Computer Networks Assignment 4

Date: 22/12/2020

Soumalya Kundu

Roll: 001810501033

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.

Brief Introduction of CDMA

CDMA employs analog-to-digital conversion (ADC) in combination with spread spectrum technology. Audio input is first digitized into binary elements. The frequency of the transmitted signal is then made to vary according to a defined pattern (code), so it can be intercepted only by a receiver whose frequency response is programmed with the same code, so it follows exactly along with the transmitter frequency.
CDMA has a number of distinguishing features that are key to spread spectrum transmission technologies:

- ***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.
- ***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.
- ***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.
- ***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.

Advantages of CDMA:

CDMA has a soft capacity. The greater the number of codes, the more the number of users. It has the following advantages:-

- 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

- 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.
- 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.
- Transmission Burst – reduces interference.

Disadvantages of CDMA:

The disadvantages of using CDMA are as follows :

- The code length must be carefully selected. A large code length can induce delay or may cause interference.
- Time synchronization is required.
- Gradual transfer increases the use of radio resources and may reduce capacity.
- As the sum of the power received and transmitted from a base station needs constant tight power control. This can result in several handovers.

Simulation and Implementation

I implemented this assignment as a standalone assignment and not like a part or extension of the previous flow control assignment. I haven't accounted for the channel noise and the flow control or error correction techniques. This probably had an effect on the outcomes or the analysis but I decided to focus on the CDMA part and just accounted for the bit transfer strategy.

So as for the implementation part, I went for a dynamic allocation of sender and receiver nodes. The numbers can be changed in the starting of the program. The sender processes use one pipe as their shared memory and send data to the channel with the help of that. Each and every sender is assigned with it's own walsh code. Senders send data through the shared pipe to the unified channel with the help of a clock. Actually

every sender is synchronised over that particular clock and the channel is incharge of that clock. Channel gathers the data from all the senders, sums it up and broadcast that data to the every receiver node. Receiver nodes then receive the data, successfully decode the data and write that data onto a file. As I haven't consider any channel error, so there is no need to check for data or flow control as such in the receiver end. All the communications have been established using python.multiprocessing.pipes and the synchronisation has been done using lock and event object.

