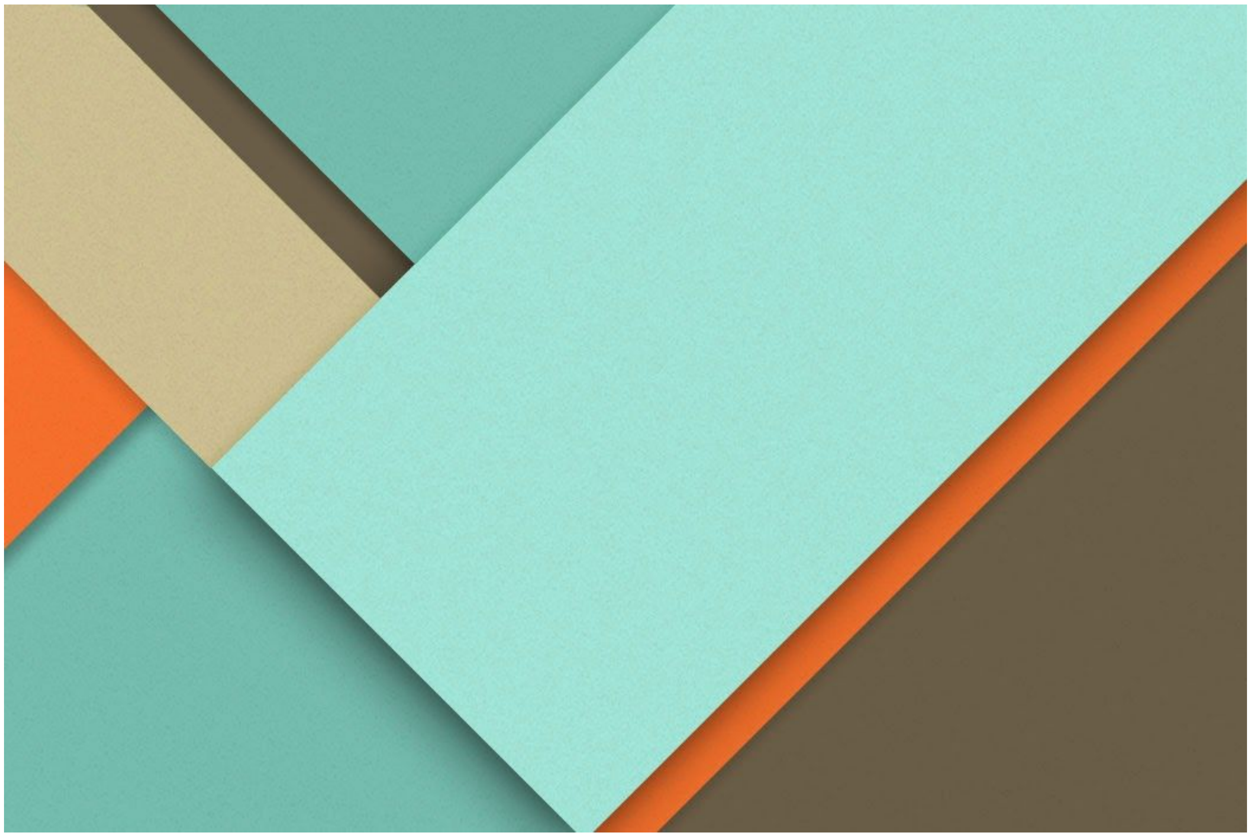


Computer Networks Lab

ASSIGNMENT 4

SAMARPAN CHAKRAVARTY
BCSE - UG III
001810501026
29/12/2020

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.

Specifications

1. Language used for implementation: Python 3.6
2. The entire module is written based on Inter Process Communication, using pipes for communication between the various processes.

Theory

WHAT IS CDMA?

Code-division multiple access (CDMA) is a channel access method used by various radio communication technologies. CDMA is an example of multiple access, where several transmitters can send information simultaneously over a single communication channel. This allows several users to share a band of frequencies. To permit this without undue interference between the users, CDMA employs spread spectrum technology and a special coding scheme where each transmitter is assigned a code, popularly known as the Walsh code.
CDMA has a number of distinguishing features that are key to spread spectrum transmission technologies:

1. Use of wide bandwidth: CDMA, like other spread spectrum technologies uses a wider bandwidth than would otherwise be needed for the transmission of the data. This results in a number of advantages including an increased immunity to interference or jamming, and multiple user access.
2. Spreading codes used: In order to achieve the increased bandwidth, the data is spread by use of a code which is independent of the data.
3. Level of security: In order to receive the data, the receiver must have a knowledge of the spreading code, without this it is not possible to decipher the transmitted data, and this gives a measure of security.
4. Multiple access: The use of the spreading codes which are independent for each user along with synchronous reception allow multiple users to access the same channel simultaneously.

WHAT ARE THE ADVANTAGES OF CDMA?

CDMA provides the following advantages over other techniques:

1. CDMA requires a tight power control, as it suffers from near-far effect. In other words, a user near the base station transmitting with the same power will drown the signal latter. All signals must have more or less equal power at the receiver
2.Rake receivers can be used to improve signal reception. Delayed versions of time (a chip or later) of the signal (multipath signals) can be collected and used to make decisions at the bit level.
3.Flexible transfer may be used. Mobile base stations can switch without changing operators. Two base stations receive mobile signal and the mobile receives signals from the two base stations.
4.Transmission Burst − reduces interference.

