Air Force Institute of Technology
Department of Electrical and Computer Engineering

CSCE 654 - Computer Communication Networks
Project #3 - Network Routing

Author: Micah Hayden

April 24, 2019

1 Introduction:

The goal of this project is to demonstrate the queuing impact of a routing network. The network consists of 6 nodes A-F. All arrivals enter the system from the source into A at rate λ. From there, messages are routed to their appropriate destination according to the network topology shown in Figure 1. The times shown along links between nodes A-F represent the propagation delays between nodes, while the percentages between a node and its sink represents the percentage of the total traffic, destined to the respective nodes. Node A has a service rate $\mu_A = 10pps$, while all other nodes have service rate $\mu_{\bar{A}} = 20pps$.

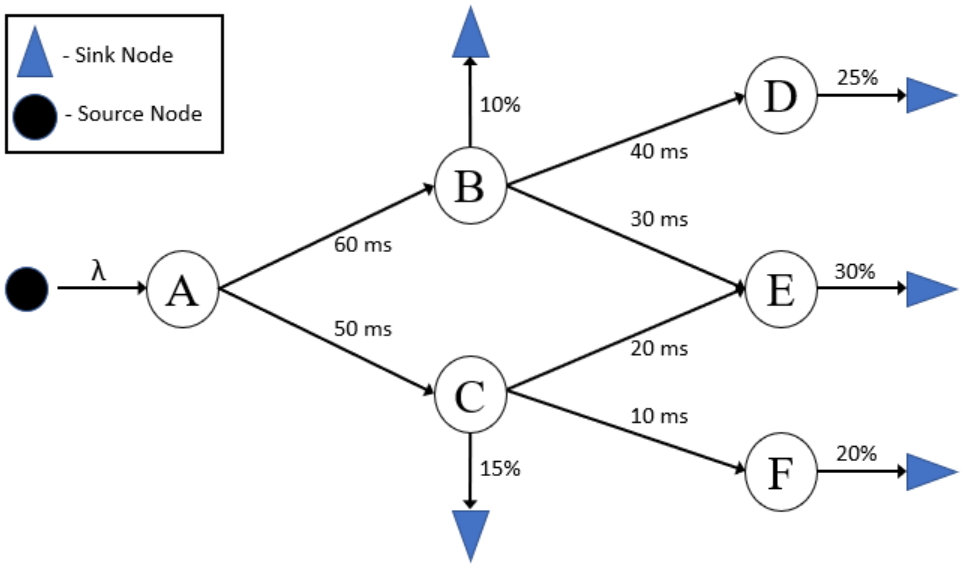

Figure 1: Network Topology of Routing Network

The key performance parameters for this experiment are the system delays - both per node, and the system as a whole; as well as the routing accuracy of the system. Case 1 has a system arrival rate $\lambda_A = 15\,pps$, and Case 2 has $\lambda_A = 17\,pps$.
Mathematically analyzing the system prior to simulation allows prediction of the system response, as well as provides an anchor point for the results. If the results match the expectation, the system functioned as expected. However, if the results do not match, several questions arise: was the simulation setup properly? Did the analysis not account for a particular parameter/factor? Was the particular queuing model used appropriate?

2 Expected Results:

This section will go through the expected results of Case 1 & Case 2. The data to be analyzed is the routing accuracy and system delay, both per node and for the system as a whole.
The expected accuracy of the routing is simply the % of packets routed to each destination. Table 1 below enumerates these expectations:

Node	% of Traffic
A	0
B	10
C	15
D	25
E	30
F	20
Total	100

Table 1: Expected traffic load per node

Traffic with destination E will be routed evenly through nodes B and C. This distributes the expected network traffic evenly across links $A \to B$ and $A \to C$, with each receiving $\frac{1}{2}$ of E's traffic.[1]

$$\begin{aligned} Load_{A\to B} &= Load_B + Load_D + \frac{1}{2} \cdot Load_E \\ &= 0.10 + 0.25 + \frac{1}{2} \cdot 0.30 \\ &= 0.50 \end{aligned} \qquad \begin{aligned} Load_{A\to C} &= Load_C + Load_F + \frac{1}{2} \cdot Load_E \\ &= 0.15 + 0.20 + \frac{1}{2} \cdot 0.30 \\ &= 0.50 \end{aligned}$$

