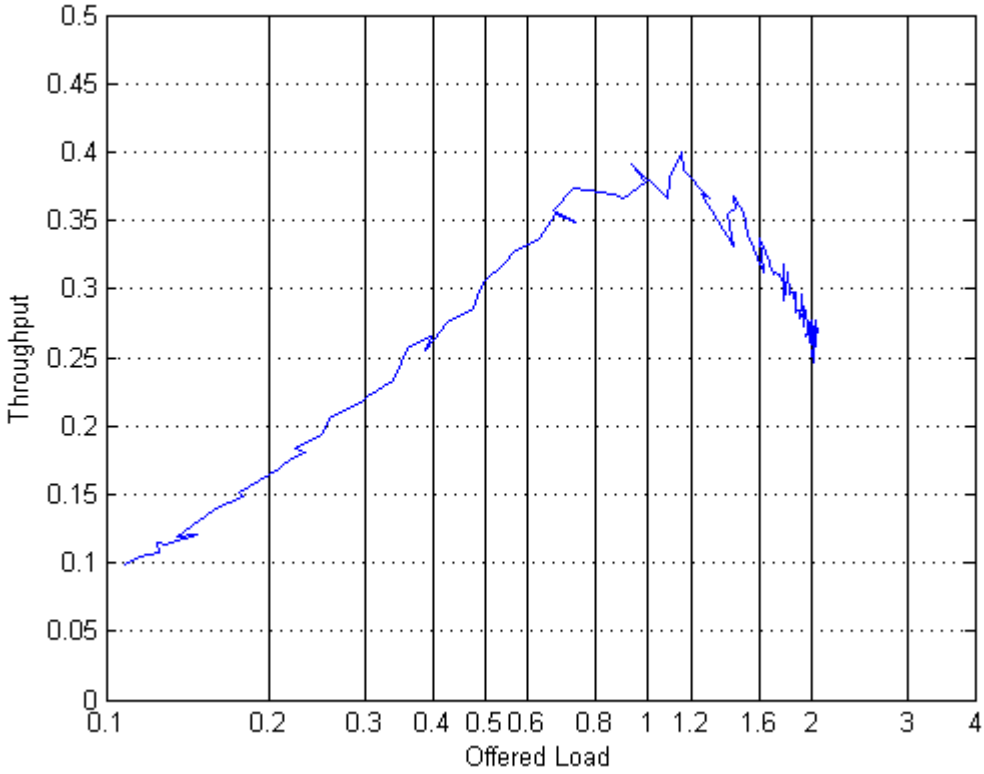
ELEC6851 Project:
Simulation of Slotted ALOHA