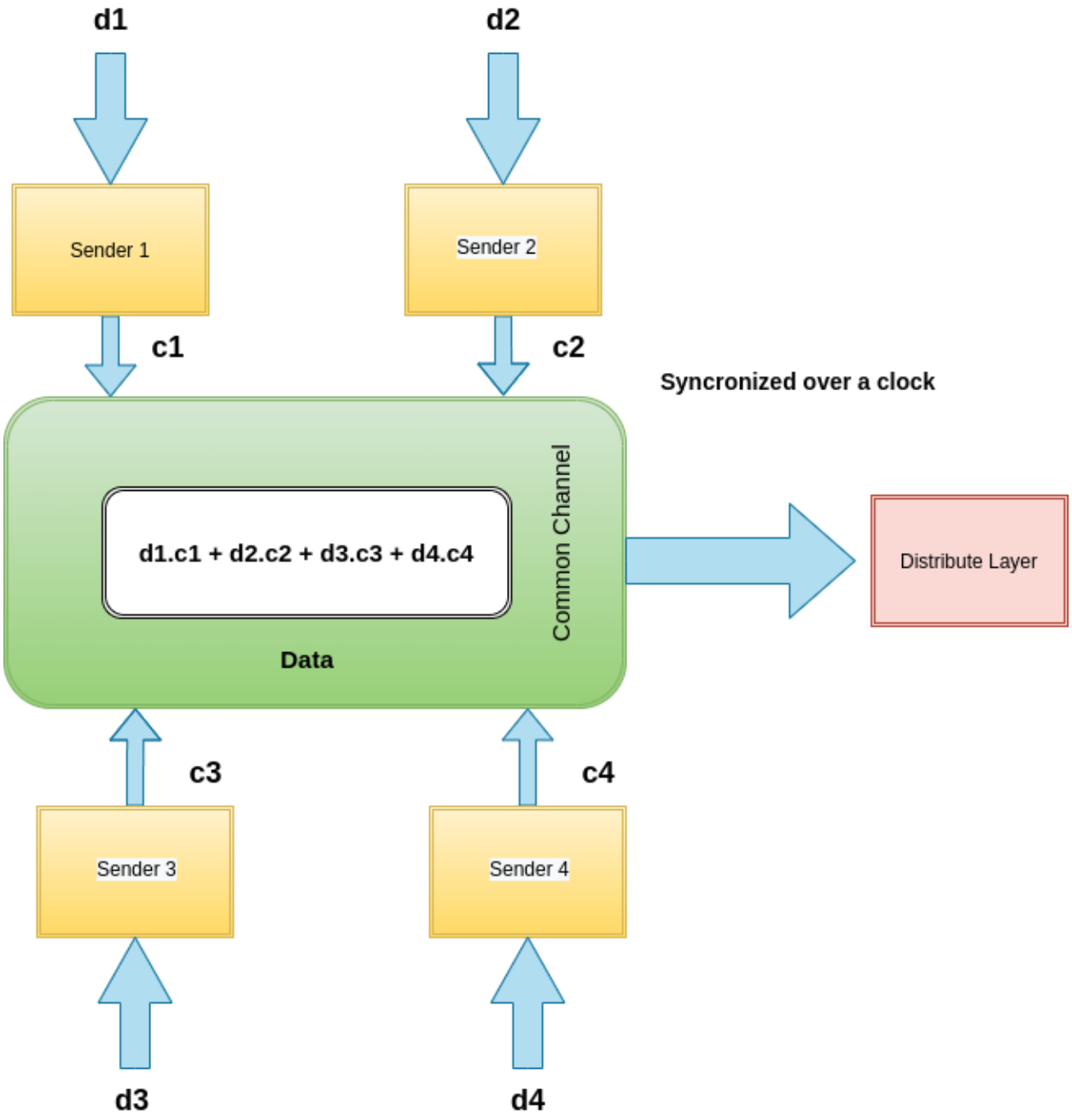


Fig: Implementation of CDMA

