Name – Jeetesh Abrol Assignment no. 4
Roll – 002210501021 BCSE III
Sub - Computer Networks Group –A1

Assignment: 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.

DESIGN:

For the simulation of the CDMA on n-stations the following design has been followed:

i) **walshCode**: The walshcode module generates the walsh table for n stations network.
ii) **Sender**: The sender module makes n threads for n stations and sends data to the channel via write file descriptor pipes.
iii) **Channel**: The channel takes all the data from pipes encodes with respective walsh code and forms the channel data with c1.d1+c2.d2+c3.d3+...+cndn, where di = ith station data, ci = ith station walsh code. Next it passes this formed data to all receiver threads.
iv) **Receiver**: The receiver on receiving the encoded data, first it decodes it with respective codes and then stores the output in respective output files.

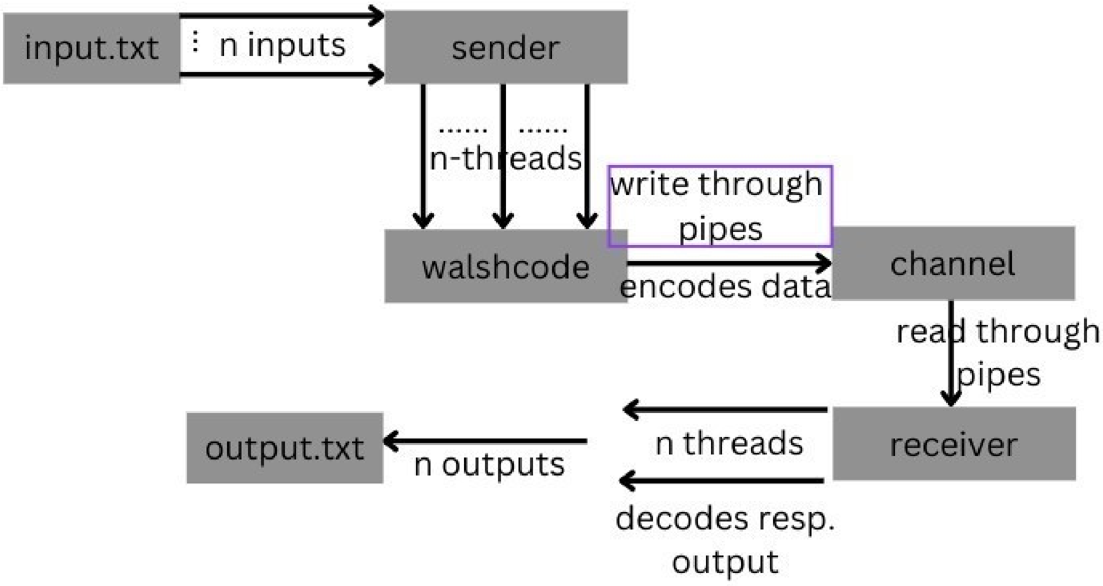


Fig: Design pattern

The outputs, data transfer at a particular time are all logged in analysis.txt and log.txt and not printed in stdout.

IMPLEMENTATION:

i) walsh table generating function:

```
def generateWalshTable(wtable, length, x1, x2, y1, y2,
  compFlag: bool): if length == 2:
    if not compFlag:
      wtable[x1][y1] = 1
      wtable[x1][y2] = 1
      wtable[x2][y1] = 1
      wtable[x2][y2] = -1
    else:
      wtable[x1][y1] = -1
      wtable[x1][y2] = -1
      wtable[x2][y1] = -1
      wtable[x2][y2] = 1
    return
  midx =
  (x1+x2)//2
  midy =
  (y1+y2)//2
  generateWalshTable(wtable, length/2, x1, midx, y1, midy, compFlag)
  generateWalshTable(wtable, length/2, x1, midx, midy+1, y2, compFlag)
  generateWalshTable(wtable, length/2, midx+1, x2, y1, midy, compFlag)
  generateWalshTable(wtable, length/2, midx+1, x2, midy+1, y2, not
  compFlag)
```

Here the length argument must be a power of 2.

ii) sender thread implementation:

```
def send_data(self):
    curr_datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    with open('logs/log.txt', 'a+', encoding='utf-8') as f:
        f.write(f"\n[{curr_datetime}] SENDER-{self.name+1} Started Sending To RECEIVER-
{self.name+1}")
    self.start = time.time()
    file = self.read_file(self.name)
    dataByte = file.read(1)
    while dataByte:
      dataBits = '{0:08b}'.format(ord(dataByte))
      for i in range(len(dataBits)):
        dataToSend = []
        bit = int(dataBits[i])
        if bit == 0:
          bit = -1
        for j in self.walshCode:
          dataToSend.append(j*bit)
        self.senderToChannel.send(dataToSend)
        self.bitcount += 1
        curr_datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
        with open('logs/log.txt', 'a+', encoding='utf-8') as f:
          f.write(f"\n[{curr_datetime}] SENDER-{self.name+1} Data Bit Sent {bit}")
        time.sleep(0.3)
      dataByte = file.read(1)
      self.delay = round((time.time()-self.start), 2)
    with open('logs/analysis.txt', 'a+', encoding='utf-8') as f:
      f.write()
    curr_datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    with open('logs/log.txt', 'a+', encoding='utf-8') as f:
      f.write(f"\n[{curr_datetime}] SENDER-{self.name+1} Ended Sending Data")
```