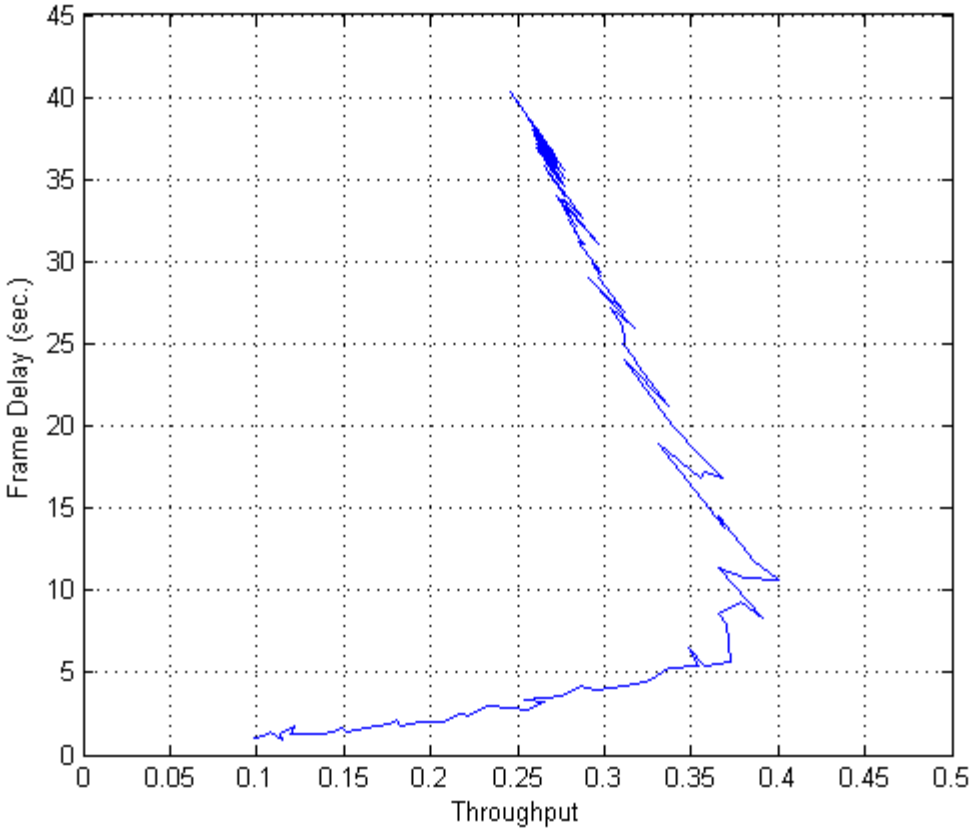
By

Masood Khosroshahy

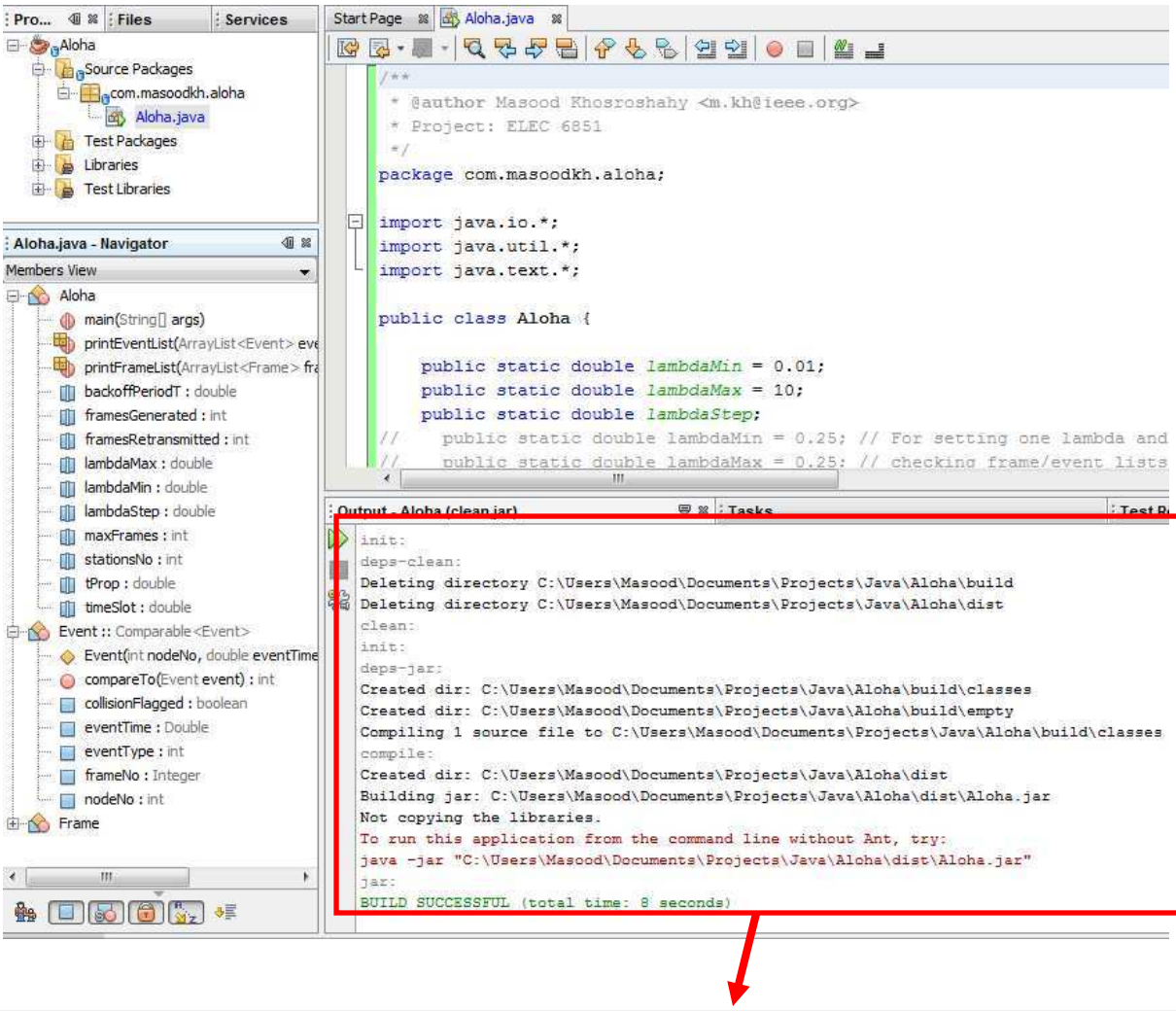
Professor: Dr. Dongyu Qiu

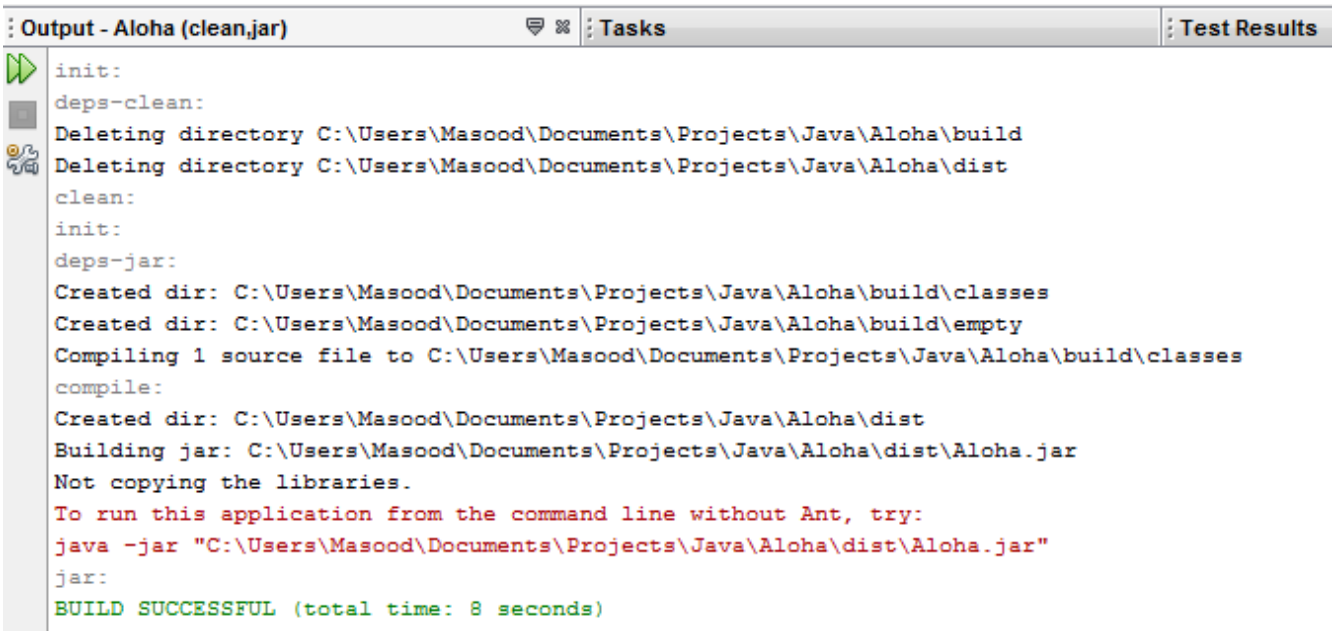
November 27, 2009


Throughput
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0.1
0.2
0.3
0.4
0.5
0.6
0.8
1
1.2
1.6
2
3
4
Offered Load