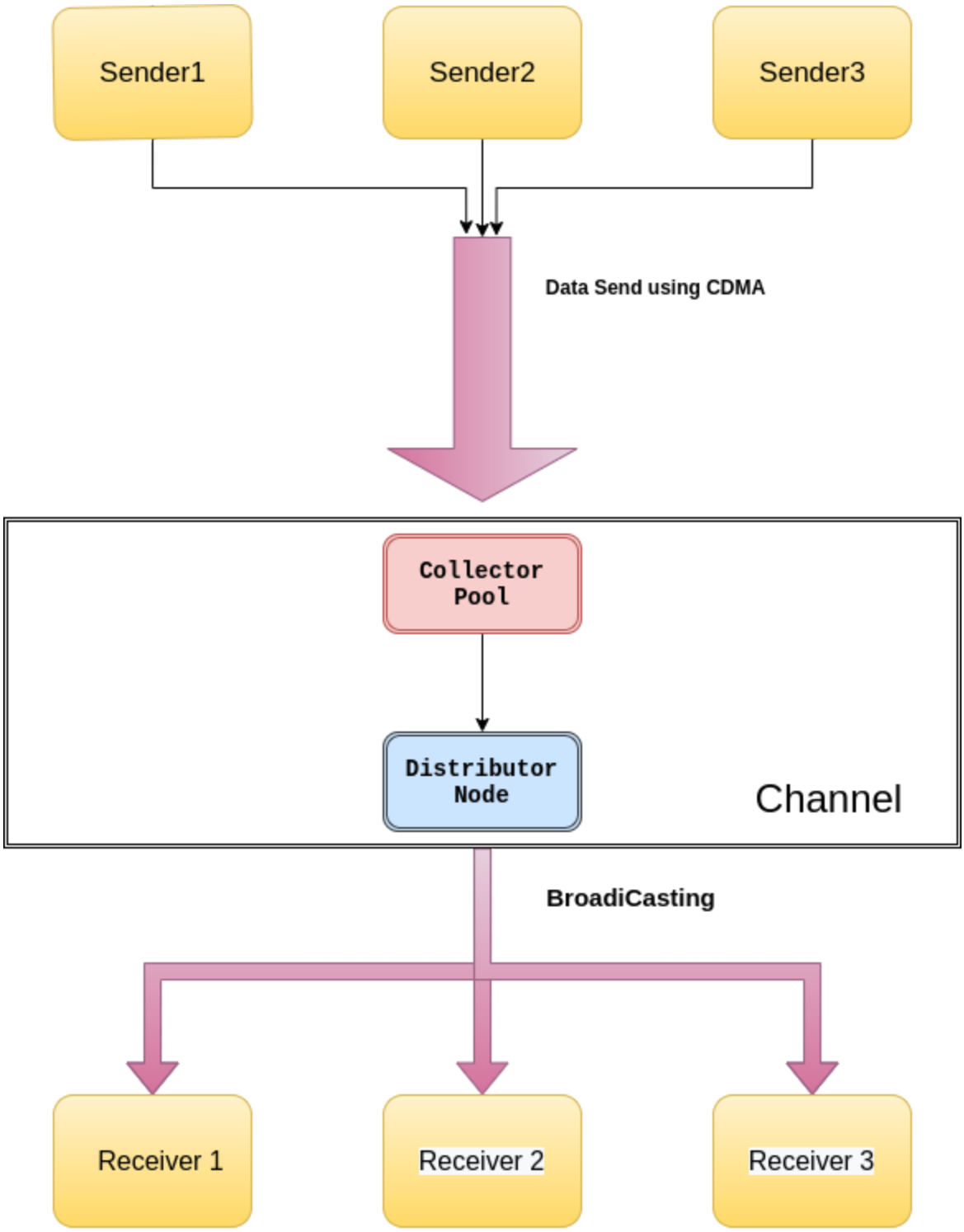


Fig: Program Structure

