Networks Lab
Assignment 3

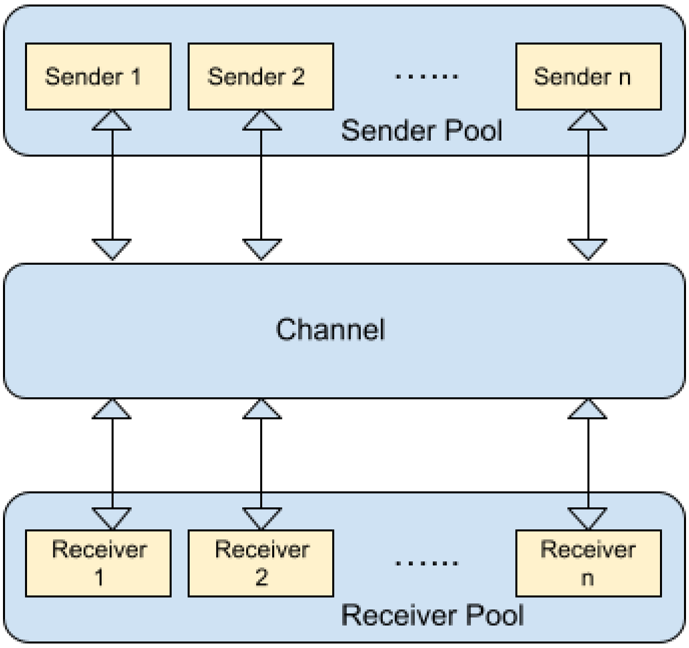
Name: Koustav Dhar Class: BCSE UG-III Group: A1 Roll: 001910501022

Problem Statement:

Implement 1-persistent, non-persistent and p-persistent CSMA techniques.

Design:

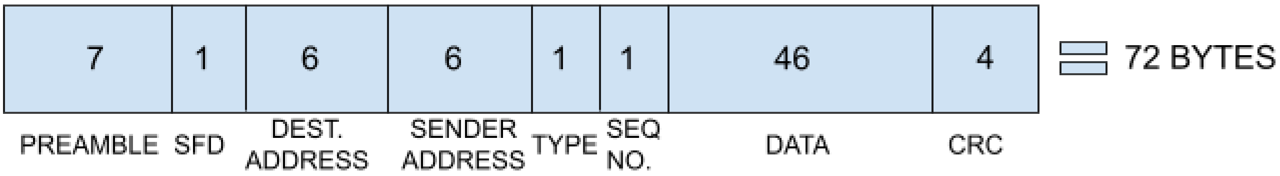
- Sender:
  - Get the sender type(1-Persistent/Non-Persistent/P-Persistent).
  - Connect with the Channel.
  - Read data from the file and build packet in IEEE 802.3 frame format.
  - Use specified persistent method and send the packet, whenever permitted.
- Channel:
  - Get station request(Connect[Receiver]/Send[Sender] request).
  - Sender:
    - Get the packet from the sender.
    - Wait for vulnerable time (Tp).
    - Check for collision.
    - Send the packet to the receiver, if no collision.
  - Receiver:
    - Add receiver station to the available list.
- Receiver:
  - Connect with the channel.
  - Wait for data packets.
  - Receive data packet from the channel.
  - Check for error, sequence no. and extract data from the frame.



Flow Diagram of the Generalised Network

Implementation:

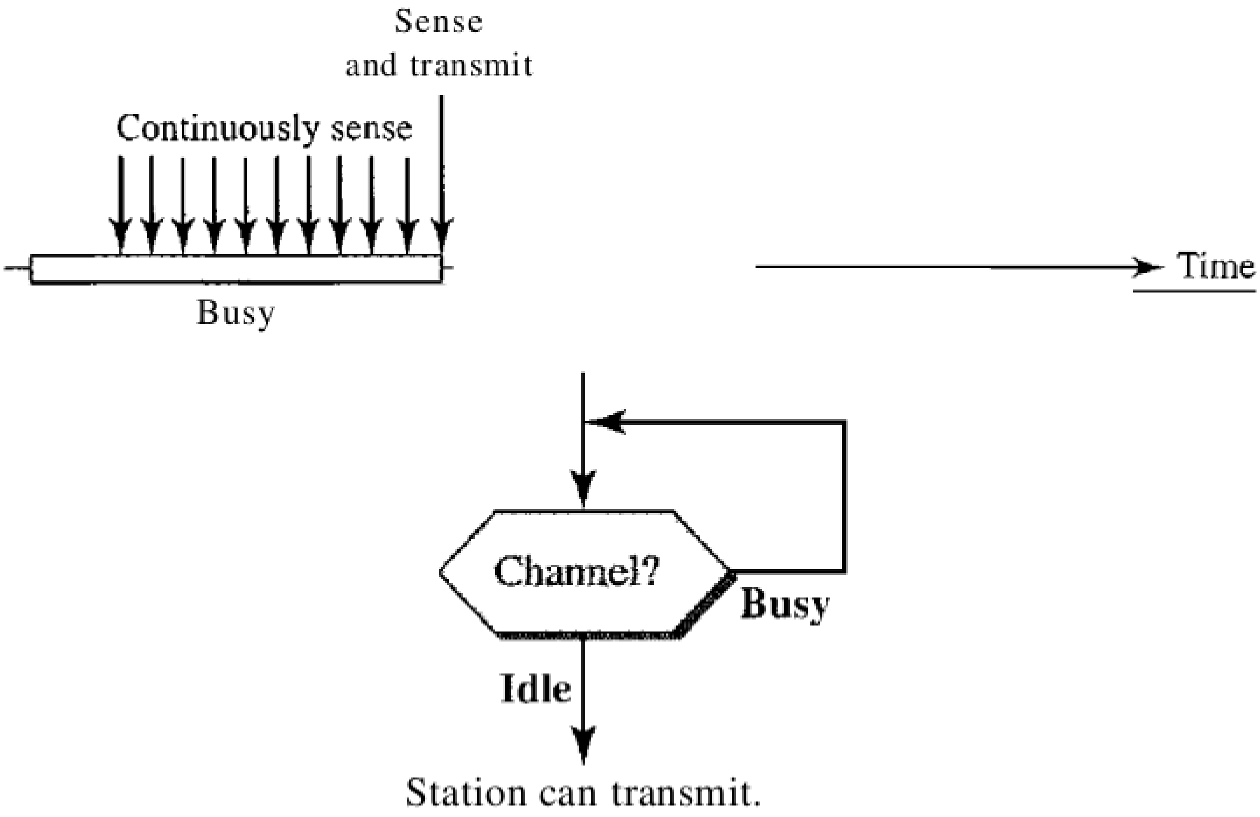
We have used the IEEE 802.3 Ethernet Frame Format for our packets here. Size of a frame is 72 bytes.