This shows that links $A \to B$ and $A \to C$ will each receive half of the traffic departing A. Because $\mu_A > \lambda_A$, the $\text{Throughput}_A = \lambda_A$. Thus, the arrival rate of nodes B and C equals $0.5 \cdot \lambda_A$. Because $\mu_B = \mu_C = 10\,pps$, and $10\,pps > 0.5 \cdot \lambda_A$ for both cases, their throughput equals their arrival rates. In summary, because all queues in this system are stable, the throughput of any node equals the node's arrival rate. It follows that any child node's arrival rate equals the percentage of the parent node's traffic that the child receives. This yields the following arrival rates for cases 1 and 2.[2]

Node	Arrival Rate (pps)
A	15.0
B	7.5
C	7.5
D	3.75
E	4.5
F	3.0

Table 2: λ for Case 1

Node	Arrival Rate (pps)
A	17.0
B	8.5
C	8.5
D	4.25
E	5.1
F	3.4

Table 3: λ for Case 2

With the arrival rates shown in Tables 2 and 3, one can use Little's Law to calculate the expected time in the system, $E[r]$, for each node, using Equation 1.

$$E[r] = \frac{1}{\mu - \lambda} \tag{1}$$

[1] $Load_{link}$ is relative to the departure rate of the source node of the particular link.
[2] λ_E is the sum of its arrival rates from B and C.

Table 4 shows these times.

Node	E[r] Case 1 (s)	E[r] Case 2 (s)
A	0.200	0.333
B	0.400	0.667
C	0.400	0.667
D	0.160	0.174
E	0.182	0.204
F	0.143	0.152

Table 4: Per-node system delays, $E[r]$

The times shown above indicate the time in the system **at** a particular node. To find the expected system delay of a packet from source to its destination, one must also account for all nodes and propagation delays along its path. For Node E, one must account for the separate paths to reach E. Because half of its traffic goes through B and the other through C, this can be represented as the average system delay of 2 packets: one going through B and the other through C to destination E.
This produces the following expected end-to-end delay (source to destination), for packets routed to each destination:

Destination:	Case 1 Delay (s)	Case 2 Delay (s)
B	0.660	1.060
C	0.650	1.050
D	0.860	1.274
E	0.862	1.284
F	0.803	1.212

Table 5: Expected end-to-end delay by destination

The overall network end-to-end delay is a weighted average of the values shown in Table 5 with the expected Traffic Loads shown in Table 1. This yields the following:

$$\text{Network Delay}_{Case\,1} = 0.798s \qquad \text{Network Delay}_{Case\,2} = 1.210s$$

3 Simulation Setup:

The basis of the omnet simulation was still the FIFO queue sample used for Project 2. Each simulation run was 10 hours of sim-time.

3.1 Routing:

To accomplish the required routing, I utilized a custom message type **Netmsg**, which uses an integer to represent its destination.
I assigned each destination node a unique integer. [3]. The source node generates a random integer between 1 and 100, inclusive. Based on the random number, it assigns a corresponding destination node. As the number of messages grows, the distribution of destinations should approach the desired proportions.
The other modifications required were to detect whether a packet had arrived at its destination; and if not, to send it out on the appropriate link. To accomplish this task, I modified the Fifo module's "endService" function. Previously, when the queue finished processing a packet, it immediately sent it out on the sole port. The modifications change this to function to check if the packet's destination matches the queue's index. If so, the queue sends the packet to the sink node. Otherwise, it uses simple logic to determine which of its outbound ports to send the message on. The code for both of these operations are shown in Appendix B.