iii)**channel thread implementation**:

```
def relayThread(self):
    while True:
      num = walshCode.nextPowerOf2(self.senderCount)
      for _ in range(self.senderCount):
        data = self.senderToChannel.recv()
        curr_datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
        with open('logs/log.txt', 'a+', encoding='utf-8') as f:
          f.write(f"\n[{curr_datetime}] CHANNEL Passed {data}")
        for j in range(num):
          #print(self.channelData, data, i, j)
          self.channelData[j] += data[j]
        self.syncVal += 1
        if self.syncVal == self.senderCount:
          for receiver in range(self.senderCount): # receiverCount = senderCount
            self.channelToReceiver[receiver].send(self.channelData)
          self.syncVal = 0
          self.channelData = [0 for _ in range(walshCode.nextPowerOf2(self.senderCount))]
```

iv)**receiver thread implementation**:

```
def receive_data(self):
    curr_datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    with open('logs/log.txt', 'a+', encoding='utf-8') as f:
        f.write(f"\n[{curr_datetime}] RECEIVER-{self.name+1} Receives Data from SENDER-
{self.senderToReceiver+1}")
    entiredata = []
    while True:
      channeldata = self.channelToReceiver.recv()
      sum = 0
      for i in range(len(channeldata)):
        sum += channeldata[i] *
      self.wTable[i] sum /= self.codeLength
      if sum == 1:
        bit = 1
      elif sum == -1:
        bit = 0
      else:
        bit = -1
      curr_datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
      with open('logs/log.txt', 'a+', encoding='utf-8') as f:
        f.write(f"\n[{curr_datetime}] RECEIVER-{self.name+1} Bit Received: {bit}")
      if len(entiredata) < 8 and bit != -1:
        entiredata.append(bit)
      if len(entiredata) == 8:
        byte = self.getByte(entiredata)
        output_file =
        self.read_file(self.senderToReceiver)
        output_file.write(byte)
        output_file.close()
        entiredata = []
```

v) **Thread initialization and pipe implementation in main.py**:

```
def simulate_environment(wTable,
    senderCount): writeRecvFd = [] #channel to
    receiver readRecvFd = [] #channel to receiver
    for _ in range(senderCount):
        readhead, writehead = multiprocessing.Pipe()
        readRecvFd.append(readhead) # file descriptor taken by receiver
        writeRecvFd.append(writehead) # file descriptor taken by channel
    readSendFd, writeSendFd = multiprocessing.Pipe() # sender to channel
    senderObjList = []
    receiverObjList = []
    senderThreads = []
    receiverThreads = []
    channel = Channel(senderCount, 0, readSendFd, writeRecvFd)
    for i in range(senderCount):
        sender = Sender(i, wTable[i],
        writeSendFd)
        senderObjList.append(sender)
        receiver = Receiver(i, wTable[i],
        readRecvFd[i])
        receiverObjList.append(receiver)
    channelThread = threading.Thread(target=channel.relayThread)
    for i in range(senderCount):
            sthread=threading.Thread(name="sender_thread"+str(i+1),
target=senderObjList[i].send_data)
        senderThreads.append(sthread)
rthread=threading.Thread(name="receiver_thread"+str(i+1), target=receiverObjList[i].receive_data)
        receiverThreads.append(rthread)
    channelThread.start()
    for thread in receiverThreads:
        thread.start()
    for thread in senderThreads:
        thread.start()
    for thread in senderThreads:
        thread.join()
    for thread in receiverThreads:
        thread.join()
    channelThread.join()
```