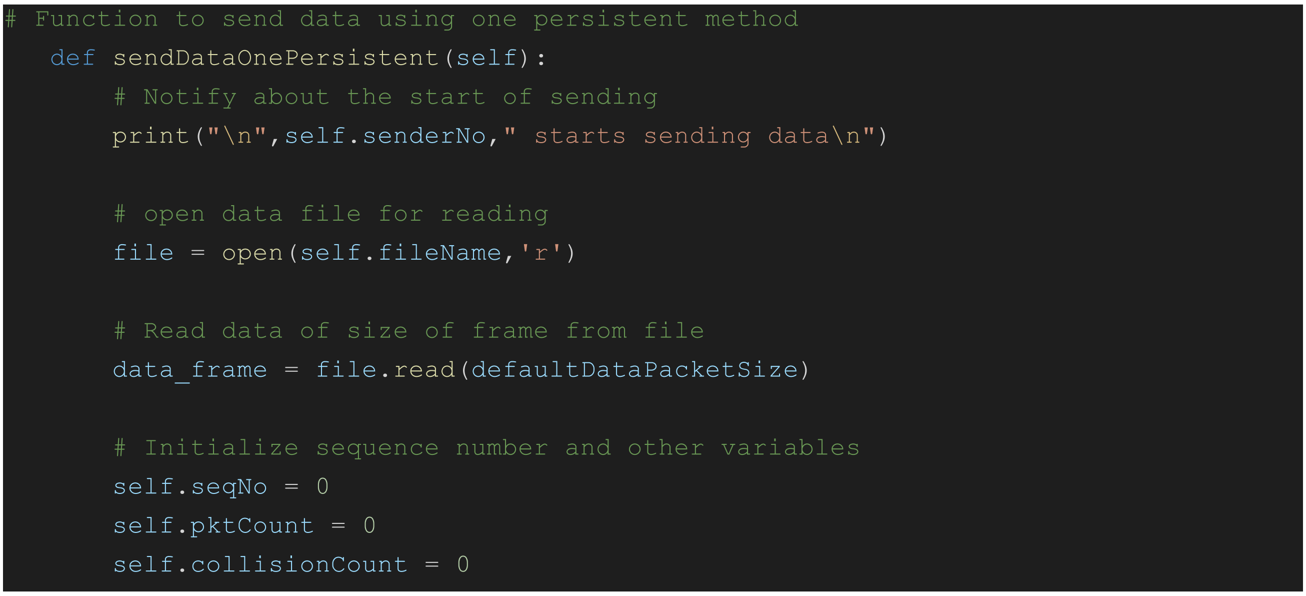
PREAMBLE	SFD	DEST. ADDRESS	SENDER ADDRESS	TYPE	SEQ NO.	DATA	CRC
7	1	6	6	1	1	46	4

IEEE 802.3 ETHERNET FRAME FORMAT

1-Persistent: 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 the station found an idle channel, it transmits the frame to the channel without any delay. It transmits the frame with probability 1. Due to probability 1, it is called 1-persistent CSMA.



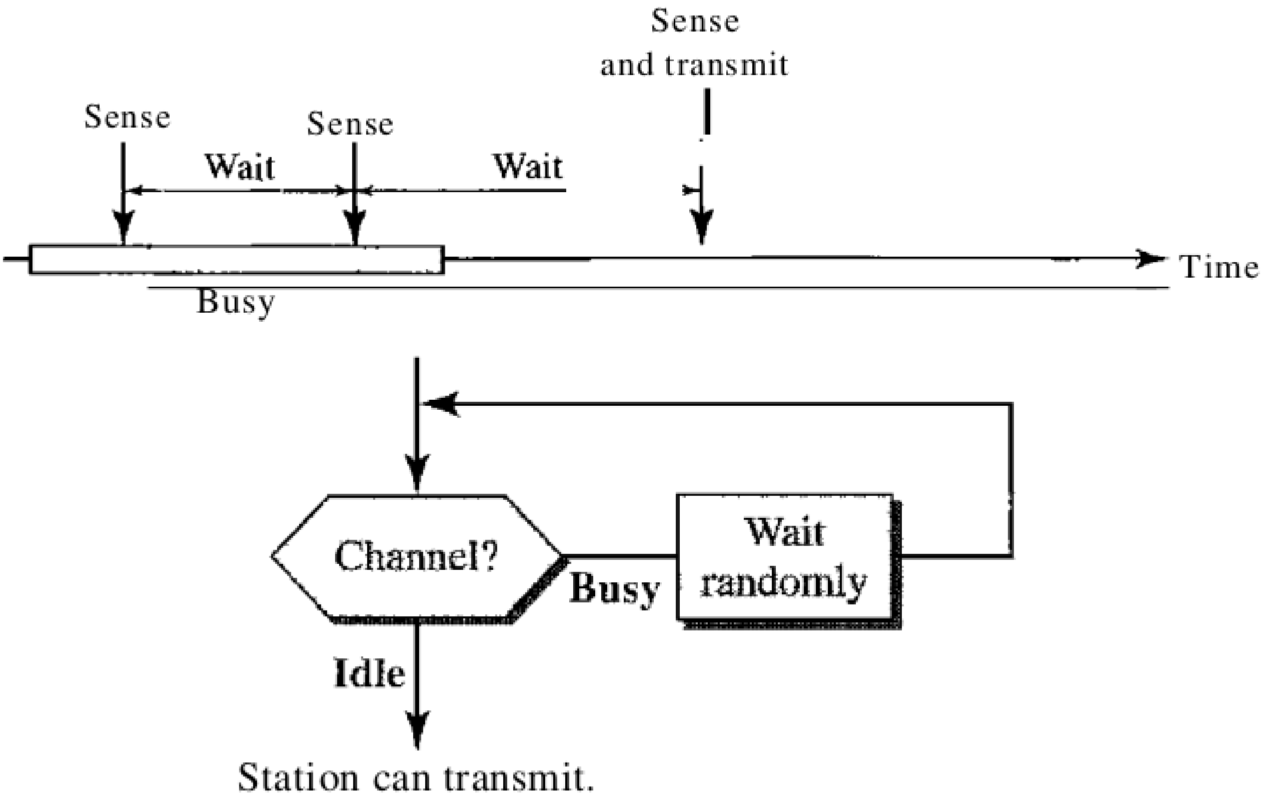
1-Persistent Method



```
# Function to send data using one persistent method
    def sendDataOnePersistent(self):
        # Notify about the start of sending
        print("\n",self.senderNo," starts sending data\n")

        # open data file for reading
        file = open(self.fileName,'r')

        # Read data of size of frame from file
        data_frame = file.read(defaultDataPacketSize)

        # Initialize sequence number and other variables
        self.seqNo = 0
        self.pktCount = 0
        self.collisionCount = 0
        previousPkt = False

        # Loop until whole data is sent
        while data_frame:
            time.sleep(0.005)
            # Get the current status of the channel (busy/idle)
            self.connection.send(str.encode('status'))
            reply = self.connection.recv(1024)
            reply = reply.decode()

            # If channel is busy, loop until it becomes idle
            if reply == 'Busy':
                continue

            # If channel is idle, send data
            else:
                if not previousPkt:
                    # Build packet using data, type and sequence number
                    packet = PacketManager.Packet(self.senderAddress,
self.receiverAddress, 0, self.seqNo, data_frame)

                    # Increment sequence number and other parameters accordingly
                    self.seqNo += 1
                    previousPkt = True

                # Notify the channel about data sending
                self.connection.send(str.encode('sending'))
                self.connection.recv(1024)

                # Send the packet and increase packet count
                self.connection.send(str.encode(packet.toBinaryString(46)))
                self.pktCount += 1

                # Print sent status
                print("Sender ",self.senderNo," sent packet no ",self.seqNo," to
channel.")

                # Wait for propagation time
                time.sleep(propagation_time)

                # Get transmission status (send successfully/collision)
                transmission_status = self.connection.recv(1024).decode()

                # If collision ocuured, increase collision count
                if transmission_status == 'collision':
                    self.collisionCount += 1
                    print("Sender ",self.senderNo,", packet no ",self.seqNo,"
collided in channel.")

                elif transmission_status == 'Sent successfully':
                    previousPkt = False
```

```
                   print("Sender ",self.senderNo,", packet no ",self.seqNo,"
delivered successfully.")

                   # Read next data frame
                   data_frame = file.read(defaultDataPacketSize)

               # If all data has been read, break
               if len(data_frame) == 0: break

       # Close the data file
       file.close()

       # Send 'end transmission' message to channel
       self.connection.send(str.encode('end'))
```

