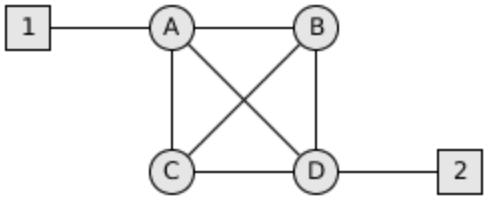
Jimi Ford
CSCI 351-01 Data Communications and Networks
Programming Project 3 Report

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?

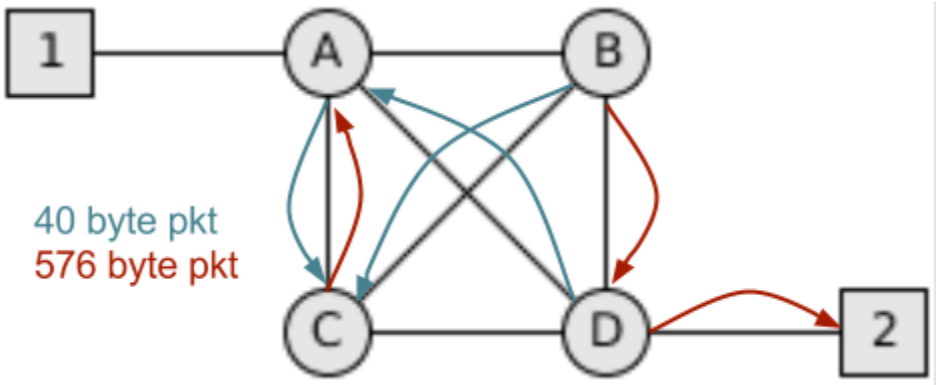
In *Q5 Supporting Data*, we observe the same recurring graph shape from the previous projects. The traversal time starts at a relatively low value, increases drastically to a global maximum, and then gradually decreases asymptotically to some value as the arrival rate approaches infinity.

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

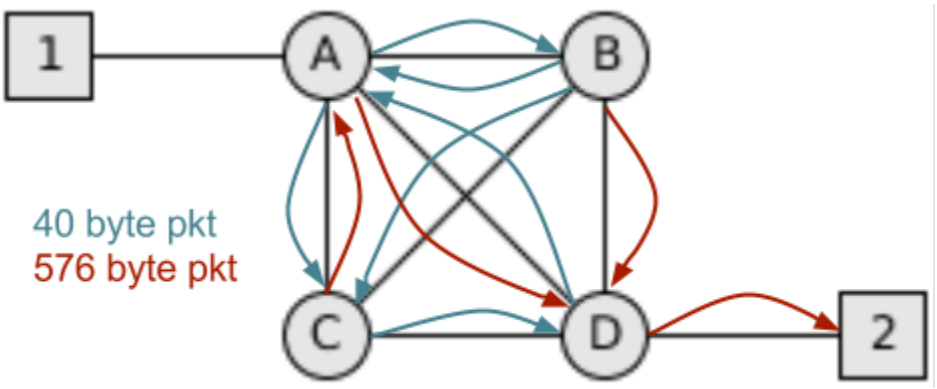
see that the smaller packets take $0.087\ seconds$ on average to finish transmitting. Note that the table data shows that 0 small packets were dropped at this rate. The theoretical amount of time that these small packets should be taking per hop (for low mean rate) is $\frac{40*8}{9600} = \frac{320}{9600} = 0.0\overline{3}\ seconds.$ 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.

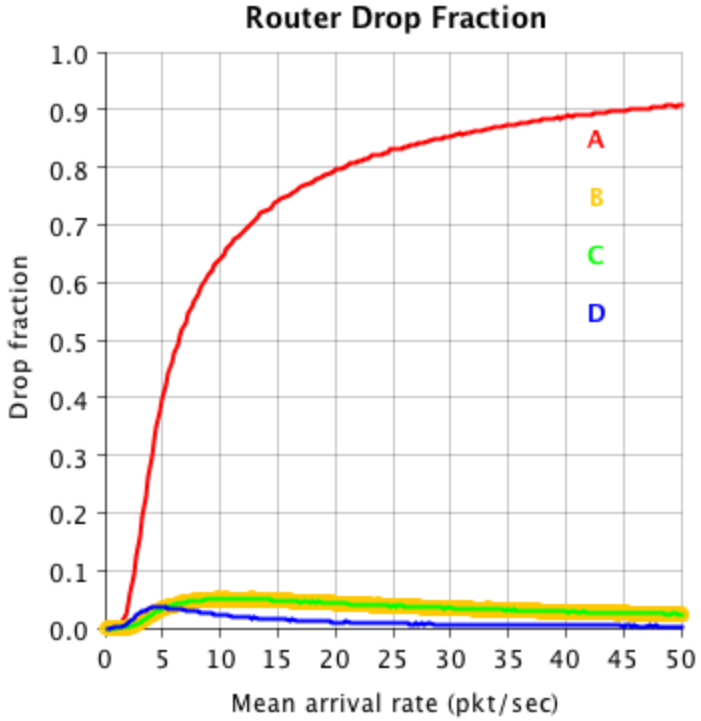


Each arrow represents a packet that is currently being transmitted. The idea I'm trying to convey here is that the more we increase the rate at which 1 generates and sends packets to $A,$ the more links we will have that are closed off at any given time. Eventually we will reach a point where the snapshots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

