Name – Jeetesh Abrol   Assignment no. 3
Roll – 002210501021   BCSE III
Sub - Computer Networks   Group –A1

Assignment: In this assignment, you have to implement 1-persistent, non-persistent and p-persistent CSMA techniques. Measure the performance parameters like throughput (i.e., average amount of data bits successfully transmitted per unit time) and forwarding delay (i.e., average end-to-end delay, including the queuing delay and the transmission delay) experienced by the CSMA frames (IEEE 802.3). Plot the comparison graphs for throughput and forwarding delay by varying p. State your observations on the impact of performance of different CSMA techniques.

DESIGN:

i) **One-persistent CSMA**: In 1-persistent CSMA, the station continuously senses the channel to check its state i.e. idle or busy so that it can transfer data or not. In case when the channel is busy, the station will wait for the channel to become idle. When a station finds an idle channel, it transmits the frame to the channel immediately. It transmits the frame with probability 1. Hence, it is called 1-persistent CSMA.
* Design Component for One-persistent:
Each **sender** threads continuously check if the channel is busy if it is, the thread will wait for a certain time and sense continuously again; if it's not then it checks if collide.txt has value 1, that means a collision occurred and increase collision value else it reads data and sends to channel and makes collide.txt value 1 for vulnerable time and again makes it zero.

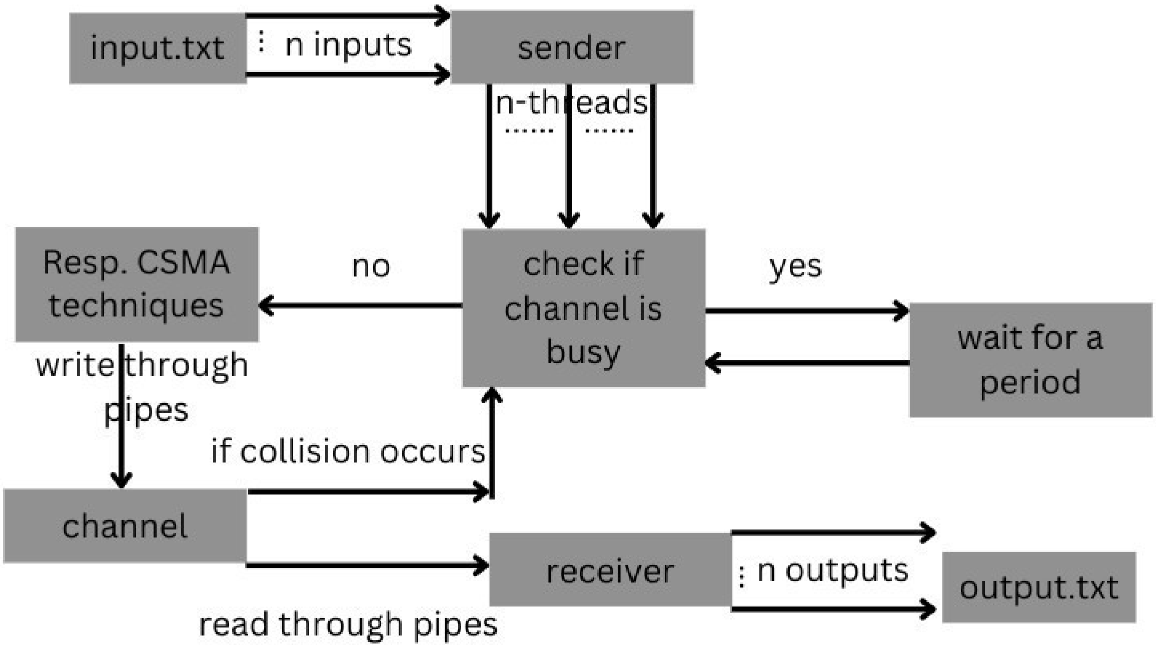
ii) **Non-persistent CSMA**: In the non persistent method, a station that has a frame to send senses the line. If the line is idle, it sends immediately. If the line is not idle, it waits for a random amount of time and then senses the line again. The non persistent approach reduces the chance of collision because it is unlikely that two or more stations will wait the same amount of time and retry to send simultaneously.
* Design Component for Non-Persistent:
The checks and calculations remain same as for One-persistent for each **sender** thread but if the thread finds a channel to be busy, it waits for random amount of time and checks again.

