

# Networks Lab Report

## Assignment 3

Md Sahil  
BCSE III  
Roll-001710501029

# 1 Objective

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

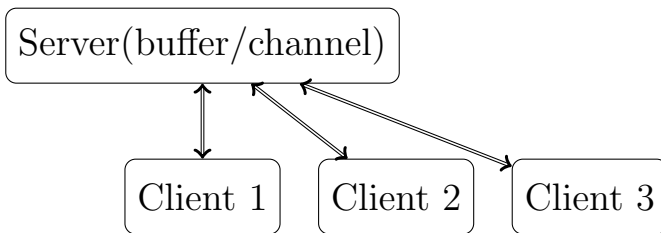
## 2 Design and Implementation

### 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).

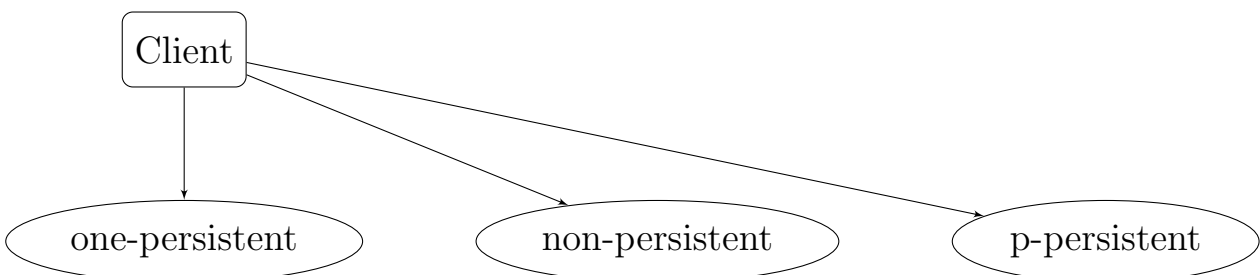
#### 2.1.1 The Server class

The server class instance acts as a medium to connect two clients. Whenever the server 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. Whenever a client tries to connect to a server. The server maps the port number with the client address. When a client tries to send 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. Whenever 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

```
1 import java.io.*;
2 import java.net.ServerSocket;
3 import java.net.Socket;
4 import java.util.*;
5 import java.time.Duration;
6 import java.time.Instant;
7
8 /**Frame format:
9  * 1 byte preamble
10  * 1 bytes destination address
11  * 1 bytes source address
12  * 1 - 10 bytes of data
13  *
14  *
15  * If the preamble is set to 00000000 then it is for dhcpLite request
16  * If the preamble is set to 00000001 then it is for dhcpLite granted
17  * If the preamble is set to 00000010 then it is for dhcpLite rejected
18  * If the preamble is set to 10000000 then it is for data transfer
19  */
20
21 public class Server {
22
23     public static final String DHCPLITE_REQUEST      = "00000000";
24     public static final String DHCPLITE_GRANTED      = "00000001";
25     public static final String DHCPLITE_REJECTED     = "00000010";
26     public static final String DATA_TRANSFER        = "10000000";
27     public static final String BUFFER_STATUS_MESSAGE = "11100000";
28     public static final String BROADCAST_ADDRESS     = "11111111";
29     public static final double ERROR_P              = 0.3;
30
31     private static Map<String, ClientHandler> dns = new HashMap<String, ClientHandler>();
32     protected static String buffer;
33     protected static Timer timer;
34
35     private static class ChannelBroadcaster implements Runnable {
36
37         protected String makeSequenceString (int n) {
38             String sq = Integer.toBinaryString(n);
39             if(sq.length() > 8)
40                 sq = sq.substring(sq.length() - 8);
41             else if(sq.length() < 8)
42                 while(sq.length() != 8)
43                     sq = "0" + sq;
44             return sq;
45         }
46
47
48         public void run() {
49             while(true) {
50                 System.out.print("");
51                 int size = dns.size();
52                 if(size != 0) {
53                     for(ClientHandler client : dns.values()) {
54                         String msg = BUFFER_STATUS_MESSAGE;
55                         msg += makeSequenceString(buffer.length());
56                         client.out.println(msg);
57                     }
58                 }
59             }
60         }
61     }
62 }
```

```

60     }
61 }
62
63 //-----
64
65 public static class Timer extends Thread{
66     private boolean running;
67     private final static int TIME = 1000;
68     private int time;
69
70     public Timer() {
71         running = false;
72         time = TIME;
73     }
74
75     public boolean isRunning() {return running;}
76
77     public void startTimer() {
78         if(running == false) {
79             running = true;
80             time = TIME;
81             resume();
82         } else {
83             time = TIME;
84         }
85     }
86
87     public void stopTimer() {
88         try {
89             time = TIME;
90             running = false;
91             suspend();
92         } catch (Exception e) {
93             e.printStackTrace();
94             System.exit(0);
95         }
96     }
97
98     public void run() {
99         try {
100             suspend();
101             while(true) {
102                 while((time--)!=0) {
103                     sleep(1);
104                 }
105                 timeout();
106             }
107         } catch (Exception e) {
108             e.printStackTrace();
109             System.exit(0);
110         }
111     }
112
113     public void timeout() {
114         buffer = new String("");
115     }
116 }
117 //-----
118
119
120 private static class ClientHandler implements Runnable {
121     private Socket soc;
122     public PrintWriter out;
123     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 listeneing 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 preamble:" + msg.substring(0,8));
250             }
251         }

