

Air Force Institute of Technology
Department of Electrical and Computer Engineering

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

Author: Micah Hayden

April 19, 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 $\mu_{\bar{A}} = 20pps$.

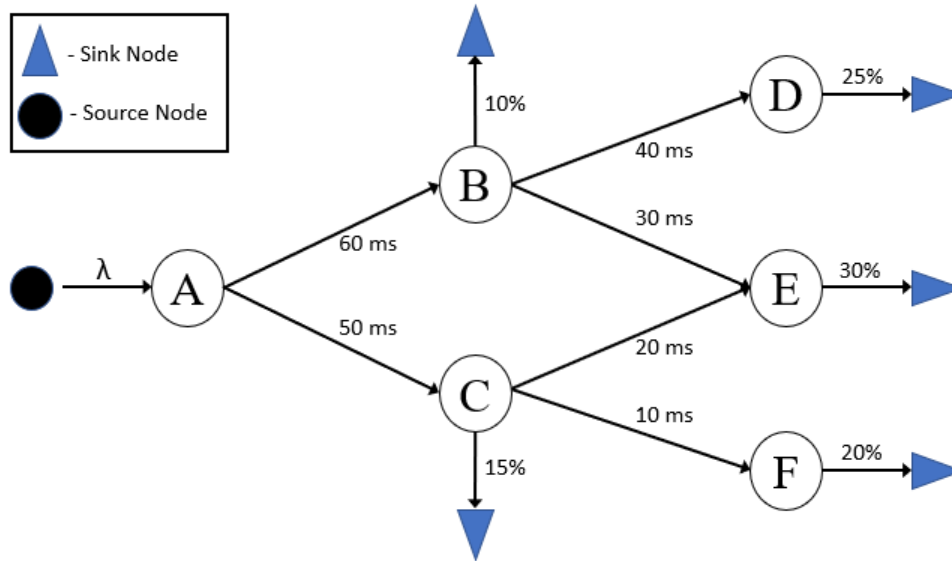


Figure 1: Network Topology of Routing Network

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 an arrival rate into A $\lambda_A = 15 pps$, and Case 2 has arrival rate $\lambda_A = 17 pps$.

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 $A \rightarrow B$ and $A \rightarrow C$, with each receiving $\frac{1}{2}$ of E 's traffic.¹

$$\begin{aligned}
 Load_{A \rightarrow B} &= Load_B + Load_D + \frac{1}{2} \cdot Load_E \\
 &= 0.10 + 0.25 + \frac{1}{2} \cdot 0.30 \\
 &= 0.50
 \end{aligned}$$

$$\begin{aligned}
 Load_{A \rightarrow C} &= Load_C + Load_F + \frac{1}{2} \cdot Load_E \\
 &= 0.15 + 0.20 + \frac{1}{2} \cdot 0.30 \\
 &= 0.50
 \end{aligned}$$

This indicates that links $A \rightarrow B$ and $A \rightarrow C$ will each receive half of the traffic departing A . Because $\mu_A > \lambda_A$, the departure rate of A equals λ_A . Thus, the arrival rate of nodes B and C equals $0.5 \cdot \lambda_A$. The departure rate from B and C must also equal their respective arrival rates because $\mu_B = \mu_C = 10 \text{ pps}$, and $10 \text{ pps} > 0.5 \cdot \lambda_A$ for both cases. Nodes B and C follow a similar pattern, but must also account for traffic routed to the sink (i.e. packets at B with destination = B). Given the departure rate equals the arrival rate for any node, the following equation must hold:

$$\lambda_{dest} = \lambda_{source} \cdot \frac{\sum \text{load of all links from destination}}{\sum \text{load of all links from source}} \quad (1)$$

I utilized the above relationship to determine the actual arrival rates of each node 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: $\lambda_{expected}$ 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: $\lambda_{expected}$ for Case 2

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 .

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 2.

$$E[r] = \frac{1}{\mu - \lambda} \quad (2)$$

Tables 4 and 5 below show these times.

Node	$E[r]_{\text{node}}$ (s)
A	0.200
B	0.400
C	0.400
D	0.160
E	0.182
F	0.143

Table 4: $E[r]$ for Case 1

Node	$E[r]_{\text{node}}$ (s)
A	0.333
B	0.667
C	0.667
D	0.174
E	0.204
F	0.152

Table 5: $E[r]$ for Case 2

The times shown above indicate the time in the system **at** a particular node. To find the expected system delay of a packet, one must also account for all previous nodes, as well as the 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. This produces the following expected end-to-end delay (source to destination), for packets routed to each node:

Node:	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 6: Expected end-to-end delay by destination

The overall network end-to-end delay is a weighted average of the values shown in Table 6 with the expected Traffic Loads shown in Table 1. This yields the following:

$$\text{End-to-End Delay}_{\text{Case 1}} = 0.798s$$

$$\text{End-to-End Delay}_{\text{Case 2}} = 1.210s$$

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 approaches the desired proportions.