3.2 Network Configuration:

I defined my network using an array of six FIFO queues and six sinks, as well as a single generator. The connections account for the propagation delay on each link. This topology matches that shown in Figure 1.[4]

3.3 Data Collection:

There are several pieces of data which need to be collected during the simulation. The first is the packet count at each sink, which will allow a comparison of each destination's traffic load. Next is the packet lifetime. This statistic is used to compare the source-destination time for each packet. Combined with the packet count, this will also allow the calculation of an overall network delay.
In order to properly analyze the system response to the network routing presented, I also need to collect data on each queues system time. I modified the AbstractFifo.h and AbstractFifo.c files, which govern the FIFO queue nodes, by adding two new statistics: serviceTime and nodeTime. These values, in conjunction with the already-existing queueingTime statistic, will allow a comparison of the time spent at each node, and are collected immediately prior to sending the message from the queue to its destination.

[3] To account for the duplicate paths to node E, there were two integers for E, one for each path
[4] There is a sink connected to A to ensure the simulator functioned properly, but is not shown on the network diagram.

4 Results & Analysis:

I conducted three simulations for each case, ensuring the results matched with different seeds and produced accurate results. I collected data to compare the lifetime of packets at each node, as well as the total lifetime of the system. The three runs were averaged for each case, producing the data used for the simulation analysis.

4.1 Routing Accuracy & Traffic Load:

To determine the routing accuracy and traffic load, I averaged the number of packets that arrived at each destination, relative to the total number of packets sent.
As seen in Figures 2 and 3, my results closely matched the expected traffic load described in Table 1

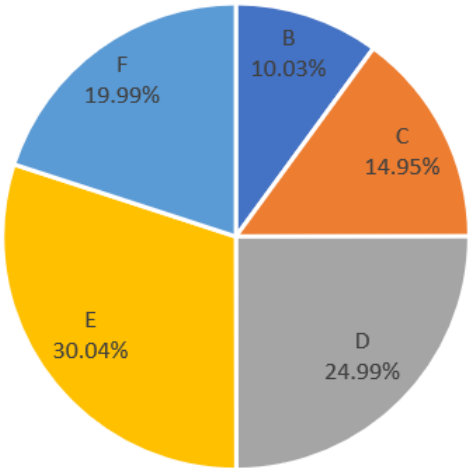

Figure 2: Traffic Load with $\lambda_A = 15\,pps$

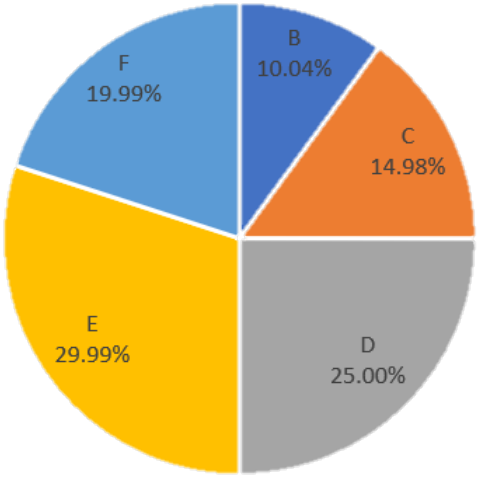

Figure 3: Traffic Load with $\lambda_A = 17\,pps$

The largest load difference was Case 1, with nodes C and E: node C had 0.05% less traffic than expected, while node E had 0.04% more.[5]

4.2 Per-Node System Times

This section discusses how well each node performed, relative to the expected system times of Nodes A-F. I used the **nodeTimeSignal** created for the FIFO nodes to measure the time spent at a given queue, to include the service time. I then compared the simulation results to the expected nodal delays described in Table 4 in Section 2. Case 1 - $\lambda_A = 15$ pps - had an average difference of 0.343% from the expected nodal delays, while Case 2 - $\lambda_A = 17$ pps - had an average difference of 0.714%. The data are shown in Tables 9 and 10 in Appendix A. The accuracy of the simulation indicates that the the model utilized for this network matches the actual implementation.

