Networks Lab Report
Assignment 3

Md Sahil
BCSE III
Roll-001710501029

1 Objective

To implement 1-persistent, non-persistent and p-persistent CSMA techniques.

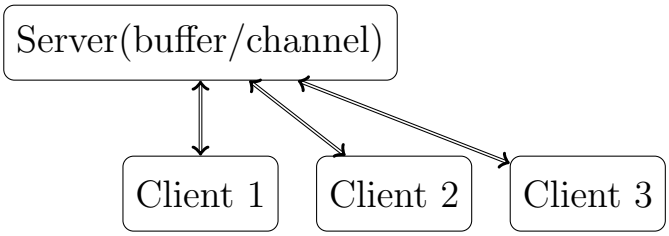
2 Design and Implementation

2.1 Program structure

The implementation is done using sockets. The clients and server communicate with each other using sockets. Listening on the channel is done through a separate thread (for both client and server). There can be multiple client instances. All the clients are connected to the central server. The central server holds the channel(a buffer).

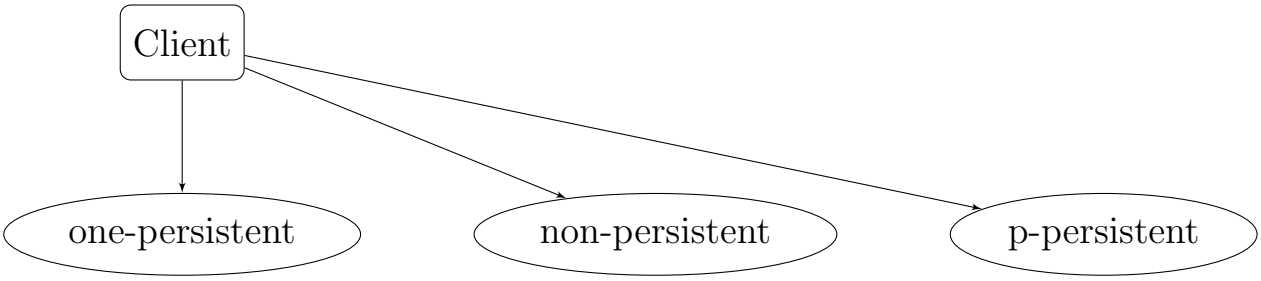
2.1.1 The Server class

The server class instance acts as a medium to connect two clients. Whenever the server is starts listening and accepting client socket connections. Each time a client connection is made, its control is passed on to the client handler thread. The client handler thread then listens for incoming frames from the assigned client socket. The server also maintains a list of mappings of clients with port numbers and client addresses. When ever a client tries to connect to a server. The server maps the port number with the client address. When a client tries to sent messages to another client, the server checks the destination address of the message, finds the port mapped to the address and forwards the message to the destination client. The server holds the incoming messages in a buffer, which acts as the channel here. When ever a new packet arrives and the buffer is non empty, a collision is registered.



2.1.2 The Client class

The instances of Client class acts as stations. It has 3 functions *onePersistent()*, *nonPersistent()* and *pPersistent()*. Each function implements its corresponding CSMA technique.



3 Code snippets

3.1 Server

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.*;
import java.time.Duration;
import java.time.Instant;

/**Frame format:
 * 1 byte premble
 * 1 bytes destination address
 * 1 bytes source address
 * 1 - 10 bytes of data
 *
 *
 * If the premble is set to 00000000 then it is for dhcpLite request
 * If the premble is set to 00000001 then it is for dhcpLite granted
 * If the premble is set to 00000010 then it is for dhcpLite rejected
 * If the premble is set to 10000000 then it is for data transfer
 */

public class Server {

  public static final String DHCPLITE_REQUEST     = "00000000";
  public static final String DHCPLITE_GRANTED     = "00000001";
  public static final String DHCPLITE_REJECTED    = "00000010";
  public static final String DATA_TRANSFER      = "10000000";
  public static final String BUFFER_STATUS_MESSAGE  = "11100000";
  public static final String BROADCAST_ADDRESS    = "11111111";
  public static final double ERROR_P          = 0.3;

  private static Map<String, ClientHandler> dns = new HashMap<String, ClientHandler>();
  protected static String buffer;
  protected static Timer timer;

  private static class ChannelBroadcaster implements Runnable {

        protected String makeSequenceString (int n) {
        String sq = Integer.toBinaryString(n);
        if(sq.length() > 8)
        sq = sq.substring(sq.length() - 8);
      else if(sq.length() < 8)
        while(sq.length()!=8)
          sq = "0" + sq;
      return sq;
    }


    public void run() {
      while(true) {
        System.out.print("");
        int size = dns.size();
        if(size!=0) {
          for(ClientHandler client : dns.values()) {
            String msg = BUFFER_STATUS_MESSAGE;
            msg += makeSequenceString(buffer.length());
            client.out.println(msg);
          }
        }
      }
```

