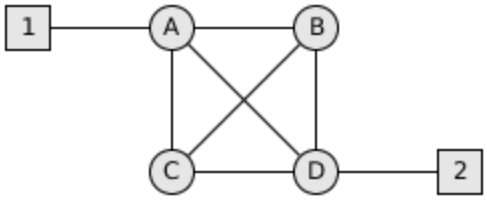
Jimi Ford
CSCI 351-01 Data Communications and Networks
Programming Project 3 Report

1. Program Inputs, Program Objective, Program Outputs

MrPotatoHead.java



This class is responsible for simulating a certain configuration of hosts and routers that transmit packets along bidirectional links between other routers and hosts. It takes in a lower bound for the smallest mean packet arrival/generation rate, and an upper bound for the quickest mean arrival rate to simulate. A mean packet rate delta is passed in to the program to tell it what delta to sweep across the packet rates at. Another parameter (or knob) that gets passed in to this program is the total number of packets to generate. The final parameter that this program uses is a seed for Prof. Alan Kaminsky's pseudorandom number generator. (Additionally, the user may specify a 6^{th} argument for the file prefix for each saved file.) The objective of this program is to study the effect that the rate at which routers/hosts receive packets has on the packet traversal time and the drop fraction within the above network that utilizes hot potato routing.

PlotHandler.java

This program contains the necessary methods to write a plot file given a set of data and then display the results of that plot file. The input to this program is what plot(s) file to display and the output is a graphical representation of the given plot file(s). The objective of this program is to produce a visual or graphical representation of the results of running the above program.

2. Exact Command Line

MrPotatoHead.java

```
Usage: java MrPotatoHead <rlb> <rub> <rdelta> <npkt> <seed>
<file-prefix>
<rlb>       - Mean packet rate lower bound
<rub>       - Mean packet rate upper bound
```

`<rdelta>` - Mean packet rate delta
`<npkt>` - Number of packets
`<seed>` - Random seed
`<file-prefix>` - Optional file prefix of saved graphs and tables

PlotHandler.java

note: This program allows for any number (greater than 0) of plot files to be specified in the command line arguments.

```
usage:
java PlotHandler <plot-file-1> (<plot-file-2> <plot-file-3>... etc.)
```

`<plot-file-1>` - The plot file (generated by MrPotatoHead) to visualize in an X-Y plot

3. Source Code *(See Appendix B for project's source code)*

---

4. Based on data generated by running your simulation program(s), discuss this question: What happens to the packet traversal time as the mean packet arrival rate increases, and why is the packet traversal time behaving this way?

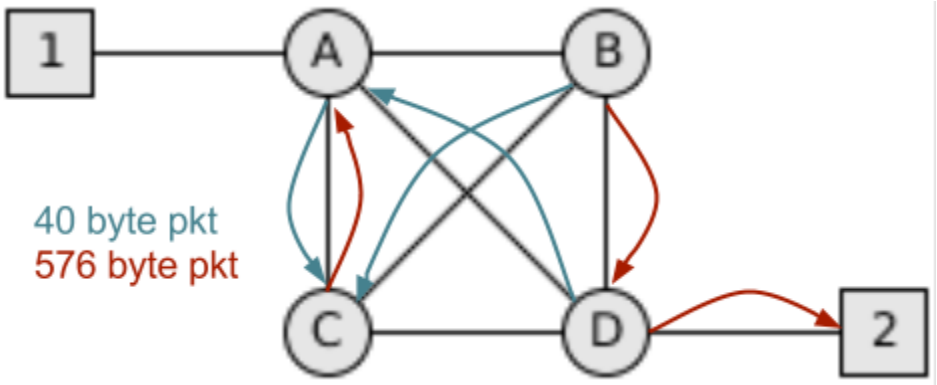
In *Q5 Supporting Data*, we observe the same recurring graph shape from the previous projects. The traversal time starts at a relatively low value, increases drastically to a global maximum, and then gradually decreases asymptotically to some value as the arrival rate approaches infinity.

The average traversal time for very small arrival rates starts at roughly the expected/theoretical value of 0.545 seconds. The average size of each packet generated is $8 * \frac{576+40}{2} = 2,464\ bits.$ Assuming no blockages (where a blockage is defined as a router not being able to transmit to its prioritized link due to a pre existing packet in transmission on that link), we would expect each packet to only hop from *1* to *A* taking up no time (due to the incredibly high bitrate), then *A* to *D* taking $0.25\overline{6}\ seconds,$ then *D* to *2* taking an additional $0.25\overline{6}\ seconds$ resulting in a total theoretical traversal time of $0.51\overline{3}\ seconds$. Since this value is very close to our observed value, it accurately explains the starting value. Note that had we started sweeping the mean arrival rate at a value closer to 0, we would see our observed traversal time start at a value much closer to the theoretical traversal time. The reason it isn't exact is because at a mean arrival rate of $0.25\ \frac{packets}{second},$ we will still see a very small amount of collisions, resulting in extra hops that some packets have to take. We know this is the cause of the difference between the expected and the observed values by looking at the observed traversal time for the smaller packets (i.e. packets that are only 40 bytes). At $0.25\ \frac{packets}{second},$ we

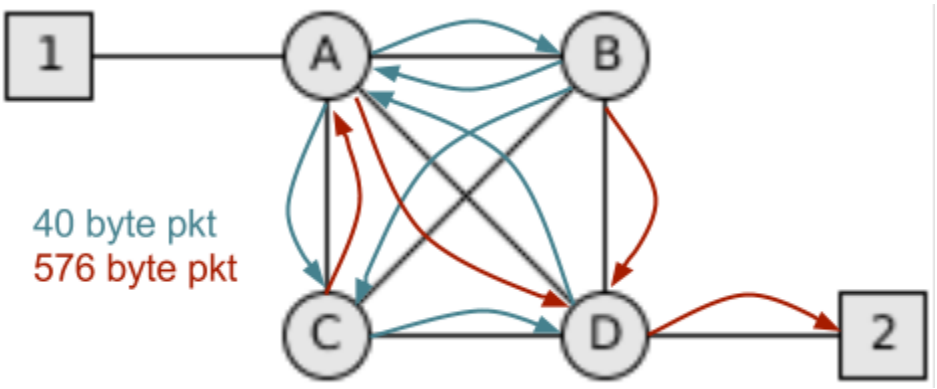
see that the smaller packets take $0.087\ seconds$ on average to finish transmitting. Note that the table data shows that 0 small packets were dropped at this rate. The theoretical amount of time that these small packets should be taking per hop (for low mean rate) is $\frac{40*8}{9600} = \frac{320}{9600} = 0.0\overline{3}\ seconds.$ So 2 hops consisting of $A \rightarrow D$ and $D \rightarrow 2$ should take $0.0\overline{6}\ seconds.$ Since we observe small packets taking $0.087\ seconds$ at a mean arrival rate of $0.25\ \frac{packets}{second},$ 30.5% higher than the expected, we know that some of these packets are being blocked from their prioritized route, causing the average traversal time to be fairly higher than the theoretical time since the packet has to hop to an intermediary routers while it waits for prioritized links to open. A better way to put it is, $0.25\ \frac{packets}{second}$ is not a low enough mean rate to ensure no blockages occur. It is interesting to point out that the larger packets' theoretical traversal time is $576 * 8/9600 = 0.96\ seconds.$ What we observe is $1.002\ seconds,$ which is only 4.38% higher than the expected value.

Next we see that the average traversal time for all packets rapidly increases to a global maximum. It is very intuitive that the traversal time increases since we are increasing the chance that a new packet will arrive at a router X while its primary links are busy. Another way of thinking of this is that we are increasing the average distance or average number of hops a packet must take to traverse all the way through the network by forcing it to take detours. It takes slightly more thorough examination to determine what's causing the rapid increase to cap off at a global maximum.

During the beginning of rapid increase in traversal time, if we took a snapshot of the state the network was in, chances are it would look something like this a majority of the time.

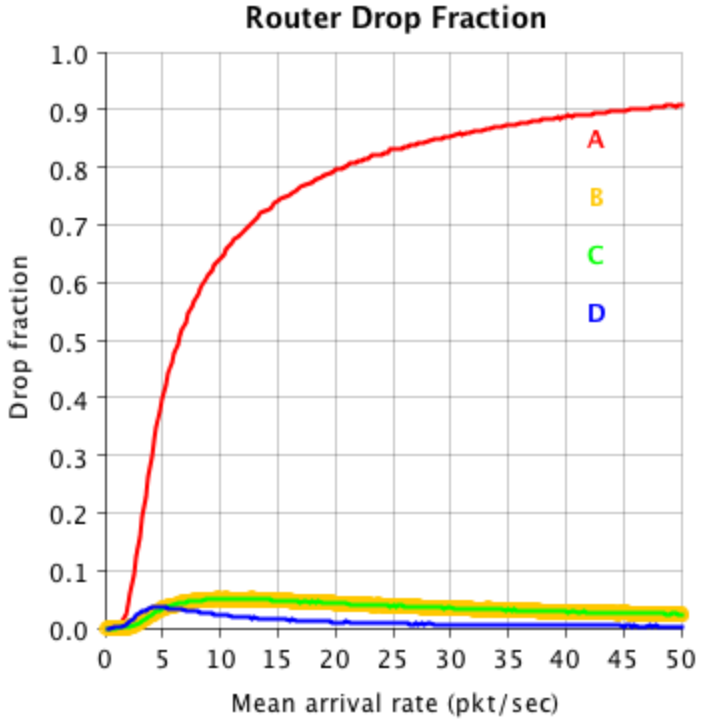


Each arrow represents a packet that is currently being transmitted. The idea I'm trying to convey here is that the more we increase the rate at which 1 generates and sends packets to $A,$ the more links we will have that are closed off at any given time. Eventually we will reach a point where the snapshots begin to look something like this as the mean arrival rate increases.