³To account for the duplicate paths to node E, there were two integers for E

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. 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 FIFO queues and 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 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.

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 for each of the three simulation runs. The traffic load was then calculated as a percentage of the number of total packets arriving to any destination for the respective case.

As seen in Figures 2 and 3, my results closely matched the expected traffic load.

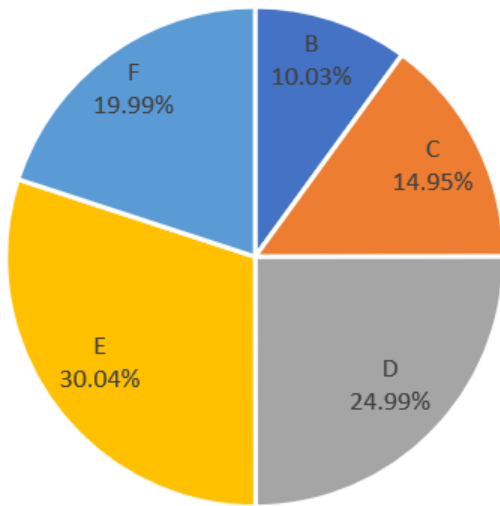


Figure 2: Traffic Load with $\lambda_A = 15 pps$

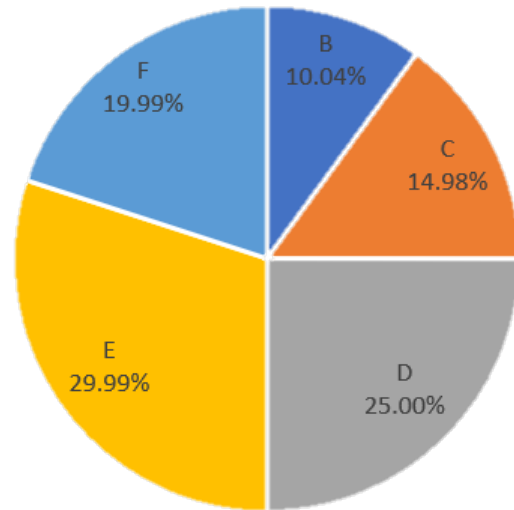


Figure 3: Traffic Load with $\lambda_A = 17 pps$

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

4.2 System Delay:

I used the statistics collected at each sink node to compare the per-node average delay for each case. Both Case 1 and Case 2 performed well, with results closely matching the expectations. As expected, the largest end-to-end delays were at Nodes D-F. This occurs because to reach those nodes, they must go through two queue nodes, and two links before arriving at their destination's queue. The per-node end-to-end delays for Case 1 produced an average percent

difference of 0.21% from the expected delays, while Case 2 had an average percent difference of 0.893%.⁴ Tables 9 and 10, shown in Appendix A, show the per-node delays.

I calculated the average end-to-end delay for Cases 1 and 2 by using a weighted average of system delays for destinations B-F. This weighted average used the observed traffic loads, as discussed in Section 4.1, as the weights; and the system delays reported by each sink as the data, shown in Tables 9 and 10.

The average end-to-end delay for Case 1 was 0.7997 seconds, a 0.26% difference than the expected result. The average end-to-end delay for Case 2 was 1.996 seconds, a 0.822% difference than the expected result.