iii) **P-persistent CSMA**: p-persistent CSMA is used when a channel has time-slots and that time- slot duration is equal to or greater than maximum propagation delay time for that channel. When station is ready to send frames, it will sense channel. If channel found to be busy, station will wait for next time-slot. But if channel is found to be idle, station transmits frame immediately with a probability p. The station thus waits for left probability i.e. q which is equal to 1-p, for beginning of next time-slot. If the next time-slot is also found idle, station transmits or waits again with probabilities p and q. This process repeats until either frame gets transmitted or another station starts transmitting.
* Design Component for P-Persistent:
The checks and calculations remain same as for One-persistent for each **sender** thread but if the thread finds a channel to be busy, it waits for certain amount of time and

checks again. Secondly it checks if a random number generated from random.random() which belongs to [0,1) is less than the probability p or not, if it is then it transmits data else it generates again.

* Other components:
packet manager: This module is meant to form packet of a particular format from raw data in order to be transmitted.
Channel: This basically relays the data sent from the sender threads into receiver threads with the help of pipes.
Receiver: Receives the packet from channel through pipes and decodes the data from packet and writes the output to respective output files.
Validate: This module helps to generate/check the checksum of a packet
* The logs are stored in a file log.txt and the final results are stored in a file

analysis.txt Below is the flow diagram of the design:



IMPLEMENTATION:

1) managing packets with
packetManager.py: class Packet:

```
   def _init_(self, _type, seq, segment_data, src, dest) ->
      None: self.type = _type
      self.seq = seq
      self.segment_data =
      segment_data self.src = src
      self.dest = dest
      self.packet = ''
   def
      generate_packet(sel
      f): preamble =
      '01'*28
      sfd = '10101011'
      src_addr = '{0:048b}'.format(int(self.src))
      dest_addr = '{0:048b}'.format(int(self.dest))
```

```
    seqbits = '{0:08b}'.format(self.seq)
    length                                        =
    '{0:08b}'.format(len(self.segment_data))
    data = ''
    for                i                in
        range(len(self.segment_data)):
        character                        =
        self.segment_data[i]
        data += '{0:08b}'.format(ord(character))
    packet = preamble + sfd + dest_addr + src_addr + seqbits +
    length + data checksum = validate.get_checksum(packet)
    packet          +=
    checksum
    self.packet         =
    packet return self
def _str_(self) -> str:
    return
    str(self.packet)
def get_datalen(self) -> int:
    return
    len(self.segment_data)
def get_type(self) ->
    int: return self.type
def get_seqno(self) -> int:
    seqbits =
    self.packet[160:168]
    return int(seqbits, 2)
def get_src(self) -> int:
    return
int(self.packet[112:160], 2) def
get_dest(self) -> int:
    return int(self.packet[64:112],
2) def get_data(self) -> str:
    datastr = ""
    databits = self.packet[176:544]
    datastr = long_to_bytes(int(databits, 2)).decode('utf-
    8') return datastr
def validate_packet(self) -> bool:
    return validate.validate_checksum(self.packet)
```