4.3 Network Delay:

I used the statistics collected at each sink node to compare the per-destination average delay. Both Case 1 and Case 2 performed well, with results closely matching the expectations. As expected, the largest end-to-end delays occurred when the destination was one of Nodes D-F. This occurs because to reach these destinations, they must go through two queue nodes, and two links before arriving at their destination's queue. The per-destination end-to-end delays for Case 1 produced an average percent difference of 0.157% from the expectation, while Case 2 had an average % difference of 0.893%.[6]

[5] The raw data used to build Figures 2 and 3 is shown in Appendix A.

[6] This % difference simply averaged the % differences for each destination's end-to-end delay, not based on the traffic load of each node.

I calculated the network's average end-to-end delay for Cases 1 and 2 by using a weighted average of each destination's end-to-end delay, using the simulated traffic loads as the weight.

Case #	Simulated Delay (s)	Expected Delay (s)	% Difference
1	0.7992	0.7976	0.200
2	1.1996	1.2095	0.822

Table 6: Average Network Delays

The average network end-to-end delay for Case 1 was 0.7992 seconds, a 0.200% difference than the expected result. The average end-to-end delay for Case 2 was 1.1996 seconds, a 0.822% difference than the expected result. Figure 4 shows these results graphically, reinforcing the impact of the arrival rate on the long-term system response.

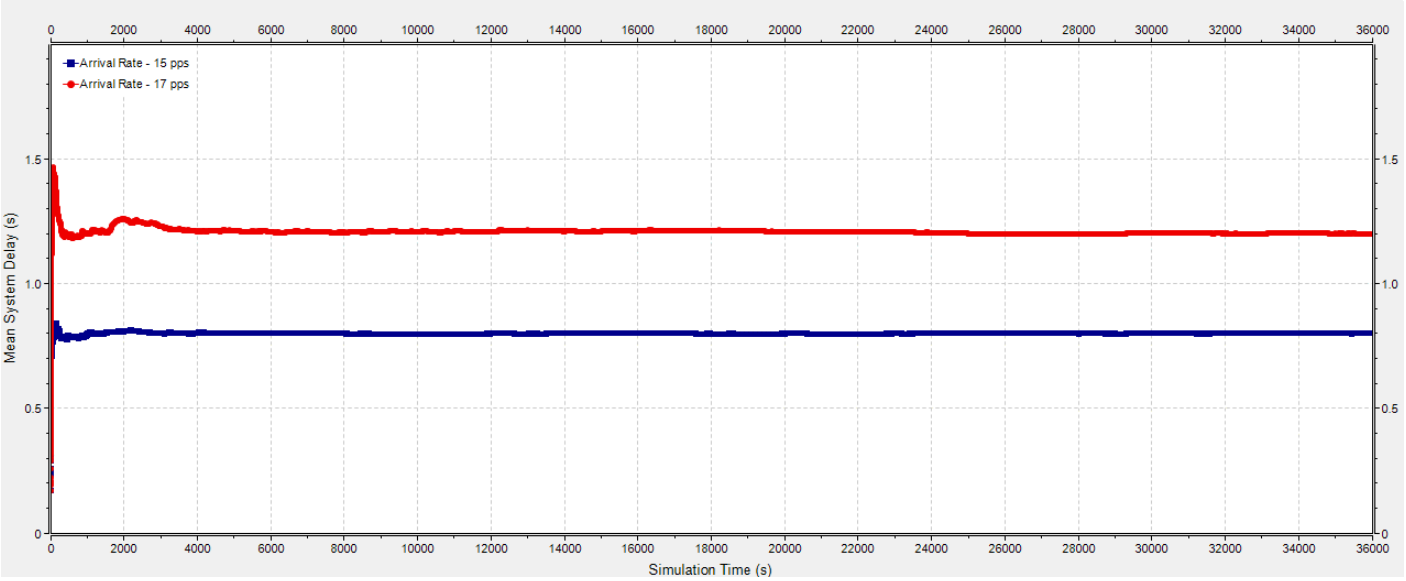

Figure 4: Mean System Delay

Despite the difference in arrival rates, both cases produced a stable system. However, change in arrival rate from 15 to 17 pps increased the expected delay by 0.4 seconds, a 50% increase.