<u>Non-Persistent:</u> In this method, the station that has frames to send, only that station senses for the channel. In case of an idle channel, it will send a frame immediately to that channel. In case when the channel is found busy, it will wait for the random time and again sense for the state of the station whether idle or busy. In this method, the station does not immediately sense the channel for only the purpose of capturing it when it detects the end of the previous transmission. The main advantage of using this method is that it reduces the chances of collision. The problem with this is that it reduces the efficiency of the network

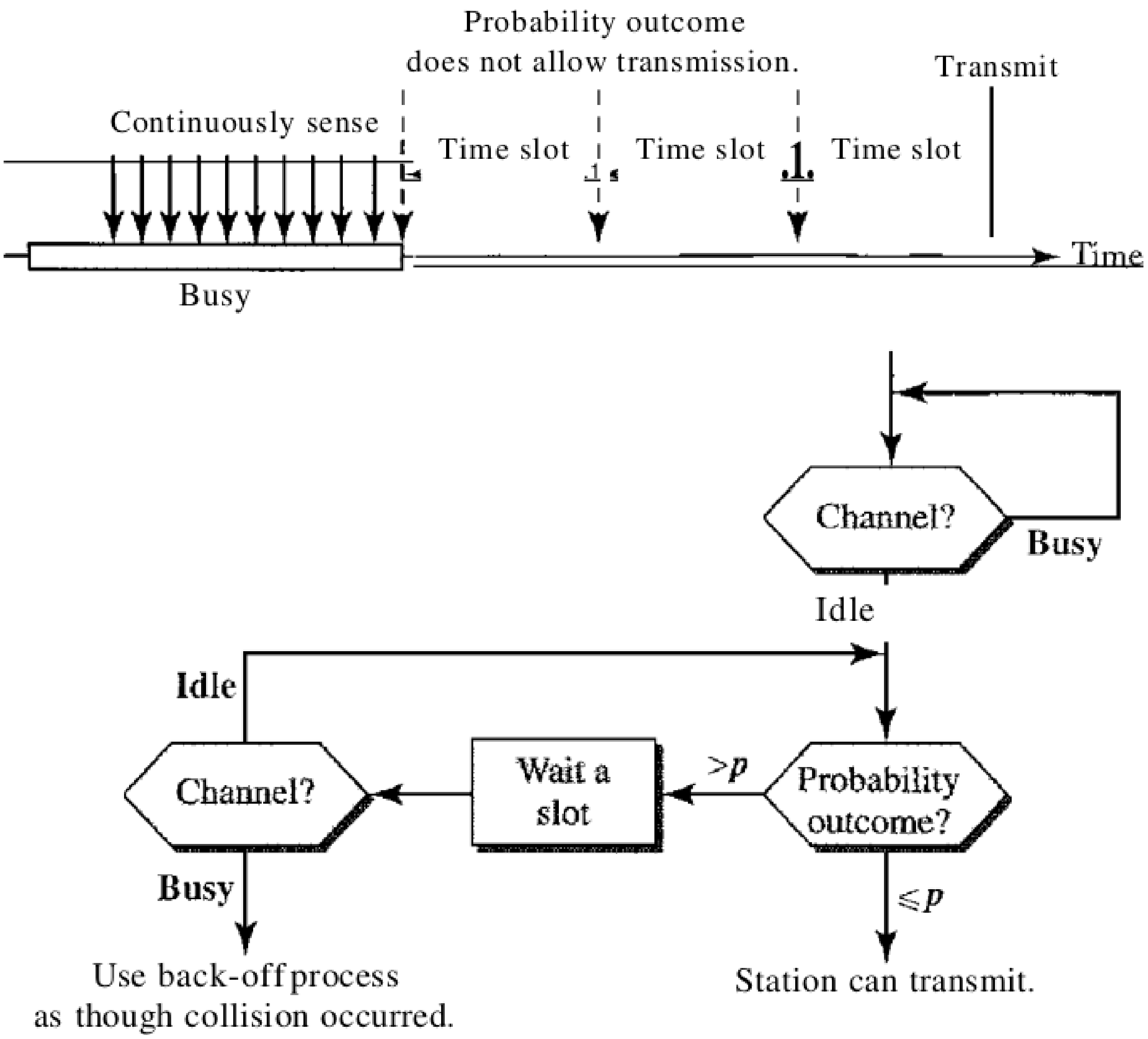


Non-Persistent Method

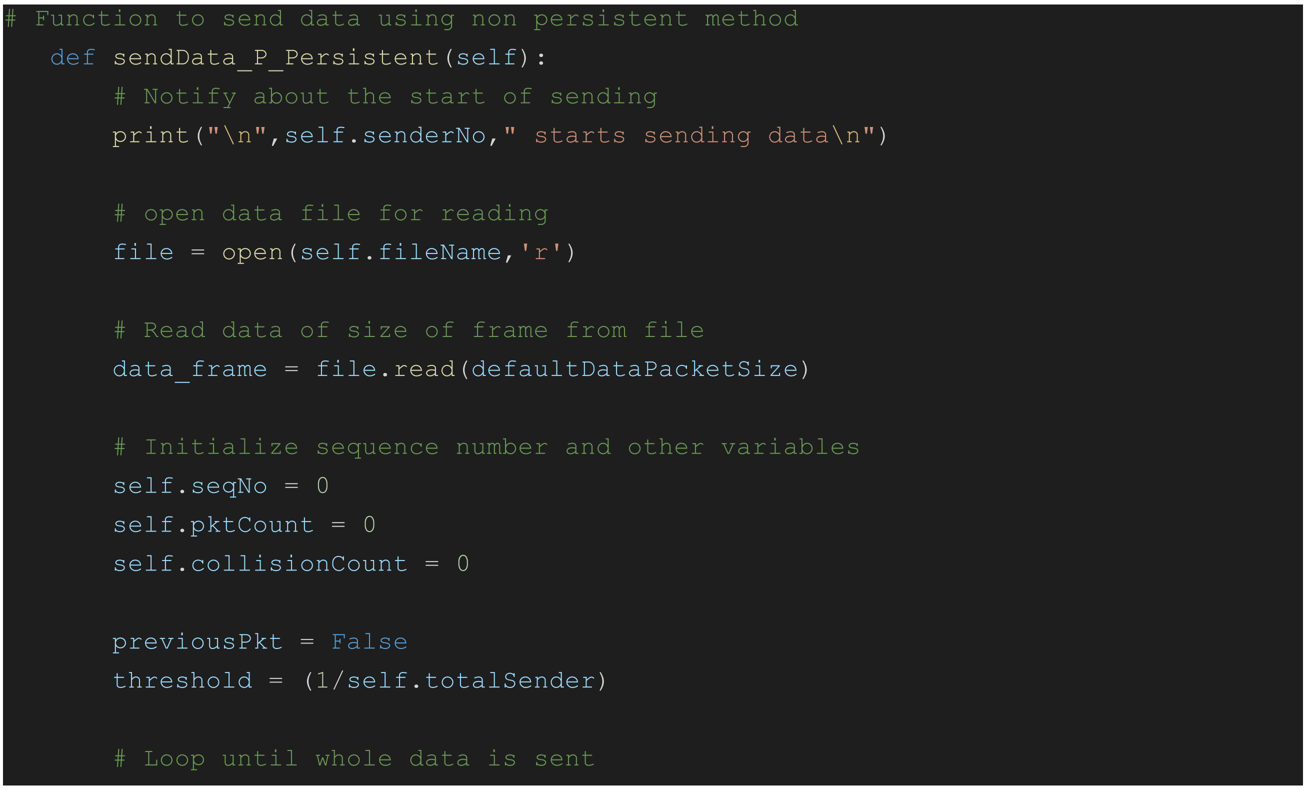
```
# Function to send data using non persistent method
   def sendDataNonPersistent(self):
       # Notify about the start of sending
       print("\n",self.senderNo," starts sending data using non-persistent\n")

       # open data file for reading
       file = open(self.fileName,'r')
```

```
        # Read data of size of frame from file
        data_frame = file.read(defaultDataPacketSize)

        # Initialize sequence number and other variables
        self.seqNo = 0
        self.pktCount = 0
        self.collisionCount = 0
        previousPkt = False

        # Loop until whole data is sent
        while data_frame:
            time.sleep(0.005)
            # Get the current status of the channel (busy/idle)
            self.connection.send(str.encode('status'))
            reply = self.connection.recv(1024)
            reply = reply.decode()

            # If channel is busy, sleep for random time then check again
            if reply == 'Busy':
                start = 1
                end = (self.totalSender)
                randomTime = random.randint(start,end)
                time.sleep(randomTime*propagation_time)
                continue

            # If channel is idle, send data
            else:
                if not previousPkt:
                    # Build packet using data, type and sequence number
                    packet = PacketManager.Packet(self.senderAddress,
self.receiverAddress, 0, self.seqNo, data_frame)

                    # Increment sequence number and other parameters accordingly
                    self.seqNo += 1
