

## Generator.java

```
1 //*****
2 //
3 // File:    Generator.java
4 // Package: ---
5 // Unit:    Class Generator
6 //
7 //*****
8
9 import edu.rit.numeric.ExponentialPrng;
10 import edu.rit.numeric.ListSeries;
11 import edu.rit.numeric.Series;
12 import edu.rit.sim.Event;
13 import edu.rit.sim.Simulation;
14 import edu.rit.util.Random;
15
16 /**
17  * Class Generator generates requests for the web server simulations.
18  *
19  * @author Alan Kaminsky
20  * @author Jimi Ford (jhf3617)
21  * @version 5-2-2015
22  */
23 public class Generator
24 {
25     // private data members
26
27     private Simulation sim;
28     private ExponentialPrng tpktPrng;
29     private Random prng;
30     private final int npkt;
31     private Routable source;
32     private Link link;
33     private ListSeries respTimeSeries;
34     private ListSeries respTimeLargePackets;
35     private ListSeries respTimeSmallPackets;
36     private int largePackets;
37     private int smallPackets;
38
39     /**
40      * Create a new request generator.
41      *
42      * @param sim Simulation.
43      * @param rpkt Packet generation mean rate.
44      * @param npkt Number of packets.
45      * @param prng Pseudorandom number generator.
46      * @param source First host in network sending the packets.
47      */
48     public Generator (Simulation sim, double rpkt, int npkt, Random prng,
49                     Routable source, Link link) {
50         this.sim = sim;
51         this.tpktPrng = new ExponentialPrng (prng, rpkt);
52         this.npkt = npkt;
53         this.source = source;
54         this.prng = prng;
55         respTimeSeries = new ListSeries();
56         respTimeLargePackets = new ListSeries();
57         respTimeSmallPackets = new ListSeries();
58         largePackets = 0;
```

```

59     smallPackets = 0;
60     this.link = link;
61     generatePacket();
62 }
63
64 /**
65  * Generate the next packet.
66  */
67 private void generatePacket() {
68     Packet p = new Packet (prng, sim, respTimeSeries, respTimeLargePackets,
69         respTimeSmallPackets);
70     if(link.ready()) {
71         source.startSending (p, link);
72     }
73     if(p.isLarge) ++largePackets;
74     else ++smallPackets;
75     if (totalPackets() < npkt) {
76         sim.doAfter (tpktPrng.next(), new Event() {
77             public void perform() {
78                 generatePacket();
79             }
80         });
81     }
82 }
83
84 /**
85  * Returns a data series containing the response time statistics of the
86  * generated requests.
87  *
88  * @return Response time series.
89  */
90 public Series responseTimeSeries() {
91     return respTimeSeries;
92 }
93
94 /**
95  * Returns a data series containing the response time statistics of the
96  * larger packets
97  *
98  * @return Response time series.
99  */
100 public Series responseTimeLarge() {
101     return respTimeLargePackets;
102 }
103
104 /**
105  * Returns a data series containing the response time statistics of the
106  * smaller packets.
107  *
108  * @return Response time series.
109  */
110 public Series responseTimeSmall() {
111     return respTimeSmallPackets;
112 }
113
114 /**
115  * Returns the total number of packets currently generated
116  */

```

```

117 public int totalPackets() {
118     return largePackets + smallPackets;
119 }
120
121 /**
122  * Returns the response time statistics of the generated requests.
123  *
124  * @return Response time statistics (mean, standard deviation, variance).
125  */
126 public Series.Stats responseTimeStats() {
127     return respTimeSeries.stats();
128 }
129
130 /**
131  * Returns the drop fraction of the generated packets.
132  */
133 public double totalDropFraction() {
134     return (double)(totalPackets() - respTimeSeries.length())
135            /(double)totalPackets();
136 }
137
138 /**
139  * Returns the drop fraction of the large packets generated
140  */
141 public double largePacketDropFraction() {
142     return (double)(largePackets - respTimeLargePackets.length())
143            /(double)largePackets;
144 }
145
146 /**
147  * Returns the drop fraction of the small packets generated
148  */
149 public double smallPacketDropFraction() {
150     return (double)(smallPackets - respTimeSmallPackets.length())
151            /(double)smallPackets;
152 }
153 }

```

## Host.java

```
1 //*****
2 //
3 // File:    Host.java
4 // Package: ---
5 // Unit:    Class Host
6 //
7 //*****
8
9 import edu.rit.sim.Simulation;
10
11 /**
12  * Class Host provides the server in the web simulation. The server's
13  * request processing time is exponentially distributed with a given mean.
14  * Requests are added to the server's queue at any time. The queue has a given
15  * maximum size.
16  *
17  * @author Jimi Ford (jhf3617)
18  * @version 22-Apr-2015
19  */
20 public class Host extends Routable
21 {
22
23     /**
24      * Construct a new server. The server's request processing time is
25      * exponentially distributed with the given mean. The server's request queue
26      * has the given maximum size.
27      *
28      * @param sim    Simulation.
29      */
30     public Host(Simulation sim) {
31         super(sim);
32     }
33
34     /**
35      * Called when this routable object finished receiving a packet on a certain
36      * link
37      * @param packet the packet this object received
38      * @param link the link that the packet was received on
39      */
40     public void receivePacket(final Packet packet, final Link link) {
41         link.open();
42         packet.finish();
43     }
44 }
```