2) relay packets through **channel.py**:

```
def transfer_data_pkts(self):
        '''Sending data packets from Sender to Receiver through the
        channel''' while True:
            packet = self.senderToChannel.recv()
            self.active = True
            time.sleep(channel_propagation_delay)
            self.active = False
            dest = packet.get_dest()
            self.channelToReceiver[dest].send(packet)
    def transfer_response(self, sender:
        int): while True:
            if self.active:
                self.channelToSender[sender].send(str(1)) #denoting
            channel busy else:
                self.channelToSender[sender].send(str(0)) #denoting
    channel idle def init_channel(self):
        curr_time =
        datetime.datetime.now() with
        open("logs/log.txt", "a+") as f:
            f.write(f"\n[{curr_time.strftime('%d/%m/%Y %H:%M:%S')}]  Channel  initialized.\n")
            channelToReceiverThreads = [] channelToSenderThreads = [] sender = 0
        dataThread = threading.Thread(name="DataThread-"+str(sender+1),
target=self.transfer_data_pkts)
        channelToReceiverThreads.append(dataThr
        ead) for _ in range(self.sendercnt):
            respThread = threading.Thread(name="RespThread-"+str(sender+1),
target=self.transfer_response, args=(sender,))
            channelToSenderThreads.append(respThr
            ead) sender += 1
        for thread in channelToReceiverThreads:
            thread.start()
        for thread in channelToSenderThreads:
            thread.start()
        for thread in channelToReceiverThreads:
            thread.join()
        for thread in channelToSenderThreads:
            thread.join()
```

3) receive data with **receiver.py**:

```
def write_file(self, filename:
        str): try:
            fd = open(filename, "a+")
        except FileNotFoundError as
        err:
            current_time = datetime.datetime.now()
                      print(f"\n  [{current_time.strftime('%d/%m/%Y  %H:%M:%S')}]  ERROR:  {err}
                      File
{filename} not found!")
            sys.exit(f"File {filename} Not Found!")
        return fd
    def
        init_receiver(self)
        : while True:
            packet = self.channelToReceiver.recv()
            sender = packet.get_src()
            if sender not in self.sender_dict.keys():
                self.sender_dict[sender] = "./logs/output/output" + str(sender+1) +
            '.txt' outfile = self.sender_dict[sender]
            fd                              =
            self.write_file(outfile)
            datastr                         =
            packet.get_data()
            fd.write(datastr)
            fd.close()
            current_time =
            datetime.datetime.now() with
            open("logs/log.txt", "a+") as f:
                     f.write(f"\n[{current_time.strftime('%d/%m/%Y %H:%M:%S')}] Receiver-
{self.id+1} received Packet SUCCESSFULLY!\n")
```

4) **one-persistent sender**:

```
def one_persistent(self,
        packet): while True:
            if not self.busy:
                f = self.read_file("./logs/collide.txt")
                collision = f.read()
                f.close()
                if collision == '1':
                    self.collisionCount += 1
                    current_time = datetime.datetime.now()
                    with open("logs/log.txt", "a+", encoding="utf-8") as fp:
                          fp.write(f"[{current_time.strftime('%d/%m/%Y %H:%M:%S')}] Sender
                          {self.id+1}
encounters COLLISION.")
                    time.sleep(0.1) # wait after
                collision else:
                    current_time = datetime.datetime.now()
                    with open("logs/log.txt", "a+", encoding="utf-8") as fp:
                          fp.write(f"[{current_time.strftime('%d/%m/%Y %H:%M:%S')}] Sender
                          {self.id+1}
sent Packet {self.packetCount+1} to Channel\n")
                    f = open('logs/collide.txt', "w", encoding='utf-
                    8') f.write(str(1))
                    f.close()
                    time.sleep(0.1) # vulnerable time
                    f = open('logs/collide.txt', "w",  encoding='utf-
                    8') f.write(str(0))
                    f.close()
                    self.senderToChannel.send(packe
                    t) time.sleep(1) # propagation
                    time break
            else:
                current_time = datetime.datetime.now()
                with open("logs/log.txt", "a+", encoding="utf-8") as fp:
                         fp.write(f"[{current_time.strftime('%d/%m/%Y %H:%M:%S')}] Sender
                         {self.id+1}
finds Channel is BUSY.")
                time.sleep(0.5)
                continue
```