Approach

I have implemented the entire process using IPC in python, with the help of pipes. The number of sender and receiver nodes are declared at the beginning of the module. Each sender node is then assigned an unique Walsh code which is used for its transmission. All the sender nodes are then assigned to the write head of a pipe. This is a common pipe which acts as the data transfer unit between the senders and the channel process. The channel process receives the data from every sender, and sums it up. This generates a Walsh encoded message. Now, this is broadcasted to every receiver node with the help of pipes. Each receiver node has a dedicated pipe from the channel, through which it receives the message. Now the receiver uses the appropriate sender's Walsh code to decode the message, and thus the transmission is finally successful.

The entire transmission takes place bit by bit. There is no well defined frame format which is considered, only the bits are taken into account. Also, the channel process has no provision to introduce errors and delay. So basically, what I have implemented is a noise-less channel. This is not quite a practical scenario, but I considered that so that I could focus on the CDMA part more. And lastly, synchronization between the sender nodes are established by using proper sleep time after sending a data bit.

The following diagram would explain the idea better:

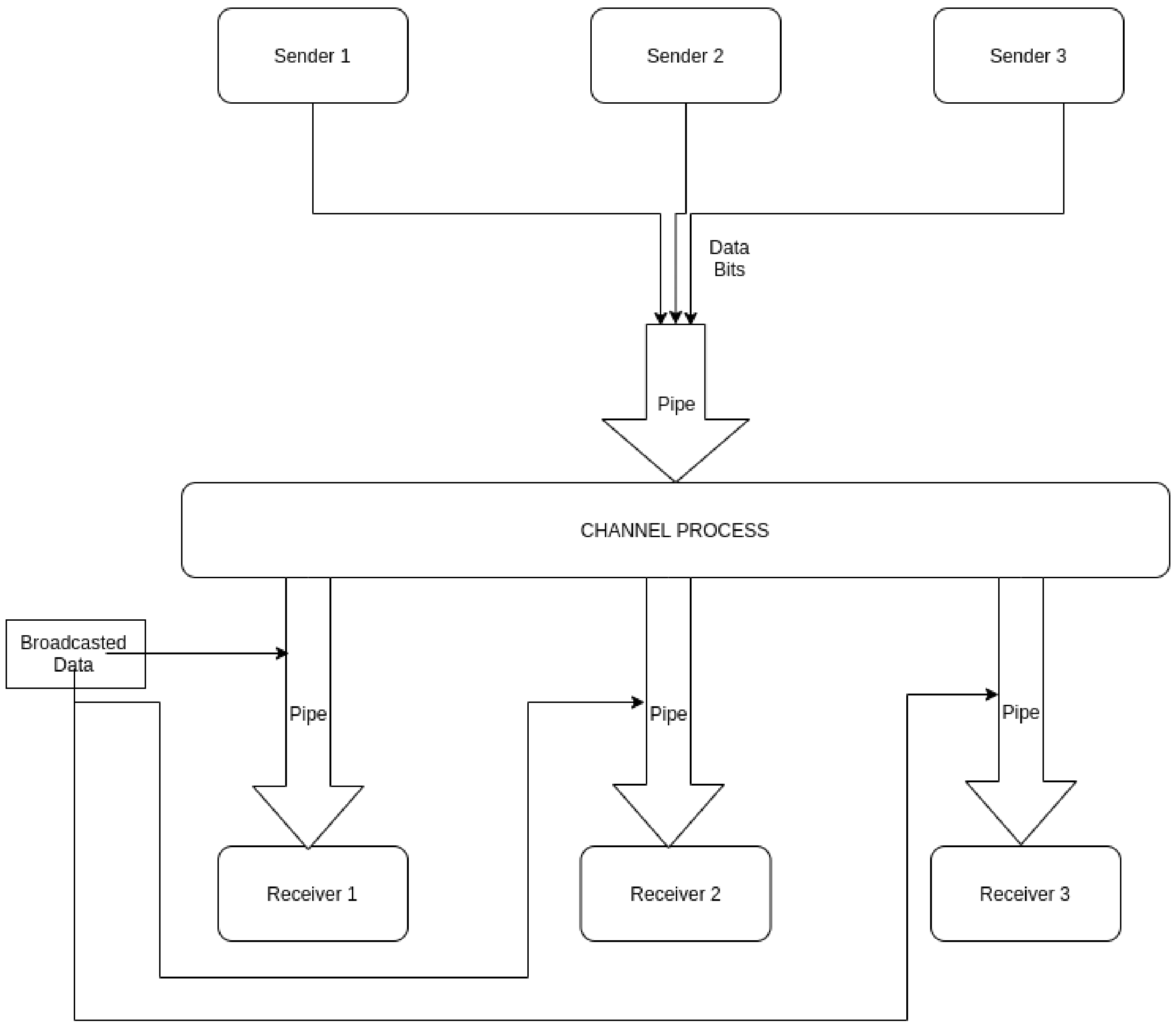


Fig: Overall implementation diagram

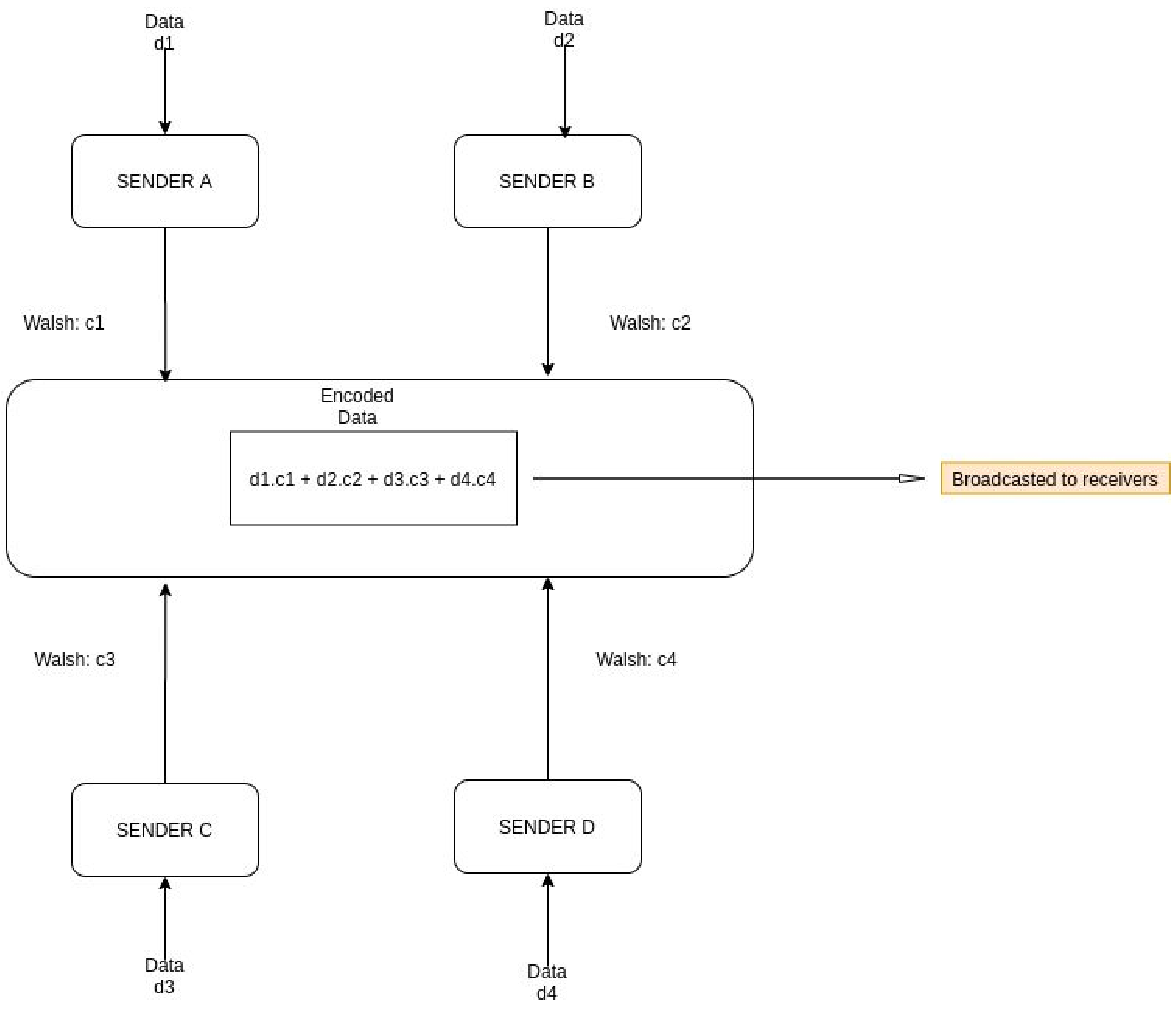
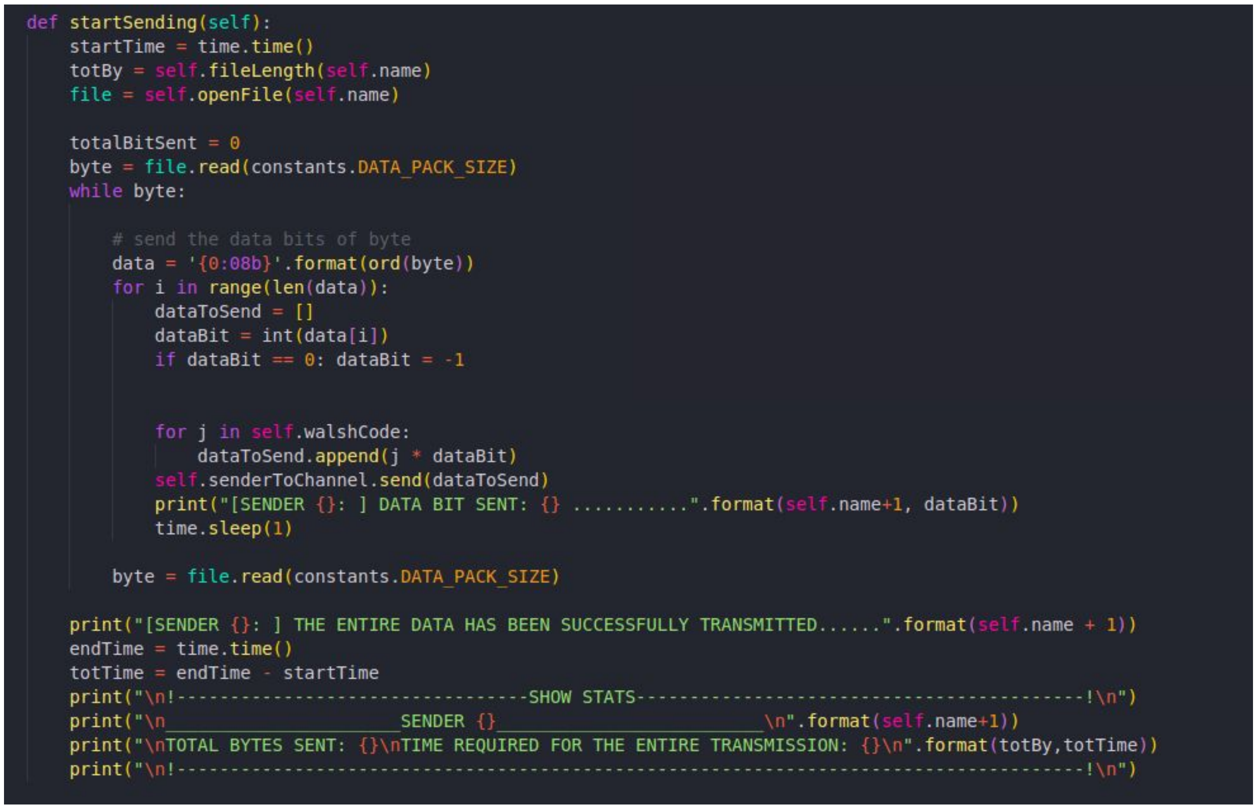