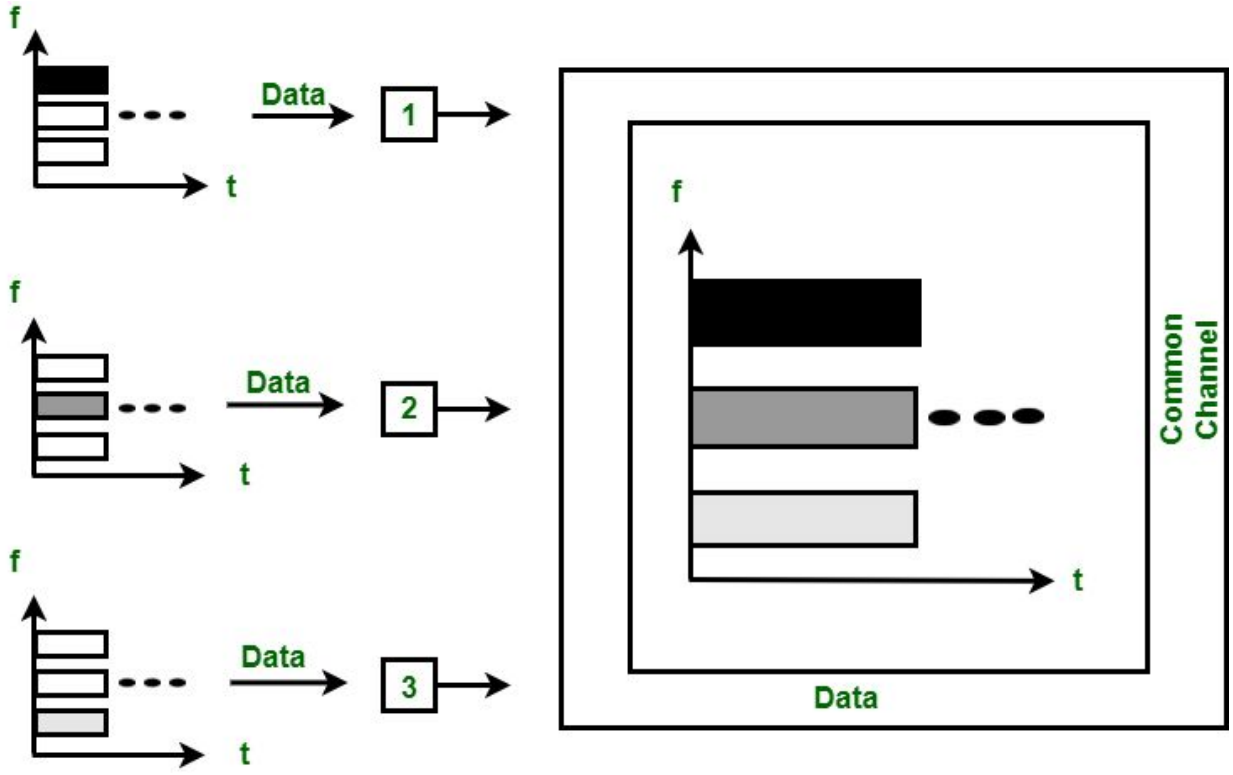


Fig Energy Diagram of Channel

Code Snippets:

Sender Node:

The sender nodes are responsible for taking the data from the input file, break the data down to every bit and club the data with the assigned walsh code. Then send the data to the sender in synchronization with all the other sender and the timeSlot.

```
def sendData(self):
    startTime = time.time()
    file = self.openFile(self.fileName)

    totalBitSent = 0
    byte = file.read(const.defaultDataPacketSize)
    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.lock.acquire()
            self.senderToChannel.send(dataToSend)
            self.lock.release()
            ##############################################
            self.nextTimeSlot.wait()

        byte = file.read(const.defaultDataPacketSize)



    while(True):
        silent = 0
        self.senderToChannel.send(silent)
        self.nextTimeSlot.wait()
```

Channel Node:

The channel process clubs the data received from the senders and Send that data to all the receiver nodes.

```
def takeDataFromSenderAndDistribute(self):
    while True:
        # clear the receiver msg of receiving msg
        self.waitTillReceived.clear()
        self.nextTimeSlot.clear()

        for i in range(const.totalSenderNumber):

            data = []
            data = self.senderToChannel.recv()

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

            self.syncValue += 1

            if self.syncValue == const.totalSenderNumber:
                # distribute over every receiver
                for receiver in range(const.totalReceiverNumber):
                    self.channelToReceiver[receiver].send(self.channelData)

                self.waitTillReceived.wait()

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

            self.nextTimeSlot.set()

        self.nextTimeSlot.set()
        time.sleep(const.sleepTime)
```

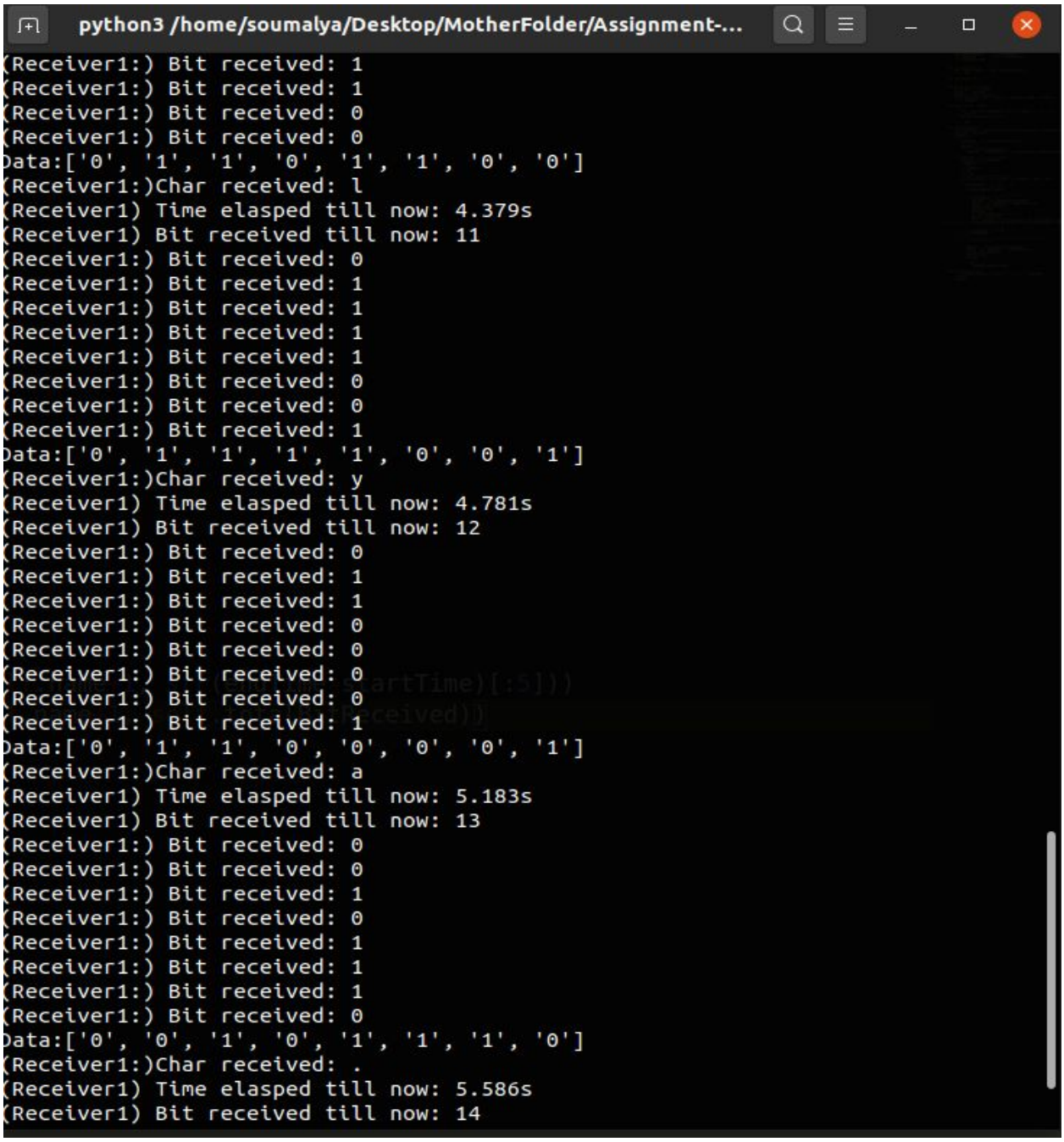
Receiver Node:

Receiver nodes are given the needed walsh set so that they can extract the data from the unified channel data. They extract the data and write them into a file.

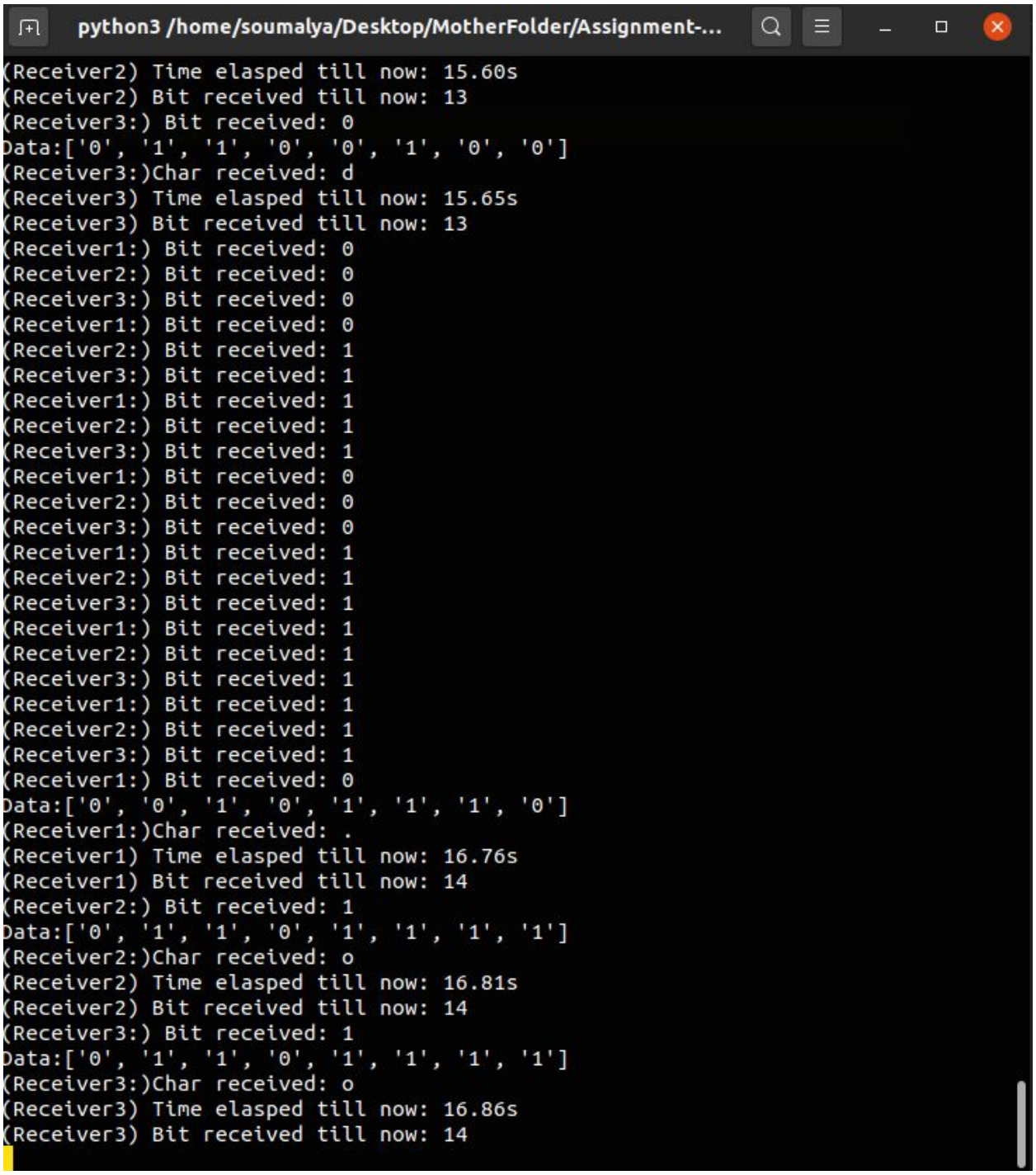
```
def receiveData(self):
    print("(Receiver{}:) Receiver{} receives data from sender{}".format(self.name+1,self.name+1,self.senderToReceive+1))
    startTime = time.time()
    totalData = []
    while True:
        channelData = self.channelToReceiver.recv()
        # extract data
        summation = 0
        for i in range(len(channelData)):
            summation += channelData[i] * self.walshTable[self.senderToReceive][i]

        ######### extract databit######
        summation /= self.codeLength
        if summation == 1:
            bit = 1
        elif summation == -1:
            bit = 0
        else: bit = -1
        ##############################
        print("(Receiver{}:) Bit received: {}".format(self.name+1, bit))

        if len(totalData) < 8 and bit != -1:
            # add the bit to totalData
            totalData.append(str(bit))

            # if totalData is full get the character
            if len(totalData) == 8:
                character = self.getCharacter(totalData)
                outFile = self.openFile(self.senderToReceive)
                outFile.write(character)
                outFile.close()
                totalData = []
                self.totalBitReceived += 1
                endTime = time.time()
                print("(Receiver{}) Time elasped till now: {}s".format(self.name+1, str(endTime-startTime)[:5]))
                print("(Receiver{}) Bit received till now: {}".format(self.name+1, self.totalBitReceived))
```

Output and ScreenShot:

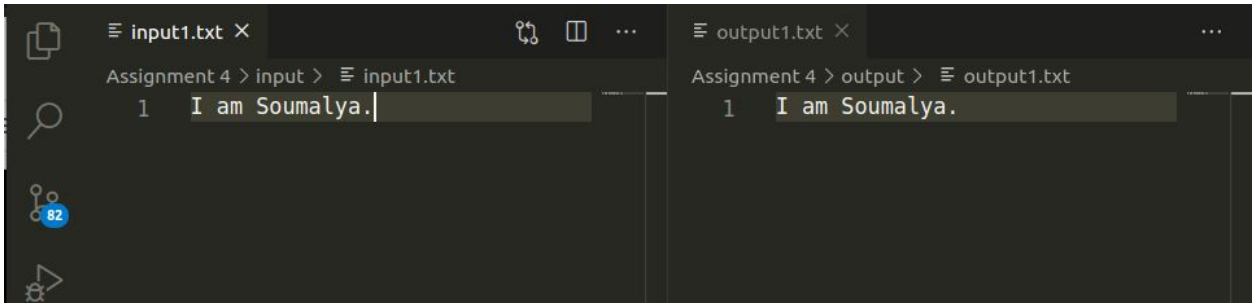
All the time estimations are taken assuming a time slot of 50 mili seconds.