4.4 Routing Decisions

For my initial simulations, I used the assumption that splitting the traffic evenly between nodes B and C would produce the best overall system delay. This assumption was used to calculate all expected results, and routed traffic to destination E evenly through B and C. After seeing my results, I decided to run another round of simulations, routing a larger portion of the traffic through C, to see if the lower propagation delays experienced on that route would produce a correspondingly lower overall system delay. Essentially, will the corresponding decrease in propagation time make up for the increase in queue time experienced on the route? By routing more traffic through C, the traffic load at C would increase, causing increased queuing time for destinations C, E (through C), and F. However, it would also decrease the queue time experienced at destinations B, D, and E (through B) which have a higher total propagation time.

I ran two additional simulations, each with three trials. Both simulations used an arrival rate of $\lambda_A = 17\,pps$. The first simulation used a small difference in the route selection to destination E: routing 46.7% of E's traffic through B, and sending the remaining 53.3% through C. This decision was made because the module generating destinations

only used 1% increments of the total traffic. Thus, the changes were from 15/15 to 14/16, and finally 10/20, where the values are the % of total traffic. Figure 5 shows the results of these trials, compared to the simulation with an even distribution.

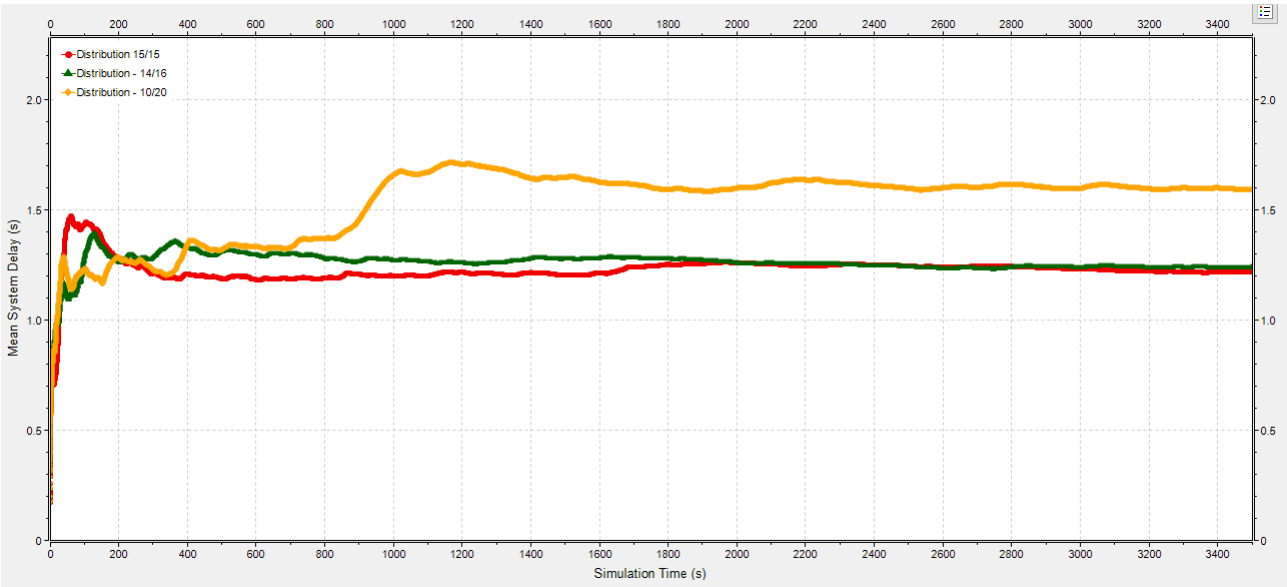

Figure 5: Network Delay with Uneven Routing to Destination E

The network delay of the original setup - with even routing had an average network delay of 1.1996 seconds. The overall network delay for the small difference had an average network delay of 1.211 seconds, while the large difference had an average delay of 1.566 seconds, over a 10 hour simulation.
Based on these results, I am confident that the optimal distribution is to evenly distribute the traffic between B and C. Any gains from reducing the propagation delay by 20 ms are negated by the corresponding increase in queue time.

