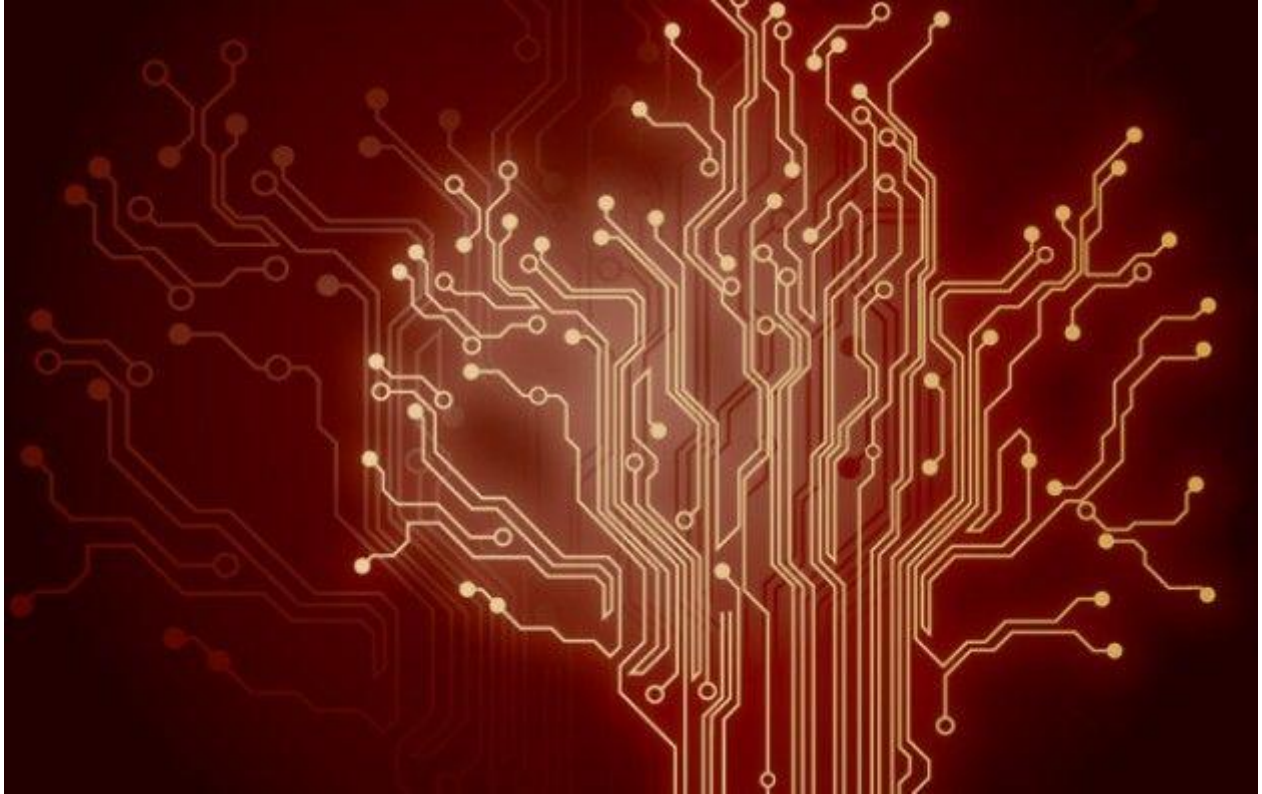


# NETWORK LAB REPORT

*CO4 : Implement CDMA with Walsh code.*



**Atanu Ghosh**

BCSE-III (2019-2023) 5th sem, Section: A-1,

Roll: 001910501005, Date: 27/09/2021

## ASSIGNMENT-4

**Implement CDMA with Walsh code.**

### 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.

.

## 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 the use of a code that 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 allows 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 a near-far effect. In other words, a user near the base station transmitting with the same power will drown the signal later. 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 signals 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.

## DESIGN

I implemented this assignment as a standalone assignment and not as 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 at 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 its own Walsh code. Senders send data through the shared pipe to the unified channel with the help of a clock. Actually, every sender is synchronized over that particular clock and the channel is in charge of that clock. Channel gathers the data from all the senders, sums it up and broadcast that data to every receiver node. Receiver nodes then receive the data, successfully decode the data and write that data onto a file.

**As I haven't considered 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 synchronization has been done using lock and event objects.