                    previousPkt = True

                # Notify the channel about data sending
                self.connection.send(str.encode('sending'))
                self.connection.recv(1024)

                # Send the packet and increase packet count
                self.connection.send(str.encode(packet.toBinaryString(46)))
                self.pktCount += 1

                # Print sent status
                print("Sender ",self.senderNo," sent packet no ",self.seqNo," to
channel.")

                # Wait for propagation time
                time.sleep(propagation_time)
                # Get transmission status (send successfully/collision)
                transmission_status = self.connection.recv(1024).decode()

                # If collision ocuured, increase collision count
                if transmission_status == 'collision':
                    self.collisionCount += 1
                    print("Sender ",self.senderNo,", packet no ",self.seqNo,"
collided in channel.")

                elif transmission_status == 'Sent successfully':
                    previousPkt = False
                    print("Sender ",self.senderNo,", packet no ",self.seqNo,"
delivered successfully.")

                    # Read next data frame
                    data_frame = file.read(defaultDataPacketSize)

                # If all data has been read, break
                if len(data_frame) == 0: break

        # Close the data file
        file.close()

        # Send 'end transmission' message to channel
        self.connection.send(str.encode('end'))
```

P-Persistent: This is the method that is used when the channel has time-slots and that time-slot duration is equal to or greater than the maximum propagation delay time. When the station is ready to send the frames, it will sense the channel. If the channel is found to be busy, the channel will wait for the next slot. If the channel is found to be idle, it transmits the frame with probability p, thus for the left probability i.e. q which is equal to 1-p the station will wait for the beginning of the next time slot. In case, when the next slot is also found idle it will transmit or wait again with the probabilities p and q. This process is repeated until either the frame gets transmitted or another station has started transmitting.