5 Conclusions:

This project worked as a functional model of simulated network traffic. The traffic loads in simulation closely matched the desired network loads, with minor variations due to the probabilistic generation of destinations. One can see the effects of the path length: nodes D-F had a larger effect of propagation delay because they had to go through two paths. On the other hand, nodes B and C only encounter a single propagation delay, on the path from node A. Because all of the queues are stable, the throughput of a node i is equivalent to the arrival rate λ_i. This allows us to analyze the network as a sequence of M/M/1 queues because the interarrival times will still be exponentially distributed. If queues B or C were unstable, destinations D-F would need further analysis to accurately model.

Appendices

A Raw Data

Traffic Load

Node	Average Count	Percentage
B	54154	10.03
C	80705	14.95
D	134853	24.99
E	162120	30.04
F	107871	19.99

Table 7: Traffic Load for Case 1

Node	Average Count	Percentage
B	61412	10.04
C	91600	14.98
D	152887	25.00
E	183458	29.99
F	122293	19.99

Table 8: Traffic Load for Case 2

Per-Node System Delay

The following tables show the delay measured at each FIFO node in the system from the time a message entered a node's queue to the time it left its service.

Node	Average Delay (s)	Expected Delay (s)	% Difference
A	0.2008	0.2000	0.3958
B	0.3996	0.4000	0.0928
C	0.4006	0.4000	0.1541
D	0.1605	0.1600	0.2938
E	0.1833	0.1818	0.7876
F	0.1433	0.1429	0.3311

Table 9: System Delay for Nodes A-F with $\lambda_A = 15$ pps

Node	Average Delay (s)	Expected Delay (s)	% Difference
A	0.3326	0.3333	0.2250
B	0.6516	0.6667	2.2587
C	0.6630	0.6667	0.5501
D	0.1738	0.1739	0.0400
E	0.2051	0.2041	0.4802
F	0.1504	0.1515	0.7313

Table 10: System Delay for Nodes A-F with $\lambda_A = 17$ pps

Per Destination End-to-End Delay

Destination	Average Delay (s)	Expected Delay (s)	% Difference
B	0.660	0.660	0.007
C	0.651	0.650	0.144
D	0.861	0.860	0.105
E	0.865	0.862	0.383
F	0.806	0.803	0.410
Network:	**0.799**	**0.798**	**0.200**

Table 11: End-to-End Delays for Case 1

Destination	Average Delay (s)	Expected Delay (s)	% Difference
B	1.042	1.060	1.690
C	1.045	1.050	0.502
D	1.257	1.274	1.292
E	1.277	1.284	0.585
F	1.207	1.212	0.397
Network:	**1.120**	**1.210**	**0.822**

Table 12: End-to-End Delays for Case 2

B Simulation Files

```
[General]
network = Network_Forward
sim-time-limit = 10h
cpu-time-limit = 300s
#debug-on-errors = true
#record-eventlog = true

[Config NetworkA]
description = "Arrival rate of 15 pps"
repeat = 3
**.gen.sendIaTime =     exponential(0.06667s)
**.fifo[0].serviceTime = exponential(0.05s)
**.fifo[1].serviceTime = exponential(0.1s)
**.fifo[2].serviceTime = exponential(0.1s)
**.fifo[3].serviceTime = exponential(0.1s)
**.fifo[4].serviceTime = exponential(0.1s)
**.fifo[5].serviceTime = exponential(0.1s)

[Config NetworkB]
description = "Arrival rate of 17 pps"
repeat = 3
**.gen.sendIaTime =     exponential(0.05882s)
**.fifo[0].serviceTime = exponential(0.05s)
**.fifo[1].serviceTime = exponential(0.1s)
**.fifo[2].serviceTime = exponential(0.1s)
**.fifo[3].serviceTime = exponential(0.1s)
**.fifo[4].serviceTime = exponential(0.1s)
**.fifo[5].serviceTime = exponential(0.1s)
```