## Link.java

```
1 //*****
2 //
3 // File:    Link.java
4 // Package: ---
5 // Unit:    Class Link
6 //
7 //*****
8
9 import edu.rit.sim.Simulation;
10
11 /**
12  * Class Link represents a connection between two routable objects. Links are
13  * <I>closed</I> if a packet is currently transmitting on them and <I>open</I>
14  * if they are ready to be transmitted on.
15  *
16  * @author Jimi Ford (jhf3617)
17  * @version 5-6-2015
18  */
19 public class Link {
20
21     /**
22      * default bit rate used in this project
23      * <P>9600 bits/sec</P>
24      */
25     public static final int DEFAULT_BIT_RATE = 9600;
26
27     /**
28      * true if this link has an infinite bit rate
29      */
30     public final boolean infiniteBitRate;
31
32     /**
33      * the bit rate of this link
34      */
35     public final double bitRate;
36
37     // private data members
38
39     private final Routable r1;
40     private final Routable r2;
41     private final Simulation sim;
42     private double closeStarted;
43     private double closeFinished;
44     private double totalTimeSpentClosed;
45     private boolean ready;
46
47
48     /**
49      * construct a link with the default finite bit rate between two routables
50      *
51      * @param sim the simulation reference
52      * @param r1 one of the routable objects
53      * @param r2 the other routable object
54      */
55     public Link(Simulation sim, Routable r1, Routable r2) {
56         this(sim, false, r1, r2);
57     }
58 }
```

```

59  /**
60  * construct a link with specified finite or infinite bit rate
61  *
62  * @param sim the simulation reference
63  * @param infiniteBitRate set to true for infinite bit rate, false for
64  * default finite bit rate
65  * @param r1 one of the routable objects
66  * @param r2 the other routable object
67  */
68  public Link(Simulation sim, boolean infiniteBitRate, Rutable r1,
69             Rutable r2) {
70      this.sim = sim;
71      this.r1 = r1;
72      this.r2 = r2;
73      this.ready = true;
74      this.infiniteBitRate = infiniteBitRate;
75      this.bitRate = infiniteBitRate ? Double.POSITIVE_INFINITY :
76          DEFAULT_BIT_RATE;
77      this.totalTimeSpentClosed = 0;
78  }
79
80  /**
81  * get the other routable object attached to this link compared to the
82  * current one
83  *
84  * @param current the current routable object querying for the other
85  * attached routable object
86  * @return the routable object that is not equal to the current one
87  */
88  public Rutable other(Rutable current) {
89      return this.r1.equals(current) ? r2 : r1;
90  }
91
92  /**
93  * get the current state of the link
94  *
95  * @return true if the link is ready to pass another packet along it; false
96  * otherwise
97  */
98  public boolean ready() {
99      return this.ready;
100  }
101
102  /**
103  * close this link off so that other packets may not be transmitted on it
104  * until open() is called
105  *
106  * @throws IllegalStateException if the link is not ready to be closed and
107  * this link has a finite bit-rate
108  */
109  public void close() throws IllegalStateException {
110      if(!this.infiniteBitRate) {
111          if(!this.ready) {
112              throw new IllegalStateException();
113          }
114          this.ready = false;
115          this.closeStarted = sim.time();
116      }

```

```
117     }
118
119     /**
120      * open this link so that other packets may be transmitted on it
121      */
122     public void open() {
123         this.ready = true;
124         this.closeFinished = sim.time();
125         this.totalTimeSpentClosed += (this.closeFinished - this.closeStarted);
126     }
127
128     /**
129      * Return the amount of time this link was closed as a fraction of the
130      * total amount of time in the simulation.
131      */
132     public double fractionClosed() {
133         return this.totalTimeSpentClosed / sim.time();
134     }
135 }
136
```

# MrPotatoHead.java

```
1 //*****
2 //
3 // File:    MrPotatoHead.java
4 // Package: ---
5 // Unit:    Class MrPotatoHead
6 //
7 //*****
8
9 import edu.rit.numeric.ListXYSeries;
10 import edu.rit.numeric.Series;
11 import edu.rit.sim.Simulation;
12 import edu.rit.util.Random;
13 import java.io.IOException;
14 import java.io.PrintWriter;
15
16 /**
17  * Class MrPotatoHead is the hot potato simulation main program. It simulates
18  * a network in which routers use hot potato routing and uses Prof. Alan
19  * Kaminsky's pj2 library to aid in this discrete event simulation.
20  *
21  * @author Alan Kaminsky
22  * @author Jimi Ford (jhf3617)
23  * @version 5-3-2015
24  */
25 public class MrPotatoHead
26 {
27     private static double rlb;
28     private static double rub;
29     private static double rdelta;
30     private static int npkt;
31     private static long seed;
32
33     private static Random prng;
34     private static Simulation sim;
35     private static String prefix;
36     private static Generator gen;
37
38     /**
39      * Main program to simulate hot-potato routing
40      *
41      * @param args command line arguments
42      */
43     public static void main(String[] args)
44     {
45         // Parse command line arguments.
46         if (args.length != 5 && args.length != 6) usage();
47         rlb = Double.parseDouble (args[0]);
48         rub = Double.parseDouble (args[1]);
49         rdelta = Double.parseDouble (args[2]);
50         if (rlb <= 0) rlb = rdelta;
51         npkt = Integer.parseInt (args[3]);
52         seed = Long.parseLong (args[4]);
53         prefix = args.length == 6 ? args[5] : "potato";
54         // Set up pseudorandom number generator.
55         prng = new Random (seed);
56
57         // Set up plot data series.
58         ListXYSeries respTimeSeries = new ListXYSeries();
```



```

59 ListXYSeries respTimeLargeSeries = new ListXYSeries();
60 ListXYSeries respTimeSmallSeries = new ListXYSeries();
61 ListXYSeries dropFracSeries = new ListXYSeries();
62 ListXYSeries dropFracLargeSeries = new ListXYSeries();
63 ListXYSeries dropFracSmallSeries = new ListXYSeries();
64
65 ListXYSeries aDrop = new ListXYSeries();
66 ListXYSeries bDrop = new ListXYSeries();
67 ListXYSeries cDrop = new ListXYSeries();
68 ListXYSeries dDrop = new ListXYSeries();
69
70 ListXYSeries aReRoute = new ListXYSeries();
71 ListXYSeries bReRoute = new ListXYSeries();
72 ListXYSeries cReRoute = new ListXYSeries();
73 ListXYSeries dReRoute = new ListXYSeries();
74
75 ListXYSeries adActivity = new ListXYSeries();
76 ListXYSeries bdActivity = new ListXYSeries();
77 ListXYSeries cdActivity = new ListXYSeries();
78 ListXYSeries d2Activity = new ListXYSeries();
79
80 ListXYSeries abActivity = new ListXYSeries();
81 ListXYSeries acActivity = new ListXYSeries();
82 ListXYSeries baActivity = new ListXYSeries();
83 ListXYSeries bcActivity = new ListXYSeries();
84 ListXYSeries caActivity = new ListXYSeries();
85 ListXYSeries cbActivity = new ListXYSeries();
86 ListXYSeries daActivity = new ListXYSeries();
87 ListXYSeries dbActivity = new ListXYSeries();
88 ListXYSeries dcActivity = new ListXYSeries();
89
90 // Sweep mean request rate.
91 System.out.printf ("Mean\tResp\tResp\tResp\tDrop\tDrop\tDrop%n");
92 System.out.printf ("Pkt\tTime\tTime\tTime\tFrac\tFrac\tFrac%n");
93 System.out.printf ("Rate\tTotal\tLarge\tSmall\tTotal\tLarge\tSmall%n");
94 StringBuilder builder = new StringBuilder();
95 builder.append(
96     String.format("Mean\tResp\tResp\tResp\tDrop\tDrop\tDrop%n"));
97 builder.append(
98     String.format("Pkt\tTime\tTime\tTime\tFrac\tFrac\tFrac%n"));
99 builder.append(
100     String.format("Rate\tTotal\tLarge\tSmall\tTotal\tLarge\tSmall%n"));
101 double rate;
102 for (int i = 0; (rate = rlb + i*rdelta) <= rub; ++ i)
103 {
104     // Set up simulation.
105     sim = new Simulation();
106     Host h1, h2;
107     Router a, b, c, d;
108
109     h1 = new Host(sim);
110     h2 = new Host(sim);
111     d = new Router(prng, sim);
112     a = new Router(prng, sim);
113     b = new Router(prng, sim);
114     c = new Router(prng, sim);
115     Link
116     ab = new Link(sim, a, b),