P-Persistent Method



```
# Function to send data using non persistent method
   def sendData_P_Persistent(self):
       # Notify about the start of sending
       print("\n",self.senderNo," starts sending data\n")

       # open data file for reading
       file = open(self.fileName,'r')

       # Read data of size of frame from file
       data_frame = file.read(defaultDataPacketSize)

       # Initialize sequence number and other variables
       self.seqNo = 0
       self.pktCount = 0
       self.collisionCount = 0

       previousPkt = False
       threshold = (1/self.totalSender)

       # Loop until whole data is sent
```

```
        while data_frame:
            time.sleep(0.005)
            # Get the current status of the channel (busy/idle)
            self.connection.send(str.encode('status'))
            reply = self.connection.recv(1024)
            reply = reply.decode()

            # If channel is idle, send data
            if reply == 'Busy':
                # Wait for next time slot
                time.sleep(propagation_time)

            else:
                if not previousPkt:
                    # Build packet using data, type and sequence number
                    packet = PacketManager.Packet(self.senderAddress,
self.receiverAddress, 0, self.seqNo, data_frame)

                    # Increment sequence number and other parameters accordingly
                    self.seqNo += 1
                    previousPkt = True

                p = random.random()
                while p >= threshold:
                    time.sleep(propagation_time)
                    #time.sleep(0.1)

                    # Get the current status of the channel (busy/idle)
                    self.connection.send(str.encode('status'))
                    reply = self.connection.recv(1024)
                    reply = reply.decode()

                    if reply == 'Busy':
                        break

                    p = random.random()

                if reply == 'Busy':
                    continue

                # Notify the channel about data sending
                self.connection.send(str.encode('sending'))
                self.connection.recv(1024)


                # Send the packet and increase packet count
                self.connection.send(str.encode(packet.toBinaryString(46)))
                self.pktCount += 1

                # Print sent status
                print("Sender ",self.senderNo," sent packet no ",self.seqNo," to
channel.")
```

```
                # Wait for propagation time
                time.sleep(propagation_time)

                # Get transmission status (send successfully/collision)
                transmission_status = self.connection.recv(1024).decode()

                # If collision ocuured, increase collision count
                if transmission_status == 'collision':
                    self.collisionCount += 1
                    print("Sender ",self.senderNo,", packet no ",self.seqNo,"
collided in channel.")

                elif transmission_status == 'Sent successfully':
                    previousPkt = False
                    print("Sender ",self.senderNo,", packet no ",self.seqNo,"
delivered successfully.")

                    # Read next data frame
                    data_frame = file.read(defaultDataPacketSize)

                # If all data has been read, break
                if len(data_frame) == 0: break

        # Close the data file
        file.close()

        # Send 'end transmission' message to channel
        self.connection.send(str.encode('end'))
```

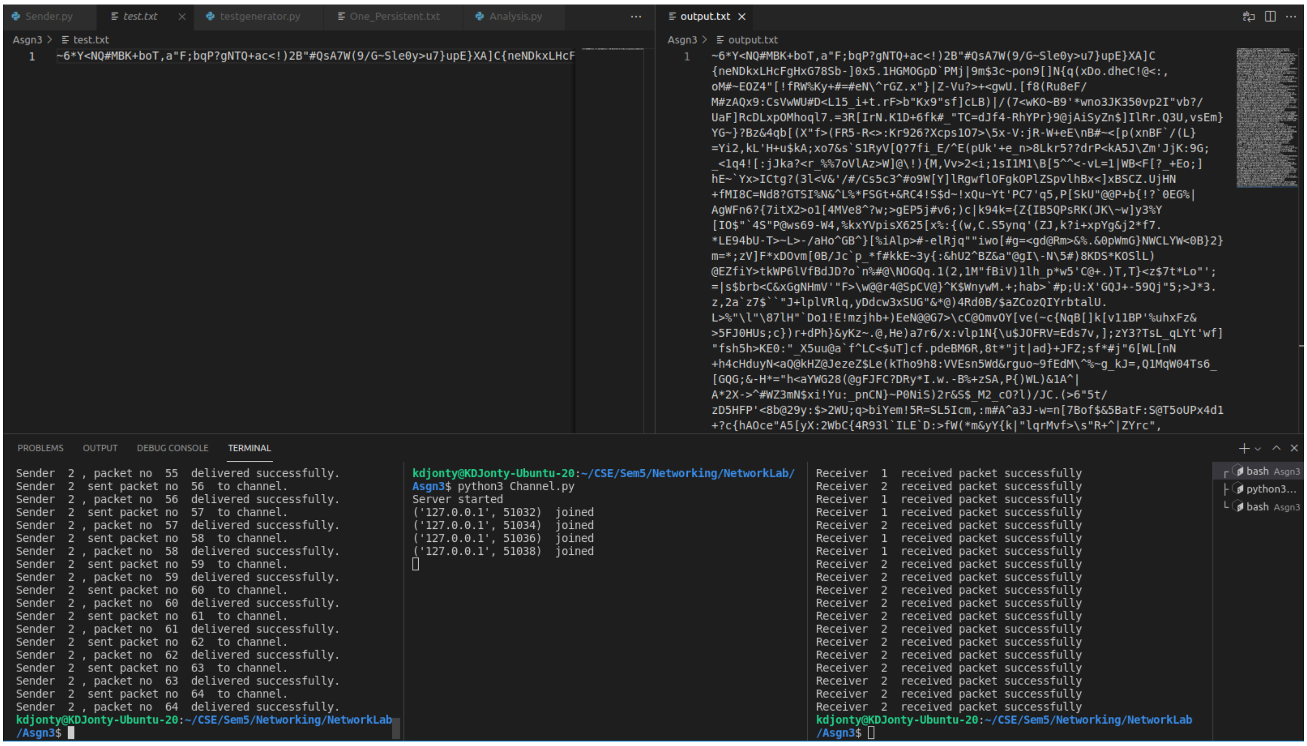
Test Cases:

We have generated random ASCII strings without whitespaces, of about 3×10^3 lengths, and performed all the tests on them.

testgenerator.py

```
import string
import random
import re

n = 23 * (128)

res = ''.join(random.choices(re.sub(r'\s+', '',string.printable), k = n))

file = open("test.txt", "w")
file.write(res)
file.close()
```