Fig: CDMA workflow

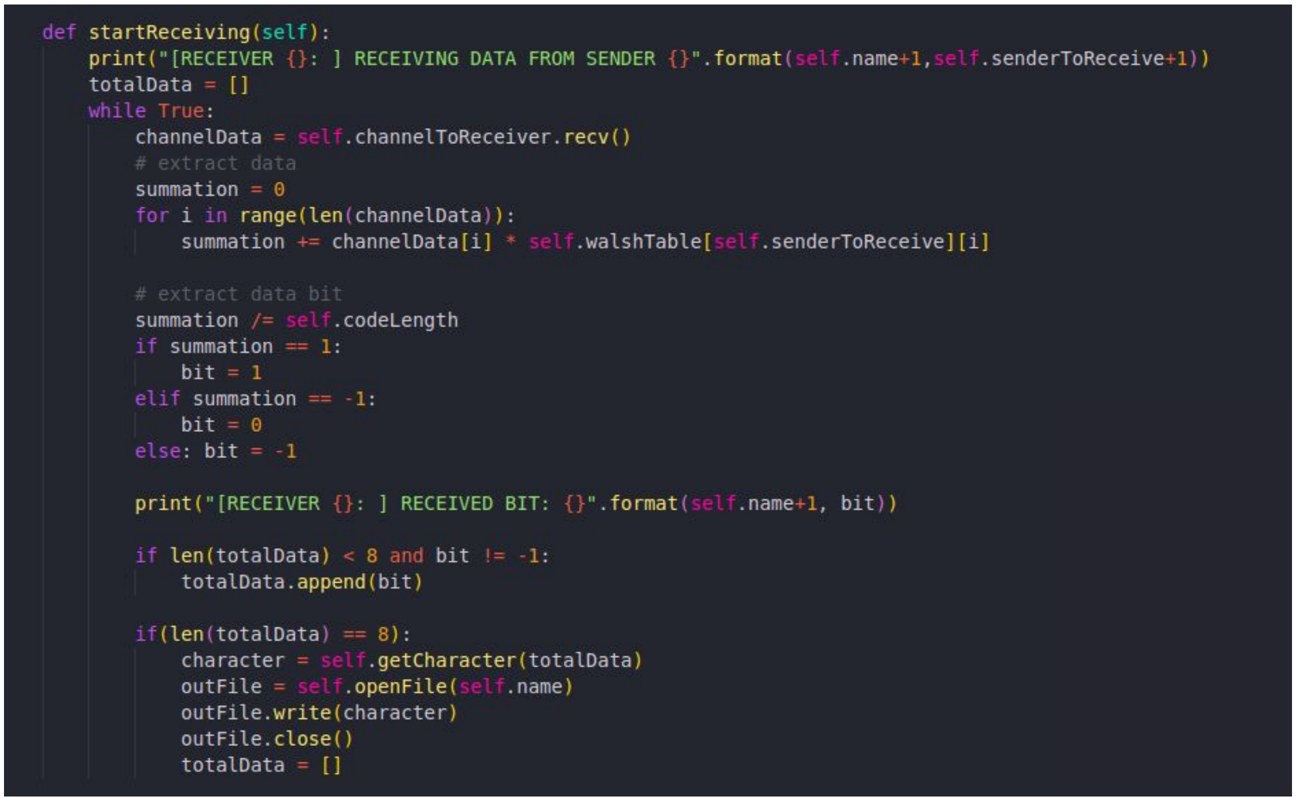
Code Structure

Sender logic:



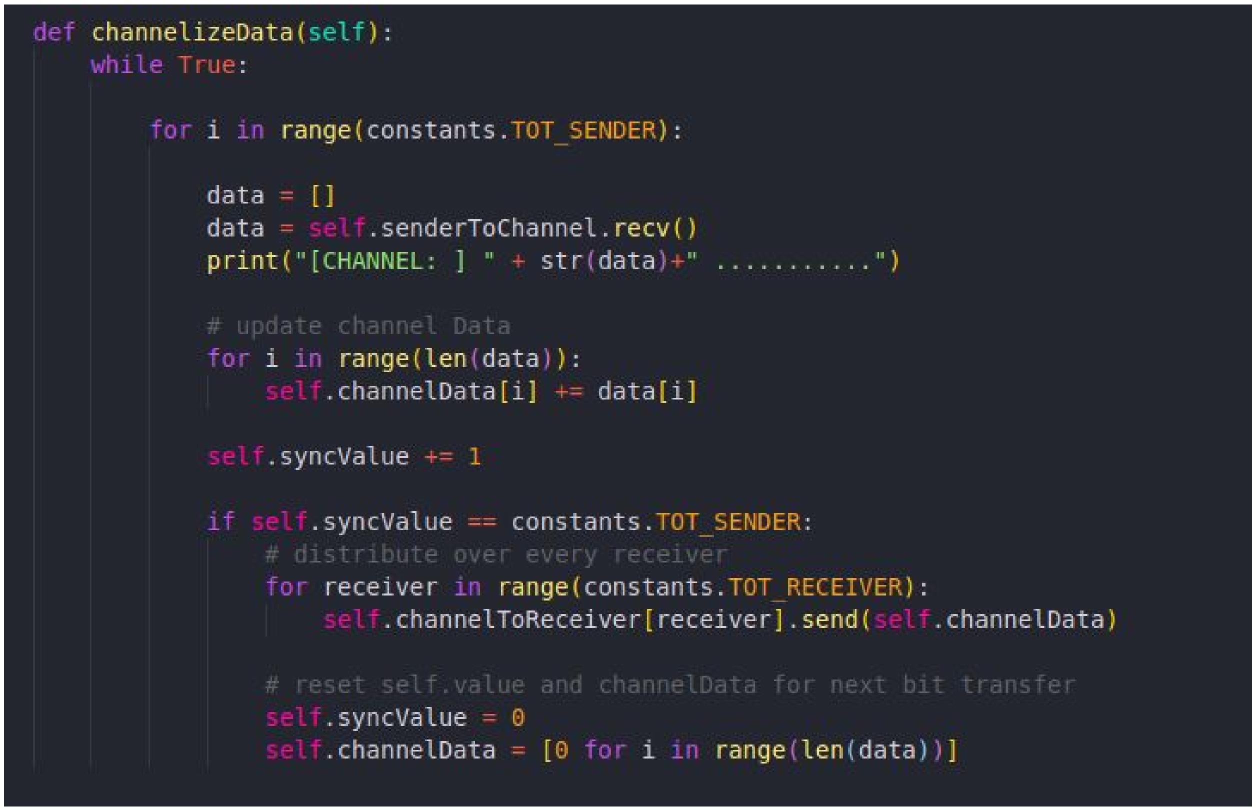
```python
def startSending(self):
    startTime = time.time()
    totBy = self.fileLength(self.name)
    file = self.openFile(self.name)

    totalBitSent = 0
    byte = file.read(constants.DATA_PACK_SIZE)
    while byte:

        # send the data bits of byte
        data = '{0:08b}'.format(ord(byte))
        for i in range(len(data)):
            dataToSend = []
            dataBit = int(data[i])
            if dataBit == 0: dataBit = -1


            for j in self.walshCode:
                dataToSend.append(j * dataBit)
            self.senderToChannel.send(dataToSend)
            print("[SENDER {}: ] DATA BIT SENT: {} ...........".format(self.name+1, dataBit))
            time.sleep(1)

        byte = file.read(constants.DATA_PACK_SIZE)

    print("[SENDER {}: ] THE ENTIRE DATA HAS BEEN SUCCESSFULLY TRANSMITTED......".format(self.name + 1))
    endTime = time.time()
    totTime = endTime - startTime
    print("\n!---------------------------------SHOW STATS------------------------------------------!\n")
    print("\n______________________SENDER {}_________________________\n".format(self.name+1))
    print("\nTOTAL BYTES SENT: {}\nTIME REQUIRED FOR THE ENTIRE TRANSMISSION: {}\n".format(totBy,totTime))
    print("\n!-------------------------------------------------------------------------------------!\n")
```

Receiver logic:



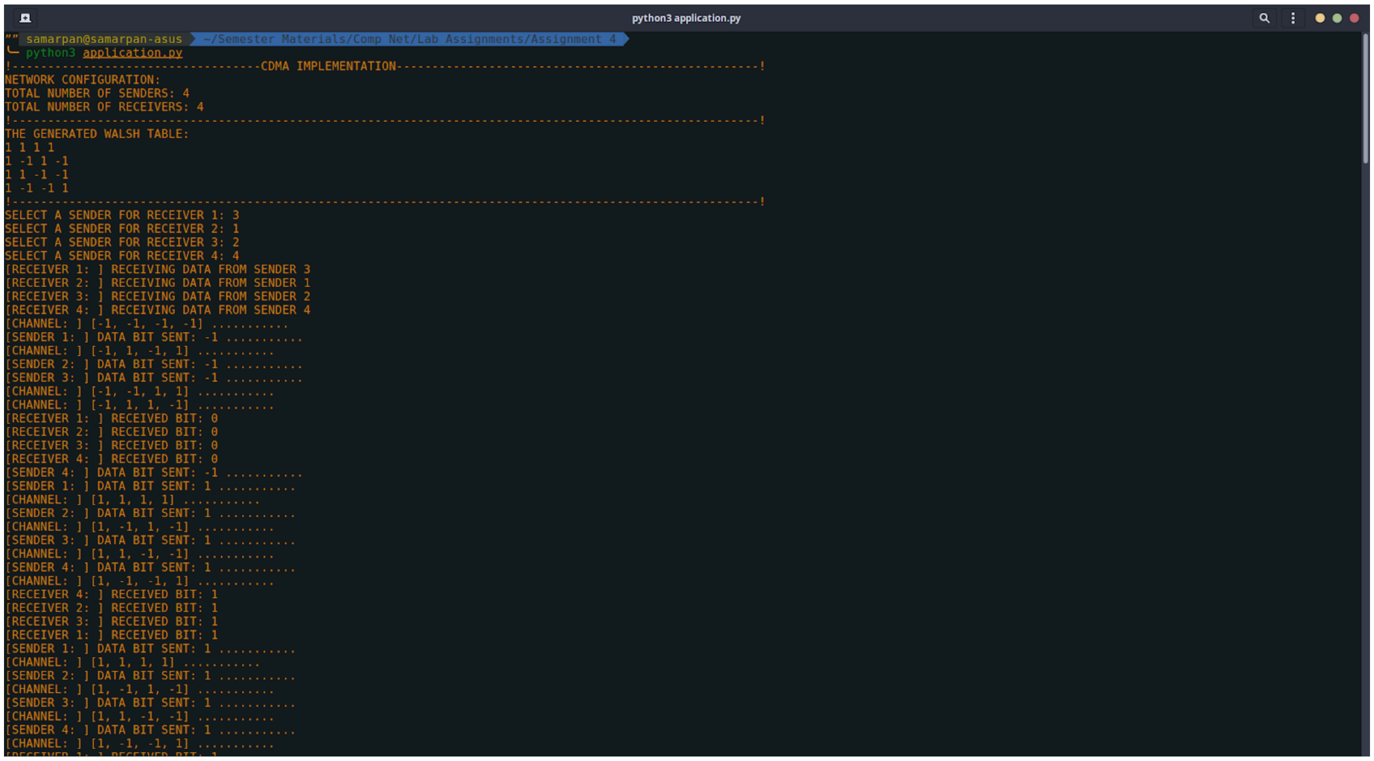
```
def startReceiving(self):
    print("[RECEIVER {}: ] RECEIVING DATA FROM SENDER {}".format(self.name+1,self.senderToReceive+1))
    totalData = []
    while True:
        channelData = self.channelToReceiver.recv()
        # extract data
        summation = 0
        for i in range(len(channelData)):
            summation += channelData[i] * self.walshTable[self.senderToReceive][i]

        # extract data bit
        summation /= self.codeLength
        if summation == 1:
            bit = 1
        elif summation == -1:
            bit = 0
        else: bit = -1

        print("[RECEIVER {}: ] RECEIVED BIT: {}".format(self.name+1, bit))

        if len(totalData) < 8 and bit != -1:
            totalData.append(bit)

        if(len(totalData) == 8):
            character = self.getCharacter(totalData)
            outFile = self.openFile(self.name)
            outFile.write(character)
            outFile.close()
            totalData = []
```

Channel logic:

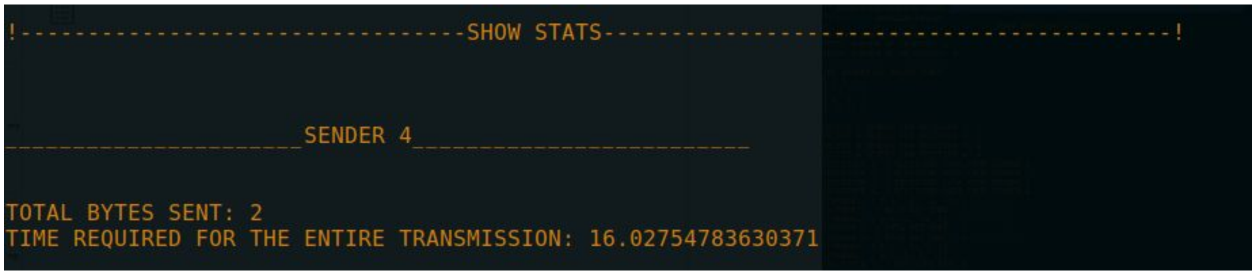


```python
def channelizeData(self):
    while True:

        for i in range(constants.TOT_SENDER):

            data = []
            data = self.senderToChannel.recv()
            print("[CHANNEL: ] " + str(data)+" ...........")

            # update channel Data
            for i in range(len(data)):
                self.channelData[i] += data[i]

            self.syncValue += 1

            if self.syncValue == constants.TOT_SENDER:
                # distribute over every receiver
                for receiver in range(constants.TOT_RECEIVER):
                    self.channelToReceiver[receiver].send(self.channelData)

                # reset self.value and channelData for next bit transfer
                self.syncValue = 0
                self.channelData = [0 for i in range(len(data))]
```