Frame Delay (sec.)
45
40
35
30
25
20
15
10
5
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Throughput

Compilation in NetBeans IDE for Java





Event List Sample

Lambda=0.25

'Node/List No'	'Event Time'	'Event Type'	'Frame No'	'Collision Flagged'
6	699.66	3	193	false
10	700.00	2	200	true
8	700.00	2	166	true
2	701.77	3	187	false
3	702.48	3	190	false
9	703.17	3	195	false
4	703.32	3	174	false
7	704.60	3	197	false
1	705.30	3	176	false
5	707.99	1	199	false
6	700.00	2	193	true
10	700.00	2	200	true
8	700.00	2	166	true
2	701.77	3	187	false
3	702.48	3	190	false
9	703.17	3	195	false
4	703.32	3	174	false
7	704.60	3	197	false
1	705.30	3	176	false
5	707.99	1	199	false
10	700.00	2	200	true
8	700.00	2	166	true
2	701.77	3	187	false
3	702.48	3	190	false
9	703.17	3	195	false
4	703.32	3	174	false
6	704.31	3	193	false
7	704.60	3	197	false
1	705.30	3	176	false
5	707.99	1	199	false
8	700.00	2	166	true
2	701.77	3	187	false
3	702.48	3	190	false
9	703.17	3	195	false
4	703.32	3	174	false
6	704.31	3	193	false
7	704.60	3	197	false
1	705.30	3	176	false
10	707.08	3	200	false
5	707.99	1	199	false

Event List Sample (continued)

Lambda=0.25

'Node/List No'	'Event Time'	'Event Type'	'Frame No'	'Collision Flagged'
1	3410.80	3	994	false
8	3411.00	2	997	false
7	3412.30	1	1000	false
2	3413.74	3	993	false
6	3414.60	3	995	false
4	3414.73	3	999	false
9	3415.90	3	980	false
3	3418.24	3	996	false
5	3419.76	3	998	false
1	3411.00	2	994	true
8	3411.00	2	997	true
7	3412.30	1	1000	false
2	3413.74	3	993	false
6	3414.60	3	995	false
4	3414.73	3	999	false
9	3415.90	3	980	false
3	3418.24	3	996	false
5	3419.76	3	998	false
8	3411.00	2	997	true
1	3412.20	3	994	false
7	3412.30	1	1000	false
2	3413.74	3	993	false
6	3414.60	3	995	false
4	3414.73	3	999	false
9	3415.90	3	980	false
3	3418.24	3	996	false
5	3419.76	3	998	false
8	3411.32	3	997	false
1	3412.20	3	994	false
7	3412.30	1	1000	false
2	3413.74	3	993	false
6	3414.60	3	995	false
4	3414.73	3	999	false
9	3415.90	3	980	false
3	3418.24	3	996	false
5	3419.76	3	998	false