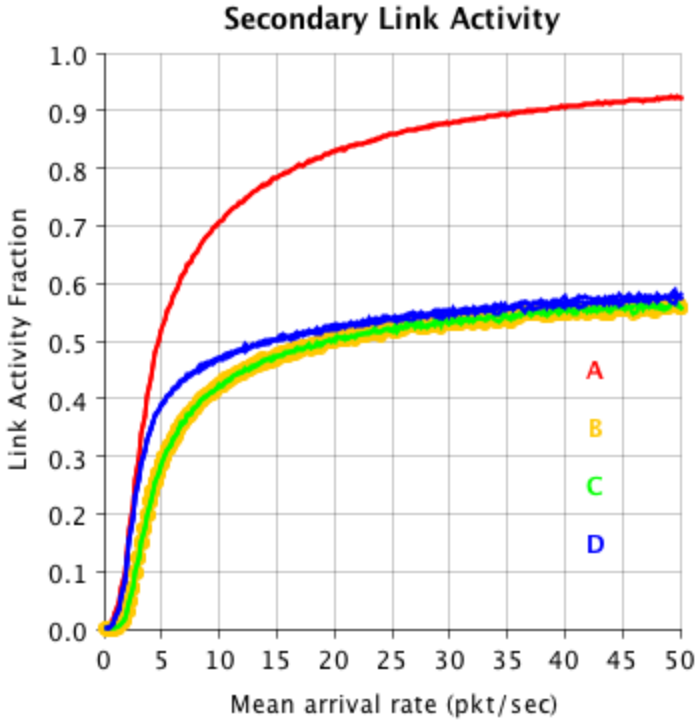


I will refer to this as *nearly saturated*. If a packet comes in to router A from $Host\ 1$, the router will have to drop it since all of its outgoing links are occupied. Once the network reaches this state, the link from D to $Host\ 2$ will be active nearly 100% of the time since that is $D's$ primary link preference and D will be bombarded with incoming packets since every other router prefers D. Packets that aren't dropped while the network is in this state will bounce along secondary routes, eventually finding its way to $D's$ primary connection to $Host\ 2$.

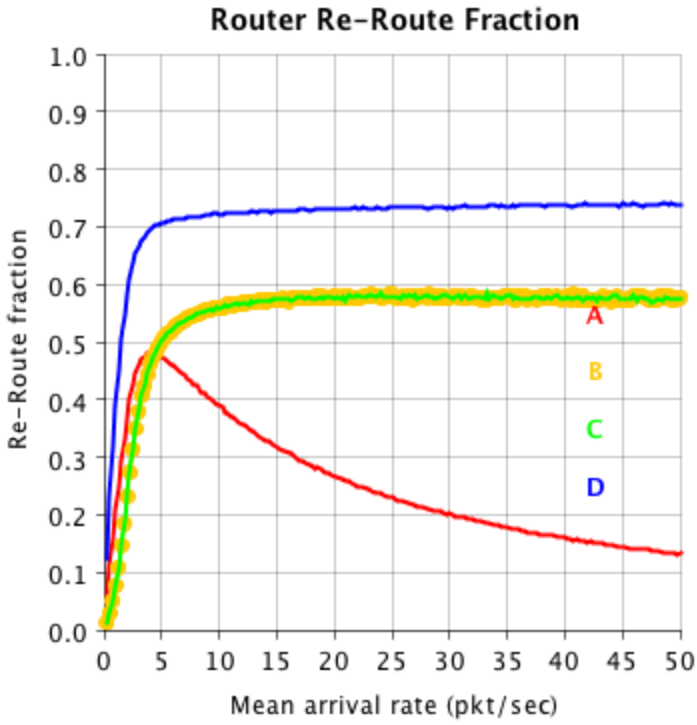
Now the fascinating observation we make is that even as we bombard the network with more and more packets at higher and higher rates, the average traversal time decreases as the mean arrival rate increases. This was completely counterintuitive for me. It caused me to go back and second guess my model for a while. At these rates, *A* will just drop the incoming packets from *Host 1* before they enter the network. We can see that in *Q5 Supporting Data* in the drop fraction plot.



A is reducing the load on the other routers because it is dropping almost every packet that comes in. The reason it is dropping so many packets is that it has incoming packets from *Host 1* at an extremely high rate and in addition, it has incoming packets that are getting re-routed from *B, C,* and *D*. Taking a look at *A's* secondary links, we see that they become very active as the mean arrival rate increases.



Every time a router chooses a secondary link since the primary link is closed, we will call this a re-route. Examining the re-route fraction, we see that *A's* secondary links become so active that *A* doesn't have the option to re-route anymore.



The other router's re-route fraction levels off sending a portion of those re-routes back to *A*, causing the links to stay busy and *A's* drop fraction to skyrocket as we saw earlier. This is what's causing the traversal time to decrease as the arrival rate increases. Most of the packets get dropped even after being re-routed.

5. Supporting Data

---

note: In order to save you time while grading, lower numbers of simulations are used. As a result, the plots produced are not extremely smooth but are smooth enough to convey the general trend of traversal time. The results in ***table form*** *that we are concerned with are in Appendix A under the columns labeled "Resp Time [Total, Large, Small]".*

Commands used:

This first command will run the discrete event simulation and save *.dwg* files as well as a *.tsv* file with the prefix *"potato"*. (See Appendix A for this table data.)

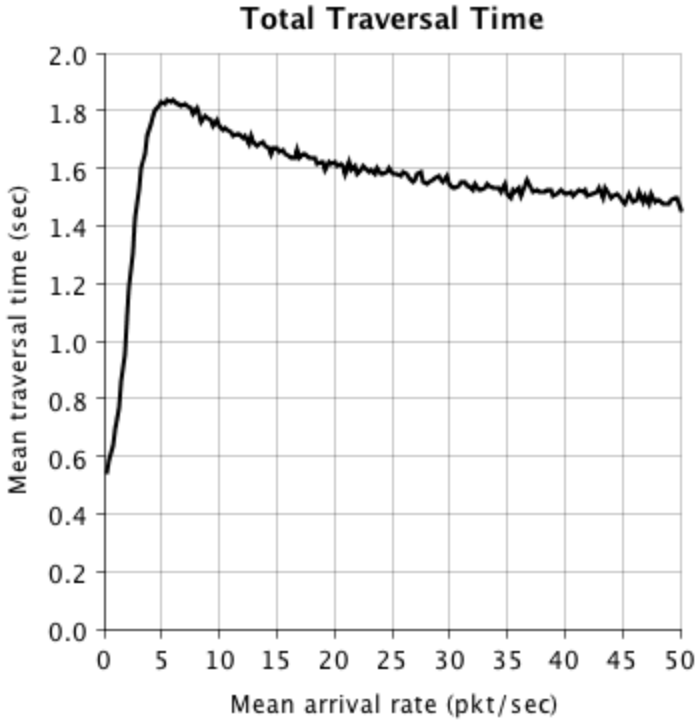
```
java MrPotatoHead 0 50 .25 200000 100123456789 potato
```

(expected runtime 30 seconds on 2.93 GHz)

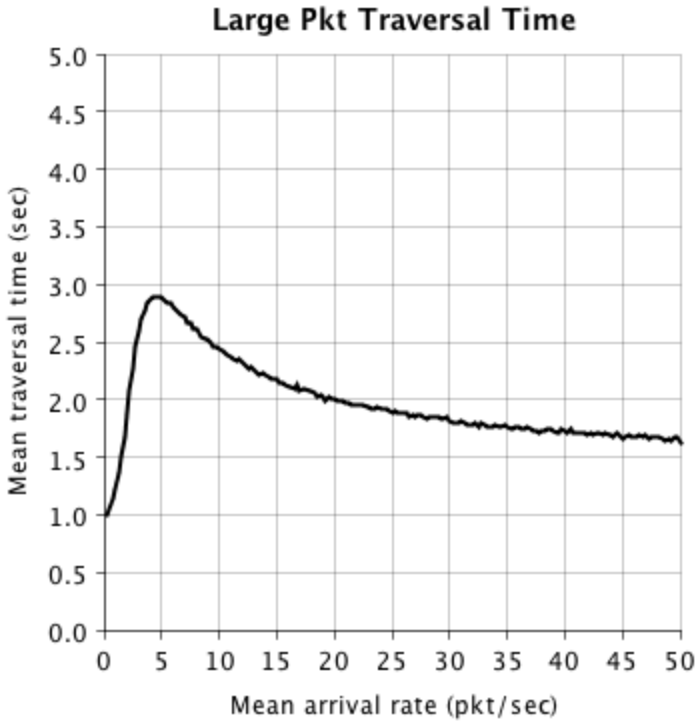
After generating the *.dwg* files, you must run the following command to visualize the plots.

```
java PlotHandler potato-traversal-time.dwg
potato-traversal-time-large.dwg potato-traversal-time-small.dwg
```

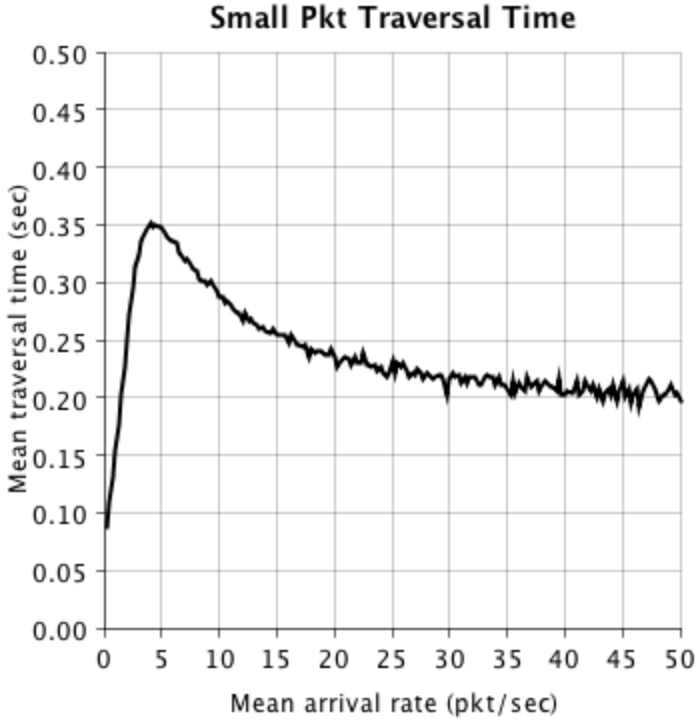
This graph below is a plot of mean (or average) traversal time for all packets generated as a function of the mean arrival rate.