One of the test cases along with terminal output screens is shown below.



```
~6*Y<NQ#MBK+boT,a"F;bqP?gNTQ+ac<!)2B"#QsA7W(9/G~Sle0y>u7}upE}XA]C{neNDkxLHcF
```

```
~6*Y<NQ#MBK+boT,a"F;bqP?gNTQ+ac<!)2B"#QsA7W(9/G~Sle0y>u7}upE}XA]C
{neNDkxLHcFgHxG78Sb-]0x5.1HGMOGpD`PMj|9m$3c~pon9[]N{q(xDo.dheC!@<:,
oM#~EOZ4"[!fRW%Ky+#=#eN\^rGZ.x"}|Z-Vu?>+<gwU.[f8(Ru8eF/
M#zAQx9:CsVwWU#D<L15_i+t.rF>b"Kx9"sf]cLB)|/(7<wKO~B9'*wno3JK350vp2I"vb?/
UaF]RcDLxpOMhoql7.=3R[IrN.K1D+6fk#_"TC=dJf4-RhYPr}9@jAiSyZn$]IlRr.Q3U,vsEm}
YG~}?Bz&4qb[(X"f>(FR5-R<>:Kr926?Xcps1O7>\5x-V:jR-W+eE\nB#~<[p(xnBF`/(L}
=Yi2,kL'H+u$kA;xo7&s`S1RyV[Q?7fi_E/^E(pUk'+e_n>8Lkr5??drP<kA5J\Zm'JjK:9G;
_<1q4![:jJka?<r_%%7oVlAz>W]@\!){M,Vv>2<i;1sI1M1\B[5^^<-vL=1|WB<F[?_+Eo;]
hE~`Yx>ICtg?(3l<V&'/#/Cs5c3^#o9W[Y]lRgwflOFgkOPlZSpvlhBx<]xBSCZ.UjHN
+fMI8C=Nd8?GTSI%N&^L%*FSGt+&RC4!S$d~!xQu~Yt'PC7'q5,P[SkU"@@P+b{!?`0EG%|
AgWFn6?{7itX2>o1[4MVe8^?w;>gEP5j#v6;)c|k94k={Z{IB5QPsRK(JK\~w]y3%Y
[IO$"`4S"P@ws69-W4,%kxYVpisX625[x%:{(w,C.S5ynq'(ZJ,k?i+xpYg&j2*f7.
*LE94bU-T>~L>-/aHo^GB^}[%iAlp>#-elRjq""iwo[#g=<gd@Rm>&%.&0pWmG}NWCLYW<0B}2}
m=*;zV]F*xDOvm[0B/Jc`p_*f#kkE~3y{:&hU2^BZ&a"@gI\-N\5#)8KDS*KOSlL)
@EZfiY>tkWP6lVfBdJD?o`n%#@\NOGQq.1(2,1M"fBiV)1lh_p*w5'C@+.)T,T}<z$7t*Lo"';
=|s$brb<C&xGgNHmV'"F>\w@@r4@SpCV@}^K$WnywM.+;hab>`#p;U:X'GQJ+-59Qj"5;>J*3.
z,2a`z7$``"J+lplVRlq,yDdcw3xSUG"&*@)4Rd0B/$aZCozQIYrbtalU.
L>%"\l"\87lH"`Do1!E!mzjhb+)EeN@@G7>\cC@OmvOY[ve(~c{NqB[]k[v11BP'%uhxFz&
>5FJ0HUs;c})r+dPh}&yKz~.@,He)a7r6/x:vlp1N{\u$JOFRV=Eds7v,];zY3?TsL_qLYt'wf]
"fsh5h>KE0:"_X5uu@a`f^LC<$uT]cf.pdeBM6R,8t*"jt|ad}+JFZ;sf*#j"6[WL[nN
+h4cHduyN<aQ@kHZ@JezeZ$Le(kTho9h8:VVEsn5Wd&rguo~9fEdM\^%~g_kJ=,Q1MqW04Ts6_
[GQG;&-H*="h<aYWG28(@gFJFC?DRy*I.w.-B%+zSA,P{)WL)&1A^|
A*2X->^#WZ3mN$xi!Yu:_pnCN}~P0NiS)2r&S$_M2_cO?l)/JC.(>6"5t/
zD5HFP'<8b@29y:$>2WU;q>biYem!5R=SL5Icm,:m#A^a3J-w=n[7Bof$&5BatF:S@T5oUPx4d1
+?c{hAOce"A5[yX:2WbC{4R93l`ILE`D:>fW(*m&yY{k|"lqrMvf>\s"R+^|ZYrc",
```

```
Sender  2 , packet no  55  delivered successfully.
Sender  2  sent packet no  56  to channel.
Sender  2 , packet no  56  delivered successfully.
Sender  2  sent packet no  57  to channel.
Sender  2 , packet no  57  delivered successfully.
Sender  2  sent packet no  58  to channel.
Sender  2 , packet no  58  delivered successfully.
Sender  2  sent packet no  59  to channel.
Sender  2 , packet no  59  delivered successfully.
Sender  2  sent packet no  60  to channel.
Sender  2 , packet no  60  delivered successfully.
Sender  2  sent packet no  61  to channel.
Sender  2 , packet no  61  delivered successfully.
Sender  2  sent packet no  62  to channel.
Sender  2 , packet no  62  delivered successfully.
Sender  2  sent packet no  63  to channel.
Sender  2 , packet no  63  delivered successfully.
Sender  2  sent packet no  64  to channel.
Sender  2 , packet no  64  delivered successfully.
kdjonty@KDJonty-Ubuntu-20:~/CSE/Sem5/Networking/NetworkLab
/Asgn3$ 
```

```
kdjonty@KDJonty-Ubuntu-20:~/CSE/Sem5/Networking/NetworkLab/
Asgn3$ python3 Channel.py
Server started
('127.0.0.1', 51032)  joined
('127.0.0.1', 51034)  joined
('127.0.0.1', 51036)  joined
('127.0.0.1', 51038)  joined
```