## SCHEMATIC DIAGRAM (as a part of DESIGN)

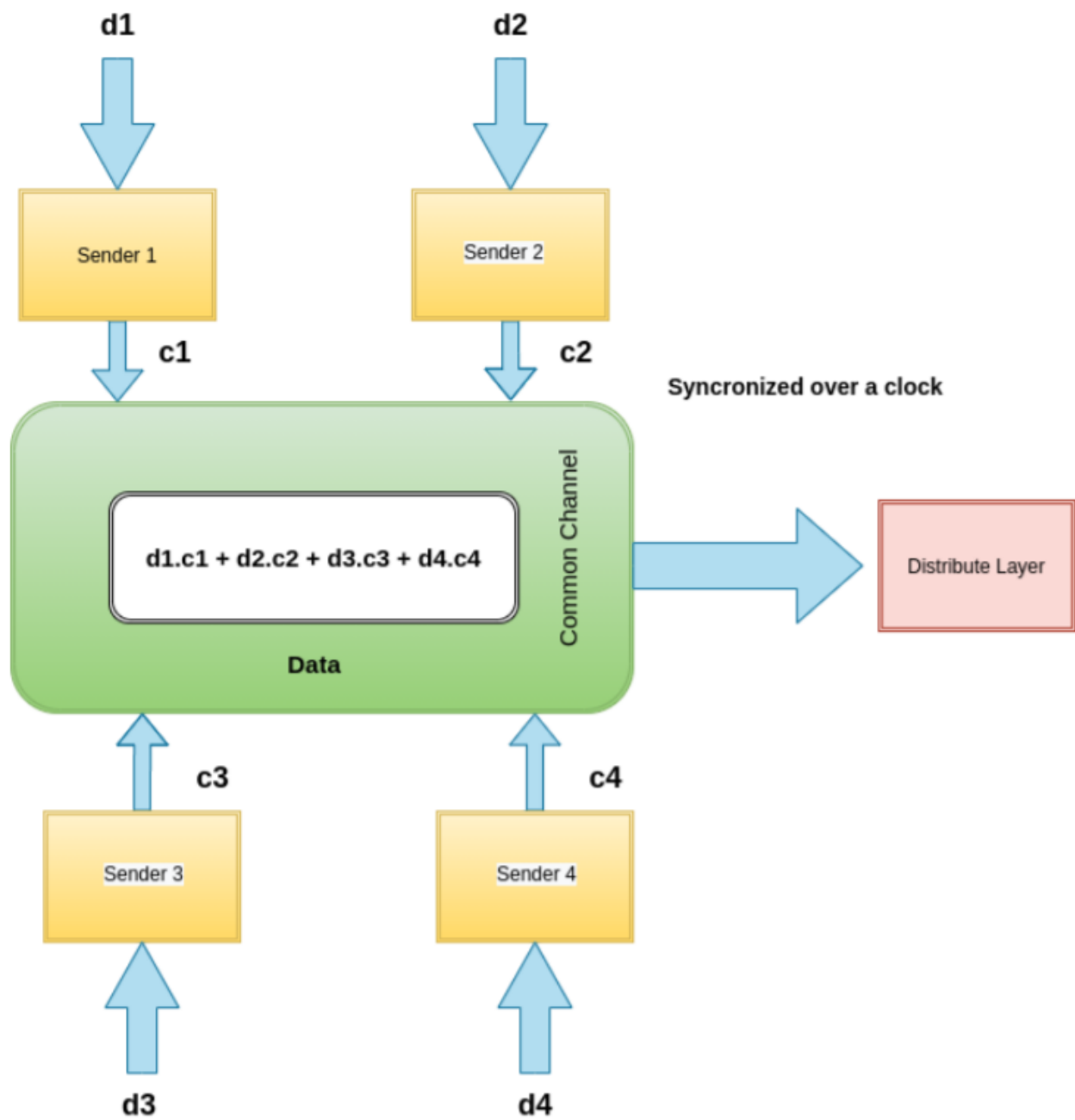


Fig: Implementation of CDMA

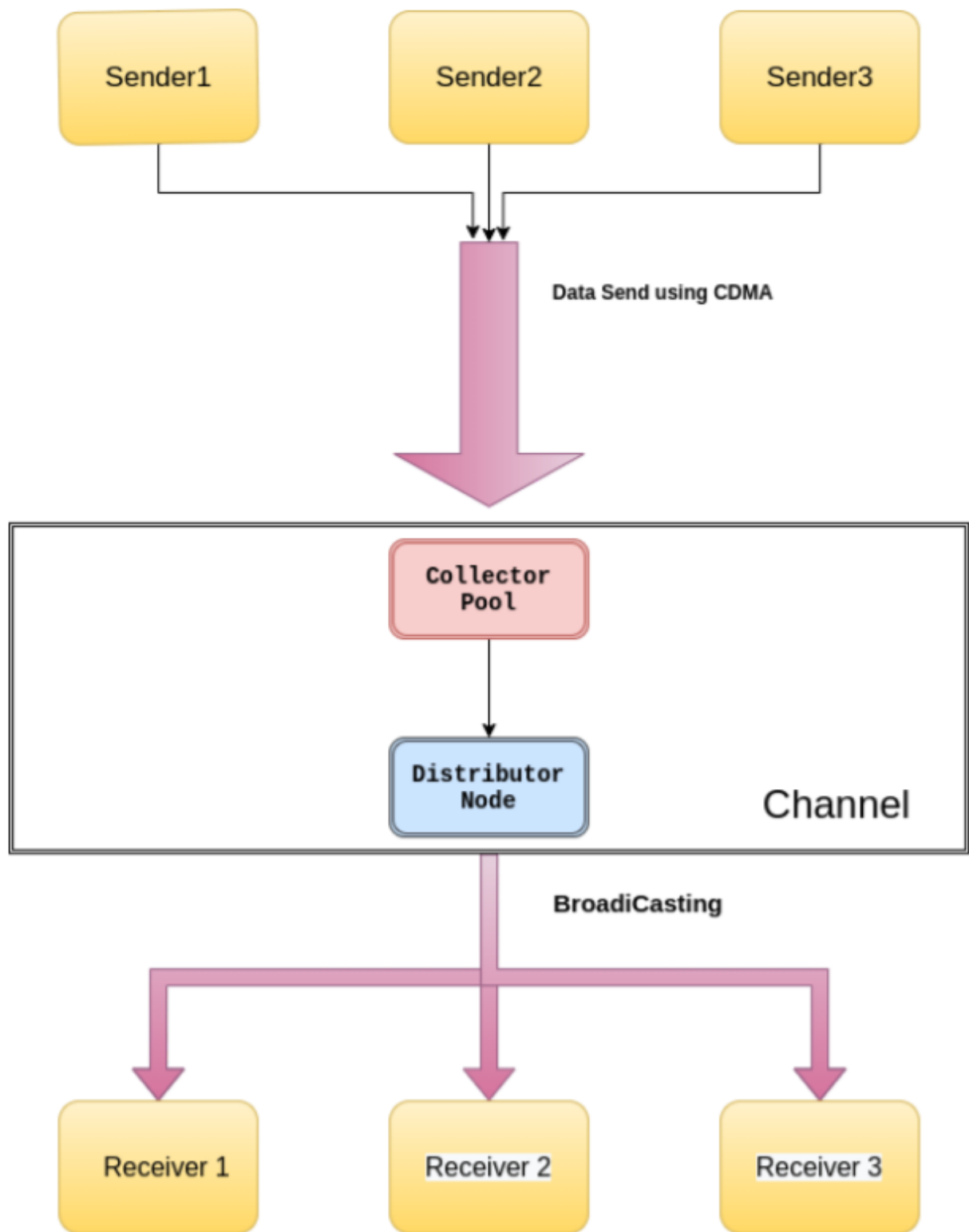


Fig: Program Structure

## IMPLEMENTATION

### A. Algorithm for Sender:

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 senders and the timeSlot

```
def send_data(self):
    '''Sends data continuously'''
    file = self.open_file(self.name)
    byte = file.read(const.default_data_packet_size)
    while byte:
        data = '{0:08b}'.format(ord(byte))      # send the data bits of byte
        for i in range(len(data)):
            data_to_send = []
            data_bit = int(data[i])
            if data_bit == 0: data_bit = -1
            for j in self.walsh_code:
                data_to_send.append(j * data_bit)
            #####
            self.sender_to_channel.send(data_to_send)
            #####
            curr_datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
            print("{} ||| SENDER-{} || DATA BIT SEND {}".format(curr_datetime, self.name+1, \
data_bit))
            with open('textfiles/report.txt', 'a+', encoding='utf-8') as rep_file:
                rep_file.write("\n{} ||| SENDER-{} || DATA BIT SEND {}".format(curr_datetime, \
self.name+1, data_bit))
            #####
            time.sleep(1)
            #####
            byte = file.read(const.default_data_packet_size)

        curr_datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
        print("{} ||| SENDER-{} || DONE SENDING...".format(curr_datetime, self.name+1))
        with open('textfiles/report.txt', 'a+', encoding='utf-8') as rep_file:
            rep_file.write("\n{} ||| SENDER-{} || DONE SENDING...".format(curr_datetime, \
self.name+1))
```