This graph below is a plot of mean (or average) traversal time for packets of $SIZE = 576$ *bytes* as a function of the mean arrival rate.

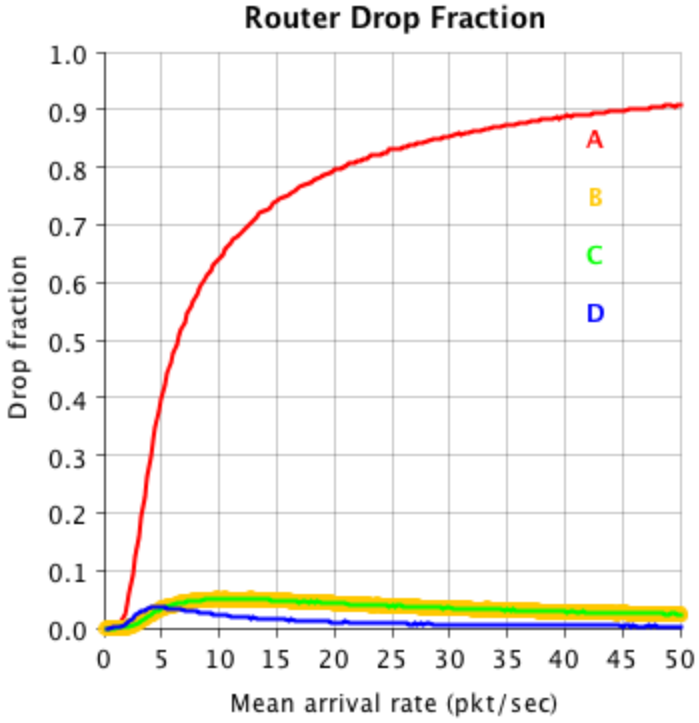


This graph below is a plot of mean (or average) traversal time for packets of $SIZE = 40\ bytes$ as a function of the mean arrival rate.



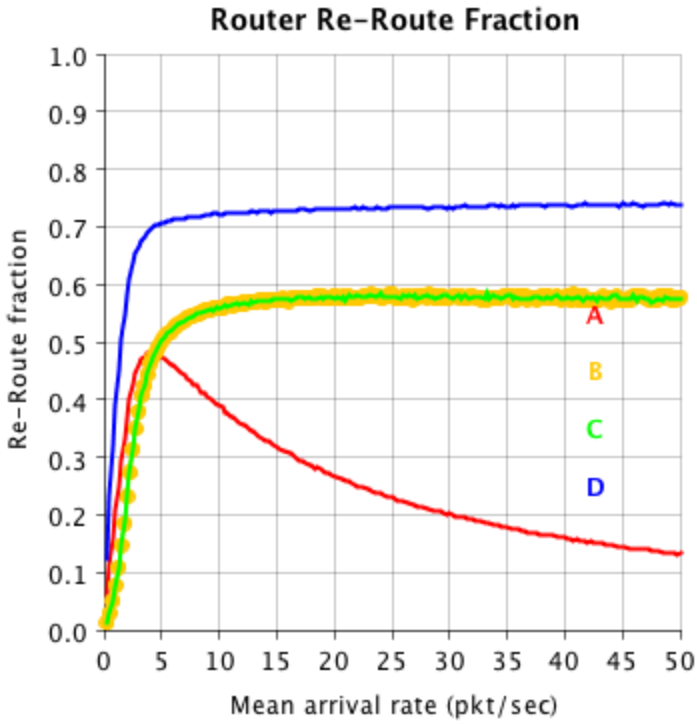
To view the individualized drop fraction of each router, run this command.

```
java PlotHandler potato-router-drop-fraction.dwg
```



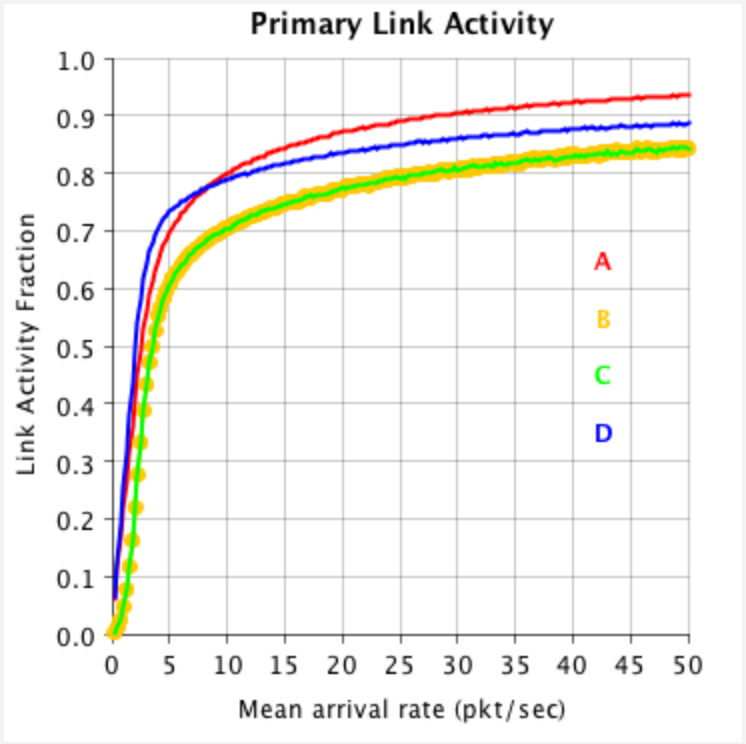
To view the individualized re-route fraction of each router, run this command. This is the fraction of packets that the routers forward along secondary links instead of their primary links.

```
java PlotHandler potato-re-route-fraction.dwg
```



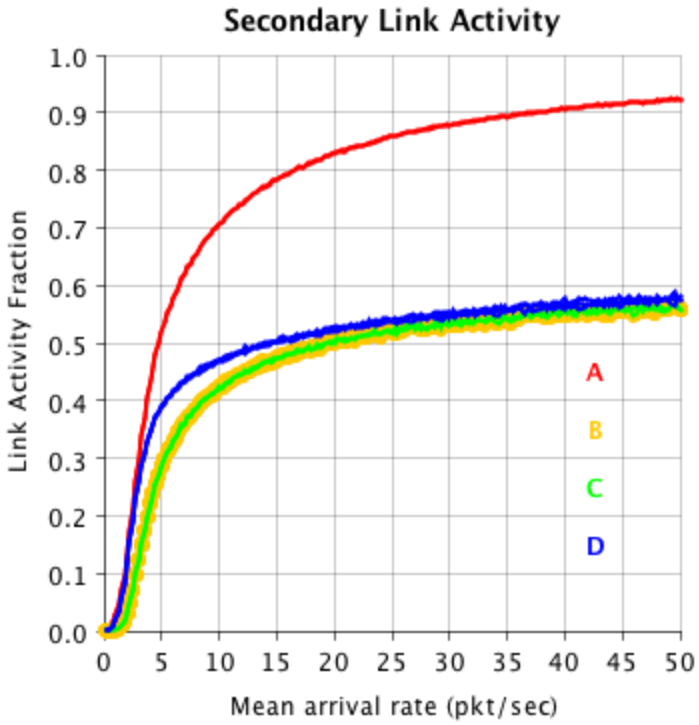
To view the individualized primary link activity fraction of each router, run this command. This is the fraction of time the router's primary outgoing link is spent transmitting a packet.

```
java PlotHandler potato-primary-link-activity-fraction.dwg
```



To view the individualized secondary link activity fraction of each router, run this command. This is the fraction of time the router's secondary outgoing links is spent transmitting a packet.

```
java PlotHandler potato-secondary-link-activity-fraction.dwg
```

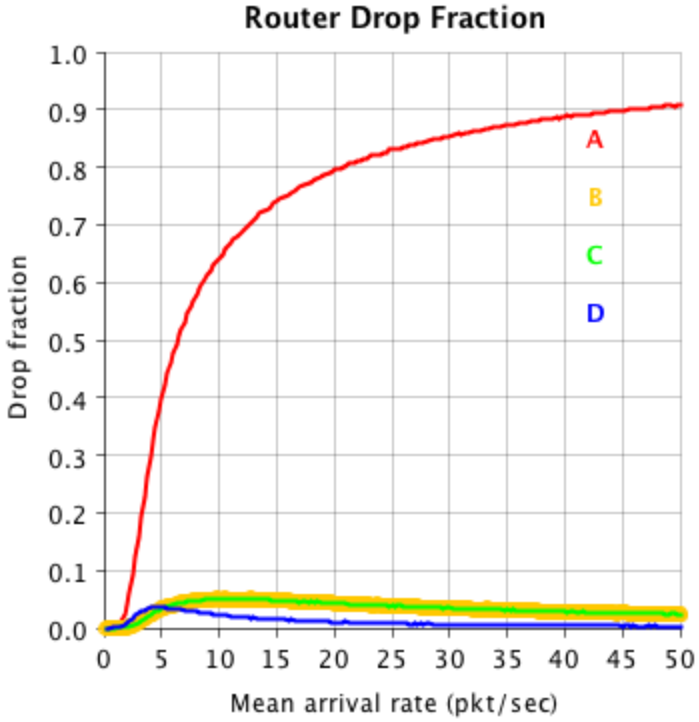


6. Based on data generated by running your simulation program(s), discuss this question: What happens to the packet drop fraction as the mean packet arrival rate increases, and why is the packet drop fraction behaving this way?