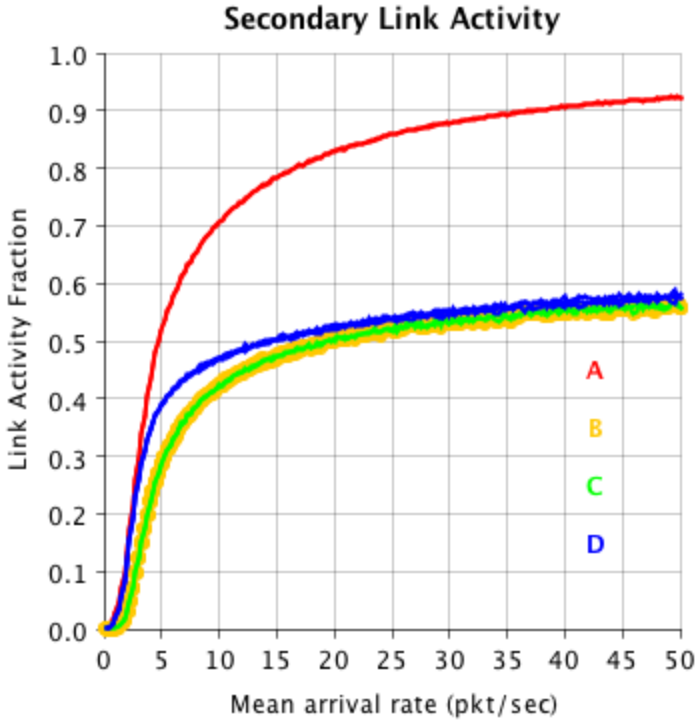


I will refer to this as *nearly saturated*. If a packet comes in to router A from $Host\ 1$, the router will have to drop it since all of its outgoing links are occupied. Once the network reaches this state, the link from D to $Host\ 2$ will be active nearly 100% of the time since that is $D's$ primary 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 $Host\ 2$.

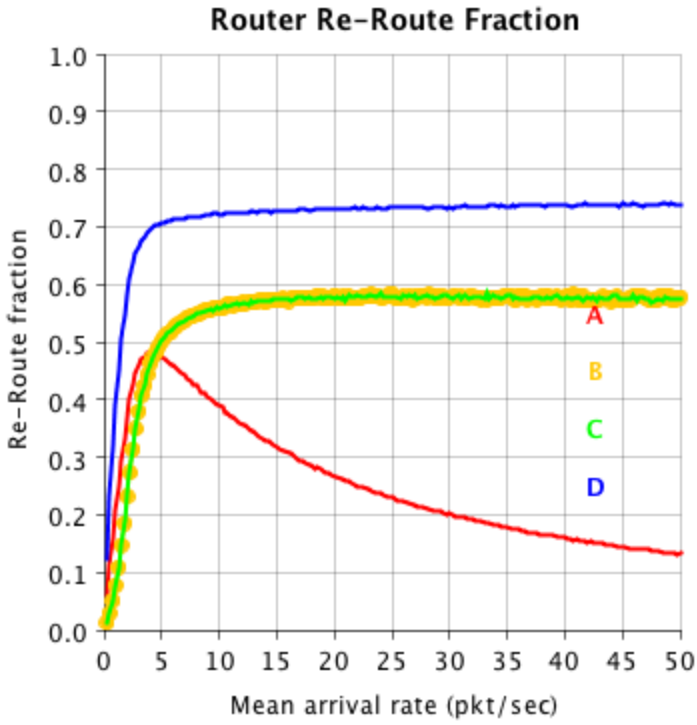
Now the fascinating observation we make is that even as we bombard the network with more and more packets at higher and higher rates, the average traversal time decreases as the mean arrival rate increases. This was completely counterintuitive for me. It caused me to go back and second guess my model 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.)

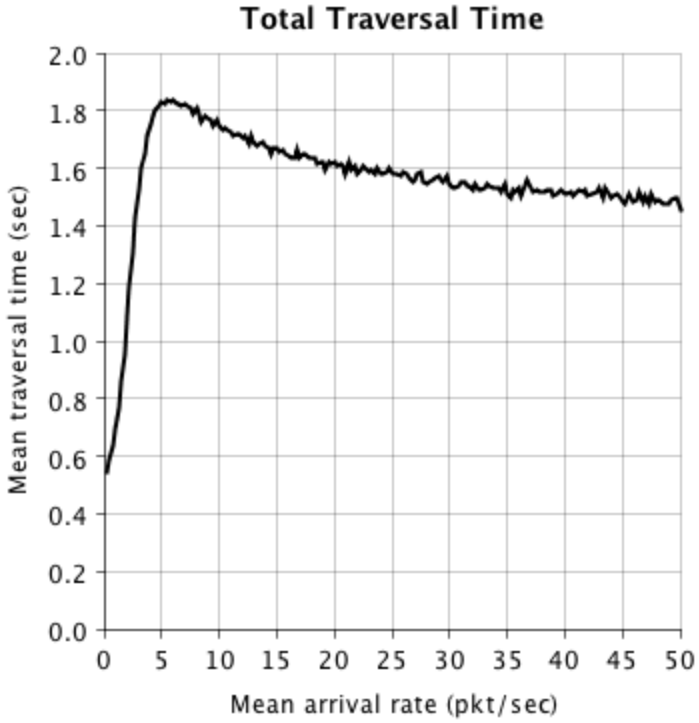
```
java MrPotatoHead 0 50 .25 200000 100123456789 potato
```