## B. Algorithm for Channel:

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

```
def take_data_from_sender_and_distribute(self):

    while True:

        for _ in range(const.total_sender_number):

            data = []

            data = self.sender_to_channel.recv()

            curr_datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")

            print("{} ||| CHANNEL... || {}".format(curr_datetime, str(data)))

            with open('textfiles/report.txt', 'a+', encoding='utf-8') as rep_file:

                rep_file.write("\n{} ||| CHANNEL... || {}".format(curr_datetime, \
str(data)))

        # update channel data

        for i in range(len(data)): self.channel_data[i] += data[i]

        self.sync_value += 1

    if self.sync_value == const.total_sender_number:

        # distribute over every receiver

        for receiver in range(const.total_receiver_number):

            self.channel_to_receiver[receiver].send(self.channel_data)

        # reset self.value and channelData for next bit transfer

        self.sync_value = 0

        self.channel_data = [0 for _ in range(len(data))]
```



### C. Algorithm for Receiver:

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 receive_data(self):

    curr_datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    print("{} ||| RECEIVER-{} || RECEIVES DATA FROM SENDER-{}".format(curr_datetime, \
self.name+1, self.sender_to_receiver+1))
    with open('textfiles/report.txt', 'a+', encoding='utf-8') as rep_file:
        rep_file.write("\n{} ||| RECEIVER-{} || RECEIVES DATA FROM SENDER-{}".format(curr_datetime, self.name+1, self.sender_to_receiver+1))
    total_data = []
    while True:
        channel_data = self.channel_to_receiver.recv()