Frame List Sample

Lambda=0.25

'Frame No'	'Node No'	'Arrival-Time'	'Departure-Time'
1	1	3.31	19.0
2	2	10.64	74.0
3	3	16.16	17.0
4	4	0.15	94.0
5	5	1.83	2.0
6	6	3.14	5.0
7	7	0.67	22.0
8	8	3.06	13.0
9	9	2.86	3.0
10	10	6.45	51.0
11	5	9.79	37.0
12	9	5.23	12.0
13	6	5.27	42.0
14	9	14.95	35.0
15	8	13.45	43.0
16	3	17.11	84.0
17	1	39.40	56.0
18	7	24.90	148.0
19	9	35.99	63.0
20	5	37.72	68.0
200	3	640.13	656.0
201	10	645.47	713.0
202	5	646.91	657.0
203	8	651.55	691.0
204	3	661.88	697.0
205	5	660.25	661.0
206	4	659.86	725.0
207	5	667.57	771.0
208	2	668.51	733.0
209	7	680.47	693.0
210	8	701.62	765.0
211	7	698.33	718.0
212	3	700.65	743.0
213	10	715.11	719.0
214	7	723.01	749.0
987	5	3461.31	3473.0
988	1	3460.11	3520.0
989	7	3470.10	3519.0
990	6	3476.37	3494.0
991	9	3479.41	3515.0
992	5	3473.94	3475.0
993	5	3475.71	3504.0
994	3	3478.15	3521.0
995	10	3492.44	3502.0
996	4	3491.34	3501.0
997	6	3494.75	3497.0
998	6	3498.74	3508.0
999	4	3504.54	3512.0
1000	10	3502.67	3522.0

Throughput vs Offered load

' Lambda'	' OfferedLoad'	' Throughput'
0.01	0.10822213500784929	0.09811616954474098
0.019799315994393975	0.22358318098720292	0.18281535648994515
0.025269501953756386	0.3397017707362535	0.23299161230195714
0.032250999437137	0.4753722794959908	0.286368843069874
0.052533479691354745	0.8631656804733728	0.3698224852071006
0.06704751154404426	1.0910823170731707	0.38109756097560976
0.08557150279516956	1.441498176997017	0.3314550878355983
0.10921333129289214	1.5278532608695652	0.33967391304347827
0.13938696110832252	1.6705844980940279	0.3176620076238882
0.2059380244827054	1.8025577043044292	0.3119151590767311
0.30426425535513835	1.8955903271692744	0.2844950213371266
0.4077432021994327	1.9213883120735686	0.29664787896766537
0.5203951312018454	1.9655742219774166	0.27540622418066646
0.6024224137575362	1.9859349145063432	0.27578599007170435
0.7322482090623714	1.9862561847168774	0.2748763056624519
0.8073036504912645	1.9986945169712793	0.26109660574412535
0.93455488839995	1.9802702702702704	0.2702702702702703
1.030346764460945	1.9809447128287707	0.2683843263553409
2.040016117336367	2.005420054200542	0.27100271002710025
3.0140329179760426	2.044305517803751	0.2718129926610492
4.039092374079943	2.017312661498708	0.25839793281653745
5.155019126272584	2.0068965517241377	0.26525198938992045
6.265957966853865	2.0221592373099715	0.2576655501159495
7.253629591379205	2.0144772117962466	0.2680965147453083
8.396982955720352	2.032051282051282	0.2670940170940171
9.257673708681688	2.0231444533120513	0.26602819898909286