(expected runtime 30 seconds on 2.93 GHz)

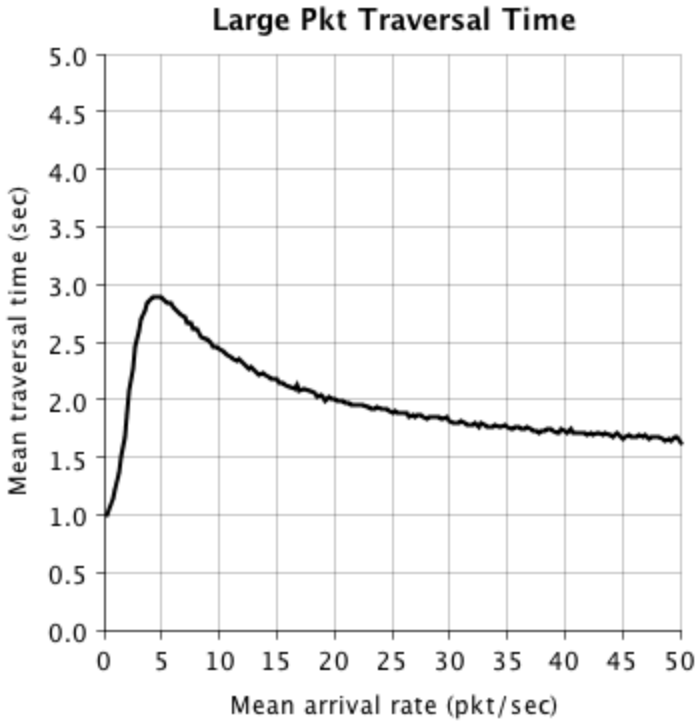
After generating the *.dwg* files, you must run the following command to visualize the plots.

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

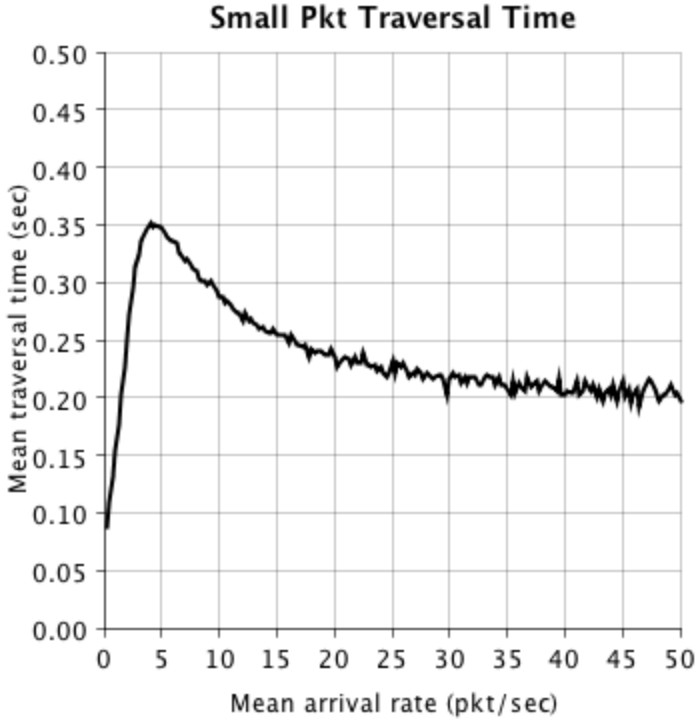
This graph below is a plot of mean (or average) traversal time for all packets generated as a function of the mean arrival rate.



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.

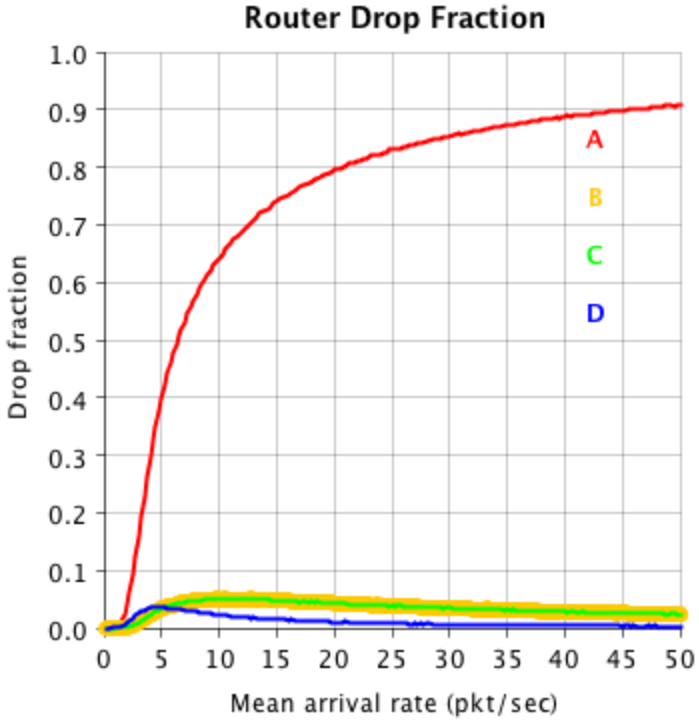


This graph below is a plot of mean (or average) traversal time for packets of $SIZE = 40\ bytes$ as a function of the mean arrival rate.