```
    }
  }

  //---------------------------------------

  public static class Timer extends Thread{
    private boolean running;
    private final static int TIME = 1000;
    private int time;

    public Timer() {
      running = false;
      time = TIME;
    }

    public boolean isRunning() {return running;}

    public  void startTimer() {
      if(running == false) {
        running = true;
        time = TIME;
        resume();
      } else {
        time = TIME;
      }
    }

    public  void stopTimer() {
      try {
        time = TIME;
        running = false;
        suspend();
      } catch (Exception e) {
        e.printStackTrace();
        System.exit(0);
      }
    }

    public void run() {
      try {
        suspend();
        while(true) {
          while((time--)!=0) {
            sleep(1);
          }
          timeout();
        }
      } catch (Exception e) {
        e.printStackTrace();
        System.exit(0);
      }
    }

    public void timeout() {
      buffer = new String("");
    }
  }
  //---------------------------------------


  private static class ClientHandler implements Runnable {
    private Socket soc;
    public PrintWriter out;
    public BufferedReader in;
```

```
124     public Instant start,finish;
125     public int rcvd;
126     public String mac_addr;
127 
128     public ClientHandler(Socket s) {
129       soc = s;
130       rcvd = 0;
131       try {
132         out = new PrintWriter(soc.getOutputStream(), true);
133         in = new BufferedReader(
134           new InputStreamReader(
135             soc.getInputStream()
136             )
137           );
138       } catch (IOException e) {
139         e.printStackTrace();
140         System.exit(0);
141       }
142     }
143 
144     public void dataTranser(String msg) {
145       /**Function for data tranfer between clients
146        * void dataTranser(String message_to_be_send)
147        * */
148       String destination = msg.substring(8,16);
149       String source = msg.substring(16,24);
150       if(destination.equals(BROADCAST_ADDRESS)) {
151         int count = Integer.parseInt(msg.substring(24,32),2);
152         if(!buffer.equals("")) {
153           System.out.println("Broadcast message received from "
154               + source
155               + " count:"
156               + count
157               + " Collision occurred!");
158         } else {
159           System.out.println("Broadcast message received from "
160               + source
161               + " count:"
162               + count);
163           rcvd++;
164           //System.out.println("Buffer:"+buffer);
165         }
166 
167         buffer = msg;
168 
169         if(count == 0)
170           start = Instant.now();
171         else if(count == 99) {
172           finish = Instant.now();
173           long timeElapsed = Duration.between(start, finish).toMillis();
174           System.out.println("Time elapsed(in ms) of:" + mac_addr + " = " + timeElapsed);
175           System.out.println("Throughput of:" + mac_addr + " = " + (100000/(float)timeElapsed));
176           System.out.println("Efficienty of:" + mac_addr + " = " + (rcvd) + "%");
177         }
178 
179         try {
180           timer.startTimer();
181         } catch (Exception e) {
182           e.printStackTrace();
183           System.exit(0);
184         }
185 
186       } else {
187       double p = Math.random();
```