Simulation Initialization File - omnetpp.ini

```
network Network_Forward
{
    @display("bgb=564,468");
    submodules:
        gen: Source {
            parameters:
                @display("p=45,136");
        }
        fifo[6]: Fifo {
            @display("p=239,136,c");
        }
        sink[6]: Sink {
            @display("p=427,136,c");
        }
    // fifo[1]-fifo[5] correspond to fifoB --> fifoF
    connections:
        gen.out --> fifo[0].in++;
        fifo[0].out++ --> {  delay = 60 ms; } --> fifo[1].in++;
        fifo[0].out++ --> {  delay = 50 ms; } --> fifo[2].in++;
        fifo[1].out++ --> {  delay = 40 ms; } --> fifo[3].in++;
        fifo[1].out++ --> {  delay = 30 ms; } --> fifo[4].in++;
        fifo[2].out++ --> {  delay = 20 ms; } --> fifo[4].in++;
        fifo[2].out++ --> {  delay = 10 ms; } --> fifo[5].in++;

        // Output gates to sinks:
        fifo[0].sink --> sink[0].in;
        fifo[1].sink --> sink[1].in;
        fifo[2].sink --> sink[2].in;
        fifo[3].sink --> sink[3].in;
        fifo[4].sink --> sink[4].in;
        fifo[5].sink --> sink[5].in;

}
```

Network Description (NED) File

```
void Fifo::endService(cMessage *msg)
{
    EV << "Completed service of " << msg->getName() << endl;
    int dest = check_and_cast<Netmsg *>(msg)->getDestination();
    // At destination:
    if (getIndex() == dest) {
        send(msg, "sink");
    }
    // At E from C
    else if (getIndex() == 4 && dest == 44 ) {
        send(msg, "sink");
    }
    // At A:
    else if (getIndex() == 0) {
        if (dest == 1 || dest == 3 || dest == 4)
            send(msg, "out", 0);
        else {
            send(msg, "out", 1);
        }
    }
    // At B:
    else if (getIndex() == 1) {
        if (dest == 3) {
            send(msg, "out", 0);
        }
        else {
            send(msg, "out", 1);
        }
    }
    // At C:
    else if (getIndex() == 2) {
        if (dest == 44) {
            send(msg, "out", 0);
        }
        else {
            send(msg, "out", 1);
        }
    }
}
```

FIFO endService Function

```
void Source::handleMessage(cMessage *msg)
{
    ASSERT(msg == sendMessageEvent);

    Netmsg *job = new Netmsg("job");
    // Produce source and destination address using probabilities.
    int probDest = intuniform(1, 100);
    int dest;
    if (probDest <= 10) {
        // Destination = B
        dest = 1;
    }
    else if (probDest <= 25) {
        // Destination = C
        dest = 2;
    }
    else if (probDest <= 50) {
        // Destination = D
        dest = 3;
    }
    else if (probDest <= 65) { //--> Using below probability to test uneven
   distribution through B/C
    //else if (probDest <= 60) {
        // Destination = E --> Send through B
        dest = 4;
    }
    else if (probDest <= 80) {
        // Destination = E --> Send through C
        dest = 44;
    }
    else {
        // Destination = F
        dest = 5;
    }
    job->setDestination( dest );
    send(job, "out");

    scheduleAt(simTime()+par("sendIaTime").doubleValue(), sendMessageEvent);
}
```

Source Node - destination logic

```
namespace fifo;

message Netmsg {
    int destination;
}
```

Netmsg Type