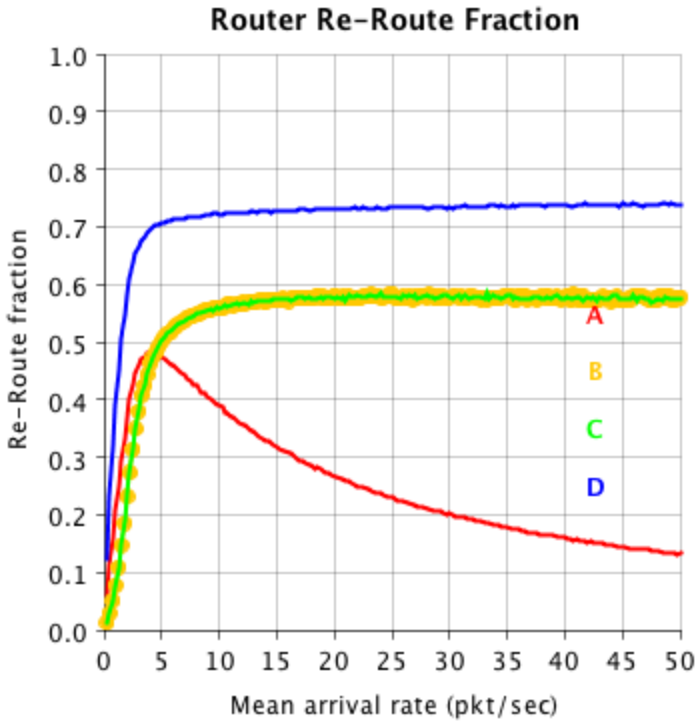
To view the individualized drop fraction of each router, run this command.

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



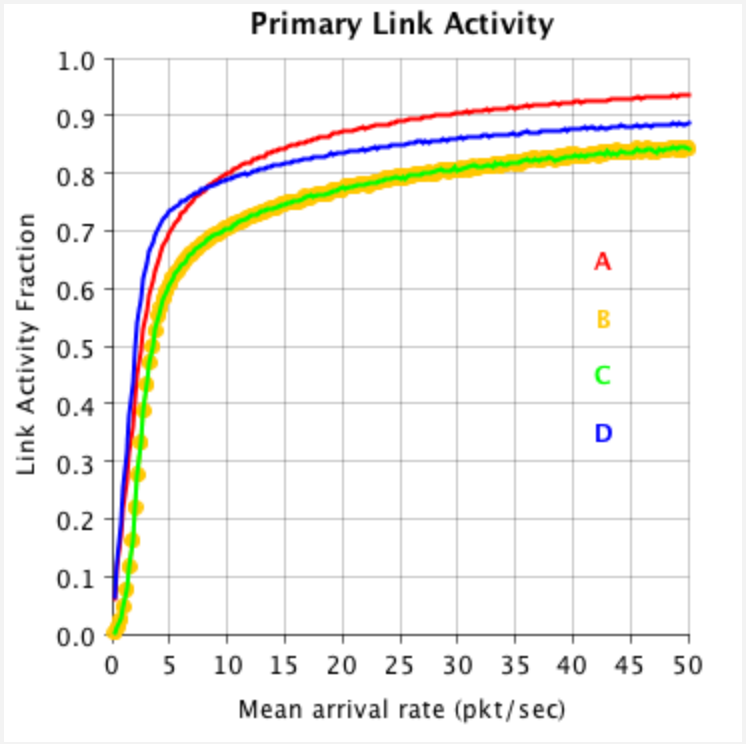
To view the individualized re-route fraction of each router, run this command. This is the fraction of packets that the routers forward along secondary links instead of their primary links.

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



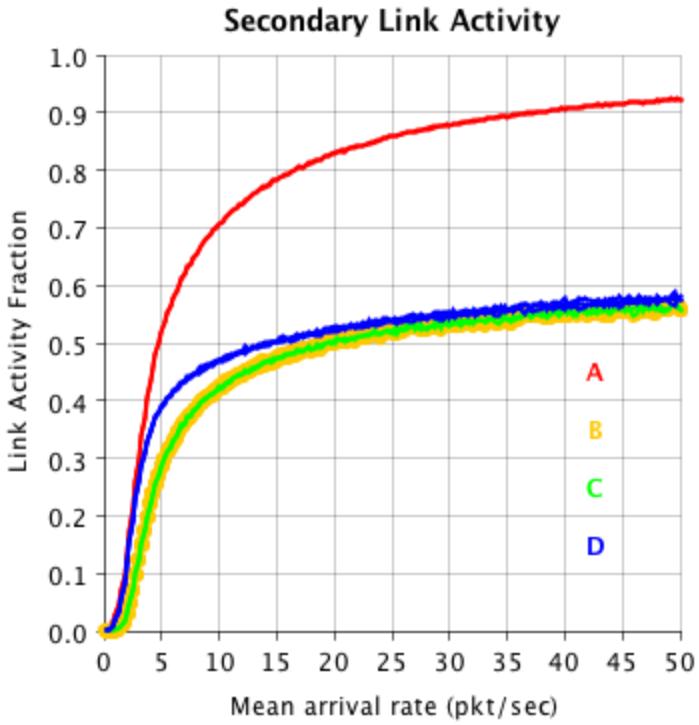
To view the individualized primary link activity fraction of each router, run this command. This is the fraction of time the router's primary outgoing link is spent transmitting a packet.

```
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
```