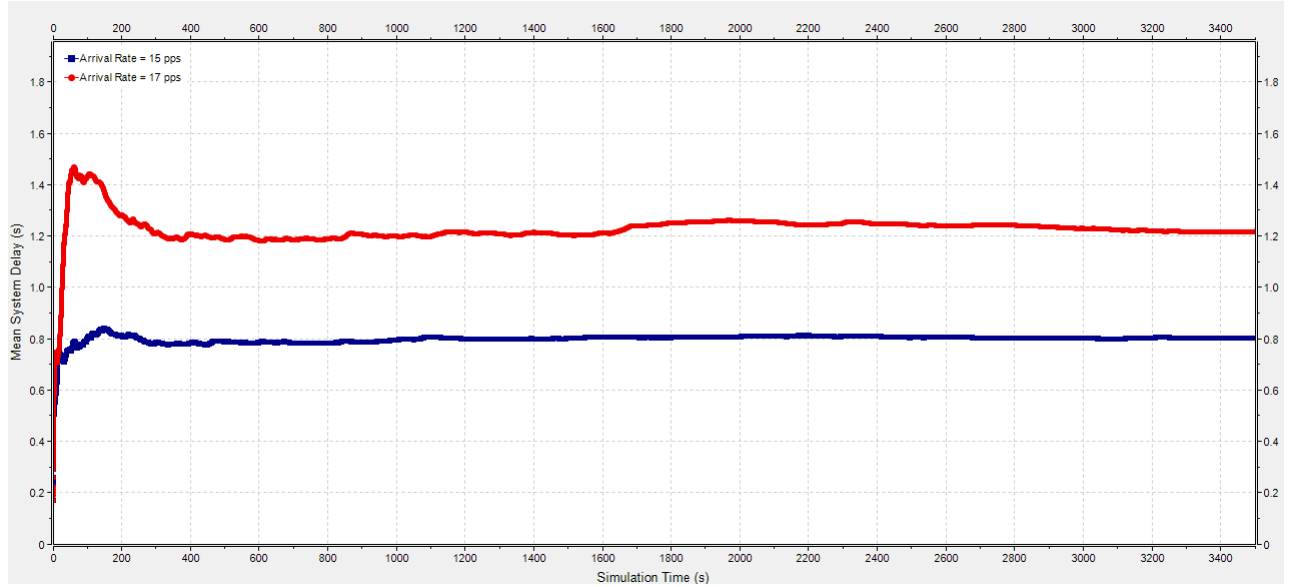


Figure 4: Mean System Delay

Figure 4 ends at $t = 3500$ seconds, although the simulation ran to 36,000 seconds. This decision was made to better show how the end-to-end delay stabilized as time increased.

4.3 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. The second simulation used a much larger difference: sending 33.3% through B, and sending 66.7% through C. Figure 5 shows the results of these trials, compared to the simulation with an even distribution.

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

⁴This % difference simply averaged the % differences for each node, not based on the traffic load of each node.

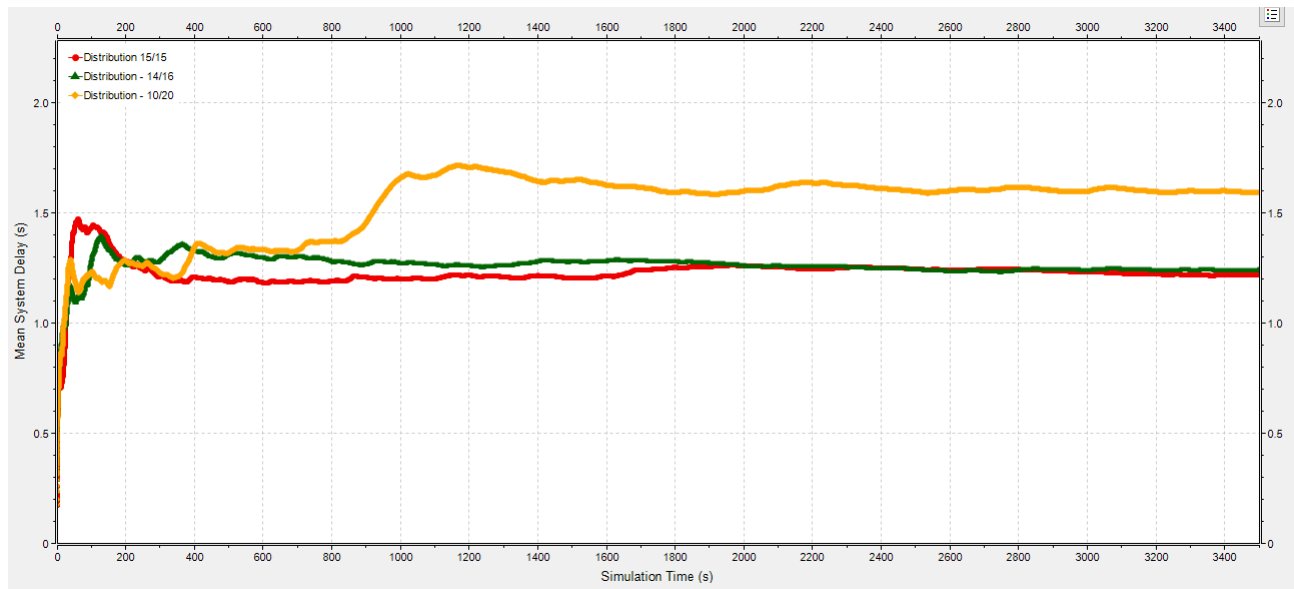


Figure 5: Network Delay with Uneven Routing to Destination E

had an average delay of 1.566 seconds.

Based on these results, I am confident that the optimal distribution is to evenly distribute the traffic between B and C. Any gains because of a reduced propagation delay are negated by the corresponding increase in queue time.