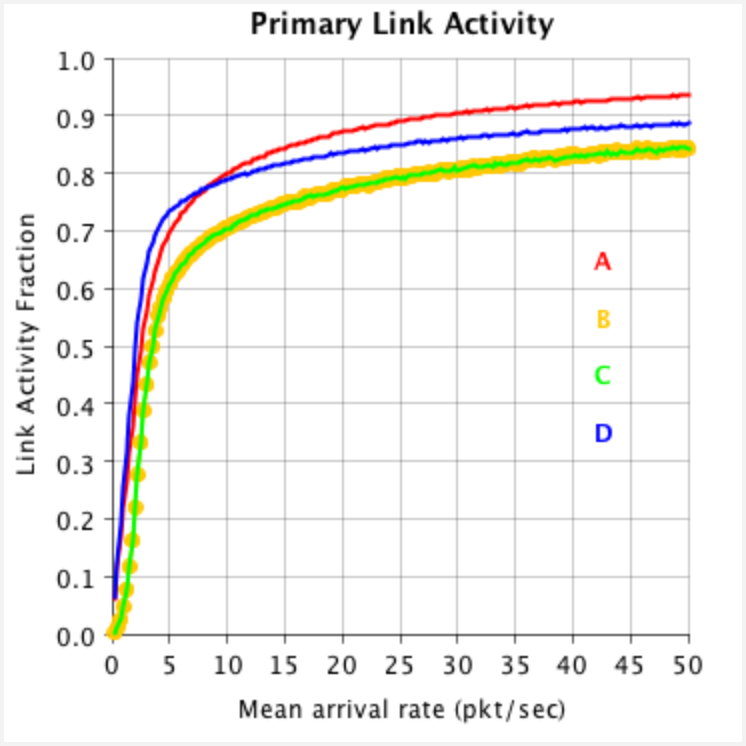
Observing *Q7. Supporting Data*, we see that the drop fraction of the network as a whole starts very low, increases rapidly and then approaches 100% as the mean arrival rate approaches infinity. This is result is much more intuitive than the average traversal time results. Let's look a little deeper into what's causing this behavior. It is very intuitive that the drop fraction starts at 0 because at such a low mean arrival rate, most of the packets have likely made it to *Host 2* before a new packet is sent from *Host 1* to *A*. This results in no "collisions" or blockages because many links are available for packets to be re-routed if need

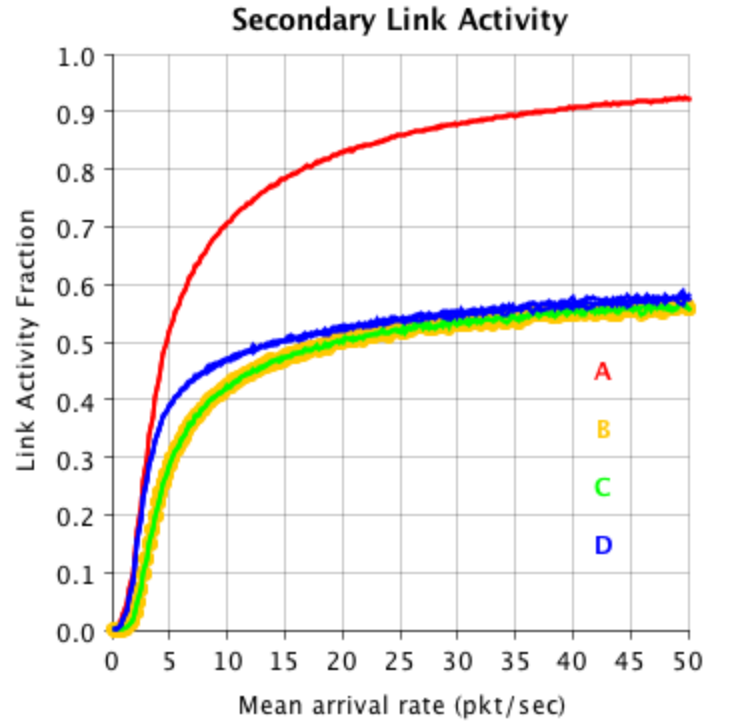
be. Examining the packet sizes and their respective drop fractions individually, we see that nearly 100% of small packets are being dropped at 50 packets/second where only 92% of the large packets are getting dropped. This was another result that I thought was incorrect at first and really made me think about why this happens. As we pointed out in *Q4*, the small packets contribute more to the total drop fraction than the large packets do. This was backwards from what my prediction was. The reason that smaller packets are dropped more than large packets (as the mean arrival rate increases) is because once a large packet is being transmitted on router *X's* link *L*, many small packets could come into *X* while this large packet is still being transmitted. Each time a router "commits" to transmitting a large packet, it is at an increased risk of dropping packets since many small packets (which move quickly through the network) can arrive at this router and congest the traffic causing many of them to be dropped.

The reason that total packets dropped increases as mean arrival rate increases is mainly because of router *A*'s inability to keep up with such high traffic.

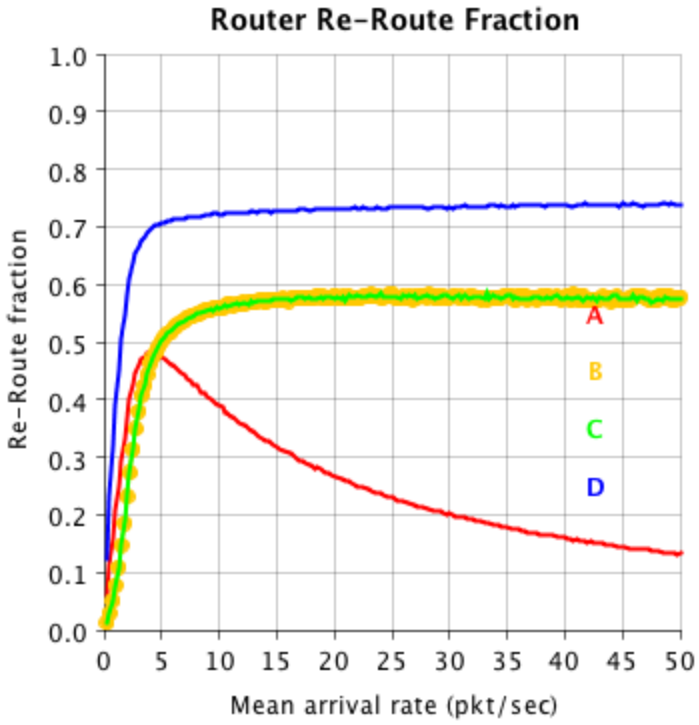


Here we see that because *A* is receiving so many packets from *Host 1* and from all the other routers when packets get re-routed, *A*'s links are closed for a majority of the time. We can confirm that with the data collected on link activity.





Here we see that *A* doesn't even have the option to re-route packets because all of its outgoing links are tied up. We can confirm this claim with the data collected on re-route fraction of router *A*.



Since *A* is so overloaded, it resorts to dropping the incoming packets instead of re-routing them because it doesn't even have the option.

7. Supporting Data

---

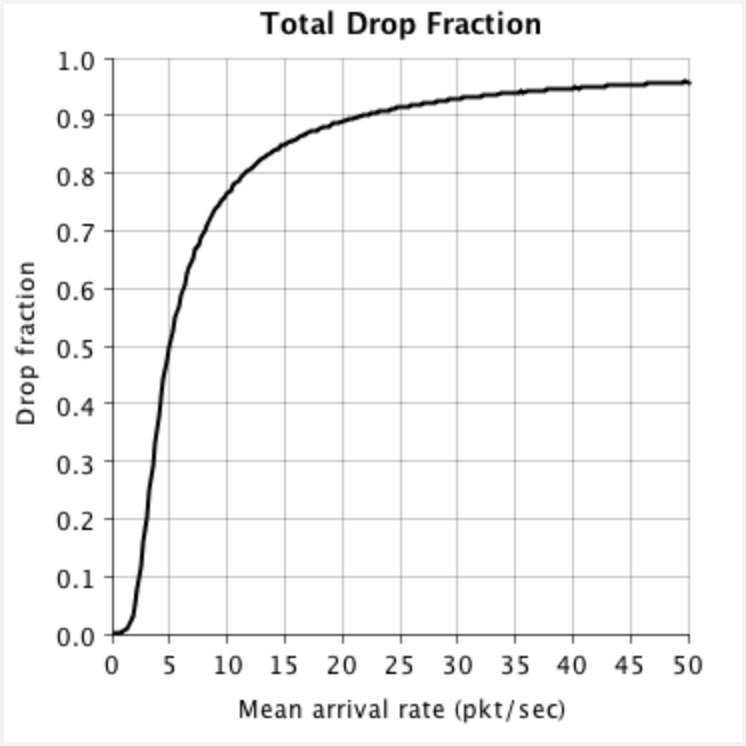
note: In order to save you time while grading, lower numbers of simulations are used. As a result, the plots produced are not extremely smooth but are smooth enough to convey the general trend of the drop fraction . The table results are found in Appendix A under the columns labeled "Drop Frac [Total, Large, Small]".

Commands used:

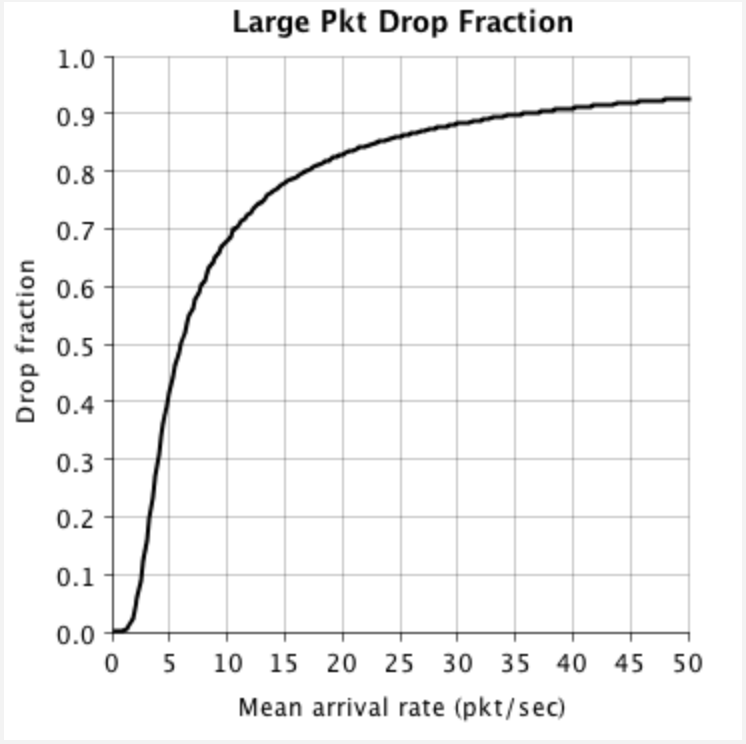
We will use the results generated from the previous run and just view the plot file generated by it:

```
java PlotHandler potato-drop-fraction.dwg
potato-drop-fraction-large.dwg potato-drop-fraction-small.dwg
```