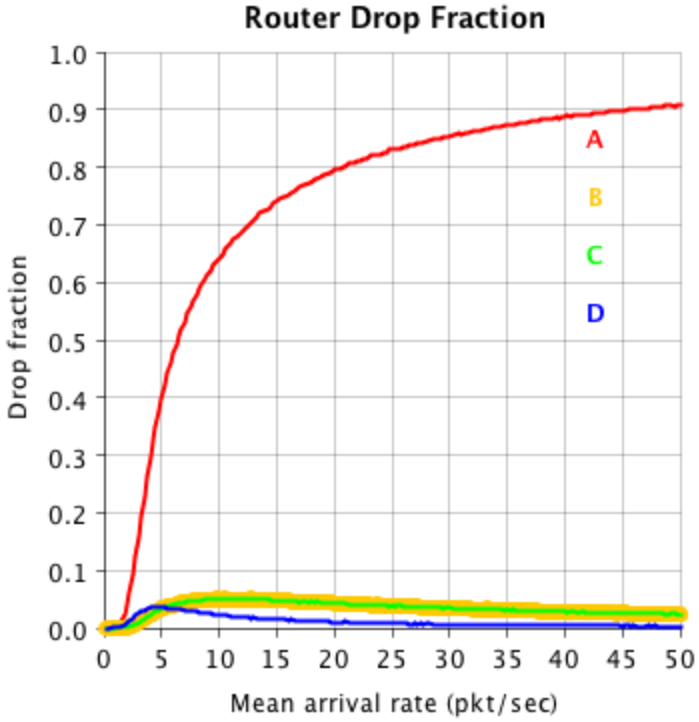


6. Based on data generated by running your simulation program(s), discuss this question: What happens to the packet drop fraction as the mean packet arrival rate increases, and why is the packet drop fraction behaving this way?

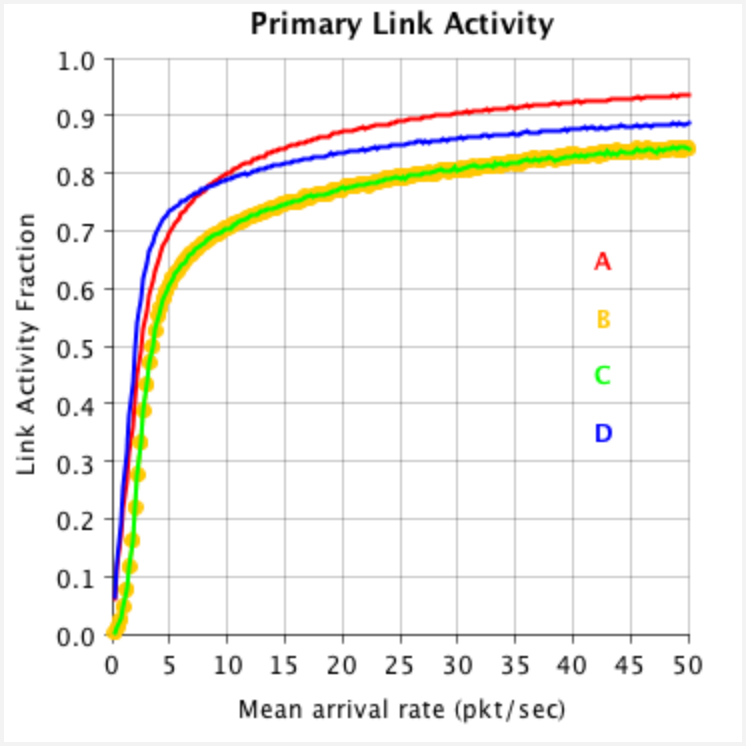
Observing *Q7. Supporting Data*, we see that the drop fraction of the network as a whole starts very low, increases rapidly and then approaches 100% as the mean arrival rate approaches infinity. This is result is much more intuitive than the average traversal time results. Let's look a little deeper 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

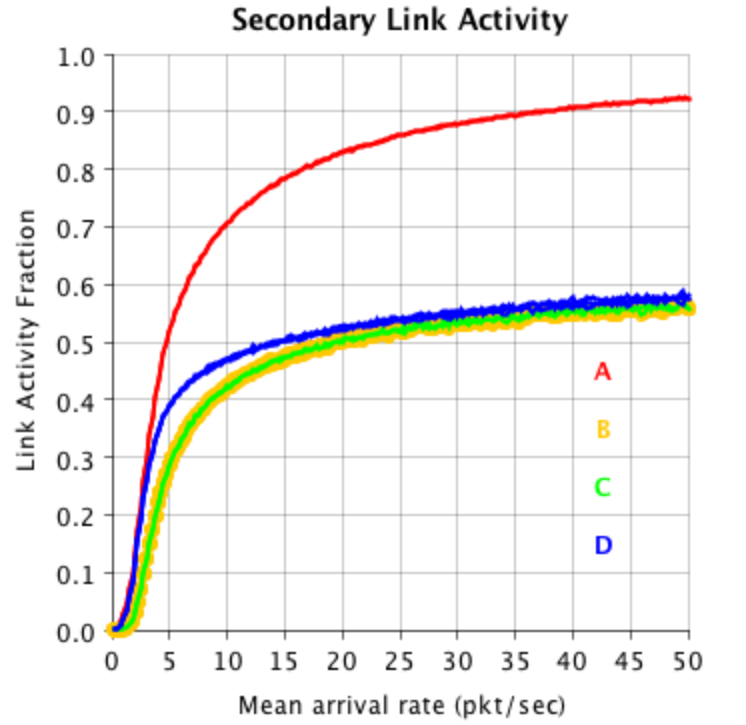
be. Examining the packet sizes and their respective drop fractions individually, we see that nearly 100% of small packets are being dropped at 50 packets/second where only 92% of the large packets are getting dropped. This was another result that I thought was incorrect at first and really made me think 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.