5 Conclusions:

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 End-to-End Delay

Node	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:	0.800	0.798	0.258

Table 9: End-to-End Delays for Case 1

Node	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 10: End-to-End Delays for Case 2

B Simulation Files

```
1 [General]
network = Network_Forward
3 sim-time-limit = 10h
cpu-time-limit = 300s
5 #debug-on-errors = true
#record-eventlog = true
7
[Config NetworkA]
9 description = "Arrival rate of 15 pps"
repeat = 3
11 **.gen.sendIaTime = exponential(0.06667s)
**.fifo[0].serviceTime = exponential(0.05s)
13 **.fifo[1].serviceTime = exponential(0.1s)
**.fifo[2].serviceTime = exponential(0.1s)
15 **.fifo[3].serviceTime = exponential(0.1s)
**.fifo[4].serviceTime = exponential(0.1s)
17 **.fifo[5].serviceTime = exponential(0.1s)

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

Simulation Initialization File - omnetpp.ini


```

network Network_Forward
2 {
    @display("bgb=564,468");
    submodules:
    4     gen: Source {
        parameters:
    6         @display("p=45,136");
    8     }
    fifo [6]: Fifo {
    10     @display("p=239,136,c");
    12     }
    sink [6]: Sink {
    14     @display("p=427,136,c");
    16     }
    // fifo [1]–fifo [5] correspond to fifoB —> fifoF
    connections:
    18     gen.out —> fifo [0].in++;
    fifo [0].out++ —> { delay = 60 ms; } —> fifo [1].in++;
    20     fifo [0].out++ —> { delay = 50 ms; } —> fifo [2].in++;
    fifo [1].out++ —> { delay = 40 ms; } —> fifo [3].in++;
    22     fifo [1].out++ —> { delay = 30 ms; } —> fifo [4].in++;
    fifo [2].out++ —> { delay = 20 ms; } —> fifo [4].in++;
    24     fifo [2].out++ —> { delay = 10 ms; } —> fifo [5].in++;

    // Output gates to sinks:
    26     fifo [0].sink —> sink [0].in;
    fifo [1].sink —> sink [1].in;
    28     fifo [2].sink —> sink [2].in;
    fifo [3].sink —> sink [3].in;
    30     fifo [4].sink —> sink [4].in;
    fifo [5].sink —> sink [5].in;
    32 }

```

Network Description (NED) File

```

1 void Fifo::endService(cMessage *msg)
2 {
3     EV << "Completed service of " << msg->getName() << endl;
4     int dest = check_and_cast<Netmsg *>(msg)->getDestination();
5     // At destination:
6     if (getIndex() == dest) {
7         send(msg, "sink");
8     }
9     // At E from C
10    else if (getIndex() == 4 && dest == 44 ) {
11        send(msg, "sink");
12    }
13    // At A:
14    else if (getIndex() == 0) {
15        if (dest == 1 || dest == 3 || dest == 4)
16            send(msg, "out", 0);
17        else {
18            send(msg, "out", 1);
19        }
20    }
21    // At B:
22    else if (getIndex() == 1) {
23        if (dest == 3) {
24            send(msg, "out", 0);
25        }
26        else {
27            send(msg, "out", 1);
28        }
29    }
30    // At C:
31    else if (getIndex() == 2) {
32        if (dest == 44) {
33            send(msg, "out", 0);
34        }
35        else {
36            send(msg, "out", 1);
37        }
38    }
39 }

```

FIFO endService Function

```

1 void Source::handleMessage(cMessage *msg)
2 {
3     ASSERT(msg == sendMessageEvent);
4
5     Netmsg *job = new Netmsg("job");
6     // Produce source and destination address using probabilities.
7     int probDest = intuniform(1, 100);
8     int dest;
9     if (probDest <= 10) {
10         // Destination = B
11         dest = 1;
12     }
13     else if (probDest <= 25) {
14         // Destination = C
15         dest = 2;
16     }
17     else if (probDest <= 50) {
18         // Destination = D
19         dest = 3;
20     }
21     else if (probDest <= 65) { //--> Using below probability to test uneven
distribution through B/C
22         //else if (probDest <= 60) {
23             // Destination = E --> Send through B
24             dest = 4;
25         }
26         else if (probDest <= 80) {
27             // Destination = E --> Send through C
28             dest = 44;
29         }
30         else {
31             // Destination = F
32             dest = 5;
33         }
34         job->setDestination( dest );
35         send(job, "out");
36
37         scheduleAt(simTime()+par("sendLaTime").doubleValue(), sendMessageEvent);
38     }
39 }

```

Source Node - destination logic

```

1 namespace fifo;
2
3 message Netmsg {
4     int destination;
5 }

```

Netmsg Type