Average Frame Delay vs Throughput

' Lambda'	' Throughput'	'Average Frame Delay'
0.01	0.09811616954474098	1.034980854741414
0.01340095640625	0.11950286806883365	1.2506026535089414
0.017103393581163136	0.16929067208396817	1.7425762481740537
0.021828745883819358	0.20644095788604458	1.9905729347486216
0.027859625904016416	0.26602819898909286	3.2196210968688805
0.03555672687944354	0.3048780487804878	4.0338136172237045
0.04538039493908195	0.34891835310537334	6.510244084990362
0.057918161359718605	0.37821482602118	9.267339712161311
0.07391988147730878	0.3865481252415926	11.706263100260342
0.12040769775041359	0.33692722371967654	21.18092856853515
0.1536741246219256	0.3117206982543641	25.254168274243554
0.19613145188829087	0.29850746268656714	28.18245388818752
0.2503189558713814	0.2975304968759298	29.422176290110762
0.31947746812289524	0.27917364600781686	32.51337272929626
0.5203951312018454	0.27540622418066646	34.320273024576444
0.6641707111676838	0.2661698163428267	35.98846706473302
1.0818641026839921	0.2677376171352075	36.298645735398026
1.380763207386186	0.2598752598752599	37.62858256024069
1.762242623765353	0.2697599136768276	36.361474545045496
2.2491177693633446	0.2728512960436562	36.06725064727653
2.870507540929564	0.26219192448872575	37.72445127937685
3.663575849505617	0.2684563758389262	36.85487847515752
4.675754309544294	0.2694691457828079	36.75680707595976
5.9675790160513	0.26434047052603754	37.49417272075419
7.616311070948165	0.26109660574412535	38.06009723914102
9.720557394115772	0.26184865147944486	37.938651350713904

```java
/**
 * @author Masood Khosroshahy <m.kh@ieee.org>
 * Project: ELEC 6851
 */
package com.masoodkh.aloha;

import java.io.*;
import java.util.*;
import java.text.*;

public class Aloha {

    public static double lambdaMin = 0.01;
    public static double lambdaMax = 10;
    public static double lambdaStep;
//    public static double lambdaMin = 0.25; // For setting one lambda and
//    public static double lambdaMax = 0.25; // checking frame/event lists
    public static double timeSlot = 1; // in Seconds
    public static double tProp = 0.01; // one-way propagation delay in
Seconds
    public static double backoffPeriodT = 10; // backoff is uniform
between 0 and this
    public static int stationsNo = 10;
    public static int maxFrames = 1000;
    public static int framesGenerated = 0;
    public static int framesRetransmitted = 0;

    public static void main(String[] args) throws IOException {

        ArrayList<Double> lambdaList = new ArrayList<Double>();
        ArrayList<Double> aveFrameDelayList = new ArrayList<Double>();
        ArrayList<Double> offeredLoadList = new ArrayList<Double>();
        ArrayList<Double> throughputList = new ArrayList<Double>();

        ArrayList<Frame> frameList = new ArrayList<Frame>();
        ArrayList<Event> eventList = new ArrayList<Event>();

        double lambda;

        for (lambda = lambdaMin; lambda <= lambdaMax; lambda +=
lambdaStep) {
            lambdaStep = lambda / 20;

            //Initialization
            frameList.clear();
            framesGenerated = 0;
            framesRetransmitted = 0;
            for (int i = 1; i <= stationsNo; i++) {
                Frame newFrame = new Frame();
                newFrame.frameNo = framesGenerated;
                newFrame.nodeNo = i;
                newFrame.arrivalTime = -(1 / lambda) *
Math.log(Math.random());
                frameList.add(newFrame);
```

```
 51
 52                 Event event = new Event(i, newFrame.arrivalTime, 1,
newFrame.frameNo);
 53                 eventList.add(event);
 54             }
 55
 56             while (eventList.size() != 0) {
 57
 58                 Collections.sort(eventList); // Sorts events based on
eventTime
 59
 60                 //This is only to show the event list in the output
 61 //                if (framesGenerated == 1000) {
 62 //                    printEventList(eventList);
 63 //                }
 64
 65                 Event event = eventList.get(0);
 66                 switch (event.eventType) {
 67                     case 1: // Scheduling Event Type 2
 68                         event.eventTime = event.eventTime + (timeSlot -
event.eventTime % timeSlot);
 69                         event.eventType = 2;
 70                         Collections.sort(eventList);
 71                         break;
 72                     case 2:
 73                         if (event.collisionFlagged) {
 74                             // Scheduding a type 3 event (recycling the
event)
 75                             event.collisionFlagged = false;
 76                             event.eventTime = event.eventTime + 2 * tProp