```

```

117         ac = new Link(sim, a, c),
118         ad = new Link(sim, a, d),
119         ba = new Link(sim, b, a),
120         bc = new Link(sim, b, c),
121         bd = new Link(sim, b, d),
122         ca = new Link(sim, c, a),
123         cb = new Link(sim, c, b),
124         cd = new Link(sim, c, d),
125         da = new Link(sim, d, a),
126         db = new Link(sim, d, b),
127         dc = new Link(sim, d, c),
128         d2 = new Link(sim, d, h2);
129     // preferred link
130     a.setPrimary(ad);
131     b.setPrimary(bd);
132     c.setPrimary(cd);
133     d.setPrimary(d2);
134     // secondary links
135     a.addSecondary(ab);
136     a.addSecondary(ac);
137     b.addSecondary(ba);
138     b.addSecondary(bc);
139     c.addSecondary(ca);
140     c.addSecondary(cb);
141     d.addSecondary(da);
142     d.addSecondary(db);
143     d.addSecondary(dc);
144
145     // Set up request generator and generate first request.
146     gen = new Generator(sim, rate, npkt, prng, h1,
147         new Link(sim, true, h1, a));
148
149     // Run the simulation.
150     sim.run();
151
152     // Print results.
153     Series.Stats totalStats = gen.responseTimeStats();
154     Series.Stats largeStats = gen.responseTimeLarge().stats();
155     Series.Stats smallStats = gen.responseTimeSmall().stats();
156     System.out.printf("%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\n",
157         rate, totalStats.mean, largeStats.mean, smallStats.mean,
158         gen.totalDropFraction(), gen.largePacketDropFraction(),
159         gen.smallPacketDropFraction());
160     builder.append(String.format(
161         "%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\n",
162         rate, totalStats.mean, largeStats.mean, smallStats.mean,
163         gen.totalDropFraction(), gen.largePacketDropFraction(),
164         gen.smallPacketDropFraction()));
165     // Record results for plot.
166     respTimeSeries.add(rate, totalStats.mean);
167     respTimeLargeSeries.add(rate, largeStats.mean);
168     respTimeSmallSeries.add(rate, smallStats.mean);
169     dropFracSeries.add(rate, gen.totalDropFraction());
170     dropFracLargeSeries.add(rate, gen.largePacketDropFraction());
171     dropFracSmallSeries.add(rate, gen.smallPacketDropFraction());
172     aDrop.add(rate, a.dropFraction(npkt));
173     bDrop.add(rate, b.dropFraction(npkt));
174     cDrop.add(rate, c.dropFraction(npkt));