```
188         if(p>ERROR_P) {
189           dns.get(destination).out.println(msg);
190           System.out.println("Message passed from:"
191               + source
192               + " to:"
193               + destination);
194         } else {
195           System.out.println("Error while passing message from:"
196               + source
197               + " to:"
198               + destination);
199         }
200       }
201     }
202
203     public void dhcpLite(String msg) {
204       /**Function mac address registration on server
205        * void dhcpLite(String dhcp_message_received_from_server)
206        * */
207       try {
208         String mac_addr = msg.substring(8,16);
209         if(dns.containsKey(mac_addr)) {
210           out.println(DHCPLITE_REJECTED);
211           soc.close();
212         } else {
213           dns.put(mac_addr,this);
214           this.mac_addr = mac_addr;
215           out.println(DHCPLITE_GRANTED);
216           System.out.println("Connection established with:"
217               + mac_addr
218               + " at socket:"
219               + soc.getRemoteSocketAddress().toString());
220         }
221       } catch (StringIndexOutOfBoundsException e) {
222         System.out.println("Socket:"
223             + soc.getRemoteSocketAddress()
224             + " Requested an invalid mac_address registration");
225         out.println(DHCPLITE_REJECTED);
226       } catch (Exception e) {
227         e.printStackTrace();
228         System.exit(0);
229       }
230     }
231
232     public void run() {
233       System.out.println("Attempting to connect:"
234           + soc.getRemoteSocketAddress().toString());
235
236       /** starts listeining to client indefinitely */
237       try {
238         while(!soc.isClosed()) {
239           if(in.ready()) {
240             String msg = in.readLine();
241             //System.out.println("Received:" + msg);
242             if(msg.substring(0,8).equals(DHCPLITE_REQUEST))
243               dhcpLite(msg);
244             else if(msg.substring(0,8).equals(DATA_TRANSFER)) {
245               //System.out.println("Data:" + msg);
246               dataTranser(msg);
247             }
248             else
249               System.out.println("Unknown premble:" + msg.substring(0,8));
250           }
251         }
```

```
      } catch (IOException e) {
        e.printStackTrace();
        System.exit(0);
      }
    }
  }

  public static int PORT = 8888;
  public static void run() {

    buffer = new String("");
    ChannelBroadcaster cb = new ChannelBroadcaster();
    Thread cdThread = new Thread(cb);
    cdThread.start();
    timer = new Timer();
    timer.start();

    try {
      ServerSocket serversocket = new ServerSocket(PORT);
      System.out.println("Server Started!");
      while(true) {
        Socket soc = serversocket.accept();
        new Thread(new ClientHandler(soc)).start();
      }
      //serversocket.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }


  public static void main(String[] args) {
    run();
  }
}
```

3.2 Client

```
import java.util.Random;

public class Client {
  public static class ClientWrapper extends ClientClass {
    public ClientWrapper() {
      super();
    }

    protected void receiveMsg(String msg) {
      //do nothing for now
    }

    protected void bufferUpdate(String buffer) {
    int newbufferStatus = Integer.parseInt(buffer,2);
    if(newbufferStatus != bufferStatus) {
        bufferStatus = newbufferStatus;
      }
    }

        protected String makeSequenceString (int n) {
        String sq = Integer.toBinaryString(n);
        if(sq.length() > 8)
        sq = sq.substring(sq.length() - 8);
      else if(sq.length() < 8)
```

```
25          while(sq.length()!=8)
26            sq = "0" + sq;
27        return sq;
28      }
29
30      protected void onePersitent() {
31        int count = 0;
32        while(true) {
33          if(bufferStatus == 0) {
34            try {
35              String msg = makeSequenceString(count);
36              System.out.println("Sending broadcast message:" + count);
37              sendMsg(msg,BROADCAST_ADDRESS);
38              count++;
39              bufferStatus = 100;
40              Thread.sleep(1000);
41            } catch (Exception e) {
42              e.printStackTrace();
43              System.exit(0);
44            }
45          } else if(bufferStatus > 0) {
46            System.out.print("");
47          }
48
49          if(count==100)
50            break;
51        }
52      }
53
54
55      protected void nonPersitent() {
56        Random rand = new Random();
57        int count = 0;
58        while(true) {
59          try {
60            System.out.print("");
61            if(bufferStatus == 0) {
62              String msg = makeSequenceString(count);
63              System.out.println("Sending broadcast message:" + count);
64              sendMsg(msg,BROADCAST_ADDRESS);
65              count++;
66              bufferStatus = 100;
67              Thread.sleep(1000);
68            } else {
69              System.out.print("");
70              int n = rand.nextInt(2000);
71              Thread.sleep(n);
72            }
73          } catch (Exception e) {
74            e.printStackTrace();
75            System.exit(0);
76          }
77
78          if(count==100)
79            break;
80        }
81      }
82
83
84      protected void pPersitent() {
85        Random rand = new Random();
86        int count = 0;
87        while(true) {
88          try {
```