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

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

        curr_datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
        print("{} ||| RECEIVER-{} || BIT RECEIVED : {}".format(curr_datetime, self.name+1, bit))
        with open('textfiles/report.txt', 'a+', encoding='utf-8') as rep_file:
            rep_file.write("\n{} ||| RECEIVER-{} || BIT RECEIVED : {}".format(curr_datetime,
self.name+1, bit))

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

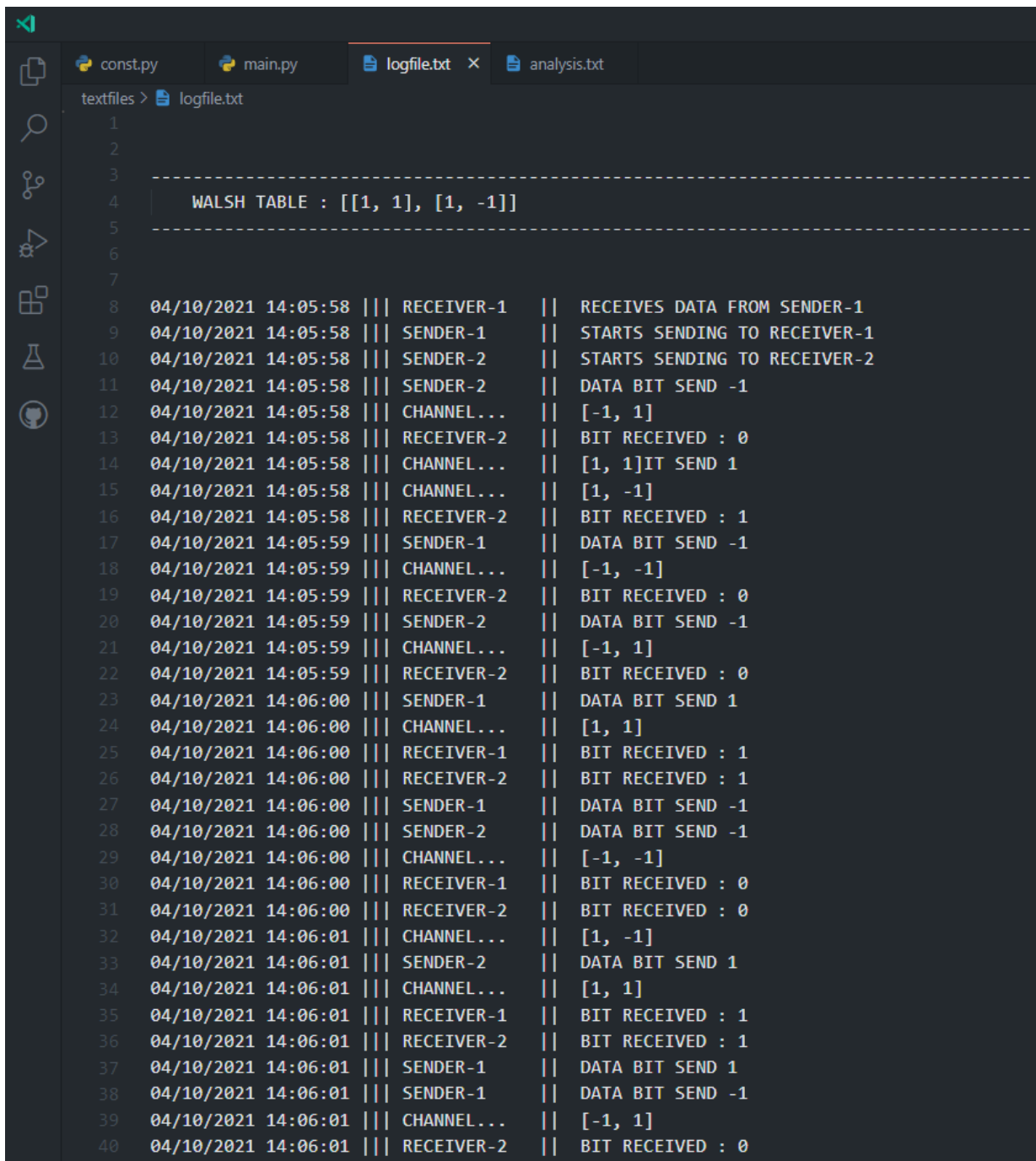
        if len(total_data) == 8:
            character = self.get_char(total_data)
            output_file = self.open_file(self.sender_to_receiver)
            output_file.write(character)
            output_file.close()
            total_data = []
```

## SOURCE CODE STRUCTURE

The Source-Code folder (`src`) contains -

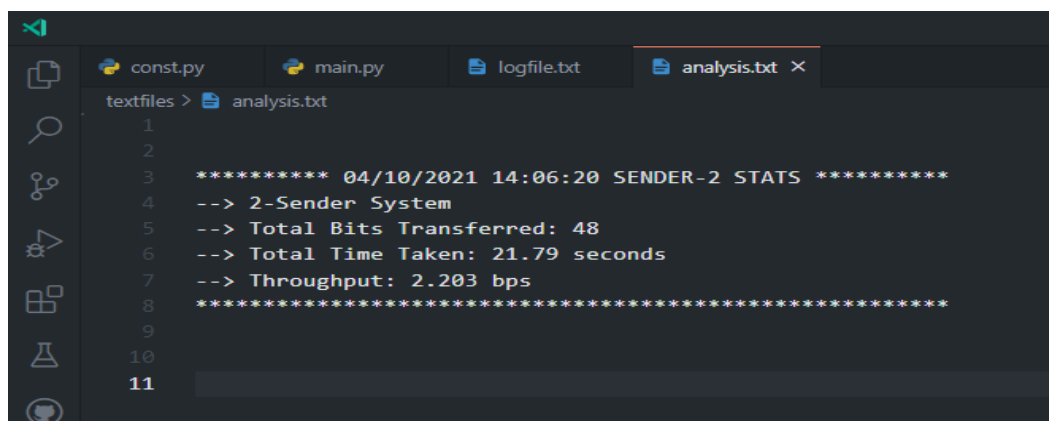
- `const.py`:  
All the constants (number of threads, time-out, input-output path, etc) are defined here.
- `gen_file.py`:  
Helps to generate a given number of input files with input strings written on those by modifying the `PATH` variable.
- `sender.py`:  
Sender Class contains methods for selecting a receiver, reading data from the input file(s), and adding the characters into the stream/channel.
- `channel.py`:  
Channel Class for channelizing packets from sender, channelizing response from the receiver.
- `receiver.py`:  
Receiver Class for extracting the data from the receiver packet, writing the output into the output file, etc.
- `main.py`:  
Contains support for adding the sender and receiver connections to separate lists one by one, adding the sender and receiver threads to separate lists one by one, and start the sending of packets and generating a performance report.
- `./textfiles` :  
Folder containing input and output files, and the log file (written number of characters sent, received and total duration and calculated throughput).

## OUTPUT & SCREENSHOTS