```

```

175         dDrop.add(rate, d.dropFraction(npkt));
176         aReRoute.add(rate, a.reRouteFraction());
177         bReRoute.add(rate, b.reRouteFraction());
178         cReRoute.add(rate, c.reRouteFraction());
179         dReRoute.add(rate, d.reRouteFraction());
180         // primary link activity
181         adActivity.add(rate, ad.fractionClosed());
182         bdActivity.add(rate, bd.fractionClosed());
183         cdActivity.add(rate, cd.fractionClosed());
184         d2Activity.add(rate, d2.fractionClosed());
185         // secondary link activity
186         abActivity.add(rate, ab.fractionClosed());
187         acActivity.add(rate, ac.fractionClosed());
188         baActivity.add(rate, ba.fractionClosed());
189         bcActivity.add(rate, bc.fractionClosed());
190         caActivity.add(rate, ca.fractionClosed());
191         cbActivity.add(rate, cb.fractionClosed());
192         daActivity.add(rate, da.fractionClosed());
193         dbActivity.add(rate, db.fractionClosed());
194         dcActivity.add(rate, dc.fractionClosed());
195     }
196
197     try {
198         new PlotHandler(prefix, dropFracSeries, respTimeSeries,
199             dropFracLargeSeries, respTimeLargeSeries,
200             dropFracSmallSeries, respTimeSmallSeries,
201             aDrop, bDrop, cDrop, dDrop,
202             aReRoute, bReRoute, cReRoute, dReRoute,
203             adActivity, bdActivity, cdActivity, d2Activity,
204             abActivity, acActivity, baActivity, bcActivity, caActivity,
205             cbActivity, daActivity, dbActivity, dcActivity).write();
206         PrintWriter tableWriter = new PrintWriter(prefix + "-table.tsv");
207         tableWriter.print(builder.toString());
208         tableWriter.close();
209     } catch (IOException e) {
210         e.printStackTrace();
211     }
212 }
213
214 /**
215  * Print a usage message and exit.
216  */
217 private static void usage()
218 {
219     System.err.println ("Usage: java MrPotatoHead <rlb> <rub> <rdelta> "
220         + "<npkt> <seed> [<file-prefix> (optional)]");
221     System.err.println ("<rlb> = Mean packet rate lower bound");
222     System.err.println ("<rub> = Mean packet rate upper bound");
223     System.err.println ("<rdelta> = Mean packet rate delta");
224     System.err.println ("<npkt> = Number of packets");
225     System.err.println ("<seed> = Random seed");
226     System.err.println ("<file-prefix> = optional file prefix, "
227         + "default = \"potato\"");
228     System.exit (1);
229 }
230 }

```

## Packet.java

```
1 //*****
2 //
3 // File:    Packet.java
4 // Package: ---
5 // Unit:    Packet Link
6 //
7 //*****
8
9 import edu.rit.numeric.ListSeries;
10 import edu.rit.sim.Simulation;
11 import edu.rit.util.Random;
12
13 /**
14  * Class Packet provides a packet model in the web simulation. It contains the
15  * logic necessary to determine the size of the packet and the amount of time
16  * it would take to transmit along a link. It also reports to several instances
17  * of ListSeries objects that keep track of the response time of packets based
18  * on their size.
19  *
20  * @author Alan Kaminsky
21  * @author Jimi Ford (jhf3617)
22  * @version 5-6-2015
23  */
24 public class Packet
25 {
26     /**
27      * size of the packet in bits
28      */
29     public final int size;
30
31     /**
32      * unique identifier across all other packets
33      */
34     public final int id;
35
36     /**
37      * true if this packet is 576 bytes, false otherwise
38      */
39     public final boolean isLarge;
40
41     // private data member
42
43     private static int idCounter = 0;
44     private Simulation sim;
45     private double startTime;
46     private double finishTime;
47     private ListSeries respTimeSeries;
48     private ListSeries respTimeLargePackets;
49     private ListSeries respTimeSmallPackets;
50
51     private static final int
52         SMALL = 40 * Byte.SIZE,
53         LARGE = 576 * Byte.SIZE;
54
55
56     /**
57      * Construct a new packet. The packet's response time will be recorded in
58      * the ListSeries.
```

```

59  * @param prng a pseudorandom number generator
60  * @param sim the current simulation object
61  * @param series the series to keep track of response times in
62  * @param seriesLargePackets series to keep track of large packet response
63  * times in
64  * @param seriesSmallPackets series to keep track of small packet response
65  * times in
66  */
67  public Packet(Random prng, Simulation sim, ListSeries series,
68               ListSeries seriesLargePackets, ListSeries seriesSmallPackets) {
69      this.id = ++ idCounter;
70      this.sim = sim;
71      this.startTime = sim.time();
72      this.size = prng.nextDouble() < .5 ? SMALL : LARGE;
73      this.isLarge = this.size == LARGE;
74      this.respTimeSeries = series;
75      this.respTimeLargePackets = seriesLargePackets;
76      this.respTimeSmallPackets = seriesSmallPackets;
77  }
78
79  /**
80   * get the time it would take this packet to transmit along a given link
81   *
82   * @param link the given link to transmit on
83   * @return the time in seconds it would take to transmit on the given link
84   */
85  public double transmitTime(Link link) {
86      if(link.infiniteBitRate) {
87          return 0;
88      }
89      return ((double)size) / link.bitRate;
90  }
91
92  /**
93   * Mark this request as finished. The request's finish time is set to the
94   * current simulation time. The request's response time is recorded in the
95   * response time series.
96   */
97  public void finish()
98  {
99      finishTime = sim.time();
100     respTimeSeries.add (responseTime());
101     if(isLarge) respTimeLargePackets.add(responseTime());
102     else respTimeSmallPackets.add(responseTime());
103 }
104
105 /**
106  * Returns this request's response time.
107  *
108  * @return Response time.
109  */
110 public double responseTime()
111 {
112     return finishTime - startTime;
113 }
114
115 /**
116  * Returns a string version of this request.