```
89            String msg = makeSequenceString(count);
90            System.out.println("Sending broadcast message:" + count);
91            int k = 0;
92            double p = 0.5;
93            int slottime = 1200;
94            while(k<15) {
95              if(bufferStatus == 0) {
96                if(p<Math.random()) {
97                  sendMsg(msg,BROADCAST_ADDRESS);
98                  count++;
99                  //bufferStatus = 100;
100                 Thread.sleep(1000);
101                 break;
102               } else {
103                 Thread.sleep(slottime);
104               }
105             } else if(bufferStatus>0) {
106               System.out.print("");
107               k++;
108               int n = rand.nextInt((int)Math.pow(2.0,k+1));
109               System.out.println("Backoff:" + n);
110               Thread.sleep(n);
111             }
112
113             if(k>15) {
114               System.out.println("Backing off limit reached, dumping packed");
115               count++;
116               break;
117             }
118           }
119         } catch (Exception e) {
120           e.printStackTrace();
121           System.exit(0);
122         }
123
124         if(count==100)
125           break;
126       }
127     }
128
129     public void run(String mac_addr, String algo) {
130       super.run(mac_addr);
131       if(algo.equals("1"))
132         onePersitent();
133       else if(algo.equals("n"))
134         nonPersitent();
135       else if(algo.equals("p"))
136         pPersitent();
137       else {
138         System.out.println("Invalid arguement!");