[For 2 senders and 1 receiver nodes]



For 4 senders and 3 receiver nodes



[Respective Input and output file]

Analysis:

To calculate and get an idea about the avg bandwidth, I turn off all the delay elements and noise in the channel and set the Channel to transmit data with a frame size of 64 bytes. It came out to be approximately 500 kbps.

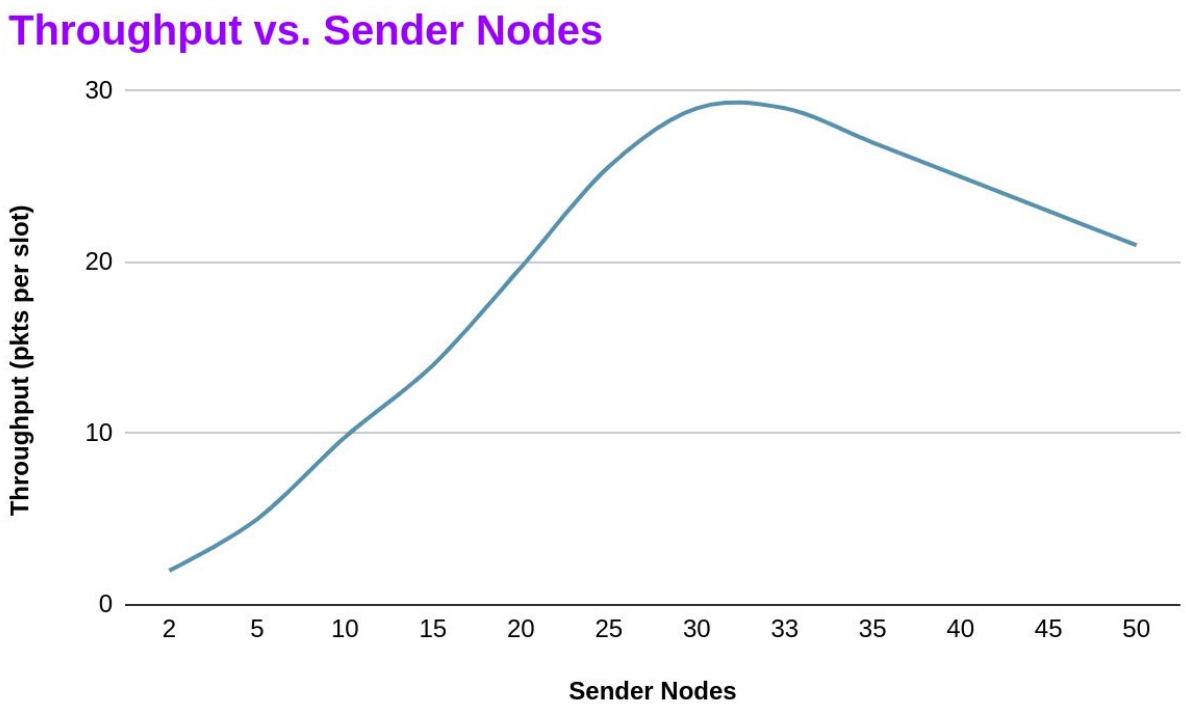
- **BandWIdth Calculated = 500 Kbps**
- **Channel is noiseless**
- **TimeSlot duration is 50 mili seconds**

Data Collected:

Number of Sender Nodes	Characters Send (per sender)	Time (in Seconds)	Throughput (Pkts per timeSlot)	Avg Throughput
2	15	5.59	2	2
	25	9.31	2	
	50	18.72	1.95	
10	15	11.89	10	9.8
	25	20.40	9.7	
	50	39.95	10	