```

```

252     } catch (IOException e) {
253         e.printStackTrace();
254         System.exit(0);
255     }
256 }
257 }
258
259 public static int PORT = 8888;
260 public static void run() {
261
262     buffer = new String("");
263     ChannelBroadcaster cb = new ChannelBroadcaster();
264     Thread cdThread = new Thread(cb);
265     cdThread.start();
266     timer = new Timer();
267     timer.start();
268
269     try {
270         ServerSocket serversocket = new ServerSocket(PORT);
271         System.out.println("Server Started!");
272         while(true) {
273             Socket soc = serversocket.accept();
274             new Thread(new ClientHandler(soc)).start();
275         }
276         //serversocket.close();
277     } catch (IOException e) {
278         e.printStackTrace();
279     }
280 }
281
282
283 public static void main(String[] args) {
284     run();
285 }
286 }

```

## 3.2 Client

```

1  import java.util.Random;
2
3  public class Client {
4      public static class ClientWrapper extends ClientClass {
5          public ClientWrapper() {
6              super();
7          }
8
9          protected void receiveMsg(String msg) {
10             //do nothing for now
11         }
12
13         protected void bufferUpdate(String buffer) {
14             int newbufferStatus = Integer.parseInt(buffer,2);
15             if(newbufferStatus != bufferStatus) {
16                 bufferStatus = newbufferStatus;
17             }
18         }
19
20         protected String makeSequenceString (int n) {
21             String sq = Integer.toBinaryString(n);
22             if(sq.length() > 8)
23                 sq = sq.substring(sq.length() - 8);
24             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 {