```
1
2
3
4  -----
5  WALSH TABLE : [[1, 1], [1, -1]]
6  -----
7
8  04/10/2021 14:05:58 || RECEIVER-1 || RECEIVES DATA FROM SENDER-1
9  04/10/2021 14:05:58 || SENDER-1 || STARTS SENDING TO RECEIVER-1
10 04/10/2021 14:05:58 || SENDER-2 || STARTS SENDING TO RECEIVER-2
11 04/10/2021 14:05:58 || SENDER-2 || DATA BIT SEND -1
12 04/10/2021 14:05:58 || CHANNEL... || [-1, 1]
13 04/10/2021 14:05:58 || RECEIVER-2 || BIT RECEIVED : 0
14 04/10/2021 14:05:58 || CHANNEL... || [1, 1]IT SEND 1
15 04/10/2021 14:05:58 || CHANNEL... || [1, -1]
16 04/10/2021 14:05:58 || RECEIVER-2 || BIT RECEIVED : 1
17 04/10/2021 14:05:59 || SENDER-1 || DATA BIT SEND -1
18 04/10/2021 14:05:59 || CHANNEL... || [-1, -1]
19 04/10/2021 14:05:59 || RECEIVER-2 || BIT RECEIVED : 0
20 04/10/2021 14:05:59 || SENDER-2 || DATA BIT SEND -1
21 04/10/2021 14:05:59 || CHANNEL... || [-1, 1]
22 04/10/2021 14:05:59 || RECEIVER-2 || BIT RECEIVED : 0
23 04/10/2021 14:06:00 || SENDER-1 || DATA BIT SEND 1
24 04/10/2021 14:06:00 || CHANNEL... || [1, 1]
25 04/10/2021 14:06:00 || RECEIVER-1 || BIT RECEIVED : 1
26 04/10/2021 14:06:00 || RECEIVER-2 || BIT RECEIVED : 1
27 04/10/2021 14:06:00 || SENDER-1 || DATA BIT SEND -1
28 04/10/2021 14:06:00 || SENDER-2 || DATA BIT SEND -1
29 04/10/2021 14:06:00 || CHANNEL... || [-1, -1]
30 04/10/2021 14:06:00 || RECEIVER-1 || BIT RECEIVED : 0
31 04/10/2021 14:06:00 || RECEIVER-2 || BIT RECEIVED : 0
32 04/10/2021 14:06:01 || CHANNEL... || [1, -1]
33 04/10/2021 14:06:01 || SENDER-2 || DATA BIT SEND 1
34 04/10/2021 14:06:01 || CHANNEL... || [1, 1]
35 04/10/2021 14:06:01 || RECEIVER-1 || BIT RECEIVED : 1
36 04/10/2021 14:06:01 || RECEIVER-2 || BIT RECEIVED : 1
37 04/10/2021 14:06:01 || SENDER-1 || DATA BIT SEND 1
38 04/10/2021 14:06:01 || SENDER-1 || DATA BIT SEND -1
39 04/10/2021 14:06:01 || CHANNEL... || [-1, 1]
40 04/10/2021 14:06:01 || RECEIVER-2 || BIT RECEIVED : 0
```

**For 2 senders and 2 receiver nodes**