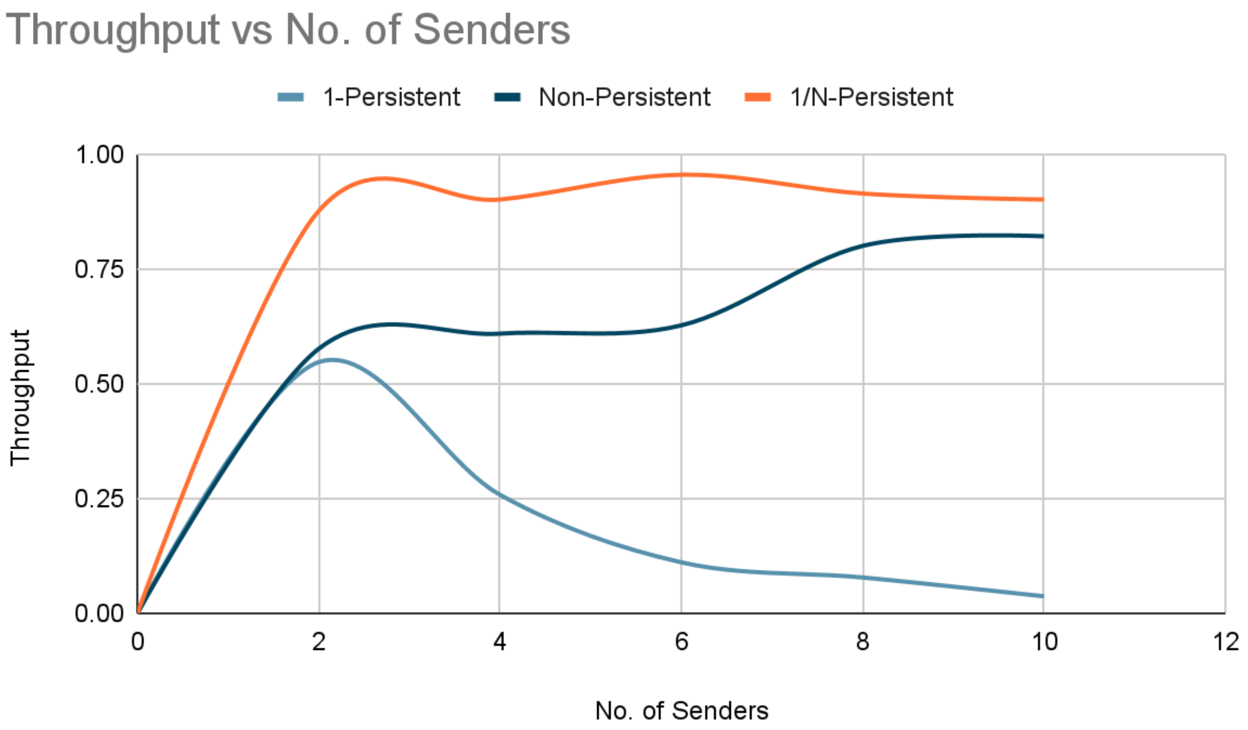
```
Receiver  1  received packet successfully
Receiver  2  received packet successfully
Receiver  1  received packet successfully
Receiver  1  received packet successfully
Receiver  2  received packet successfully
Receiver  1  received packet successfully
Receiver  1  received packet successfully
Receiver  2  received packet successfully
Receiver  2  received packet successfully
Receiver  2  received packet successfully
Receiver  2  received packet successfully
Receiver  2  received packet successfully
Receiver  2  received packet successfully
Receiver  2  received packet successfully
Receiver  2  received packet successfully
Receiver  2  received packet successfully
Receiver  2  received packet successfully
Receiver  2  received packet successfully
Receiver  2  received packet successfully
kdjonty@KDJonty-Ubuntu-20:~/CSE/Sem5/Networking/NetworkLab
/Asgn3$ 
```

Results:

We have used no. of collisions, throughput, and delay per packet as the parameters to compare the 3 persistent methods, i.e. 1-Persistent, Non-Persistent, and P-Persistent with P=1/N, where N=no. of senders.

1-Persistent

No of senders	Total packets sent	Effective packets sent	No. of collisions	Total time (in min)	Throughput	Delay per packet (in sec)
2	117	64	53	0.337	0.547	0.316
4	248	64	184	0.929	0.258	0.871
6	575	64	511	2.444	0.111	2.292
8	812	64	748	3.804	0.078	3.566
10	1692	64	1628	7.870	0.037	7.378