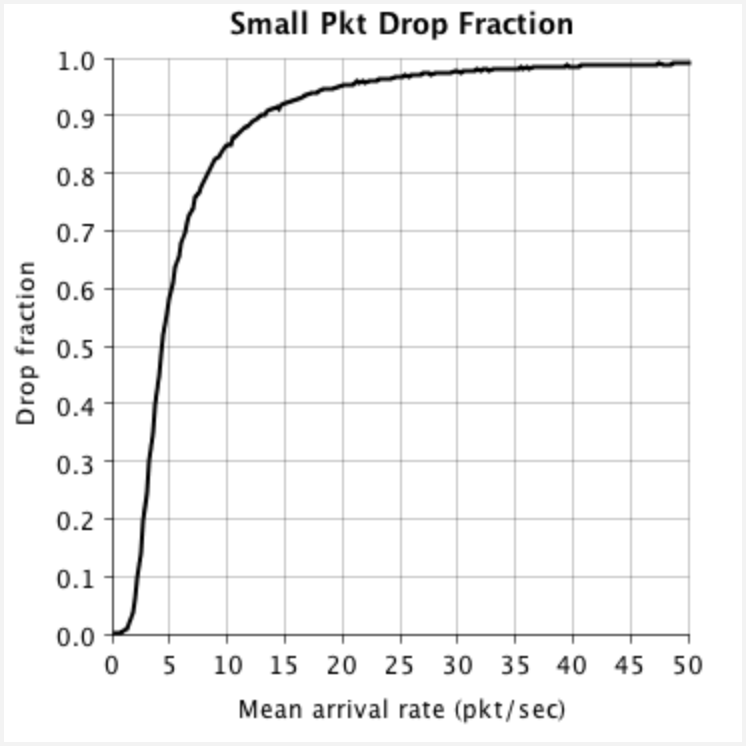
This graph below is a plot of the fraction of all packets that were dropped as a function of the mean arrival rate at which packets were generated.



This graph below is a plot of the fraction of the large packets (i.e. 576 bytes) that were dropped as a function of the mean arrival rate at which packets were generated.

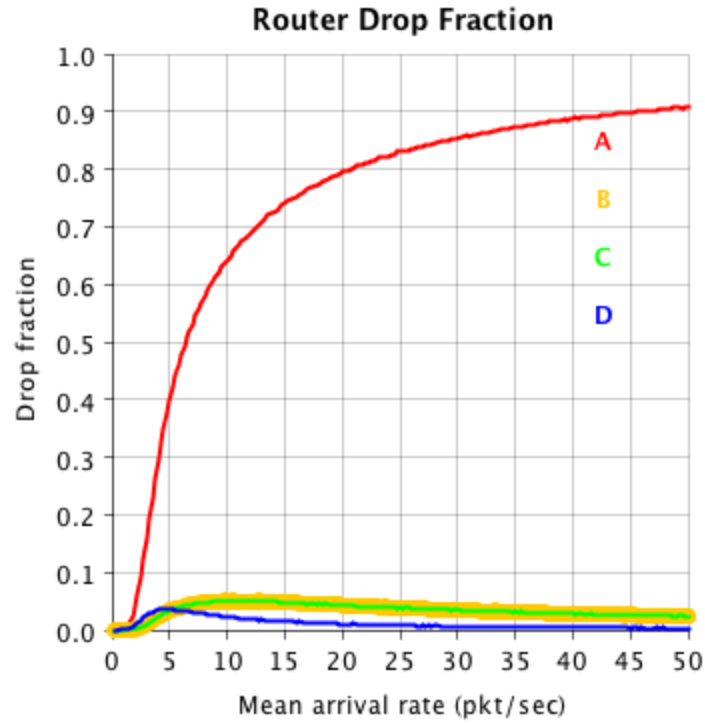


This graph below is a plot of the fraction of the small packets (i.e. 40 bytes) that were dropped as a function of the mean arrival rate at which packets were generated.



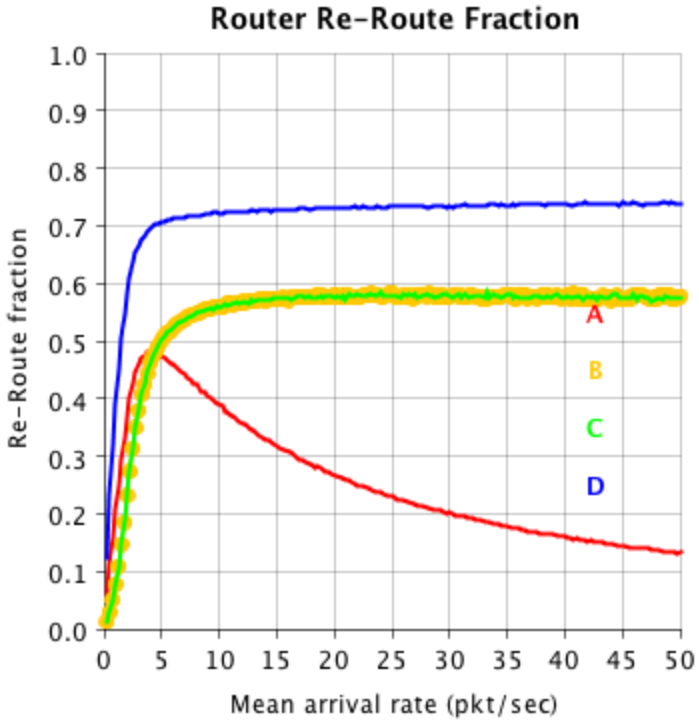
To view the individualized drop fraction of each router, run this command.

```
java PlotHandler potato-router-drop-fraction.dwg
```



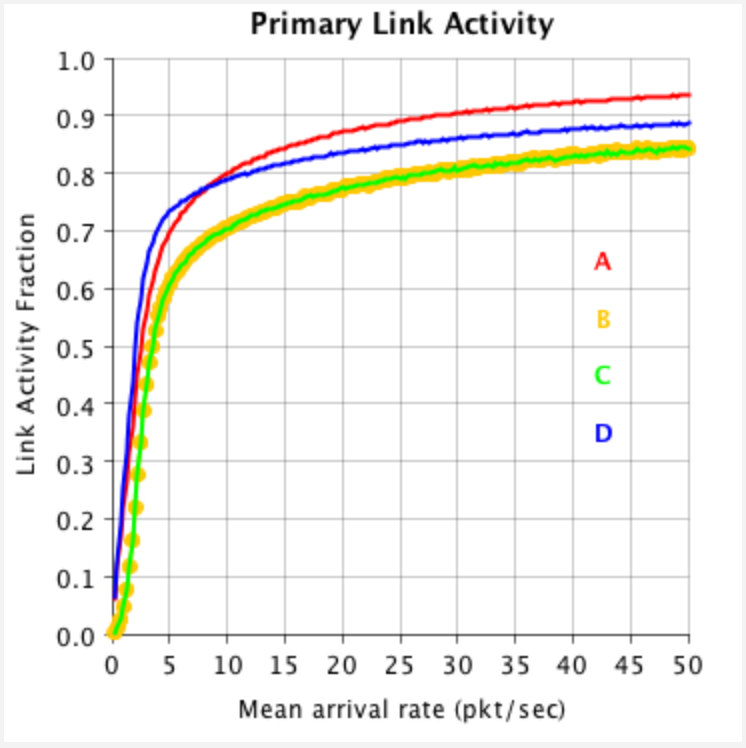
To view the individualized re-route fraction of each router, run this command. This is the fraction of packets that the routers forward along secondary links instead of their primary links.

```
java PlotHandler potato-re-route-fraction.dwg
```



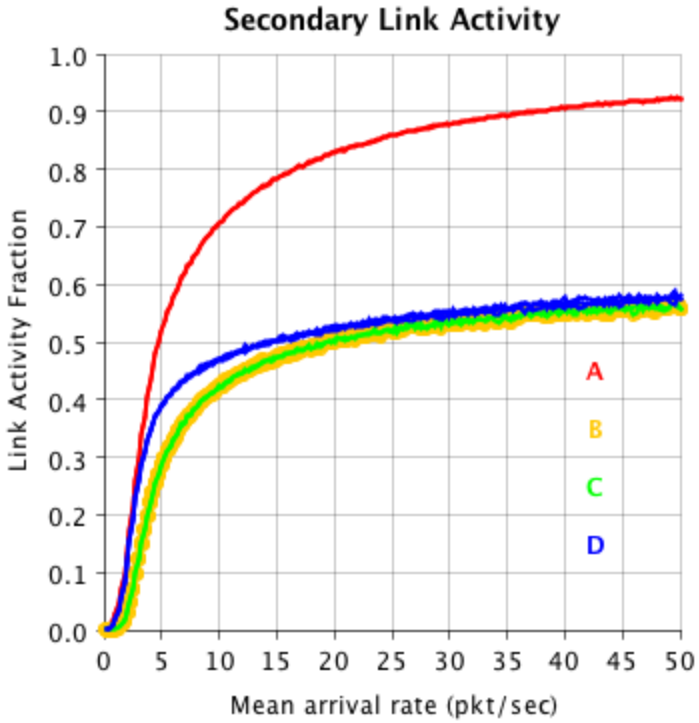
To view the individualized primary link activity fraction of each router, run this command. This is the fraction of time the router's primary outgoing link is spent transmitting a packet.

```
java PlotHandler potato-primary-link-activity-fraction.dwg
```



To view the individualized secondary link activity fraction of each router, run this command. This is the fraction of time the router's secondary outgoing links is spent transmitting a packet.

```
java PlotHandler potato-secondary-link-activity-fraction.dwg
```



8. What I learned

I know I said this about project 3, but this project was truly my favorite project we've done so far. I have never simulated time in a program before, so it was very satisfying to finally learn how to do that. I learned that Hot-Potato routing is not at all close to ideal for an infrastructure that demands a low number of dropped packets in high network traffic. I also got a little better at using Prof. Alan Kaminsky's Plot library. Until this project, I didn't realize it supported multiple plots on the same "image/graph" since I hadn't explored the javadoc enough and because I hadn't really needed this feature until this project. This may come as a surprise, but I learned and now understand the value of abstract classes in Java. I think this is the first project at RIT (that I can remember) that had an appropriate application of abstract classes. If past projects called for abstract classes then it was because the project was designed to teach us about them. This was the first time that I chose to use an abstract class on a project that I wasn't *forced* to use an abstract class on. In the past, I would've thought that maybe an Interface would work well, but that would've resulted in duplicate code since both routers and hosts send packets in the exact same manner. This project reinforced my understanding of how crucial the design step is in software engineering. Just a few days before the project was due, our professor clarified some things on the project that people (including myself) had misunderstood. Fortunately, my design was so easy to work with, I fixed my errors in very few lines of code. Clarifications like this are going to be a part of

everyday life in “the real world” so it’s important that my code is written in a way that modifications are easy to make.