```
1
2
3 ***** 04/10/2021 14:06:20 SENDER-2 STATS *****
4 --> 2-Sender System
5 --> Total Bits Transferred: 48
6 --> Total Time Taken: 21.79 seconds
7 --> Throughput: 2.203 bps
8 *****
9
10
11
```

```

const.py  main.py  logfile.txt  analysis.txt
textfiles > logfile.txt
1
2
3 -----
4 WALSH TABLE : [[1, 1, 1, 1], [1, -1, 1, -1], [1, 1, -1, -1], [1, -1, -1, 1]]
5 -----
6
7
8 04/10/2021 14:09:29 ||| RECEIVER-1 || RECEIVES DATA FROM SENDER-1
9 04/10/2021 14:09:29 ||| RECEIVER-2 || RECEIVES DATA FROM SENDER-2
10 04/10/2021 14:09:29 ||| RECEIVER-3 || RECEIVES DATA FROM SENDER-3
11 04/10/2021 14:09:29 ||| RECEIVER-4 || RECEIVES DATA FROM SENDER-4
12 04/10/2021 14:09:29 ||| SENDER-1 || STARTS SENDING TO RECEIVER-1
13 04/10/2021 14:09:29 ||| SENDER-2 || STARTS SENDING TO RECEIVER-2
14 04/10/2021 14:09:29 ||| SENDER-4 || STARTS SENDING TO RECEIVER-4
15 04/10/2021 14:09:29 ||| CHANNEL... || [-1, -1, -1, -1]
16 04/10/2021 14:09:29 ||| SENDER-1 || DATA BIT SEND -1
17 04/10/2021 14:09:29 ||| CHANNEL... || [-1, -1, 1, 1]-1
18 04/10/2021 14:09:29 ||| SENDER-2 || DATA BIT SEND -1
19 04/10/2021 14:09:29 ||| SENDER-4 || DATA BIT SEND -1
20 04/10/2021 14:09:29 ||| CHANNEL... || [-1, 1, -1, 1]
21 04/10/2021 14:09:29 ||| CHANNEL... || [-1, 1, 1, -1]
22 04/10/2021 14:09:29 ||| RECEIVER-1 || BIT RECEIVED : 0
23 04/10/2021 14:09:29 ||| RECEIVER-4 || BIT RECEIVED : 0
24 04/10/2021 14:09:29 ||| RECEIVER-3 || BIT RECEIVED : 0
25 04/10/2021 14:09:30 ||| SENDER-3 || DATA BIT SEND 1
26 04/10/2021 14:09:30 ||| SENDER-1 || DATA BIT SEND 1
27 04/10/2021 14:09:30 ||| CHANNEL... || [1, -1, -1, 1]
28 04/10/2021 14:09:30 ||| CHANNEL... || [1, 1, -1, -1]
29 04/10/2021 14:09:30 ||| CHANNEL... || [1, 1, 1, 1]
30 04/10/2021 14:09:30 ||| RECEIVER-3 || BIT RECEIVED : 1
31 04/10/2021 14:09:30 ||| RECEIVER-4 || BIT RECEIVED : 1
32 04/10/2021 14:09:30 ||| SENDER-4 || DATA BIT SEND -1
33 04/10/2021 14:09:30 ||| SENDER-3 || DATA BIT SEND -1
34 04/10/2021 14:09:30 ||| CHANNEL... || [-1, -1, 1, 1]
35 04/10/2021 14:09:30 ||| CHANNEL... || [-1, 1, 1, -1]
36 04/10/2021 14:09:30 ||| CHANNEL... || [-1, 1, -1, 1]
37 04/10/2021 14:09:30 ||| RECEIVER-1 || BIT RECEIVED : 0
38 04/10/2021 14:09:30 ||| RECEIVER-3 || BIT RECEIVED : 0
39 04/10/2021 14:09:30 ||| RECEIVER-4 || BIT RECEIVED : 0

```

**For 4 senders and 4 receiver nodes**