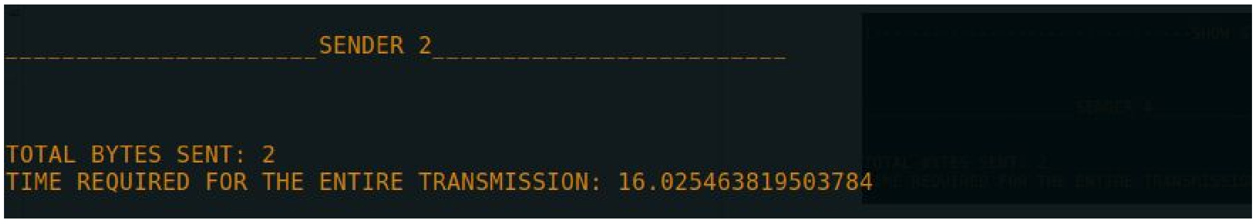
Sample execution



```
python3 application.py
"" samarpan@samarpan-asus  ~/Semester Materials/Comp Net/Lab Assignments/Assignment 4
└─ python3 application.py
!-----------------------------------CDMA IMPLEMENTATION-------------------------------------------------!
NETWORK CONFIGURATION:
TOTAL NUMBER OF SENDERS: 4
TOTAL NUMBER OF RECEIVERS: 4
!------------------------------------------------------------------------------------------------------!
THE GENERATED WALSH TABLE:
1 1 1 1
1 -1 1 -1
1 1 -1 -1
1 -1 -1 1
!------------------------------------------------------------------------------------------------------!
SELECT A SENDER FOR RECEIVER 1: 3
SELECT A SENDER FOR RECEIVER 2: 1
SELECT A SENDER FOR RECEIVER 3: 2
SELECT A SENDER FOR RECEIVER 4: 4
[RECEIVER 1: ] RECEIVING DATA FROM SENDER 3
[RECEIVER 2: ] RECEIVING DATA FROM SENDER 1
[RECEIVER 3: ] RECEIVING DATA FROM SENDER 2
[RECEIVER 4: ] RECEIVING DATA FROM SENDER 4
[CHANNEL: ] [-1, -1, -1, -1] ...........
[SENDER 1: ] DATA BIT SENT: -1 ...........
[CHANNEL: ] [-1, 1, -1, 1] ...........
[SENDER 2: ] DATA BIT SENT: -1 ...........
[SENDER 3: ] DATA BIT SENT: -1 ...........
[CHANNEL: ] [-1, -1, 1, 1] ...........
[CHANNEL: ] [-1, 1, 1, -1] ...........
[RECEIVER 1: ] RECEIVED BIT: 0
[RECEIVER 2: ] RECEIVED BIT: 0
[RECEIVER 3: ] RECEIVED BIT: 0
[RECEIVER 4: ] RECEIVED BIT: 0
[SENDER 4: ] DATA BIT SENT: -1 ...........
[SENDER 1: ] DATA BIT SENT: 1 ...........
[CHANNEL: ] [1, 1, 1, 1] ...........
[SENDER 2: ] DATA BIT SENT: 1 ...........
[CHANNEL: ] [1, -1, 1, -1] ...........
[SENDER 3: ] DATA BIT SENT: 1 ...........
[CHANNEL: ] [1, 1, -1, -1] ...........
[SENDER 4: ] DATA BIT SENT: 1 ...........
[CHANNEL: ] [1, -1, -1, 1] ...........
[RECEIVER 4: ] RECEIVED BIT: 1
[RECEIVER 2: ] RECEIVED BIT: 1
[RECEIVER 3: ] RECEIVED BIT: 1
[RECEIVER 1: ] RECEIVED BIT: 1
[SENDER 1: ] DATA BIT SENT: 1 ...........
[CHANNEL: ] [1, 1, 1, 1] ...........
[SENDER 2: ] DATA BIT SENT: 1 ...........
[CHANNEL: ] [1, -1, 1, -1] ...........
[SENDER 3: ] DATA BIT SENT: 1 ...........
[CHANNEL: ] [1, 1, -1, -1] ...........
[SENDER 4: ] DATA BIT SENT: 1 ...........
[CHANNEL: ] [1, -1, -1, 1] ...........
```



```
!--------------------------------SHOW STATS------------------------------------------!


_____________________SENDER 4________________________


TOTAL BYTES SENT: 2
TIME REQUIRED FOR THE ENTIRE TRANSMISSION: 16.02754783630371
```

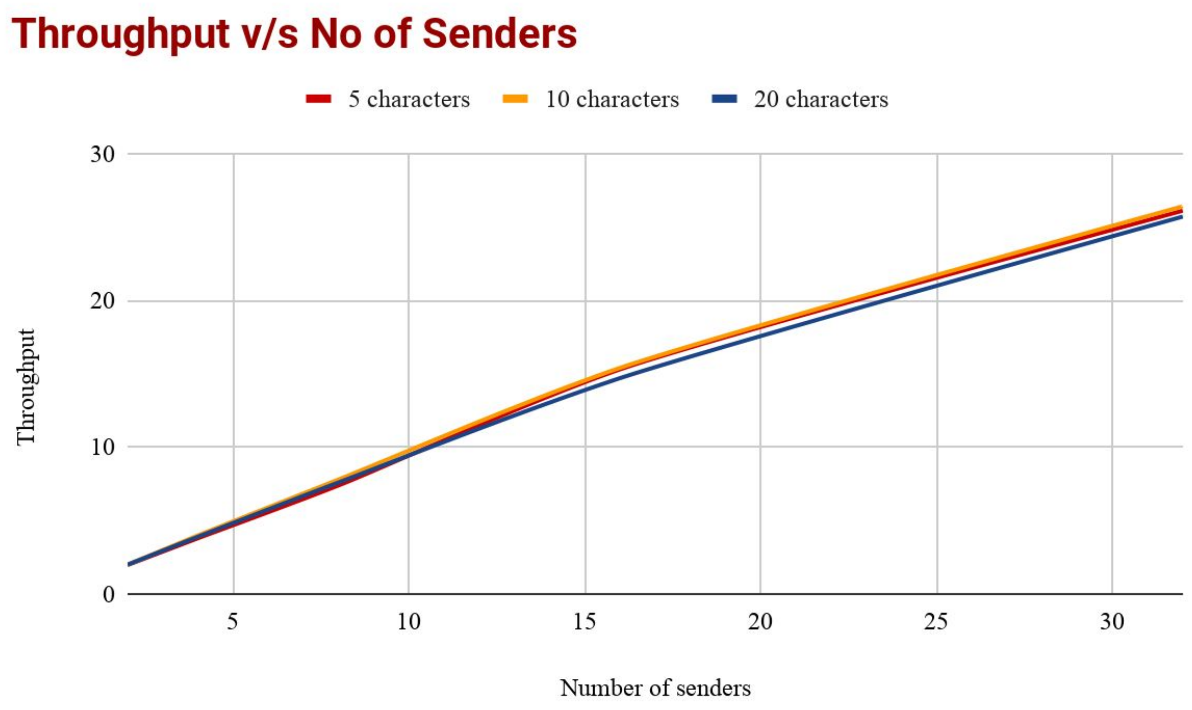


```
______________________SENDER 2________________________


TOTAL BYTES SENT: 2
TIME REQUIRED FOR THE ENTIRE TRANSMISSION: 16.025463819503784
```

Analysis

1.