```

```
117     *  
118     * @return String version.  
119     */  
120     public String toString()  
121     {  
122         return "Packet " + id;  
123     }  
124 }
```

## PlotHandler.java

```
1 //*****
2 //
3 // File:    PlotHandler.java
4 // Package: ---
5 // Unit:    Class PlotHandler
6 //
7 //*****
8
9 import java.awt.BasicStroke;
10 import java.awt.Color;
11 import java.io.IOException;
12 import java.text.DecimalFormat;
13
14 import edu.rit.numeric.ListXYSeries;
15 import edu.rit.numeric.plot.Dots;
16 import edu.rit.numeric.plot.Plot;
17
18 /**
19  * Class PlotHandler is the delegate for dealing with visualizing the data
20  * generated by the "number crunching" program, MrPotatoHead.
21  * Its purpose is to be instantiated in MrPotatoHead with the data to plot,
22  * where the write() method should then be called.
23  * <P>
24  * Running this program and specifying in the command line arguments the plot
25  * files previously generated will open a graphical representation of these
26  * plots for each file.
27  * </P>
28  * @author Jimi Ford
29  * @version 5-6-2015
30  *
31  */
32 public class PlotHandler {
33
34     // private data members
35     private final String rtTotalFile;
36     private final String dfTotalFile;
37     private final String rtlargeFile;
38     private final String dflargeFile;
39     private final String rtSmallFile;
40     private final String dfSmallFile;
41     private final String routerDropFile;
42     private final String reRouteFile;
43     private final String primaryActivityFile;
44     private final String secondaryActivityFile;
45     private final ListXYSeries dfTotal;
46     private final ListXYSeries rtTotal;
47     private final ListXYSeries dfLarge;
48     private final ListXYSeries rtlarge;
49     private final ListXYSeries dfSmall;
50     private final ListXYSeries rtSmall;
51     private final ListXYSeries aDrop;
52     private final ListXYSeries bDrop;
53     private final ListXYSeries cDrop;
54     private final ListXYSeries dDrop;
55     private final ListXYSeries aReRoute;
56     private final ListXYSeries bReRoute;
57     private final ListXYSeries cReRoute;
58     private final ListXYSeries dReRoute;
```

```

59 private final ListXYSeries adActivity;
60 private final ListXYSeries bdActivity;
61 private final ListXYSeries cdActivity;
62 private final ListXYSeries d2Activity;
63 private final ListXYSeries abActivity;
64 private final ListXYSeries acActivity;
65 private final ListXYSeries baActivity;
66 private final ListXYSeries bcActivity;
67 private final ListXYSeries caActivity;
68 private final ListXYSeries cbActivity;
69 private final ListXYSeries daActivity;
70 private final ListXYSeries dbActivity;
71 private final ListXYSeries dcActivity;
72
73
74 /**
75  * Construct a new PlotHandler object
76  *
77  * @param prefix the prefix to use for saving the files
78  * @param dfTotal the xy-series that contains the drop fraction info
79  * @param rtTotal the xy-series that contains the response time info
80  * @param dfLarge series containing the drop fraction for large packets
81  * @param rtLarge series containing the response time for large packets
82  * @param dfSmall series containing the drop fraction for small packets
83  * @param rtSmall series containing the response time for small packets
84  * @param aDrop series containing the drop fraction of router a
85  * @param bDrop series containing the drop fraction of router b
86  * @param cDrop series containing the drop fraction of router c
87  * @param dDrop series containing the drop fraction of router d
88  * @param aReRoute series containing the re-route fraction of router a
89  * @param bReRoute series containing the re-route fraction of router b
90  * @param cReRoute series containing the re-route fraction of router c
91  * @param dReRoute series containing the re-route fraction of router d
92  * @param adActivity series containing activity fraction of link ad
93  * @param bdActivity series containing activity fraction of link bd
94  * @param cdActivity series containing activity fraction of link cd
95  * @param d2Activity series containing activity fraction of link d2
96  * @param abActivity series containing activity fraction of link ab
97  * @param acActivity series containing activity fraction of link ac
98  * @param baActivity series containing activity fraction of link ba
99  * @param bcActivity series containing activity fraction of link bc
100 * @param caActivity series containing activity fraction of link ca
101 * @param cbActivity series containing activity fraction of link cb
102 * @param daActivity series containing activity fraction of link da
103 * @param dbActivity series containing activity fraction of link db
104 * @param dcActivity series containing activity fraction of link dc
105 */
106 public PlotHandler(String prefix,
107     ListXYSeries dfTotal, ListXYSeries rtTotal,
108     ListXYSeries dfLarge, ListXYSeries rtLarge,
109     ListXYSeries dfSmall, ListXYSeries rtSmall,
110     ListXYSeries aDrop, ListXYSeries bDrop, ListXYSeries cDrop,
111     ListXYSeries dDrop, ListXYSeries aReRoute, ListXYSeries bReRoute,
112     ListXYSeries cReRoute, ListXYSeries dReRoute,
113     ListXYSeries adActivity, ListXYSeries bdActivity,
114     ListXYSeries cdActivity, ListXYSeries d2Activity,
115     ListXYSeries abActivity, ListXYSeries acActivity,
116     ListXYSeries baActivity, ListXYSeries bcActivity,