```
1 |
2
3 ***** 04/10/2021 14:10:01 SENDER-1 STATS *****
4 --> 4-Sender System
5 --> Total Bits Transferred: 48
6 --> Total Time Taken: 32.22 seconds
7 --> Throughput: 1.49 bps
8 *****
9
10
11
12
13 ***** 04/10/2021 14:10:01 SENDER-4 STATS *****
14 --> 4-Sender System
15 --> Total Bits Transferred: 48
16 --> Total Time Taken: 32.24 seconds
17 --> Throughput: 1.489 bps
18 *****
19
20
21
22
23 ***** 04/10/2021 14:10:01 SENDER-2 STATS *****
24 --> 4-Sender System
25 --> Total Bits Transferred: 48
26 --> Total Time Taken: 32.25 seconds
27 --> Throughput: 1.488 bps
28 *****
29
30
31
32
33 ***** 04/10/2021 14:10:01 SENDER-3 STATS *****
34 --> 4-Sender System
35 --> Total Bits Transferred: 48
36 --> Total Time Taken: 32.25 seconds
37 --> Throughput: 1.488 bps
38 *****
39
40
```

```
input1.txt
1 JUBCSE

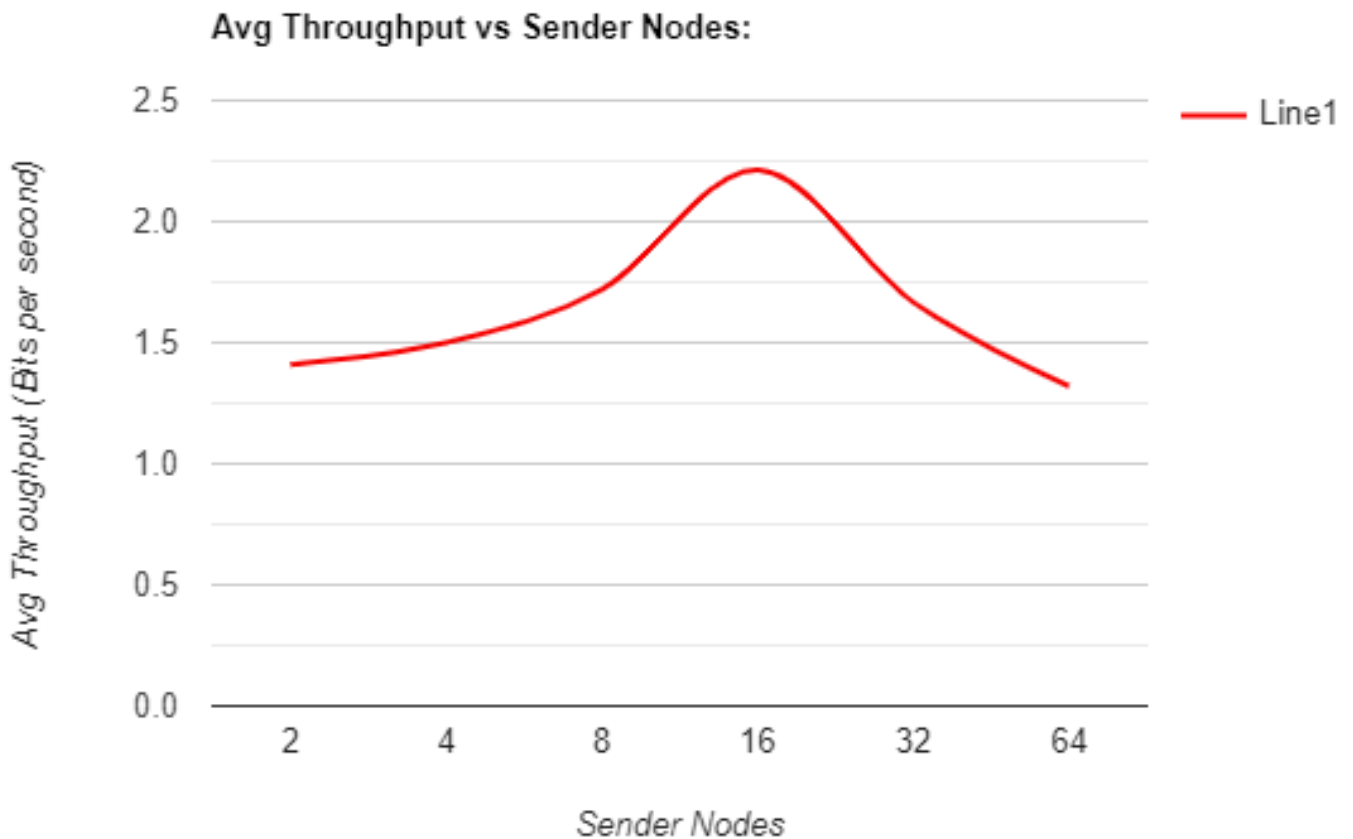
output1.txt
1 JUBCSE
```

## INPUT and OUTPUT files

## RESULTS and ANALYSIS

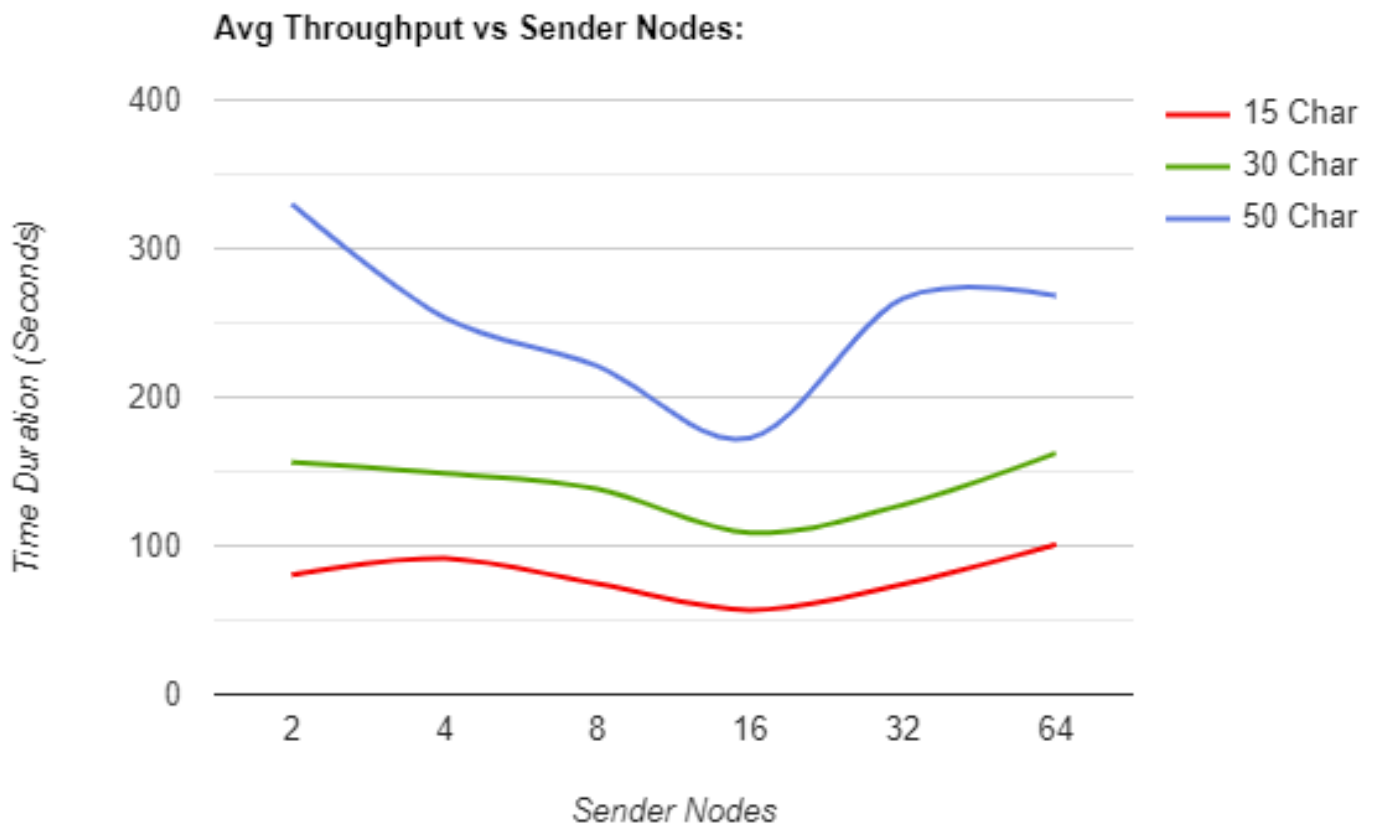
Number of Sender Nodes	Characters Sent (per sender)	Time Duration (in Seconds)	Throughput (Bits per second)	Avg Throughput (Bps)
2	15	80.48	1.491	1.410
	30	156.12	1.537	
	50	330.02	1.212	
4	15	91.49	1.312	1.501
	30	148.79	1.613	
	50	253.64	1.578	
8	15	74.48	1.611	1.719
	30	138.25	1.736	
	50	220.99	1.810	
16	15	56.81	2.112	2.213
	30	108.65	2.209	
	50	172.48	2.319	
32	15	74.15	1.618	1.667
	30	127.59	1.881	
	50	266.39	1.502	
64	15	100.76	1.191	1.320
	30	162.27	1.379	
	50	268.18	1.391	

## Avg Throughput vs Sender Nodes:



This is a very interesting result indeed. Theoretically, there is no chance of collision and hence no timeslot 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 a 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 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 gets increased. The highest value I've achieved is at around 31 active senders. The results will be higher with more capable hardware.

## Time Duration vs Characters Sent:



Theoretically, all the lines should be parallel (which they almost are) as 50 packets should take more time than 25 packets and 15 packets. But what's interesting is that they are not straight lines. Theoretically, CDMA should achieve a throughput of  $n$  (= total active sender nodes), which implies the curves should be a perfectly straight line parallel to the x-axis, and **these curves ARE almost parallel in the lower to middle 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

1. 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.
2. I didn't consider any packet structure either.

## COMMENTS

This assignment has helped me in understanding the CDMA technique of message transfer in Computer Networks by researching and implementing them. It has also helped in understanding the demerits of this protocol, and how such demerits are overcome.