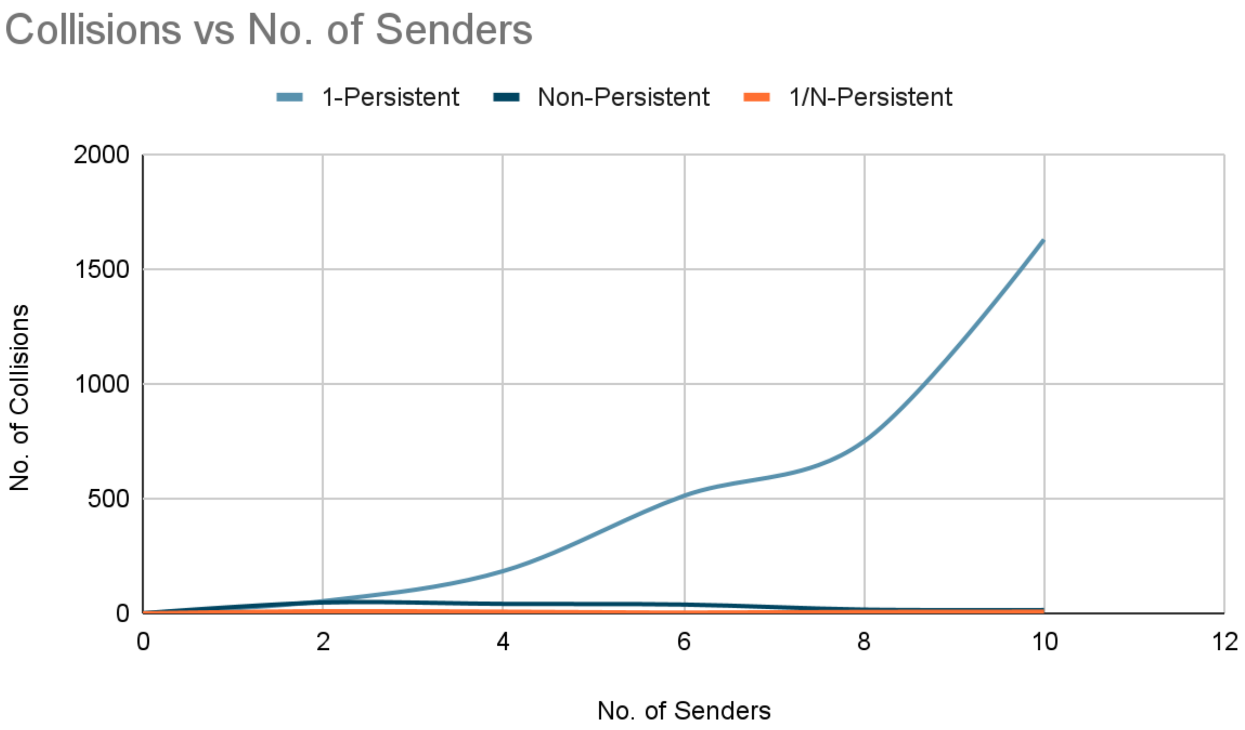


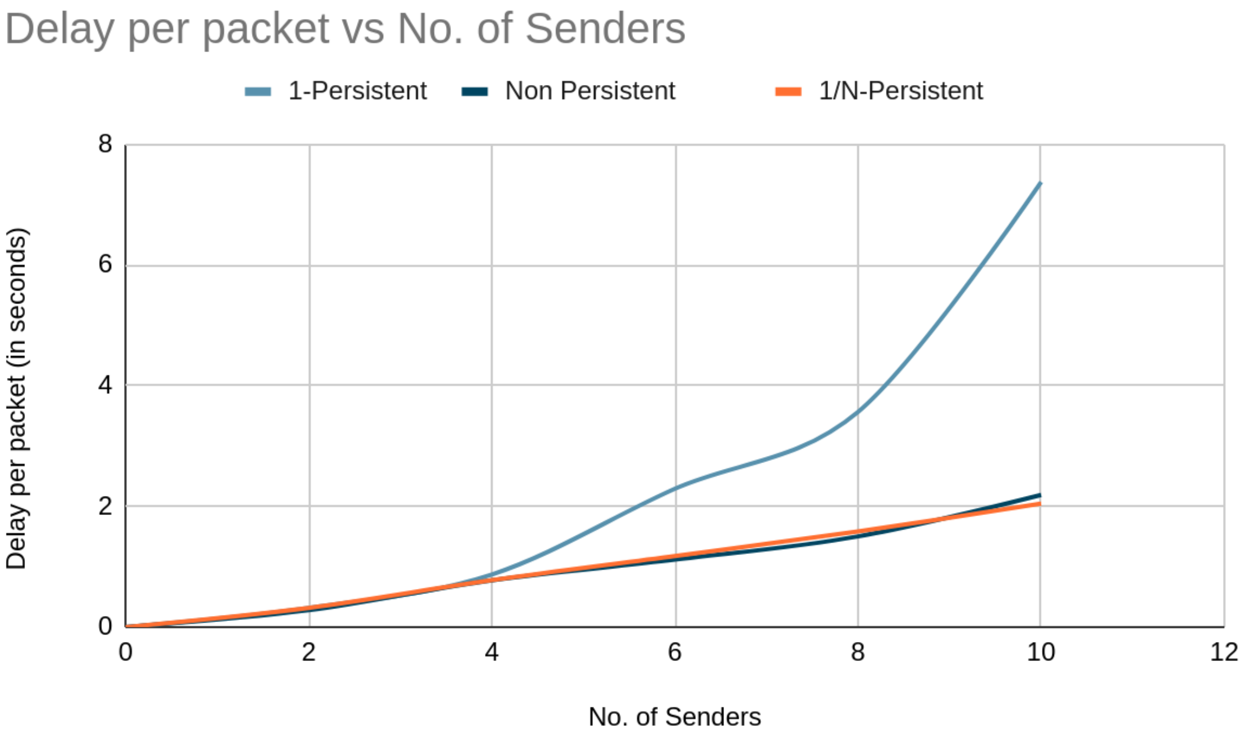
Non-Persistent

No of senders	Total packets sent	Effective packets sent	No. of collisions	Total time (in min)	Throughput	Delay per packet (in sec)
2	111	64	47	0.297	0.576	0.279
4	105	64	41	0.826	0.609	0.774
6	102	64	38	1.191	0.627	1.116
8	80	64	16	1.602	0.800	1.502
10	78	64	14	2.189	0.821	2.189

P-Persistent(1/N)

No of senders	Total packets sent	Effective packets sent	No. of collisions	Total time (in min)	Throughput	Delay per packet (in sec)
2	73	64	9	0.338	0.876	0.317
4	71	64	7	0.829	0.901	0.777
6	67	64	3	1.256	0.955	1.178
8	70	64	6	1.690	0.914	1.585
10	71	64	7	2.181	0.901	2.045


Collisions vs No. of Senders
1-Persistent
Non-Persistent
1/N-Persistent
No. of Collisions
2000
1500
1000
500
0
0
2
4
6
8
10
12
No. of Senders


Delay per packet vs No. of Senders
1-Persistent
Non Persistent
1/N-Persistent
Delay per packet (in seconds)
8
6
4
2
0
0
2
4
6
8
10
12
No. of Senders

Analysis:

- **Collision:**
  - In the 1-Persistent method, a frame is transmitted immediately after it senses the channel idle, it has the maximum chances of collision. When the number of senders increases, no. of collisions increases exponentially.
  - In the Non-Persistent method, it waits for a random time when the channel is found to be busy. An average number of collisions remains almost the same with a slight increase with an increasing no. of senders, since the random waiting range also increases, hence reducing the chances of sensing the idle channel simultaneously.
  - In the P-Persistent method, whenever it senses an idle channel, it generates a random value which must be less than p(1/no. of senders) to transmit the frame, else waits for a time, and tries again. It is unlikely for different senders to get in the same slot, which reduces the collision probability. Average number of collisions remains almost same with a slight increase with increasing number of senders as value of p decreases too.
- **Throughput:**
  - In the 1-Persistent method, since no. of collisions increases with an increase in no. of senders, throughput decreases.
  - In the Non-Persistent method, throughput increases slowly up to a certain point and then saturates.
  - The P-Persistent method provides the best throughput. Throughput is greater than the other two methods and remains almost saturated at all times.
- **Average Delay per Packet:**
  - In the 1-Persistent method, since no. of collisions increases exponentially with an increase in no. of senders, delay per packet also increases exponentially.
  - In the Non-Persistent method, with increasing no. of senders, delay per packet also increases linearly.
  - In the P-Persistent method too, delay per packet increases linearly with the increase in no. of senders.

Among all of the methods, the P-Persistent method with probability = 1/N, where N=no. of senders is the most efficient.

Improvements:

- ➢ Since the receiver sends an acknowledgment, which is also a form of the data packet, and the receiver is also a station, this assignment can be extended further such that, both the sender and the receiver follow the persistent methods.
- ➢ This would have been more efficient if it was implemented in a language closer to the system such as C/C++.