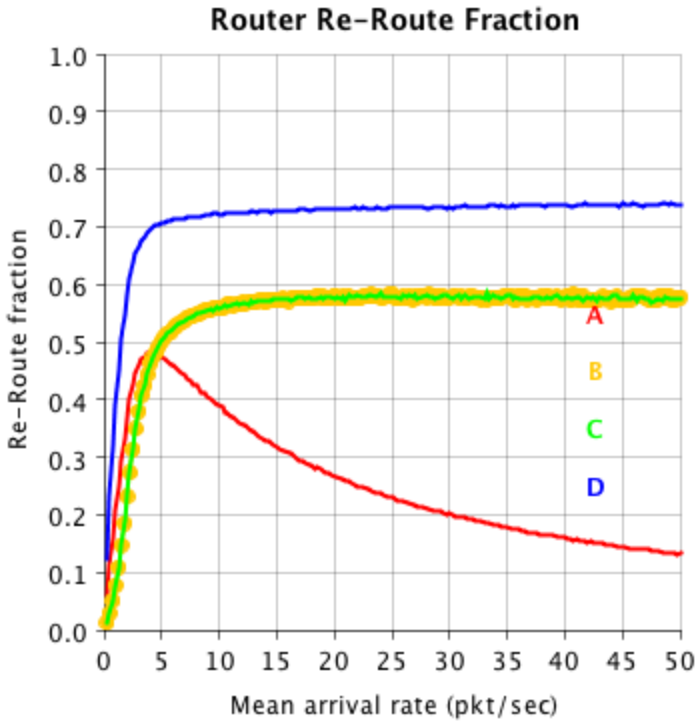


Here we see that because *A* is receiving so many packets from *Host 1* and from all the other routers when packets get re-routed, *A*'s links are closed for a majority of the time. We can confirm that with the data collected on link activity.





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

---

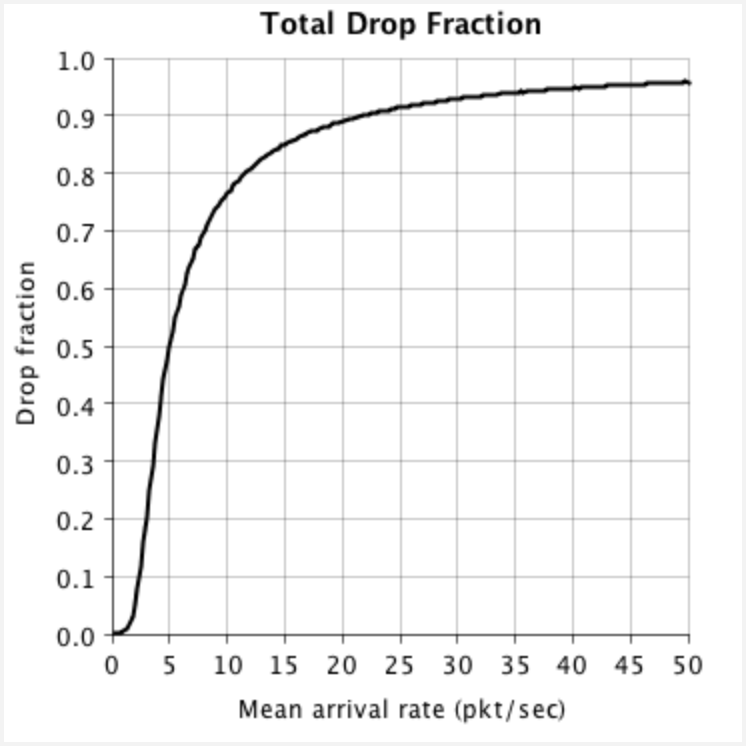
note: In order to save you time while grading, lower numbers of simulations are used. As a result, the plots produced are not extremely smooth but are smooth enough to convey the general trend of the drop fraction . The table results are found in Appendix A under the columns labeled "Drop Frac [Total,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

Commands used:

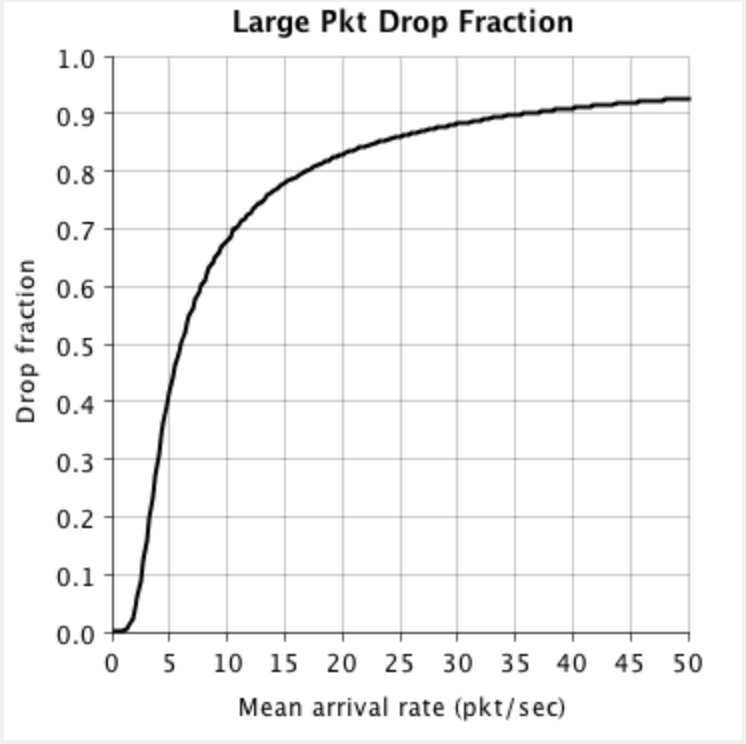
We will use the results generated from the previous run and just view the plot file generated by it:

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

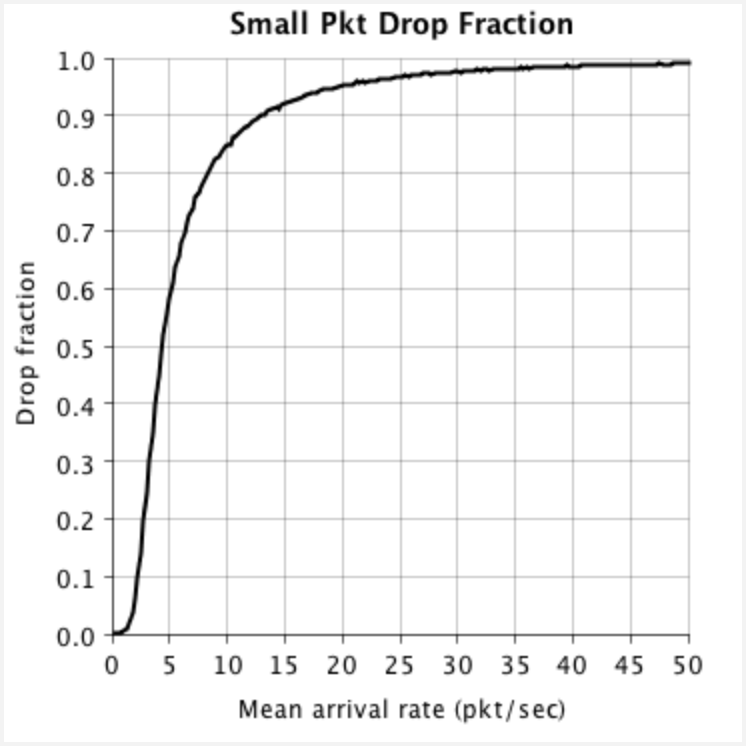
This graph below is a plot of the fraction of all packets that were dropped as a function of the mean arrival rate at which packets were generated.



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.

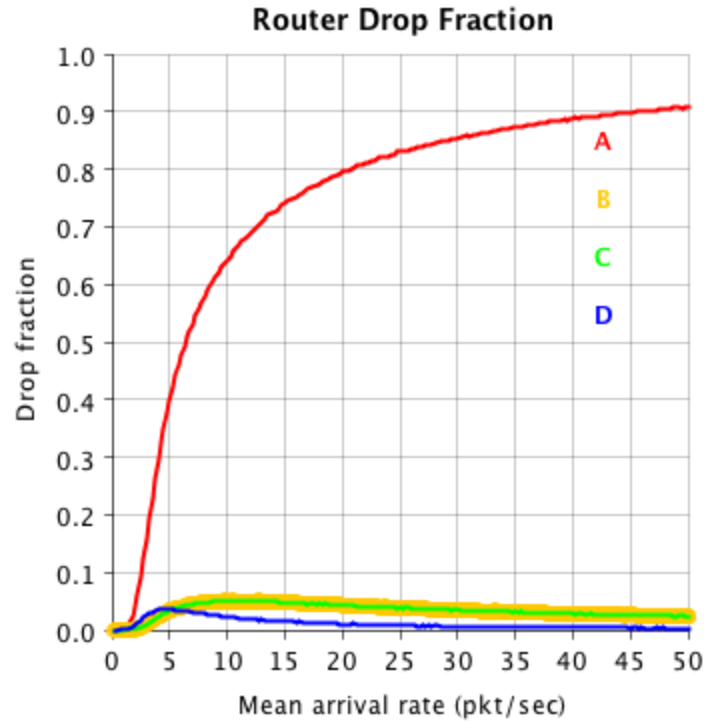


This graph below is a plot of the fraction of the small packets (i.e. 40 bytes) that were dropped as a function of the mean arrival rate at which packets were generated.