+ backoffPeriodT * Math.random();
 77                             event.eventType = 3;
 78                             Collections.sort(eventList);
 79                         } else {
 80                             // Marking in the frame: transmitted flag and
departure time
 81                             if
(event.frameNo.equals(frameList.get(event.frameNo - 1).frameNo)) {
 82                                 frameList.get(event.frameNo -
1).transmitted = true;
 83                                 frameList.get(event.frameNo -
1).departureTime = event.eventTime;
 84                             } else {
 85                                 System.err.println("No matching between
frame numbers in frameList and eventList!");
 86                             }
 87
 88                             if (framesGenerated < maxFrames) {
 89                                 // Creating a new frame and recycling the
event
 90                                 Frame newFrame = new Frame();
 91                                 newFrame.frameNo = framesGenerated;
 92                                 newFrame.nodeNo = event.nodeNo;
 93                                 newFrame.arrivalTime = event.eventTime + 2
```

```
* tProp - (1 / lambda) * Math.log(Math.random());
 94                                     frameList.add(newFrame);
 95
 96                                     event.eventTime = newFrame.arrivalTime;
 97                                     event.eventType = 1;
 98                                     event.frameNo = newFrame.frameNo;
 99                                     Collections.sort(eventList);
100                                 } else {
101                                     // Enough frames generated. Deleting the
event
102                                     if (eventList.remove(event)) {
103                                         //System.out.println("Node " +
event.nodeNo + " is finished.");
104                                     } else {
105                                         System.out.println("Error. Event
couldn't be found or removed.");
106                                     }
107                                     Collections.sort(eventList);
108                                 }
109                             }
110                             break;
111                         case 3:// Scheduling Event Type 2
112                             event.eventTime = event.eventTime + (timeSlot -
event.eventTime % timeSlot);
113                             event.eventType = 2;
114                             Collections.sort(eventList);
115                             framesRetransmitted++;
116                             break;
117                         default:
118                             System.out.println("Invalid event type.");
119                             break;
120                 }
121
122                 // This is for detecting and flagging the collisions
123                 Collections.sort(eventList); // Sorts events based on
eventTime
124                 for (int index = 0; index < eventList.size(); index++) {
125                     event = eventList.get(index);
126                     if (event.eventType == 2) {
127                         for (int i = index + 1; i < eventList.size(); i++)
{
128                             if
(event.eventTime.equals(eventList.get(i).eventTime)) {
129                                 event.collisionFlagged = true;
130                                 eventList.get(i).collisionFlagged = true;
131                             }
132                         }
133                     }
134                 }
135
136             }
137
138             double simulationRunTime = frameList.get(frameList.size() -
1).departureTime;
```

```
139             for (int index = 0; index < frameList.size(); index++) {
140                 if (simulationRunTime <
frameList.get(index).departureTime) {
141                     simulationRunTime =
frameList.get(index).departureTime;
142                 }
143             }
144 
145             double totalFrameDelay = 0;
146             for (int index = 0; index < frameList.size(); index++) {
147                 totalFrameDelay += frameList.get(index).departureTime -
frameList.get(index).arrivalTime;
148             }
149             double aveFrameDelay = totalFrameDelay / frameList.size();
150             double throughput = frameList.size() / simulationRunTime;
151             double offeredLoad = (frameList.size() + framesRetransmitted)
/ simulationRunTime;
152 
153             /*
154             printFrameList(frameList);
155             System.out.println("framesGenerated:" + framesGenerated);