5) **Non-persistent sender**:

```
def non_persistent(self,
        packet): while True:
            if not self.busy:
                f = self.read_file("./logs/collide.txt")
                collision = f.read()
                f.close()
                if collision == '1':
                    self.collisionCount += 1
                    current_time = datetime.datetime.now()
                    with open("logs/log.txt", "a+", encoding="utf-8") as fp:
                          fp.write(f"[{current_time.strftime('%d/%m/%Y %H:%M:%S')}] Sender
                          {self.id+1}
encounters COLLISION.")
                    time.sleep(0.1) # wait after
                collision else:
```

```
          current_time = datetime.datetime.now()
          with open("logs/log.txt", "a+", encoding="utf-8") as fp:
             fp.write(f"[{current_time.strftime('%d/%m/%Y %H:%M:%S')}] Sender
             {self.id+1}
sent Packet {self.packetCount+1} to Channel\n")
          f = open('logs/collide.txt', "w", encoding='utf-
          8') f.write(str(1))
          f.close()
          time.sleep(0.1) # vulnerable time
          f = open('logs/collide.txt', "w",  encoding='utf-
          8') f.write(str(0))
          f.close()
          self.senderToChannel.send(packe
          t) time.sleep(1) # propagation
          time break
      else:
        random.seed(time.time())
        current_time = datetime.datetime.now()
        with open("logs/log.txt", "a+", encoding="utf-8") as fp:
             fp.write(f"[{current_time.strftime('%d/%m/%Y %H:%M:%S')}] Sender
             {self.id+1}
finds Channel is BUSY.")
        time.sleep(random.randint(1
        , 5)) continue
```

6) **p-persistent sender**:

```
def p_persistent(self, packet):
    while True:
      if not self.busy:
        x =
        random.random() p
        =
        1/self.senderCount
        if x <= p:
          f = self.read_file("./logs/collide.txt")
          collision = f.read()
          f.close()
          if collision == '1':
            self.collisionCount +=
            1
            current_time = datetime.datetime.now()
            with open("logs/log.txt", "a+", encoding="utf-8") as fp:
                      fp.write(f"[{current_time.strftime('%d/%m/%Y %H:%M:%S')}]
                      Sender
{self.id+1} encounters COLLISION.")
            time.sleep(0.1) # wait after
          collision else:
            current_time = datetime.datetime.now()
            with open("logs/log.txt", "a+", encoding="utf-8") as fp:
                      fp.write(f"[{current_time.strftime('%d/%m/%Y %H:%M:%S')}]
                      Sender
{self.id+1} sent Packet {self.packetCount+1} to Channel\n")
            f = open('logs/collide.txt', "w",
            encoding='utf-8') f.write(str(1))
            f.close()
            time.sleep(0.1) # vulnerable time
```

```
                f = open('logs/collide.txt', "w",
                encoding='utf-8') f.write(str(0))
                f.close()
                self.senderToChannel.send(pac
                ket) time.sleep(1) #
                propagation time break
            else:
              current_time = datetime.datetime.now()
              with open("logs/log.txt", "a+", encoding="utf-8") as fp:
              fp.write(f"[{current_time.strftime('%d/%m/%Y %H:%M:%S')}] Sender
              {self.id+1} is WAITING, wait period:0.25secs.\n")
              time.sleep(0.25) # wait a certain time
          else:
            current_time = datetime.datetime.now()
            with open("logs/log.txt", "a+", encoding="utf-8") as fp:
                  fp.write(f"[{current_time.strftime('%d/%m/%Y %H:%M:%S')}] Sender
                  {self.id+1}
finds Channel is BUSY.")
            time.sleep(0.5)
            continue
```

RESULTS:

For a particular input, the results for different CSMA techniques are as follows:
i) For **one-persistent CSMA**, performance metrics are:

```
+---------- 2022-10-29 00:30:55.387419 SENDER-3 STATS---------+
[*]     Total packets: 20
[*]     Total Delay: 91.66
secs [*]        Total
collisions: 4
[*]     Throughput: 0.833
+-------------------------------------------------------- +
+---------- 2022-10-29 00:30:57.881816 SENDER-4 STATS---------+
[*]     Total packets: 20
[*]     Total Delay: 94.16
secs [*]        Total
collisions: 4
[*]     Throughput: 0.833
+-------------------------------------------------------- +
+---------- 2022-10-29 00:31:01.897271 SENDER-6 STATS---------+
[*]     Total packets: 20
[*]     Total Delay: 98.17
secs [*]        Total
collisions: 13
[*]     Throughput: 0.606
+-------------------------------------------------------- +
+---------- 2022-10-29 00:31:08.282059 SENDER-2 STATS---------+
[*]     Total packets: 20
[*]     Total Delay: 104.55
secs [*]        Total collisions:
16
[*]     Throughput: 0.556
+-------------------------------------------------------- +
+---------- 2022-10-29 00:31:08.426892 SENDER-5 STATS---------+
[*]     Total packets: 20
[*]     Total Delay: 104.7
secs [*]        Total
collisions: 13
[*]     Throughput: 0.606
+------------------------------------------------------------ +
+---------- 2022-10-29 00:31:12.744027 SENDER-1 STATS---------+
[*]     Total packets: 20
[*]     Total Delay: 109.02
secs [*]        Total collisions:
11
[*]     Throughput: 0.645
+------------------------------------------------------------ +
```

ii) For non-**persistent CSMA**, performance metrics are:

```
iii)    +---------- 2022-10-29 00:33:35.334795 SENDER-5 STATS      +
[*]     Total packets: 20
[*]     Total Delay: 62.41
secs [*]        Total
collisions: 1
[*]     Throughput: 0.952
+------------------------------------------------------------ +
+---------- 2022-10-29 00:34:06.583895 SENDER-6 STATS---------+
[*]     Total packets: 20
[*]     Total Delay: 93.66
secs [*]        Total
collisions: 3
[*]     Throughput: 0.87
+------------------------------------------------------------ +
+---------- 2022-10-29 00:34:18.702857 SENDER-4 STATS---------+
[*]     Total packets: 20
[*]     Total Delay: 105.78
secs [*]        Total collisions:
4
[*]     Throughput: 0.833
+------------------------------------------------------------ +
+---------- 2022-10-29 00:34:33.827996 SENDER-1 STATS---------+
[*]     Total packets: 20
[*]     Total Delay: 120.91
secs [*]        Total collisions:
5
[*]     Throughput: 0.8
+------------------------------------------------------------ +
+---------- 2022-10-29 00:34:37.411350 SENDER-3 STATS---------+
[*]     Total packets: 20
[*]     Total Delay: 124.49
secs [*]        Total collisions:
1
[*]     Throughput: 0.952
+------------------------------------------------------------ +
+---------- 2022-10-29 00:34:45.640098 SENDER-2 STATS---------+
[*]     Total packets: 20
[*]     Total Delay: 132.72
secs [*]        Total collisions:
3
[*]     Throughput: 0.87
+------------------------------------------------------------ +
```