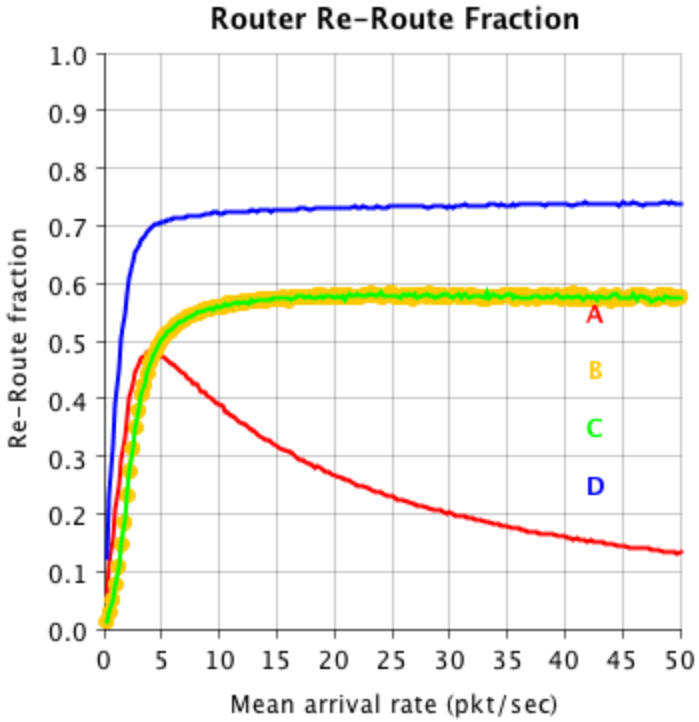
To view the individualized drop fraction of each router, run this command.

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



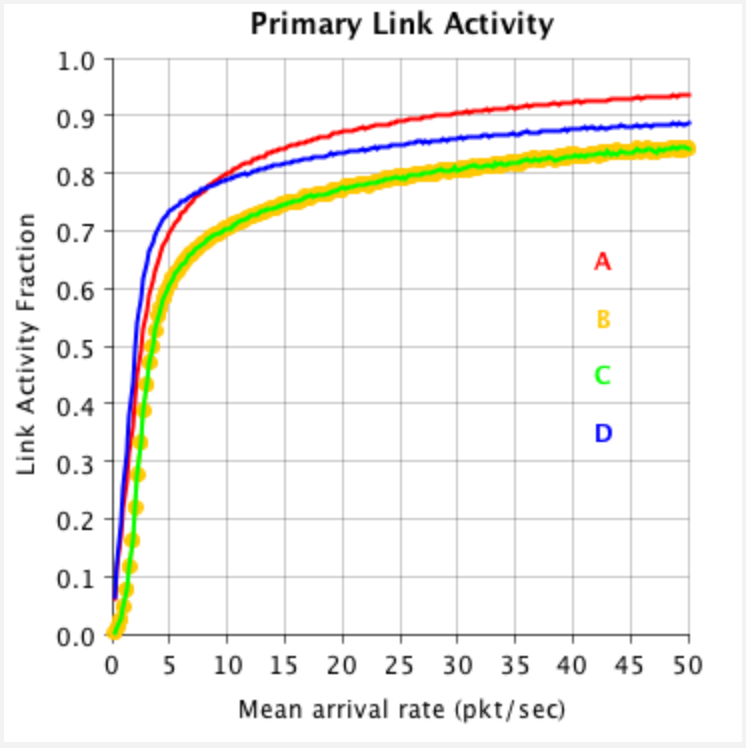
To view the individualized re-route fraction of each router, run this command. This is the fraction of packets that the routers forward along secondary links instead of their primary links.

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



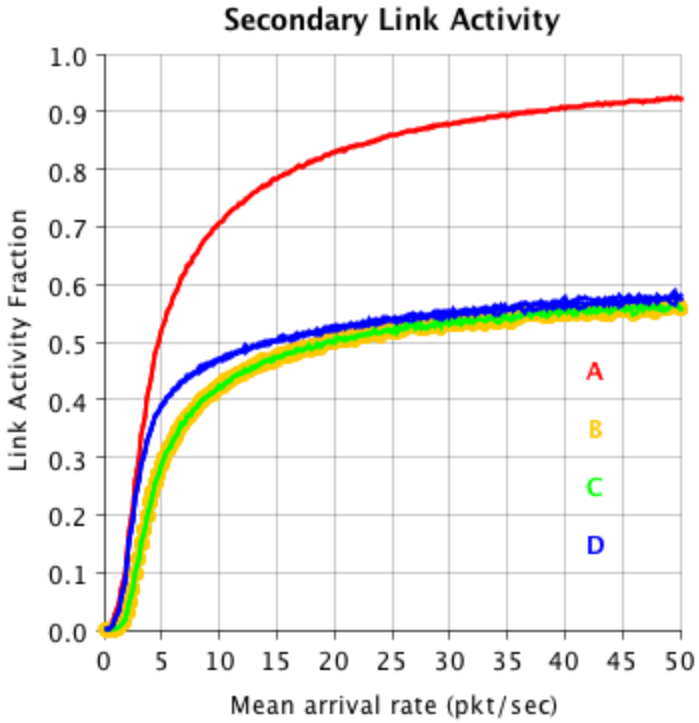
To view the individualized primary link activity fraction of each router, run this command. This is the fraction of time the router's primary outgoing link is spent transmitting a packet.

```
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