender  0  is sending coded data :  [-1, -1, -1, -1]
```

RESULTS

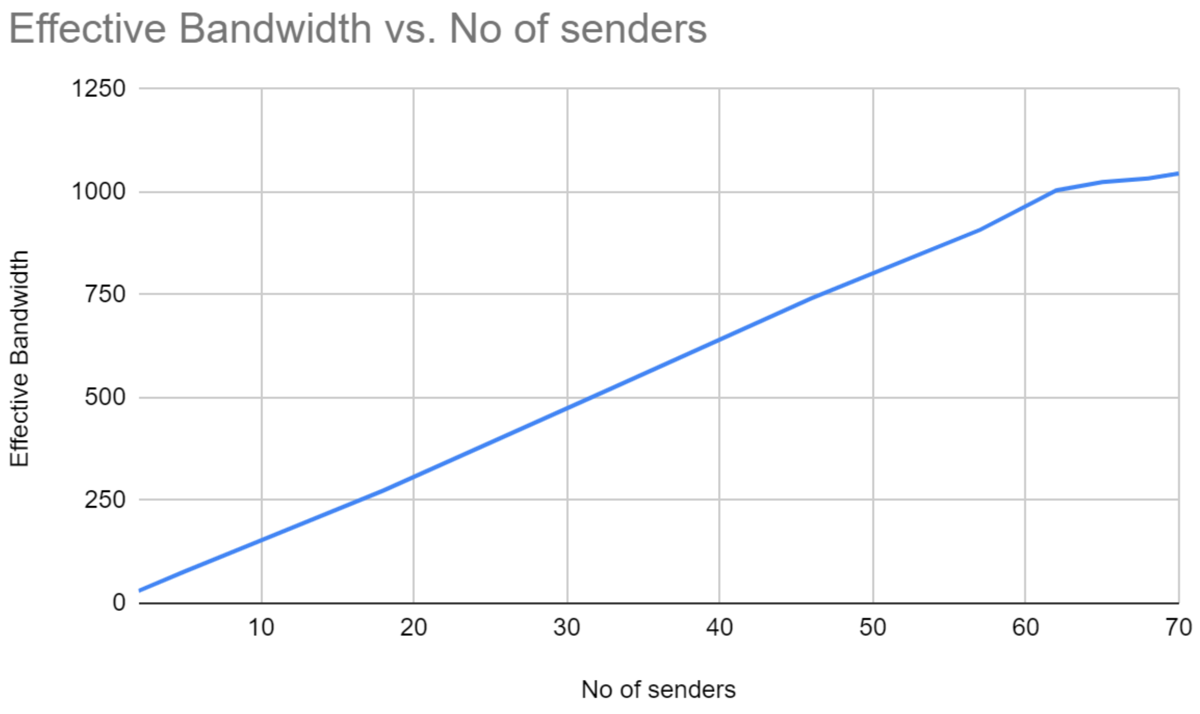
Time Slot - 5 ms

Effective Bandwidth = Total data bits sent by all senders / Total time taken

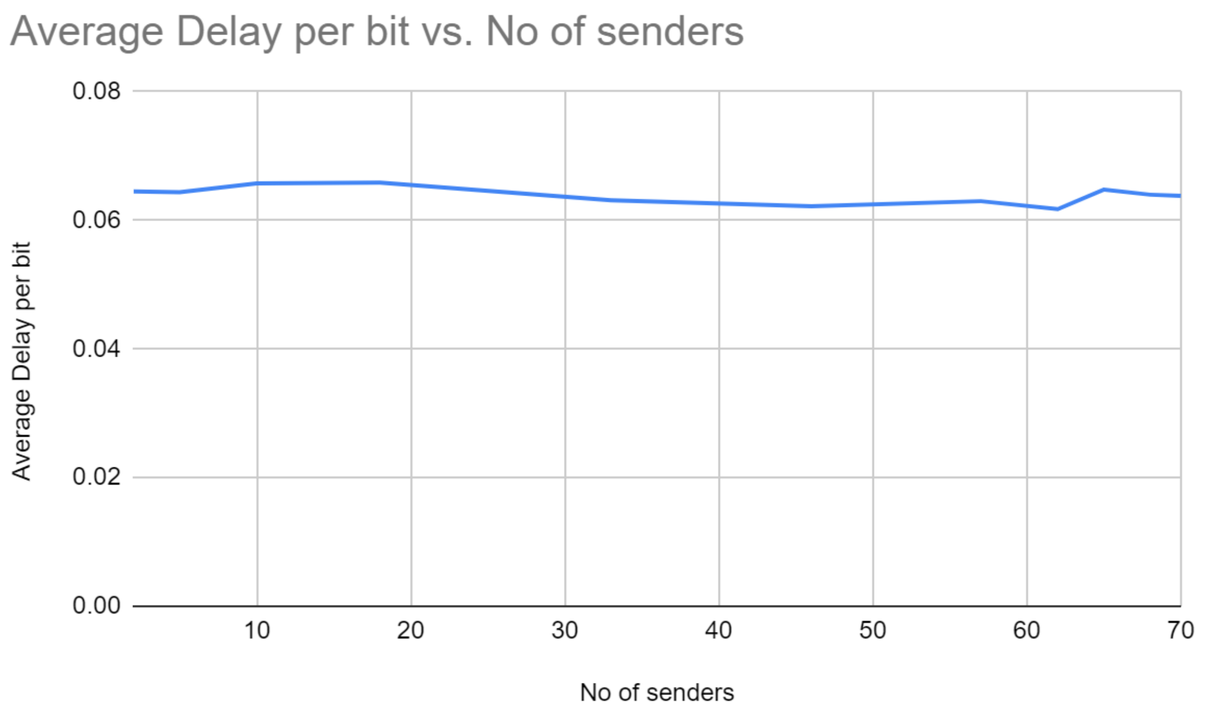
Average Delay = Total time taken / Total data bits sent by each sender

No of senders	Bits per sender	Total time taken (in sec)	Effective Bandwidth (bps)	Average Delay (sec/bit)
2	96	6.193621	30	0.064517
5	96	6.246387	77	0.064396
10	96	6.507643	153	0.065734
18	96	6.983813	274	0.065885
33	96	6.942727	524	0.063116
46	96	6.840396	742	0.062185
57	96	7.117216	908	0.062984
62	96	7.101081	1004	0.061749
65	96	7.375795	1024	0.064789
68	96	7.124794	1033	0.063997
70	96	7.120487	1045	0.063819

ANALYSIS



As the number of senders increases, the bits sent per slot increase and so does the effective bandwidth. Also the time needed to generate the Walsh code increases. Above 64 senders, 128 Walsh codes are to be generated which may take long. So some additional time slots may be needed. It is this point that the effective bandwidth shows a slight dip in its increasing nature.



The average increases a bit in the beginning when there is a frequent increase in the length of Walsh code. Then it decreases slowly because more bits pass through the same time slot and then again when the Walsh code becomes huge, the delay increases a bit.

COMMENTS

The assignment tells how CDMA can be implemented using Walsh code. It is a very simple implementation. So the data is quite reliable and less prone to errors. No frame format or noise was considered to make the data more interpretable and easy to analyse.