139         System.exit(0);
140       }
141     }
142
143
144   }
145
146   public static void main(String args[]) {
147     new ClientWrapper().run(args[0], args[1]);
148   }
149 }
```

4 Results

Observations have been taken by sending 100 packets from each station. Here *throughput* is defined as the no of packets transmitted by the sender in unit time. And *efficiency* is the no of packets received correctly (without collision) divided by the total number of messages sent by the sender station.

Table 1: With 2 stations

	one-persistent		non-persistent		p-persistent	
Sender	Throughput	Efficiency	Throughput	Efficiency	Throughput	Efficiency
1	0.9459	0.72	0.2996	1.00	0.2504	1.00
2	0.9458	0.62	0.2710	0.99	0.2465	0.99

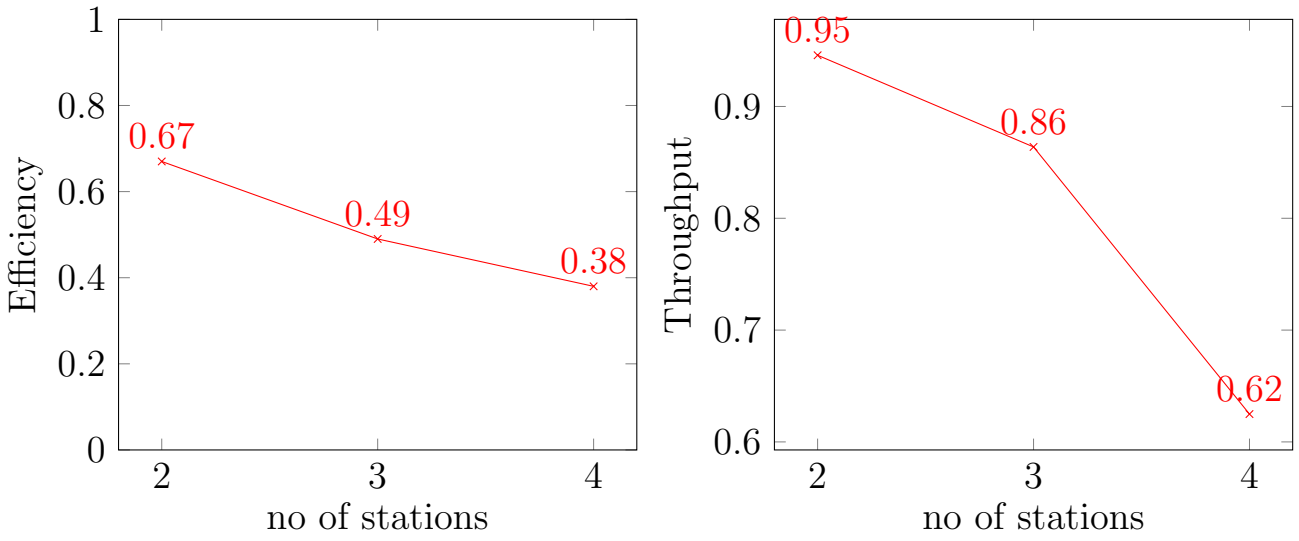
Table 2: With 3 stations

	one-persistent		non-persistent		p-persistent	
Sender	Throughput	Efficiency	Throughput	Efficiency	Throughput	Efficiency
1	0.8748	0.46	0.2658	0.99	0.1804	1.00
2	0.8663	0.49	0.2490	1.00	0.1711	0.99
3	0.8505	0.52	0.2342	0.99	0.1697	0.99

Table 3: With 4 stations

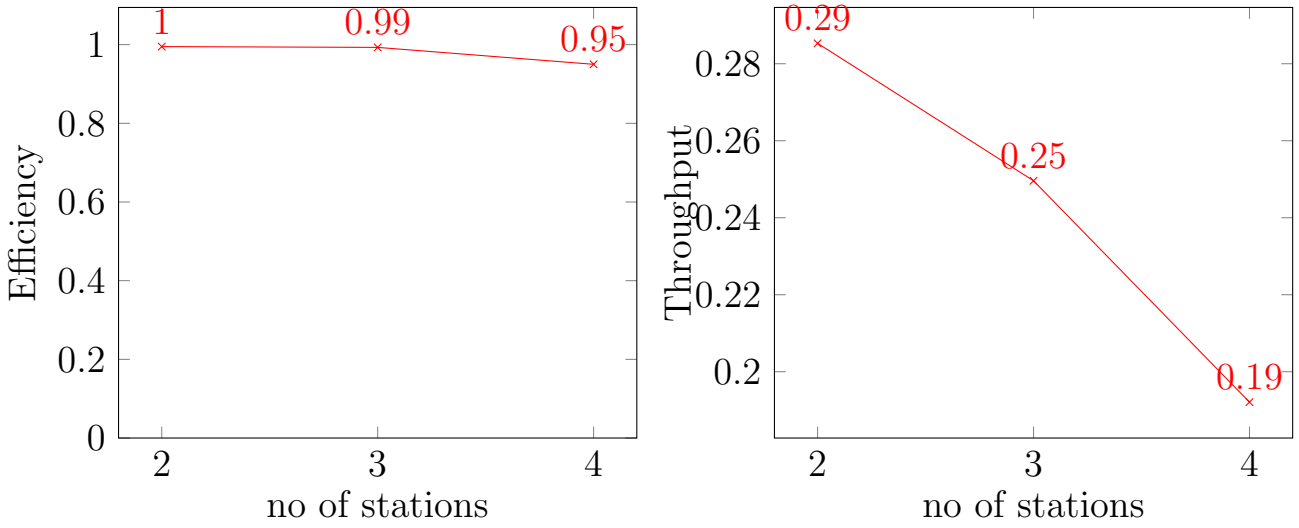
	one-persistent		non-persistent		p-persistent	
Sender	Throughput	Efficiency	Throughput	Efficiency	Throughput	Efficiency
1	0.6447	0.35	0.1980	0.93	0.1841	0.98
2	0.6196	0.34	0.1962	0.97	0.1712	0.99
3	0.6197	0.47	0.1891	0.95	0.1816	0.99
4	0.6157	0.37	0.1852	0.95	0.1555	0.97

4.1 One-persistent



One persistent has the highest throughput among the three schemes. But takes a hit on efficiency, because the sender transmit the package as soon as it senses the channel idle. More than one channel may sense the channel idle at the same time and thus collision occurs. As evident from the efficiency graph, the collision frequency increase with the number of stations.