Appendix A)

Table generated

Mean Pkt Rate	Resp Time Total	Resp Time Large	Resp Time Small	Drop Frac Total	Drop Frac Large	Drop Frac Small
0.25	0.545	1.002	0.087	0	0	0
0.5	0.584	1.058	0.108	0	0	0.001
0.75	0.634	1.133	0.131	0.001	0.001	0.001
1	0.693	1.232	0.153	0.004	0.003	0.004
1.25	0.769	1.358	0.176	0.008	0.006	0.009
1.5	0.857	1.507	0.202	0.016	0.013	0.019
1.75	0.955	1.676	0.227	0.029	0.021	0.036
2	1.08	1.886	0.25	0.05	0.038	0.062
2.25	1.193	2.08	0.272	0.078	0.06	0.096
2.5	1.308	2.273	0.294	0.115	0.088	0.141
2.75	1.429	2.448	0.313	0.159	0.124	0.195
3	1.522	2.597	0.324	0.202	0.159	0.246
3.25	1.6	2.699	0.336	0.248	0.196	0.299
3.5	1.646	2.766	0.343	0.291	0.234	0.348
3.75	1.711	2.835	0.347	0.333	0.268	0.398
4	1.756	2.876	0.352	0.378	0.309	0.446
4.25	1.78	2.894	0.349	0.411	0.337	0.485

4.5	1.8	2.894	0.35	0.441	0.364	0.519
4.75	1.817	2.887	0.349	0.475	0.393	0.557
5	1.832	2.89	0.348	0.5	0.416	0.583
5.25	1.823	2.854	0.343	0.527	0.443	0.61
5.5	1.836	2.841	0.34	0.551	0.463	0.638
5.75	1.833	2.831	0.336	0.569	0.483	0.655
6	1.836	2.8	0.336	0.59	0.5	0.679
6.25	1.824	2.764	0.334	0.607	0.518	0.696
6.5	1.823	2.752	0.325	0.626	0.536	0.714
6.75	1.819	2.729	0.324	0.638	0.549	0.727
7	1.82	2.719	0.319	0.65	0.562	0.738
7.25	1.815	2.665	0.32	0.668	0.577	0.759
7.5	1.808	2.663	0.315	0.677	0.589	0.765
7.75	1.789	2.616	0.312	0.688	0.602	0.775
8	1.806	2.617	0.31	0.7	0.611	0.789
8.25	1.786	2.582	0.303	0.71	0.623	0.797
8.5	1.762	2.543	0.301	0.719	0.633	0.804
8.75	1.782	2.534	0.302	0.73	0.642	0.818
9	1.774	2.519	0.298	0.737	0.651	0.824
9.25	1.764	2.494	0.301	0.744	0.659	0.83
9.5	1.748	2.459	0.298	0.752	0.667	0.836
9.75	1.767	2.463	0.293	0.76	0.674	0.846
10	1.745	2.44	0.287	0.765	0.68	0.849
10.25	1.729	2.421	0.288	0.769	0.686	0.85

10.5	1.743	2.405	0.283	0.779	0.699	0.862
10.75	1.731	2.388	0.284	0.783	0.701	0.865
11	1.725	2.376	0.28	0.788	0.707	0.868
11.25	1.713	2.356	0.276	0.793	0.715	0.872
11.5	1.721	2.343	0.274	0.799	0.718	0.879
11.75	1.719	2.344	0.273	0.803	0.724	0.882
12	1.703	2.311	0.267	0.807	0.729	0.886
12.25	1.711	2.299	0.274	0.812	0.734	0.891
12.5	1.683	2.265	0.267	0.816	0.74	0.893
12.75	1.709	2.279	0.268	0.821	0.744	0.898
13	1.694	2.261	0.265	0.823	0.746	0.9
13.25	1.677	2.235	0.263	0.827	0.753	0.902
13.5	1.684	2.217	0.26	0.833	0.758	0.908
13.75	1.688	2.225	0.262	0.835	0.761	0.91
14	1.675	2.206	0.258	0.838	0.765	0.912
14.25	1.673	2.197	0.256	0.841	0.768	0.914
14.5	1.646	2.176	0.257	0.843	0.773	0.913
14.75	1.671	2.178	0.259	0.848	0.778	0.919
15	1.669	2.172	0.254	0.85	0.778	0.921
15.25	1.656	2.151	0.255	0.852	0.782	0.923
15.5	1.665	2.148	0.254	0.856	0.785	0.926
15.75	1.652	2.135	0.254	0.858	0.788	0.927
16	1.639	2.114	0.248	0.86	0.791	0.929
16.25	1.635	2.104	0.254	0.862	0.794	0.93

16.5	1.634	2.095	0.249	0.865	0.798	0.932
16.75	1.666	2.126	0.246	0.867	0.799	0.935
17	1.635	2.08	0.246	0.869	0.802	0.937
17.25	1.648	2.087	0.245	0.872	0.804	0.939
17.5	1.647	2.084	0.246	0.873	0.806	0.94
17.75	1.633	2.077	0.236	0.875	0.81	0.939
18	1.635	2.068	0.243	0.877	0.811	0.942
18.25	1.635	2.049	0.239	0.88	0.815	0.945
18.5	1.615	2.027	0.241	0.881	0.816	0.945
18.75	1.618	2.038	0.241	0.882	0.819	0.945
19	1.624	2.028	0.239	0.884	0.821	0.948
19.25	1.595	1.996	0.238	0.886	0.824	0.948
19.5	1.621	2.021	0.237	0.887	0.825	0.95
19.75	1.613	2.006	0.243	0.889	0.827	0.95
20	1.624	2.013	0.234	0.891	0.83	0.952
20.25	1.604	1.989	0.227	0.892	0.831	0.953
20.5	1.618	1.995	0.232	0.894	0.833	0.954
20.75	1.61	1.99	0.233	0.894	0.835	0.954
21	1.581	1.971	0.236	0.895	0.836	0.953
21.25	1.624	1.975	0.234	0.898	0.838	0.959
21.5	1.592	1.953	0.228	0.899	0.841	0.958
21.75	1.606	1.957	0.235	0.9	0.841	0.959
22	1.583	1.955	0.23	0.9	0.844	0.957
22.25	1.594	1.953	0.231	0.902	0.845	0.959

22.5	1.609	1.952	0.238	0.904	0.847	0.962
22.75	1.591	1.937	0.227	0.905	0.848	0.962
23	1.588	1.934	0.228	0.906	0.85	0.962
23.25	1.596	1.927	0.227	0.908	0.851	0.964
23.5	1.583	1.914	0.228	0.908	0.853	0.964
23.75	1.601	1.939	0.223	0.909	0.854	0.964
24	1.582	1.92	0.227	0.909	0.855	0.964
24.25	1.581	1.911	0.221	0.911	0.856	0.965
24.5	1.588	1.912	0.218	0.912	0.858	0.966
24.75	1.598	1.904	0.222	0.914	0.86	0.969
25	1.579	1.888	0.233	0.914	0.861	0.968
25.25	1.577	1.894	0.222	0.915	0.862	0.968
25.5	1.583	1.891	0.23	0.916	0.863	0.969
25.75	1.571	1.882	0.227	0.916	0.864	0.969
26	1.586	1.889	0.23	0.918	0.866	0.97
26.25	1.581	1.881	0.224	0.918	0.866	0.97
26.5	1.565	1.857	0.218	0.919	0.868	0.971
26.75	1.551	1.862	0.222	0.919	0.869	0.969
27	1.559	1.841	0.219	0.921	0.87	0.973
27.25	1.581	1.865	0.225	0.922	0.871	0.973
27.5	1.586	1.865	0.222	0.923	0.873	0.974
27.75	1.555	1.853	0.217	0.922	0.873	0.972
28	1.544	1.833	0.222	0.924	0.875	0.973
28.25	1.553	1.84	0.219	0.924	0.875	0.973

28.5	1.561	1.841	0.216	0.926	0.877	0.974
28.75	1.568	1.848	0.217	0.926	0.877	0.975
29	1.57	1.849	0.22	0.926	0.878	0.975
29.25	1.561	1.835	0.22	0.927	0.879	0.975
29.5	1.549	1.824	0.216	0.928	0.88	0.975
29.75	1.571	1.842	0.203	0.929	0.881	0.976
30	1.544	1.808	0.216	0.929	0.882	0.977
30.25	1.531	1.803	0.222	0.929	0.884	0.975
30.5	1.532	1.789	0.217	0.931	0.884	0.977
30.75	1.541	1.798	0.219	0.931	0.885	0.977
31	1.549	1.816	0.213	0.931	0.885	0.977
31.25	1.552	1.797	0.218	0.933	0.886	0.979
31.5	1.535	1.784	0.212	0.933	0.888	0.979
31.75	1.539	1.78	0.217	0.934	0.888	0.98
32	1.526	1.779	0.218	0.934	0.888	0.979
32.25	1.547	1.798	0.218	0.934	0.889	0.979
32.5	1.523	1.77	0.212	0.935	0.89	0.979
32.75	1.523	1.79	0.211	0.935	0.891	0.978
33	1.53	1.779	0.218	0.936	0.892	0.98
33.25	1.547	1.77	0.219	0.937	0.893	0.982
33.5	1.538	1.77	0.218	0.937	0.893	0.981
33.75	1.53	1.764	0.218	0.938	0.894	0.981
34	1.533	1.772	0.211	0.938	0.895	0.981
34.25	1.532	1.768	0.218	0.939	0.896	0.981