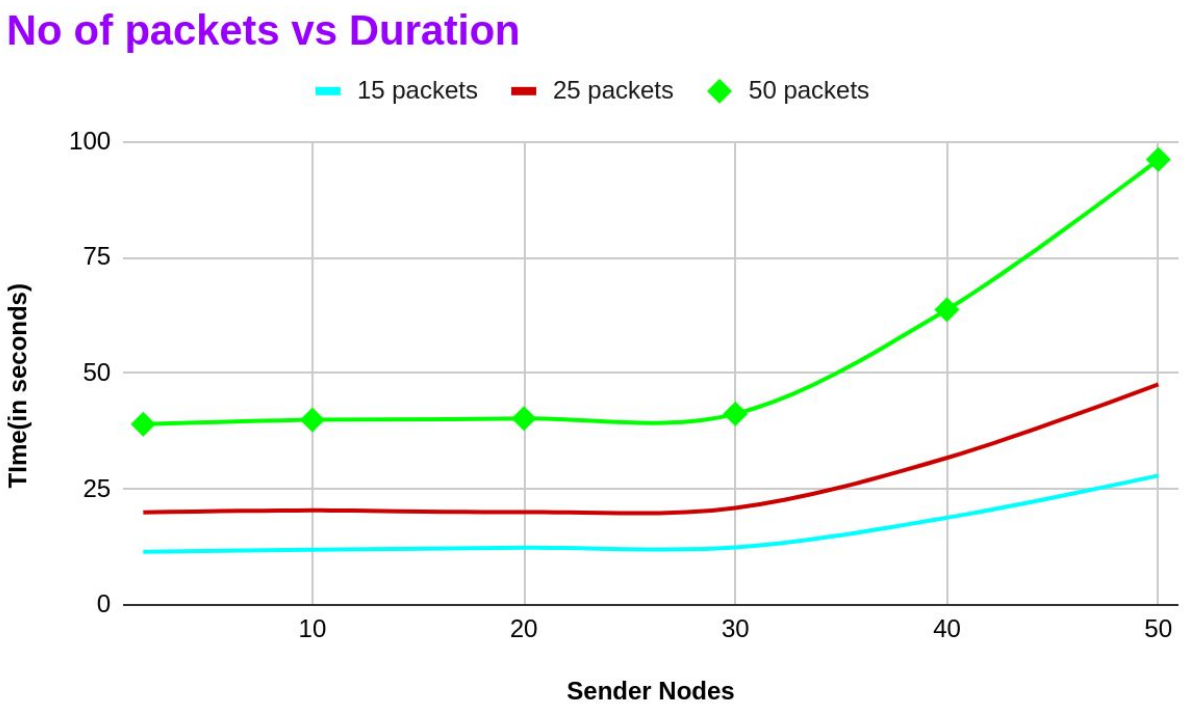
20	15	12.34	19.5	19.7
	25	20.02	20	
	50	40.23	19.9	
30	15	12.41	29	29
	25	20.93	28.7	
	50	41.23	29.1	
40	15	18.82	25.5	25.2
	25	31.74	25.2	
	50	63.74	25.1	
50	15	27.90	21.5	21.1
	25	47.61	21	
	50	96.15	20.8	

Throughput and Sender Nodes:



This is a very interesting result indeed. Theoretically there is no chance of collision and hence no timesolt loss. So the result should be a linear increasing curve with throughput equals to the total number of senders (as every single packet is sent in the same time slot) and actually this is the case for lower number of senders. The curve is linear but interestingly enough the curve shows a dip as the number of senders increase. The only possible explanation is, at a higher number of active senders, the channel takes **more time than a timeslot(i.e 50 ms)** to club the data and send that clubbed data to the receivers. So some of the timeslots get lapsed without sending any data. This thing gets prevalent as the number of senders get increased. The highest value I've achieved is at **around 31 active senders**. The results will be higher with more capable hardware.

Duration vs No of packets:



Theoretically all the lines should be parallel as 50 packets should take more time than 25 packets and 15 packets. But what's interesting is that they are not perfectly straight lines. Theoretically CDMA should achieve a throughput of n(= total active sender nodes), which implies the curves should be a perfect straight line parallel to x axis, and these curves ARE parallel in the lower region but they start to increase as the number of senders get increased. This is due to the timeSlot lapse that occurs due to channel processing time exceeds the time for a time slot.

Conclusion:

As the number of Active sender stations increases, throughput for CDMA gets worse. But at lower numbers CDMA provides a near perfect throughput. **So for a lower number of senders, this seems a good and feasible protocol.**

Scopes of Improvement:

- I haven’t considered any flow control protocol, so I didn’t consider any errors introduced by the channel. This is not a practical scenario at all.
- I didn’t consider any packet structure either.