```



```

117     ListXYSeries caActivity, ListXYSeries cbActivity,
118     ListXYSeries daActivity, ListXYSeries dbActivity,
119     ListXYSeries dcActivity) {
120         rtTotalFile = prefix + "-traversal-time.dwg";
121         dfTotalFile = prefix + "-drop-fraction.dwg";
122         rtLargeFile = prefix + "-traversal-time-large.dwg";
123         rtSmallFile = prefix + "-traversal-time-small.dwg";
124         dfLargeFile = prefix + "-drop-fraction-large.dwg";
125         dfSmallFile = prefix + "-drop-fraction-small.dwg";
126         routerDropFile = prefix + "-router-drop-fraction.dwg";
127         reRouteFile = prefix + "-re-route-fraction.dwg";
128         primaryActivityFile = prefix + "-primary-link-activity-fraction.dwg";
129         secondaryActivityFile = prefix +
130             "-secondary-link-activity-fraction.dwg";
131         this.dfTotal = dfTotal;
132         this.rtTotal = rtTotal;
133         this.dfLarge = dfLarge;
134         this.rtLarge = rtLarge;
135         this.dfSmall = dfSmall;
136         this.rtSmall = rtSmall;
137         this.aDrop = aDrop;
138         this.bDrop = bDrop;
139         this.cDrop = cDrop;
140         this.dDrop = dDrop;
141         this.aReRoute = aReRoute;
142         this.bReRoute = bReRoute;
143         this.cReRoute = cReRoute;
144         this.dReRoute = dReRoute;
145         this.adActivity = adActivity;
146         this.bdActivity = bdActivity;
147         this.cdActivity = cdActivity;
148         this.d2Activity = d2Activity;
149         this.abActivity = abActivity;
150         this.acActivity = acActivity;
151         this.baActivity = baActivity;
152         this.bcActivity = bcActivity;
153         this.caActivity = caActivity;
154         this.cbActivity = cbActivity;
155         this.daActivity = daActivity;
156         this.dbActivity = dbActivity;
157         this.dcActivity = dcActivity;
158     }
159
160     /**
161     * Save the plot information into files and display the plots.
162     *
163     * @throws IOException if it can't write to the file specified
164     */
165     public void write() throws IOException {
166         write("Total", "0.0", dfTotal, dfTotalFile, rtTotal, rtTotalFile);
167         write("Large Pkt", "0.0", dfLarge, dfLargeFile, rtLarge, rtLargeFile);
168         write("Small Pkt", "0.00", dfSmall, dfSmallFile, rtSmall, rtSmallFile);
169         writeRouterDrop();
170         writeRouterReRoute();
171         writePrimaryLinkActivity();
172         writeSecondaryLinkActivity();
173     }
174

```

```

175
176 /**
177  * write the router drop fraction plot
178  * @throws IOException if it can't write to the file specified
179  */
180 private void writeRouterDrop() throws IOException {
181     Plot routerDropFraction = new Plot()
182         .plotTitle("Router Drop Fraction")
183         .xAxisTitle ("Mean arrival rate (pkt/sec)")
184         .yAxisTitle ("Drop fraction")
185         .yAxisStart (0.0)
186         .yAxisEnd (1.0)
187         .yAxisTickFormat (new DecimalFormat ("0.0"))
188         .seriesDots(null)
189         .seriesColor(Color.RED)
190         .xySeries(aDrop)
191         .seriesColor(Color.ORANGE)
192         .seriesDots(Dots.circle(Color.ORANGE, new BasicStroke(),
193             Color.ORANGE, 7))
194         .xySeries(bDrop)
195         .seriesDots(null)
196         .seriesColor(Color.GREEN)
197         .xySeries(cDrop)
198         .seriesColor(Color.BLUE)
199         .xySeries(dDrop)
200         .labelColor(Color.RED)
201         .label("<b>A</b>", 42.5, .85)
202         .labelColor(Color.ORANGE)
203         .label("<b>B</b>", 42.5, .75)
204         .labelColor(Color.GREEN)
205         .label("<b>C</b>", 42.5, .65)
206         .labelColor(Color.BLUE)
207         .label("<b>D</b>", 42.5, .55);
208     Plot.write(routerDropFraction, routerDropFile);
209 }
210
211 /**
212  * write the primary link activity plot
213  * @throws IOException if it can't write to the file specified
214  */
215 private void writePrimaryLinkActivity() throws IOException {
216     Plot linkActivity = new Plot()
217         .plotTitle("Primary Link Activity")
218         .xAxisTitle ("Mean arrival rate (pkt/sec)")
219         .yAxisTitle ("Link Activity Fraction")
220         .yAxisStart (0.0)
221         .yAxisEnd (1.0)
222         .yAxisTickFormat (new DecimalFormat ("0.0"))
223         .seriesDots(null)
224         .seriesColor(Color.RED)
225         .xySeries(adActivity)
226         .seriesColor(Color.ORANGE)
227         .seriesDots(Dots.circle(Color.ORANGE, new BasicStroke(),
228             Color.ORANGE, 7))
229         .xySeries(bdActivity)
230         .seriesDots(null)
231         .seriesColor(Color.GREEN)
232         .xySeries(cdActivity)

```

```

233     .seriesColor(Color.BLUE)
234     .xySeries(d2Activity)
235     .labelColor(Color.RED)
236     .label("<b>A</b>", 42.5, .65)
237     .labelColor(Color.ORANGE)
238     .label("<b>B</b>", 42.5, .55)
239     .labelColor(Color.GREEN)
240     .label("<b>C</b>", 42.5, .45)
241     .labelColor(Color.BLUE)
242     .label("<b>D</b>", 42.5, .35);
243     Plot.write(linkActivity, primaryActivityFile);
244 }
245
246 /**
247  * write the secondary link activity plot
248  * @throws IOException if it can't write to the file specified
249  */
250 private void writeSecondaryLinkActivity() throws IOException {
251     Plot linkActivity = new Plot()
252     .plotTitle("Secondary Link Activity")
253     .xAxisTitle ("Mean arrival rate (pkt/sec)")
254     .yAxisTitle ("Link Activity Fraction")
255     .yAxisStart (0.0)
256     .yAxisEnd (1.0)
257     .yAxisTickFormat (new DecimalFormat ("0.0"))
258     .seriesDots(null)
259     .seriesColor(Color.RED)
260     .xySeries(abActivity)
261     .xySeries(acActivity)
262     .seriesColor(Color.ORANGE)
263     .seriesDots(Dots.circle(Color.ORANGE, new BasicStroke(),
264         Color.ORANGE, 7))
265     .xySeries(bcActivity)
266     .xySeries(baActivity)
267     .seriesColor(Color.GREEN)
268     .seriesDots(null)
269     .xySeries(caActivity)
270     .xySeries(cbActivity)
271     .seriesColor(Color.BLUE)
272     .xySeries(daActivity)
273     .xySeries(dbActivity)
274     .xySeries(dcActivity)
275     .labelColor(Color.RED)
276     .label("<b>A</b>", 42.5, .45)
277     .labelColor(Color.ORANGE)
278     .label("<b>B</b>", 42.5, .35)
279     .labelColor(Color.GREEN)
280     .label("<b>C</b>", 42.5, .25)
281     .labelColor(Color.BLUE)
282     .label("<b>D</b>", 42.5, .15);
283     Plot.write(linkActivity, secondaryActivityFile);
284 }
285
286 /**
287  * write the router re-route fraction plot
288  *
289  * @throws IOException if it can't write to the file specified
290  */