156             System.out.println("framesRetransmitted:" +
framesRetransmitted);
157             System.out.println("aveFrameDelay:" + aveFrameDelay);
158             System.out.println("simulationRunTime: " + simulationRunTime);
159             System.out.println("throughput: " + throughput);
160             System.out.println("offeredLoad: " + offeredLoad);
161              */
162 
163             lambdaList.add(lambda);
164             aveFrameDelayList.add(aveFrameDelay);
165             offeredLoadList.add(offeredLoad);
166             throughputList.add(throughput);
167         }
168 
169         File f = new File("throughputVSofferedLoad.txt");
170         FileOutputStream fop = new FileOutputStream(f);
171         String crlf = System.getProperty("line.separator");
172 
173         if (f.exists()) {
174             String str = "Lambda    OfferedLoad Throughput" + crlf;
175             fop.write(str.getBytes());
176             for (int index = 0; index < throughputList.size(); index++) {
177                 str = lambdaList.get(index) + "  " +
offeredLoadList.get(index) + "    " + throughputList.get(index) + crlf;
178                 fop.write(str.getBytes());
179             }
180             fop.flush();
181             fop.close();
182         } else {
183             System.out.println("The file throughputVSofferedLoad.txt does
not exist");
184         }
185 
```

```
186         File f2 = new File("aveDelayVSthroughput.txt");
187         FileOutputStream fop2 = new FileOutputStream(f2);
188 
189         if (f2.exists()) {
190             String str = "Lambda    Throughput  AveFrameDelay" + crlf;
191             fop2.write(str.getBytes());
192             for (int index = 0; index < throughputList.size(); index++) {
193                 str = lambdaList.get(index) + "  " +
throughputList.get(index) + "  " + aveFrameDelayList.get(index) + crlf;
194                 fop2.write(str.getBytes());
195             }
196             fop2.flush();
197             fop2.close();
198         } else {
199             System.out.println("The file aveDelayVSthroughput.txt does not
exist");
200         }
201     }
202 
203     static void printEventList(ArrayList<Event> eventList) {
204         System.out.println("##### Events:");
205         System.out.println("'Node/List No'    'Event Time'  'Event Type'
'Frame No'    'Collision Flagged'");
206         DecimalFormat timeFormat = new DecimalFormat("0.00");
207         for (int index = 0; index <
208                 eventList.size(); index++) {
209             Event event = eventList.get(index);
210 //            System.out.println("Node/List No=" + event.nodeNo + ", Event
Time=" + timeFormat.format(event.eventTime) + ", Event Type=" +
event.eventType + ", Frame No=" + event.frameNo + ", collisionFlagged=" +
event.collisionFlagged);
211             System.out.println(event.nodeNo + " " +
timeFormat.format(event.eventTime) + "  " + event.eventType + " " +
event.frameNo + "   " + event.collisionFlagged);
212         }
213 
214     }
215 
216     static void printFrameList(ArrayList<Frame> frameList) {
217         System.out.println("##### Frames:");
218         System.out.println("'Frame No'  'Node No'   'A-Time'
'D-Time'");
219         DecimalFormat timeFormat = new DecimalFormat("0.00");
220         for (int index = 0; index <
221                 frameList.size(); index++) {
222             Frame frame = frameList.get(index);
223             System.out.println(frame.frameNo + "    " + frame.nodeNo + "
" + timeFormat.format(frame.arrivalTime) + "    " + frame.departureTime);
224         }
225     }
226 }
227 
228 class Frame {
229 
```

```
230     public Integer frameNo;
231     public int nodeNo;
232     public double arrivalTime;
233     public double departureTime;
234     public boolean transmitted;
235 
236     public Frame() {
237         frameNo = 0;
238         nodeNo = 0;
239         arrivalTime = 0;
240         departureTime = 0;
241         transmitted = false;
242         Aloha.framesGenerated++;
243     }
244 }
245 
246 class Event implements Comparable<Event> {
247 
248     public int nodeNo;
249     public Double eventTime;
250     public int eventType;
251     public Integer frameNo;
252     public boolean collisionFlagged = false;
253 
254     public Event(int nodeNo, double eventTime, int eventType, int frameNo)
{
255         this.nodeNo = nodeNo;
256         this.eventTime = eventTime;
257         this.eventType = eventType;
258         this.frameNo = frameNo;
259     }
260 
261     public int compareTo(Event event) {
262         return eventTime.compareTo(event.eventTime);
263     }
264 }
```