Number of Senders	Characters sent	Throughput	Avg throughput
2	5	1.98	2
	10	2	
	20	2	
4	5	3.80	3.9
	10	3.99	
	20	3.90	
8	5	7.4	7.4
	10	7.8	
	20	7.6	
16	5	15.3	15.1
	10	15.4	
	20	14.7	
32	5	26.1	26
	10	26.4	
	20	25.7	

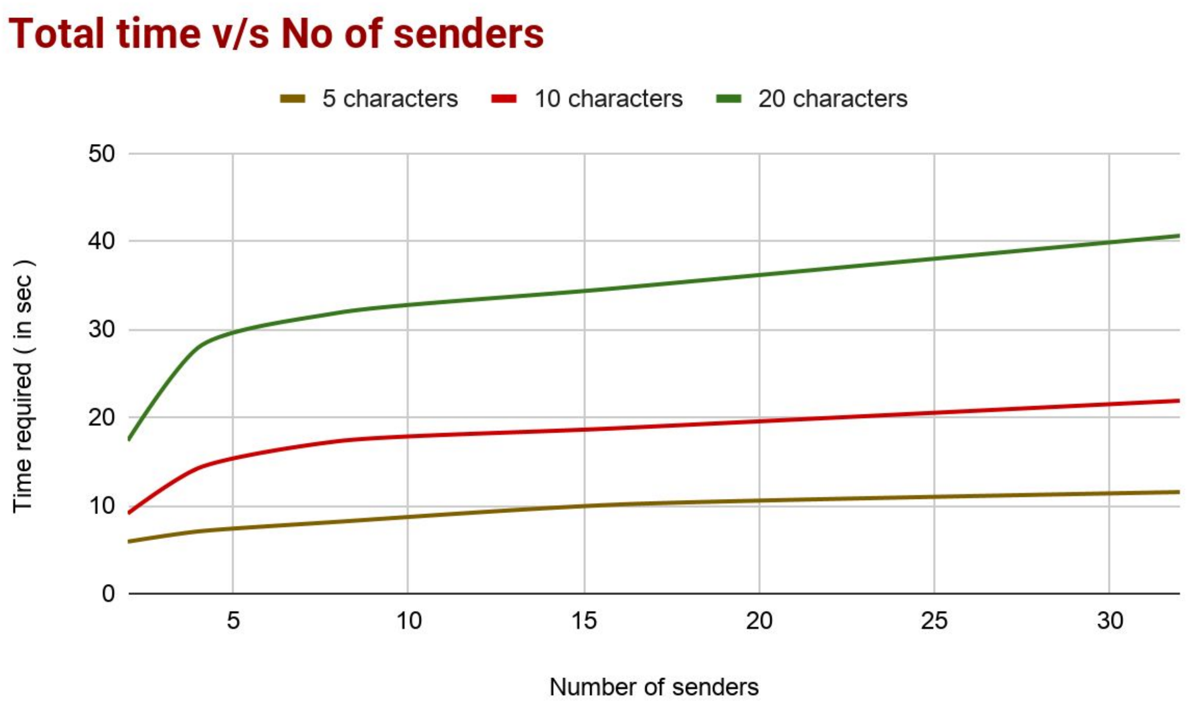


As per the theory, the throughput should always be equal to the number of senders, since I have not introduced any delay or errors during transmission. So, the curve was expected to be a straight line with slope 1. But it is found that there is no linear increase in throughput as we increase the number of senders. The increase in throughput follows a gradual curved path. This can be reasoned with the fact that on increasing the number of senders, the encoding of the data by the channel takes more time and thus, a few time slots are wasted in the calculation.

In addition to it, it is observed that the throughput remains almost the same irrespective of the number of characters sent, dropping by a minute margin for large numbers. Thus we can say that the number of characters to be sent does not affect the throughput so much.

2.

Number of Senders	Characters sent	Time required(s)
2	5	5.9
	10	9.1
	20	17.37
4	5	7.06
	10	14.23
	20	27.91
8	5	8.16
	10	17.31
	20	31.89
16	5	10.12
	10	18.79
	20	34.70
32	5	11.53
	10	21.90
	20	40.63



Total time required for the entire transmission, theoretically, should not depend on the number of senders. But, as evident from the above graph, the time required increases with the increase in the number of senders. This is in accordance with the throughput observation. For a larger number of senders, more time is required to perform the calculations, so total time required for the entire transmission also increases.

As expected, the time required for transmission of 5 characters is the least for all cases, followed by the time required by 10 and 20 characters.

Conclusion

So, as per the analysis conducted, the throughput of CDMA is largely affected by the number of senders, for larger networks. In most practical scenarios, the number of sender nodes is greater than equal to 50, so the throughput is quite low in such cases, compared to the expected value.

Scopes of improvement

1. Error and delay factors are not considered here, which is not what practical networks are like. So, this is more like a theoretical scenario.
2. Due to hardware limitations, a huge number of sender nodes could not be tested.