```

```

291 private void writeRouterReRoute() throws IOException {
292     Plot reRouteFraction = new Plot()
293         .plotTitle("Router Re-Route Fraction")
294         .xAxisTitle ("Mean arrival rate (pkt/sec)")
295         .yAxisTitle ("Re-Route fraction")
296         .yAxisStart (0.0)
297         .yAxisEnd (1.0)
298         .yAxisTickFormat (new DecimalFormat ("0.0"))
299         .seriesDots(null)
300         .seriesColor(Color.RED)
301         .xySeries(aReRoute)
302         .seriesColor(Color.ORANGE)
303         .seriesDots(Dots.circle(Color.ORANGE, new BasicStroke(),
304             Color.ORANGE, 7))
305         .xySeries(bReRoute)
306         .seriesDots(null)
307         .seriesColor(Color.GREEN)
308         .xySeries(cReRoute)
309         .seriesColor(Color.BLUE)
310         .xySeries(dReRoute)
311         .labelColor(Color.RED)
312         .label("<b>A</b>", 42.5, .55)
313         .labelColor(Color.ORANGE)
314         .label("<b>B</b>", 42.5, .45)
315         .labelColor(Color.GREEN)
316         .label("<b>C</b>", 42.5, .35)
317         .labelColor(Color.BLUE)
318         .label("<b>D</b>", 42.5, .25);
319     Plot.write(reRouteFraction, reRouteFile);
320 }
321
322 /**
323  * Save the plot information into files.
324  *
325  * @param titlePrefix Prefix of the plot's title
326  * @param yFormat decimal format of the traversal time y-axis labels
327  * @param df drop fraction series
328  * @param dfFile drop fraction file name
329  * @param rt response time series
330  * @param rtFile response time file
331  * @throws IOException if it fails to write to any of the specified files
332  */
333 private void write(String titlePrefix, String yFormat, ListXYSeries df,
334     String dfFile, ListXYSeries rt, String rtFile) throws IOException {
335     Plot responseTime = new Plot()
336         .plotTitle (titlePrefix+" Traversal Time")
337         .xAxisTitle ("Mean arrival rate (pkt/sec)")
338         .yAxisTitle ("Mean traversal time (sec)")
339         .yAxisTickFormat (new DecimalFormat (yFormat))
340         .seriesDots (null)
341         .xySeries (rt);
342     Plot dropFraction = new Plot()
343         .plotTitle (titlePrefix+" Drop Fraction")
344         .xAxisTitle ("Mean arrival rate (pkt/sec)")
345         .yAxisTitle ("Drop fraction")
346         .yAxisStart (0.0)
347         .yAxisEnd (1.0)
348         .yAxisTickFormat (new DecimalFormat ("0.0"))

```

```

349     .seriesDots (null)
350     .xySeries (df);
351     Plot.write(responseTime, rtFile);
352     Plot.write(dropFraction, dfFile);
353 }
354
355 /**
356  * Open a GUI for each plot in order to visualize the results of a
357  * previously run set of simulations.
358  *
359  * @param args each plot file generated that you wish to visualize
360  */
361 public static void main(String args[]) {
362     if(args.length < 1) {
363         System.err.println("Must specify at least 1 plot file.");
364         usage();
365     }
366
367     for(int i = 0; i < args.length; i++) {
368         try {
369             Plot plot = Plot.read(args[i]);
370             plot.getFrame().setVisible(true);
371         } catch (ClassNotFoundException e) {
372             System.err.println("Could not deserialize " + args[i]);
373         } catch (IOException e) {
374             System.err.println("Could not open " + args[i]);
375         } catch (IllegalArgumentException e) {
376             System.err.println("Error in file " + args[i]);
377         }
378     }
379 }
380
381 /**
382  * Print the usage message for this program and gracefully exit.
383  */
384 private static void usage() {
385     System.err.println("usage: java PlotHandler <plot-file-1> "+
386         "<(<plot-file-2> <plot-file-3>... etc.)");
387     System.exit(1);
388 }
389 }
390

```

## Routable.java

```
1 //*****
2 //
3 // File:    Routable.java
4 // Package: ---
5 // Unit:    Class Routable
6 //
7 //*****
8
9 import edu.rit.sim.Event;
10 import edu.rit.sim.Simulation;
11
12 /**
13  * Class Routable is the abstract base class that defines objects that contain
14  * routing logic with the ability to be linked together. Known implementations
15  * include Router and Host.
16  *
17  * @author Jimi Ford (jhf3617)
18  * @version 5-6-2015
19  */
20 public abstract class Routable {
21
22     private static int count = 0;
23
24     /**
25      * the simulation reference
26      */
27     protected final Simulation sim;
28     private final int id;
29
30
31     /**
32      * Construct a routable object
33      * @param sim the simulation this object should belong to
34      */
35     public Routable(Simulation sim) {
36         this.sim = sim;
37         id = ++ count;
38     }
39
40     /**
41      * compare this instance with another object and determine whether this
42      * instance is equal to the other object.
43      *
44      * @param o the other object to compare to
45      * @return true if this object is equal to the other object
46      */
47     public boolean equals(Object o) {
48         if(o == this) {
49             return true;
50         }
51         if(o instanceof Routable) {
52             return this.id == ((Routable)o).id;
53         }
54         return false;
55     }
56
57     /**
58      * Called when this routable object finished receiving a packet on a certain
```