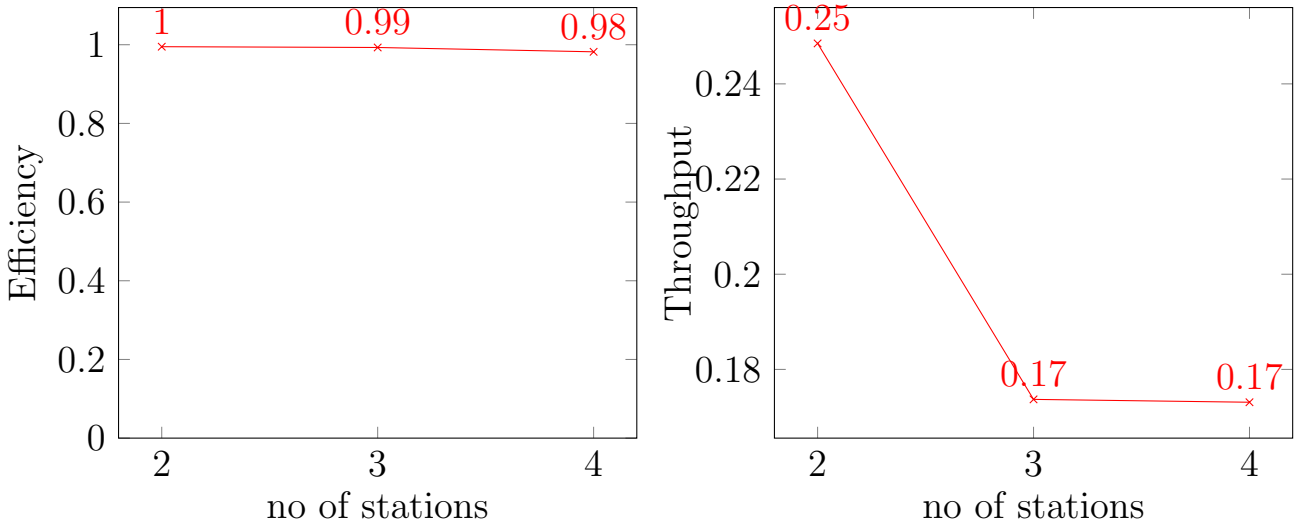
4.2 Non-persistent



Non persistent scheme provides very good efficiency. The collision frequency is very low. But the throughput is worse than one persistent scheme, because when ever the sender senses the channel as busy it waits of a random amount of time. The chances of collision reduces drastically. The throughput also decreases with increase in number of sender stations.

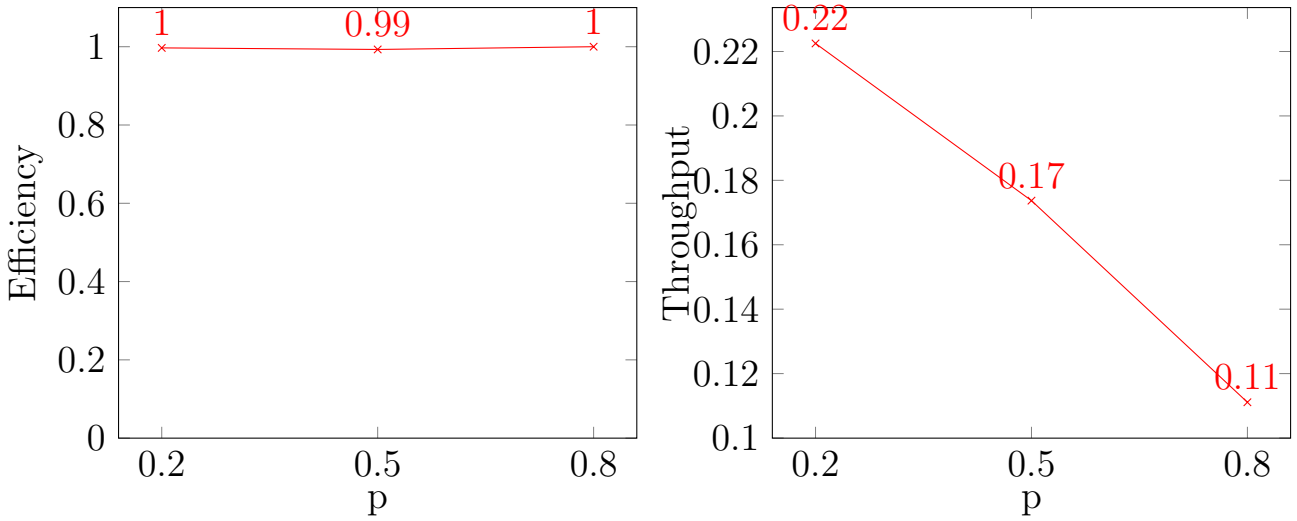
4.3 p-persistent

For The following graphs we keep the value of p at 0.5 and vary the number of stations.



P-persistent scheme also provides very good efficiency but suffers in throughput. If the sender senses the channel as busy it executes its back-off scheme. If the sender senses the channel as idle, it transmits the message with a probability of *1-p*, else it waits for a timeslot and senses the channel again.

For the following graphs we keep the number of stations fixed at 3 and vary the value of p.



The efficiency doesn't vary much with the change in the value of p. The throughput decreases drastically as the value of p is increased. With the increase in the value of p, the probability of transmitting (when the channel is idle) decreases. Thus the throughput decreases.

5 Comments

- The architecture of the code here is identical to the previous assignment.
- I faced some thread scheduling problems. For example in certain infinite while loops, adding a blank print statement changed the execution.