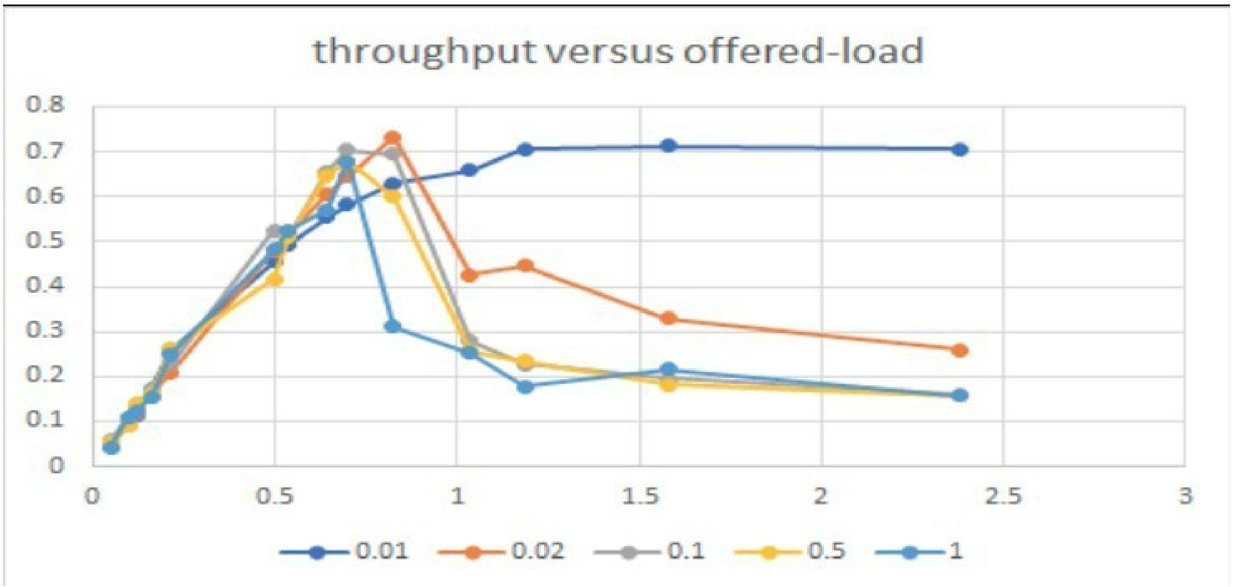
iv) For **p-persistent CSMA**, performance metrics are:

```
+---------- 2022-10-29 00:39:18.596823 SENDER-2 STATS---------+
[*]	Total packets: 20
[*]	Total Delay: 106.31
secs [*]	Total collisions:
1
[*]	Throughput: 0.952
+-------------------------------------------------------- +
+---------- 2022-10-29 00:39:31.137221 SENDER-3 STATS---------+
[*]	Total packets: 20
[*]	Total Delay: 118.85
secs [*]	Total collisions:
3
[*]	Throughput: 0.87
+-------------------------------------------------------- +
+---------- 2022-10-29 00:39:38.090426 SENDER-4 STATS---------+
[*]	Total packets: 20
[*]	Total Delay: 125.8 secs [*]		Total collisions: 2
[*]	Throughput: 0.909
+-------------------------------------------------------- +
+---------- 2022-10-29 00:39:55.875035 SENDER-1 STATS---------+
[*]	Total packets: 20
[*]	Total Delay: 143.59
secs [*]	Total collisions:
6
[*]	Throughput: 0.769
+-------------------------------------------------------- +
+---------- 2022-10-29 00:40:09.398525 SENDER-6 STATS---------+
[*]	Total packets: 20
[*]	Total Delay: 157.11
secs [*]	Total collisions:
7
[*]	Throughput: 0.741
+-------------------------------------------------------- +
+---------- 2022-10-29 00:40:15.518801 SENDER-5 STATS---------+
[*]	Total packets: 20
[*]	Total Delay: 163.23
secs [*]	Total collisions:
3
[*]	Throughput: 0.87
+-------------------------------------------------------- +
```

Justification

1. In one persistent the frame is sent immediately after the sender senses the channel idle, therefore it has the maximum chances of collision.
2. In non persistent, even if the sender sends the packet immediately as soon as the channel is detected idle, but it waits for a random time interval to send the packet as it detects the channel is busy.
3. P persistent method uses combination of above two methods. When it senses an idle channel, it doesn’t transmit immediately. It generates a random value ‘x’ which must be less than p (= 1/n) to transmit the frame. If ‘x’ exceeds p, then it waits for a timeslot (Tp) then again senses the channel and repeats the above process. It is unlikely for different senders to generate ‘x’ (<p) in the same slot which reduces collision probability.

The below graph compares the different values of p:



ANALYSIS:

From the above results, the points we can clearly see that:

i) Collision count for one-persistent > non-persistent >= p-persistent.

ii) Average Throughput for each sender one-persistent < p-persistent < non-persistent

iii) The throughput for p-persistent is maximum when p = 1/n, where n is the number of stations/senders.

COMMENTS:

This lab assignment gives insights to different CSMA techniques. Implementation of the schemes give greater understanding of the topic (how the protocols are different from each other in terms of performance, collision avoidance, throughput). Though the implementation is quite resource intensive, the performance metrics is quite accurate and hence we can get an idea in real world how these different CSMA methods performs against each other.