# Routable.java

```

59  * link
60  * @param packet the packet this object received
61  * @param link the link that the packet was received on
62  */
63  public abstract void receivePacket(final Packet packet, final Link link);
64
65  /**
66   * Send a given packet along a given link to another routable
67   *
68   * @param packet the packet to send
69   * @param link the link to send the packet along
70   */
71  public void startSending(final Packet packet, final Link link) {
72      final Routable other = link.other(this);
73      final double transmitTime = packet.transmitTime(link);
74      link.close();
75      sim.doAfter(transmitTime, new Event() {
76          public void perform() {
77              other.receivePacket(packet, link);
78          }
79      });
80  }
81 }
82

```

## Router.java

```
1 //*****
2 //
3 // File:    Router.java
4 // Package: ---
5 // Unit:    Class Router
6 //
7 //*****
8
9 import edu.rit.sim.Simulation;
10 import edu.rit.util.AList;
11 import edu.rit.util.Random;
12
13 /**
14  * Class models a router's behavior where packets are transmitted on a
15  * preferred link if that link is available, otherwise a secondary link is
16  * chosen at random until an available link is found. If no secondary links
17  * are available, the packet is dropped.
18  *
19  * @author Jimi Ford (jhf3617)
20  * @version 5-2-2015
21  */
22 public class Router extends Routable {
23
24     // private data members
25
26     private final Random prng;
27     private Link primary;
28     private int dropCount;
29     private int receiveCount;
30     private int reRouteCount;
31     private final AList<Link> secondary;
32
33     /**
34      * Construct a router object
35      *
36      * @param prng the pseudorandom number generator to use for choosing what
37      * secondary routables to use
38      * @param sim the simulation object this router should be associated with
39      */
40     public Router(Random prng, Simulation sim) {
41         super(sim);
42         this.prng = prng;
43         this.dropCount = 0;
44         this.receiveCount = 0;
45         this.reRouteCount = 0;
46         this.secondary = new AList<Link>();
47     }
48
49     /**
50      * Set the primary link this router should prefer to send its received
51      * packets on
52      *
53      * @param link the link to prioritize
54      */
55     public void setPrimary(Link link) {
56         this.primary = link;
57     }
58 }
```



```

59  /**
60   * add a secondary link to the list of secondary links
61   *
62   * @param link the link to add
63   */
64  public void addSecondary(Link link) {
65      this.secondary.addLast(link);
66  }
67
68  /**
69   * Called when this routable object finished receiving a packet on a certain
70   * link
71   * @param packet the packet this object received
72   * @param link the link that the packet was received on
73   */
74  public void receivePacket(final Packet packet, final Link l) {
75      l.open();
76      Link link = null;
77      ++receiveCount;
78      boolean goodToGo = false;
79      if(primary.ready()) {
80          goodToGo = true;
81          link = primary;
82      } else if(secondary.size() > 0) {
83
84          int[] indices = ShuffleHelper.shuffledArray(prng, secondary.size());
85          for(int i = 0; i < indices.length && !goodToGo; i++) {
86              link = secondary.get(indices[i]);
87              if(link.ready()) {
88                  goodToGo = true;
89                  ++reRouteCount;
90              }
91          }
92      }
93      if(goodToGo) {
94          startSending(packet, link);
95      } else {
96          // drop packet
97          ++dropCount;
98      }
99  }
100
101  /**
102   * Get the fraction of packets that this router dropped
103   *
104   * @param totalPacketCount the total number of packets generated in the
105   * simulation
106   * @return a number between 0 and 1
107   */
108  public double dropFraction(int totalPacketCount) {
109      return ((double)this.dropCount)/((double)totalPacketCount;
110  }
111
112  /**
113   * Get the fraction of the packets that the router had to re-route along
114   * a secondary route
115   */
116  public double reRouteFraction() {

```

```
117         return receiveCount == 0 ? 0 :  
118             ((double)reRouteCount)/((double)receiveCount;  
119     }  
120 }  
121
```

## ShuffleHelper.java

```
1 //*****
2 //
3 // File:    ShuffleHelper.java
4 // Package: ---
5 // Unit:    Class ShuffleHelper
6 //
7 //*****
8
9 import edu.rit.util.Random;
10
11 /**
12  * Class provides helper methods for picking secondary links to transmit
13  * packets on when primary links are currently in use.
14  *
15  * @author Jimi Ford (jhf3617)
16  * @version 5-3-2015
17  */
18 public class ShuffleHelper {
19
20     /**
21      * Shuffle an array in place
22      *
23      * @param prng pseudorandom number generator used with shuffling
24      * @param array the array to shuffle
25      */
26     private static void shuffleArray(Random prng, int[] array) {
27         for (int i = array.length - 1; i > 0; i--) {
28             int index = prng.nextInt(i + 1);
29             int a = array[index];
30             array[index] = array[i];
31             array[i] = a;
32         }
33     }
34
35     /**
36      * Create an array with <I>size</I> elements ranging from 0 to
37      * <I>size - 1</I>.
38      *
39      * @param size the number of elements the array should contain
40      * @return the array containing elements from 0 to <I>size - 1</I>
41      */
42     private static int[] indexArray(int size) {
43         int[] retval = new int[size];
44         for (int i = 0; i < size; i++) {
45             retval[i] = i;
46         }
47         return retval;
48     }
49
50     /**
51      * Create a shuffled array with <I>size</I> elements ranging from 0 to
52      * <I>size - 1</I>.
53      *
54      * @param prng pseudorandom number generator used for shuffling
55      * @param size number of elements to contain
56      * @return the shuffled array
57      */
58     public static int[] shuffledArray(Random prng, int size) {
```

ShuffleHelper.java

```
59     int[] arr = indexArray(size);
60     shuffleArray(prng, arr);
61     return arr;
62 }
63 }
64
```