```



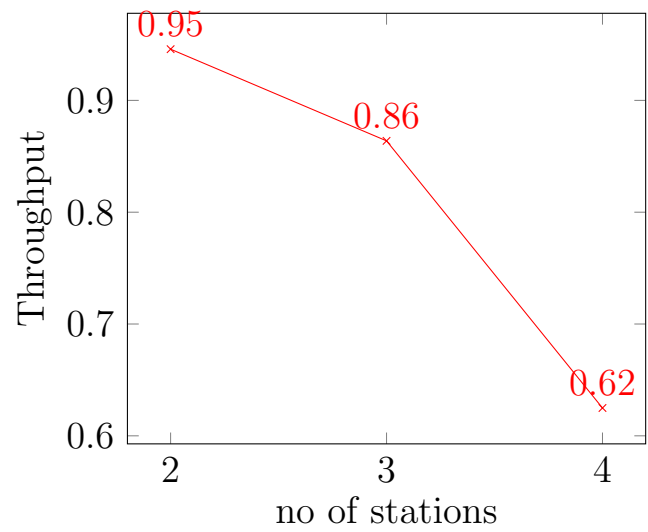
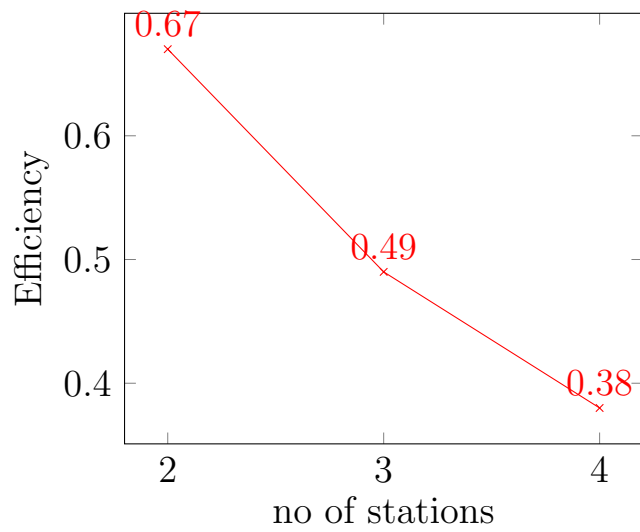
```

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 argument!");
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 }

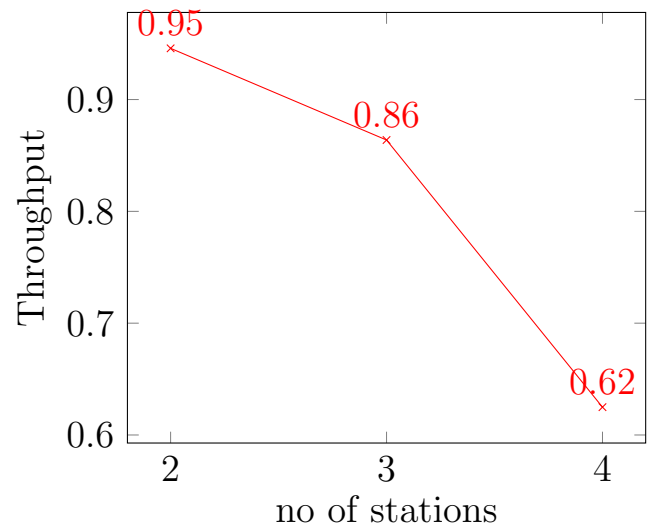
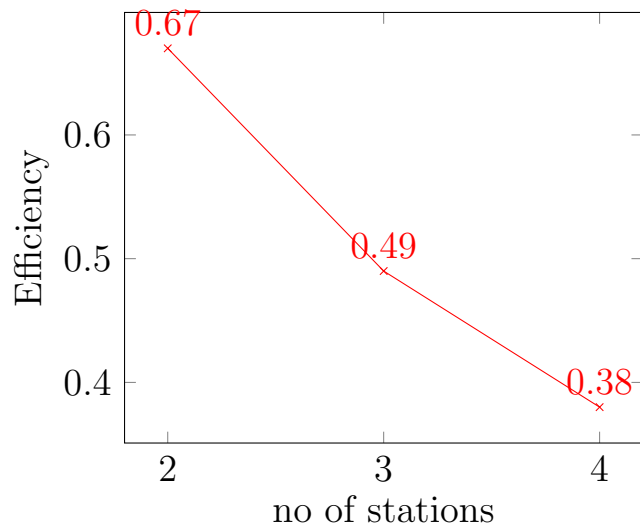
```

## 4 Results

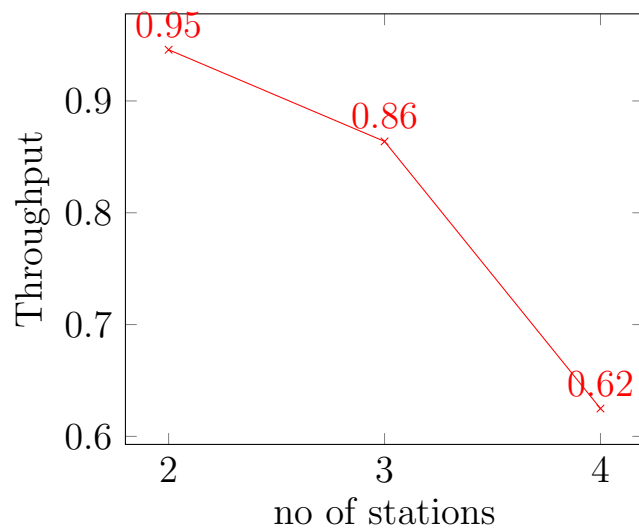
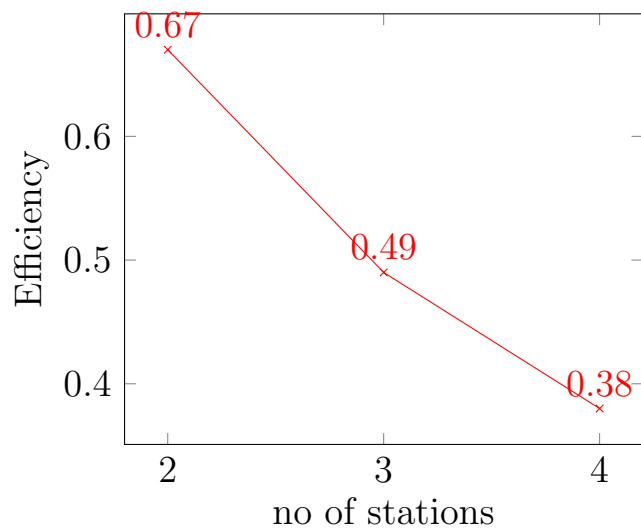
### 4.1 One-persisent



### 4.2 Non-persisent



### 4.3 p-persistent



## 5 Comments