34.5	1.518	1.759	0.21	0.939	0.897	0.981
34.75	1.543	1.772	0.21	0.94	0.897	0.982
35	1.513	1.768	0.207	0.939	0.898	0.98
35.25	1.498	1.743	0.202	0.94	0.898	0.981
35.5	1.526	1.74	0.214	0.941	0.899	0.984
35.75	1.52	1.759	0.205	0.941	0.9	0.982
36	1.532	1.759	0.212	0.941	0.9	0.983
36.25	1.505	1.745	0.21	0.942	0.901	0.982
36.5	1.536	1.748	0.206	0.943	0.902	0.984
36.75	1.556	1.769	0.219	0.943	0.903	0.984
37	1.528	1.742	0.21	0.944	0.903	0.984
37.25	1.514	1.728	0.211	0.944	0.903	0.984
37.5	1.525	1.735	0.214	0.944	0.904	0.985
37.75	1.515	1.718	0.206	0.945	0.905	0.985
38	1.515	1.729	0.209	0.945	0.905	0.984
38.25	1.518	1.732	0.215	0.945	0.906	0.985
38.5	1.528	1.74	0.213	0.946	0.907	0.985
38.75	1.523	1.741	0.21	0.946	0.907	0.985
39	1.503	1.728	0.209	0.946	0.907	0.984
39.25	1.511	1.717	0.206	0.947	0.908	0.985
39.5	1.518	1.716	0.216	0.947	0.908	0.986
39.75	1.527	1.739	0.202	0.947	0.909	0.986
40	1.511	1.726	0.203	0.948	0.91	0.985
40.25	1.509	1.713	0.206	0.948	0.91	0.986

40.5	1.516	1.739	0.204	0.948	0.91	0.985
40.75	1.501	1.703	0.203	0.949	0.911	0.986
41	1.516	1.704	0.215	0.95	0.912	0.987
41.25	1.523	1.718	0.202	0.95	0.913	0.987
41.5	1.515	1.712	0.206	0.95	0.912	0.987
41.75	1.527	1.715	0.215	0.95	0.913	0.988
42	1.503	1.697	0.21	0.951	0.914	0.987
42.25	1.51	1.704	0.207	0.951	0.914	0.987
42.5	1.507	1.693	0.211	0.951	0.914	0.988
42.75	1.514	1.716	0.201	0.951	0.915	0.987
43	1.528	1.716	0.208	0.952	0.916	0.988
43.25	1.494	1.693	0.197	0.952	0.916	0.987
43.5	1.528	1.713	0.203	0.952	0.916	0.988
43.75	1.509	1.694	0.207	0.953	0.917	0.988
44	1.5	1.692	0.202	0.952	0.917	0.988
44.25	1.503	1.677	0.212	0.953	0.918	0.989
44.5	1.513	1.71	0.197	0.953	0.918	0.988
44.75	1.5	1.694	0.206	0.953	0.918	0.988
45	1.484	1.658	0.214	0.954	0.919	0.989
45.25	1.473	1.668	0.201	0.954	0.919	0.988
45.5	1.51	1.688	0.207	0.954	0.92	0.989
45.75	1.497	1.681	0.195	0.955	0.92	0.989
46	1.481	1.674	0.202	0.955	0.921	0.988
46.25	1.491	1.684	0.208	0.955	0.921	0.988

46.5	1.518	1.696	0.192	0.955	0.922	0.989
46.75	1.485	1.683	0.205	0.955	0.923	0.988
47	1.505	1.687	0.209	0.956	0.922	0.989
47.25	1.476	1.654	0.216	0.956	0.922	0.989
47.5	1.512	1.68	0.214	0.957	0.923	0.99
47.75	1.485	1.673	0.208	0.956	0.923	0.989
48	1.49	1.672	0.203	0.957	0.924	0.989
48.25	1.492	1.675	0.197	0.957	0.924	0.989
48.5	1.473	1.652	0.202	0.957	0.925	0.989
48.75	1.477	1.642	0.202	0.957	0.925	0.99
49	1.477	1.654	0.208	0.958	0.925	0.99
49.25	1.492	1.648	0.212	0.958	0.926	0.991
49.5	1.496	1.668	0.203	0.958	0.926	0.99
49.75	1.5	1.672	0.203	0.958	0.927	0.99
50	1.455	1.63	0.197	0.958	0.927	0.99

Appendix B)

Source code

```java
//******************************************************************************
//
// File:    Generator.java
// Package: ---
// Unit:    Class Generator
//
//******************************************************************************

import edu.rit.numeric.ExponentialPrng;
import edu.rit.numeric.ListSeries;
import edu.rit.numeric.Series;
import edu.rit.sim.Event;
import edu.rit.sim.Simulation;
import edu.rit.util.Random;

/**
 * Class Generator generates requests for the web server simulations.
 *
 * @author  Alan Kaminsky
 * @author Jimi Ford (jhf3617)
 * @version 5-2-2015
 */
public class Generator
{
    // private data members

    private Simulation sim;
    private ExponentialPrng tpktPrng;
    private Random prng;
    private final int npkt;
    private Routable source;
    private Link link;
    private ListSeries respTimeSeries;
    private ListSeries respTimeLargePackets;
    private ListSeries respTimeSmallPackets;
    private int largePackets;
    private int smallPackets;

    /**
     * Create a new request generator.
     *
     * @param  sim     Simulation.
     * @param  rpkt    Packet generation mean rate.
     * @param  npkt    Number of packets.
     * @param  prng    Pseudorandom number generator.
     * @param  source  First host in network sending the packets.
     */
    public Generator (Simulation sim, double rpkt, int npkt, Random prng,
            Routable source, Link link) {
        this.sim = sim;
        this.tpktPrng = new ExponentialPrng (prng, rpkt);
        this.npkt = npkt;
        this.source = source;
        this.prng = prng;
        respTimeSeries = new ListSeries();
        respTimeLargePackets = new ListSeries();
        respTimeSmallPackets = new ListSeries();
        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 }
```

```java
//********************************************************************************
//
// File:    Host.java
// Package: ---
// Unit:    Class Host
//
//********************************************************************************

import edu.rit.sim.Simulation;

/**
 * Class Host provides the server in the web simulation. The server's
 * request processing time is exponentially distributed with a given mean.
 * Requests are added to the server's queue at any time. The queue has a given
 * maximum size.
 *
 * @author Jimi Ford (jhf3617)
 * @version 22-Apr-2015
 */
public class Host extends Routable
{

    /**
     * Construct a new server. The server's request processing time is
     * exponentially distributed with the given mean. The server's request queue
     * has the given maximum size.
     *
     * @param  sim    Simulation.
     */
    public Host(Simulation sim) {
        super(sim);
    }

    /**
     * Called when this routable object finished receiving a packet on a certain
     * link
     * @param packet the packet this object received
     * @param link the link that the packet was received on
     */
    public void receivePacket(final Packet packet, final Link link) {
        link.open();
        packet.finish();
    }
}
```

```java
//********************************************************************************
//
// File:    Link.java
// Package: ---
// Unit:    Class Link
//
//********************************************************************************

import edu.rit.sim.Simulation;

/**
 * Class Link represents a connection between two routable objects. Links are
 * <I>closed</I> if a packet is currently transmitting on them and <I>open</I>
 * if they are ready to be transmitted on.
 *
 * @author Jimi Ford (jhf3617)
 * @version 5-6-2015
 */
public class Link {

    /**
     * default bit rate used in this project
     * <P>9600 bits/sec</P>
     */
    public static final int DEFAULT_BIT_RATE = 9600;

    /**
     * true if this link has an infinite bit rate
     */
    public final boolean infiniteBitRate;

    /**
     * the bit rate of this link
     */
    public final double bitRate;

    // private data members

    private final Routable r1;
    private final Routable r2;
    private final Simulation sim;
    private double closeStarted;
    private double closeFinished;
    private double totalTimeSpentClosed;
    private boolean ready;


    /**
     * construct a link with the default finit bit rate between two routables
     *
     * @param sim the simulation reference
     * @param r1 one of the routable objects
     * @param r2 the other routable object
     */
    public Link(Simulation sim, Routable r1, Routable r2) {
        this(sim, false, r1, r2);
    }