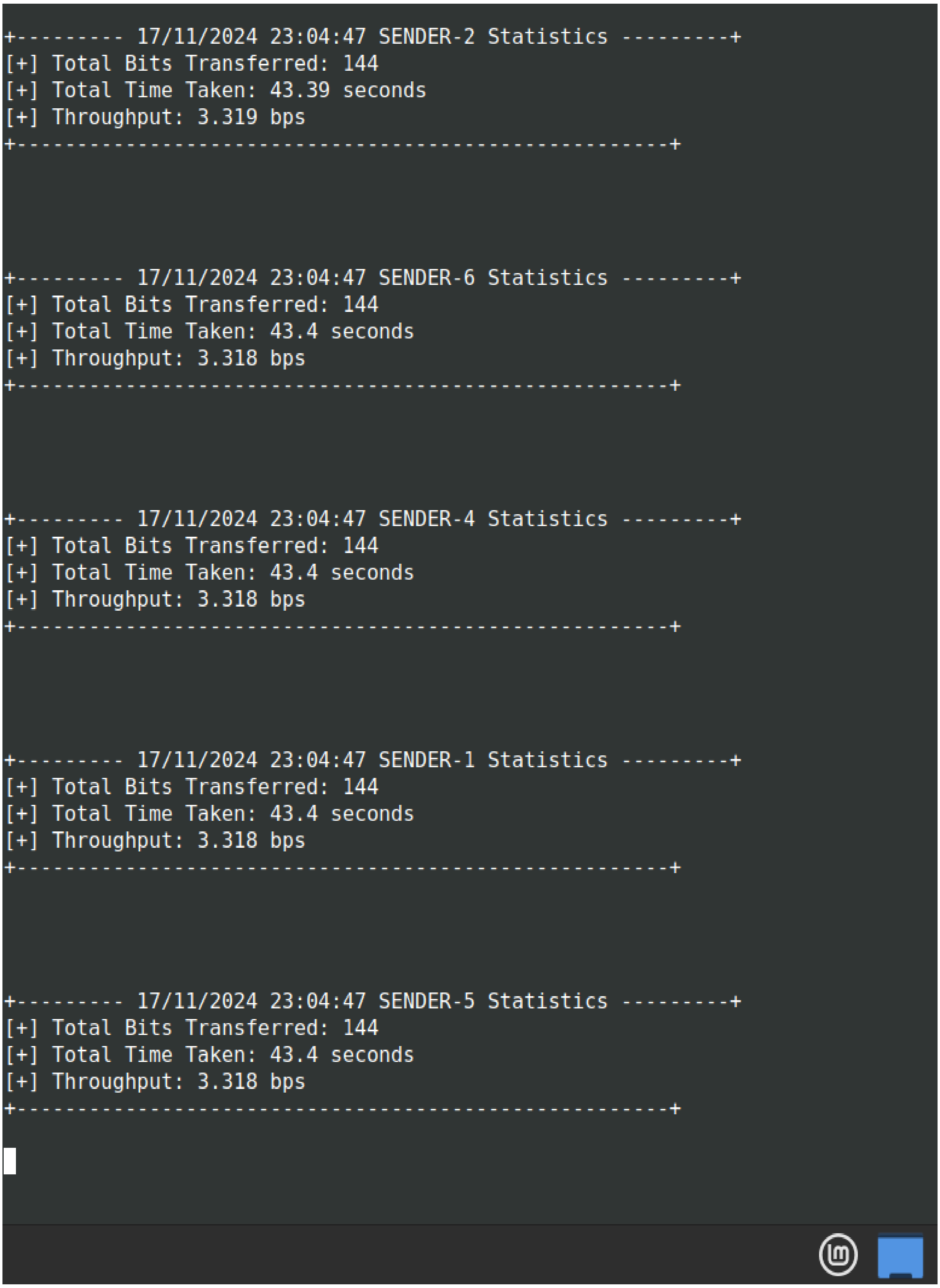
RESULTS:

1) For 5 stations and data word as: “JEETESH ABROL”, the metrics for each sender was about:
[+] Total Bits Transferred: 104
[+] Total Time Taken: 31.32 seconds
[+] Throughput: 3.321 bps

2) For 6 stations and data word as “Jeetesh Abrol test”, the metrics for each sender was about:

[+] Total Bits Transferred: 144
[+] Total Time Taken: 43.4 seconds
[+] Throughput: 3.318 bps

The below image shows the output in the file: analysis.txt:



```
+--------- 17/11/2024 23:04:47 SENDER-2 Statistics ---------+
[+] Total Bits Transferred: 144
[+] Total Time Taken: 43.39 seconds
[+] Throughput: 3.319 bps
+----------------------------------------------------+




+--------- 17/11/2024 23:04:47 SENDER-6 Statistics ---------+
[+] Total Bits Transferred: 144
[+] Total Time Taken: 43.4 seconds
[+] Throughput: 3.318 bps
+----------------------------------------------------+




+--------- 17/11/2024 23:04:47 SENDER-4 Statistics ---------+
[+] Total Bits Transferred: 144
[+] Total Time Taken: 43.4 seconds
[+] Throughput: 3.318 bps
+----------------------------------------------------+




+--------- 17/11/2024 23:04:47 SENDER-1 Statistics ---------+
[+] Total Bits Transferred: 144
[+] Total Time Taken: 43.4 seconds
[+] Throughput: 3.318 bps
+----------------------------------------------------+




+--------- 17/11/2024 23:04:47 SENDER-5 Statistics ---------+
[+] Total Bits Transferred: 144
[+] Total Time Taken: 43.4 seconds
[+] Throughput: 3.318 bps
+----------------------------------------------------+
```

ANALYSIS

1) No flow control protocol is considered here, hence the channel is assumed to be noise and disturbance free which isn’t the practical scenario.
2) No particular packet format is considered, but again this is just a simulation environment.
3) All the outputs are taken in different files which makes the implementation very slow and also very resource intensive but on the other hand we get a cleaner and better analysis output well formatted in a file.

Though the above points can make a difference in the results in real world situations, the results are quite accurate otherwise.