```

```java
    /**
     * construct a link with specified finite or infinite bit rate
     *
     * @param sim the simulation reference
     * @param infiniteBitRate set to true for infinite bit rate, false for
     * default finite bit rate
     * @param r1 one of the routable objects
     * @param r2 the other routable object
     */
    public Link(Simulation sim, boolean infiniteBitRate, Routable r1,
            Routable r2) {
        this.sim = sim;
        this.r1 = r1;
        this.r2 = r2;
        this.ready = true;
        this.infiniteBitRate = infiniteBitRate;
        this.bitRate = infiniteBitRate ? Double.POSITIVE_INFINITY :
            DEFAULT_BIT_RATE;
        this.totalTimeSpentClosed = 0;
    }

    /**
     * get the other routable object attached to this link compared to the
     * current one
     *
     * @param current the current routable object querying for the other
     * attached routable object
     * @return the routable object that is not equal to the current one
     */
    public Routable other(Routable current) {
        return this.r1.equals(current) ? r2 : r1;
    }

    /**
     * get the current state of the link
     *
     * @return true if the link is ready to pass another packet along it; false
     * otherwise
     */
    public boolean ready() {
        return this.ready;
    }

    /**
     * close this link off so that other packets may not be transmitted on it
     * until open() is called
     *
     * @throws IllegalStateException if the link is not ready to be closed and
     * this link has a finite bit-rate
     */
    public void close() throws IllegalStateException {
        if(!this.infiniteBitRate) {
            if(!this.ready) {
                throw new IllegalStateException();
            }
            this.ready = false;
            this.closeStarted = sim.time();
        }
```

```
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
```

```java
//********************************************************************************
//
// File:    MrPotatoHead.java
// Package: ---
// Unit:    Class MrPotatoHead
//
//********************************************************************************

import edu.rit.numeric.ListXYSeries;
import edu.rit.numeric.Series;
import edu.rit.sim.Simulation;
import edu.rit.util.Random;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * Class MrPotatoHead is the hot potato simulation main program. It simulates
 * a network in which routers use hot potato routing and uses Prof. Alan
 * Kaminsky's pj2 library to aid in this discrete event simulation.
 *
 * @author  Alan Kaminsky
 * @author Jimi Ford (jhf3617)
 * @version 5-3-2015
 */
public class MrPotatoHead
{
    private static double rlb;
    private static double rub;
    private static double rdelta;
    private static int npkt;
    private static long seed;

    private static Random prng;
    private static Simulation sim;
    private static String prefix;
    private static Generator gen;

    /**
     * Main program to simulate hot-potato routing
     *
     * @param args command line arguments
     */
    public static void main(String[] args)
    {
        // Parse command line arguments.
        if (args.length != 5 && args.length != 6) usage();
        rlb = Double.parseDouble (args[0]);
        rub = Double.parseDouble (args[1]);
        rdelta = Double.parseDouble (args[2]);
        if(rlb <= 0) rlb = rdelta;
        npkt = Integer.parseInt (args[3]);
        seed = Long.parseLong (args[4]);
        prefix = args.length == 6 ? args[5] : "potato";
        // Set up pseudorandom number generator.
        prng = new Random (seed);

        // Set up plot data series.
        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 }
```

```java
//********************************************************************************
//
// File:    Packet.java
// Package: ---
// Unit:    Packet Link
//
//********************************************************************************

import edu.rit.numeric.ListSeries;
import edu.rit.sim.Simulation;
import edu.rit.util.Random;

/**
 * Class Packet provides a packet model in the web simulation. It contains the
 * logic necessary to determine the size of the packet and the amount of time
 * it would take to transmit along a link. It also reports to several instances
 * of ListSeries objects that keep track of the response time of packets based
 * on their size.
 *
 * @author  Alan Kaminsky
 * @author Jimi Ford (jhf3617)
 * @version 5-6-2015
 */
public class Packet
{
    /**
     * size of the packet in bits
     */
    public final int size;

    /**
     * unique identifier across all other packets
     */
    public final int id;

    /**
     * true if this packet is 576 bytes, false otherwise
     */
    public final boolean isLarge;

    // private data member

    private static int idCounter = 0;
    private Simulation sim;
    private double startTime;
    private double finishTime;
    private ListSeries respTimeSeries;
    private ListSeries respTimeLargePackets;
    private ListSeries respTimeSmallPackets;

    private static final int
        SMALL = 40 * Byte.SIZE,
        LARGE = 576 * Byte.SIZE;


    /**
     * Construct a new packet. The packet's response time will be recorded in
     * 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 }
```

```java
//*******************************************************************************
//
// File:    PlotHandler.java
// Package: ---
// Unit:    Class PlotHandler
//
//*******************************************************************************

import java.awt.BasicStroke;
import java.awt.Color;
import java.io.IOException;
import java.text.DecimalFormat;

import edu.rit.numeric.ListXYSeries;
import edu.rit.numeric.plot.Dots;
import edu.rit.numeric.plot.Plot;

/**
 * Class PlotHandler is the delegate for dealing with visualizing the data
 * generated by the "number crunching" program, MrPotatoHead.
 * Its purpose is to be instantiated in MrPotatoHead with the data to plot,
 * where the write() method should then be called.
 * <P>
 * Running this program and specifying in the command line arguments the plot
 * files previously generated will open a graphical representation of these
 * plots for each file.
 * </P>
 * @author Jimi Ford
 * @version 5-6-2015
 *
 */
public class PlotHandler {

    // private data members
    private final String rtTotalFile;
    private final String dfTotalFile;
    private final String rtLargeFile;
    private final String dfLargeFile;
    private final String rtSmallFile;
    private final String dfSmallFile;
    private final String routerDropFile;
    private final String reRouteFile;
    private final String primaryActivityFile;
    private final String secondaryActivityFile;
    private final ListXYSeries dfTotal;
    private final ListXYSeries rtTotal;
    private final ListXYSeries dfLarge;
    private final ListXYSeries rtLarge;
    private final ListXYSeries dfSmall;
    private final ListXYSeries rtSmall;
    private final ListXYSeries aDrop;
    private final ListXYSeries bDrop;
    private final ListXYSeries cDrop;
    private final ListXYSeries dDrop;
    private final ListXYSeries aReRoute;
    private final ListXYSeries bReRoute;
    private final ListXYSeries cReRoute;
    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 
```

```java
//*********************************************************************************
//
// File:    Routable.java
// Package: ---
// Unit:    Class Routable
//
//*********************************************************************************

import edu.rit.sim.Event;
import edu.rit.sim.Simulation;

/**
 * Class Routable is the abstract base class that defines objects that contain
 * routing logic with the ability to be linked together. Known implementations
 * include Router and Host.
 *
 * @author Jimi Ford (jhf3617)
 * @version 5-6-2015
 */
public abstract class Routable {

    private static int count = 0;

    /**
     * the simulation reference
     */
    protected final Simulation sim;
    private final int id;


    /**
     * Construct a routable object
     * @param sim the simulation this object should belong to
     */
    public Routable(Simulation sim) {
        this.sim = sim;
        id = ++ count;
    }

    /**
     * compare this instance with another object and determine whether this
     * instance is equal to the other object.
     *
     * @param o the other object to compare to
     * @return true if this object is equal to the other object
     */
    public boolean equals(Object o) {
        if(o == this) {
            return true;
        }
        if(o instanceof Routable) {
            return this.id == ((Routable)o).id;
        }
        return false;
    }

    /**
     * Called when this routable object finished receiving a packet on a certain
```

```
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
```

```java
//********************************************************************************
//
// File:    Router.java
// Package: ---
// Unit:    Class Router
//
//********************************************************************************

import edu.rit.sim.Simulation;
import edu.rit.util.AList;
import edu.rit.util.Random;

/**
 * Class models a router's behavior where packets are transmitted on a
 * preferred link if that link is available, otherwise a secondary link is
 * chosen at random until an available link is found. If no secondary links
 * are available, the packet is dropped.
 *
 * @author Jimi Ford (jhf3617)
 * @version 5-2-2015
 */
public class Router extends Routable {

    // private data members

    private final Random prng;
    private Link primary;
    private int dropCount;
    private int receiveCount;
    private int reRouteCount;
    private final AList<Link> secondary;

    /**
     * Construct a router object
     *
     * @param prng the pseudorandom number generator to use for choosing what
     * secondary routables to use
     * @param sim the simulation object this router should be associated with
     */
    public Router(Random prng, Simulation sim) {
        super(sim);
        this.prng = prng;
        this.dropCount = 0;
        this.receiveCount = 0;
        this.reRouteCount = 0;
        this.secondary = new AList<Link>();
    }

    /**
     * Set the primary link this router should prefer to send its received
     * packets on
     *
     * @param link the link to prioritize
     */
    public void setPrimary(Link link) {
        this.primary = link;
    }

```

```java
    /**
     * add a secondary link to the list of secondary links
     *
     * @param link the link to add
     */
    public void addSecondary(Link link) {
        this.secondary.addLast(link);
    }

    /**
     * Called when this routable object finished receiving a packet on a certain
     * link
     * @param packet the packet this object received
     * @param link the link that the packet was received on
     */
    public void receivePacket(final Packet packet, final Link l) {
        l.open();
        Link link = null;
        ++receiveCount;
        boolean goodToGo = false;
        if(primary.ready()) {
            goodToGo = true;
            link = primary;
        } else if(secondary.size() > 0) {

            int[] indices = ShuffleHelper.shuffledArray(prng, secondary.size());
            for(int i = 0; i < indices.length && !goodToGo; i++) {
                link = secondary.get(indices[i]);
                if(link.ready()) {
                    goodToGo = true;
                    ++reRouteCount;
                }
            }
        }
        if(goodToGo) {
            startSending(packet, link);
        } else {
            // drop packet
            ++dropCount;
        }
    }

    /**
     * Get the fraction of packets that this router dropped
     *
     * @param totalPacketCount the total number of packets generated in the
     * simulation
     * @return a number between 0 and 1
     */
    public double dropFraction(int totalPacketCount) {
        return ((double)this.dropCount)/(double)totalPacketCount;
    }

    /**
     * Get the fraction of the packets that the router had to re-route along
     * a secondary route
     */
    public double reRouteFraction() {
```

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

```java
//********************************************************************************
//
// File:    ShuffleHelper.java
// Package: ---
// Unit:    Class ShuffleHelper
//
//********************************************************************************

import edu.rit.util.Random;

/**
 * Class provides helper methods for picking secondary links to transmit
 * packets on when primary links are currently in use.
 *
 * @author Jimi Ford (jhf3617)
 * @version 5-3-2015
 */
public class ShuffleHelper {

    /**
     * Shuffle an array in place
     *
     * @param prng pseudorandom number generator used with shuffling
     * @param array the array to shuffle
     */
    private static void shuffleArray(Random prng, int[] array) {
        for (int i = array.length - 1; i > 0; i--) {
            int index = prng.nextInt(i + 1);
            int a = array[index];
            array[index] = array[i];
            array[i] = a;
        }
    }

    /**
     * Create an array with <I>size</I> elements ranging from 0 to
     * <I>size - 1</I>.
     *
     * @param size the number of elements the array should contain
     * @return the array containing elements from 0 to <I>size - 1</I>
     */
    private static int[] indexArray(int size) {
        int[] retval = new int[size];
        for(int i = 0; i < size; i++) {
            retval[i] = i;
        }
        return retval;
    }

    /**
     * Create a shuffled array with <I>size</I> elements ranging from 0 to
     * <I>size - 1</I>.
     *
     * @param prng pseudorandom number generator used for shuffling
     * @param size number of elements to contain
     * @return the shuffled array
     */
    public static int[] shuffledArray(Random prng, int size) {
```

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