ECS 152A/EEC173A Computer Networks Project

Spring Quarter 2020

Simulation Analysis of an IEEE 802.x Based Network

Project I

Department of Computer Science
University of California
Davis, CA 95616

1. Introduction

In this project we will build a discrete event simulation model to study the behavior of an IEEE 802.x based Network. The following references and resources will be used in this project.

- This write-up
- Class lectures and discussions
- Additional material and references provided if necessary

Schedule, milestones, and other administrative issues related to this project are the following:

- It will be a team project with at least 2 members and at most 3 members per team.
- Project I will be the implementation of a single server queue using programming language of your preference. Successful completion of Project I is important and will help in quickly completing Project II.
- You will have to submit both a soft copy of the code and a report explaining your implementation and the results (canvas upload). For both Projects, you may be asked to explain your code. More details about submission guidelines will be made available later.
- Project I is due **May. 1, 2020 by 5:00PM.**
- There may be questions related to this project both in quizzes and the final exam.

This write-up contains a) a brief description of discrete event simulation, b) description of Project 1, c) discussion of the overall logic of Project 1, and d) results of Project 1.

2 Discrete Event Simulation

Consider the transmitter shown in Figure 1. It consists of a link processor, which transmits packets into the link and a finite buffer to hold packets. Packet arrival process is random and the inter-arrival time between packets follow a particular distribution function. Furthermore, packets come in different sizes and the time to transmit a packet depends on its size. When a packet arrives at the transmitter, there can be one of two cases – 1) the transmitter is idle and 2) the transmitter is busy transmitting another packet

and there are 0 or more packets waiting in the buffer. In the first case, the arriving packet is given to the link processor, which immediately starts transmitting the packet; while in the second case, the packet is queued in the buffer behind other queued packets. While the buffer is not empty, the link processor retrieves a packet at a time from the buffer in a first-in-first-out (FIFO) order and transmits it onto the link. The link processor is never idle when there is a packet waiting in the buffer.

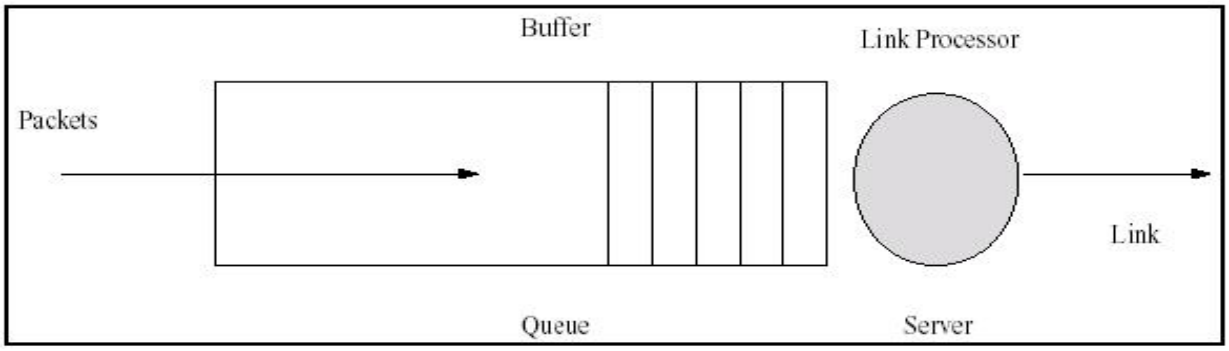

Figure 1: Model of a simple transmitter.

We would like to study the behavior of the transmitter using discrete event simulation. In particular, given a distribution of the packet inter-arrival times and a distribution of the packet transmission times, we are interested in determining the average number of packets in the buffer, the total number of dropped packets, and the percentage of time the link processor is busy. The latter is also referred to as the link processor utilization.

In discrete event simulation, we study the system by considering key events in the system and letting time proceed in discrete steps associated with those events. This is opposed to time-based simulation, where the simulated system is studied as time progresses in small fixed increments; all the events that occur in a particular time interval are processed in differing groups. In this study, we will use discrete event simulation.

In the case of our transmitter, the key events are 1) arrival of a packet and 2) departure of a packet, i.e., the end of the transmission of a packet. To simulate the transmitter, we will monitor the transmitter only at times when these particular events occur and advance time in discrete steps from one event to the next. It turns out that for the specific distribution of the inter-arrival and the transmission time that we will use in this study, the average number of packets in the transmitter at any arbitrary time is the same as the average number of packets seen by an arriving or a departing packet.

Figure 2 shows one possible sequence of packet arrivals and departures. For example, A1 is the arrival time of the first packet and D1 is the time when transmission of the first packet is completed, and the packet departs. Similarly, A2 and D2 are the arrival and departure times of the second packet, respectively. Note that the inter-arrival times and the packet transmission times are random.

From the arrival and the departure times, we can obtain the queue-length, i.e., the total number of packets in the transmitter (which the sum of the packets in the queue and including any in the processor) as a function of time. Figure 3 shows the queue-length as a function of time for the example sequence. To obtain the mean number of packets in the transmitter, we can determine the area under the curve and

divide it by the total time.

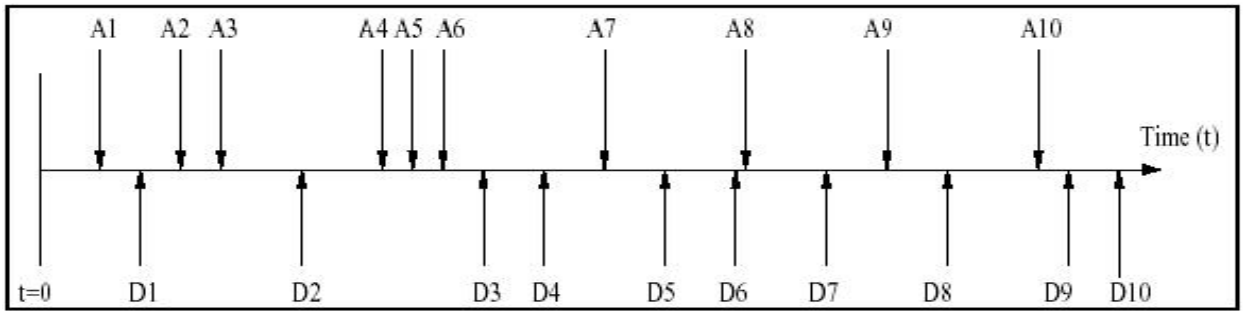

Figure 2: An example sequence of arrivals and departures.

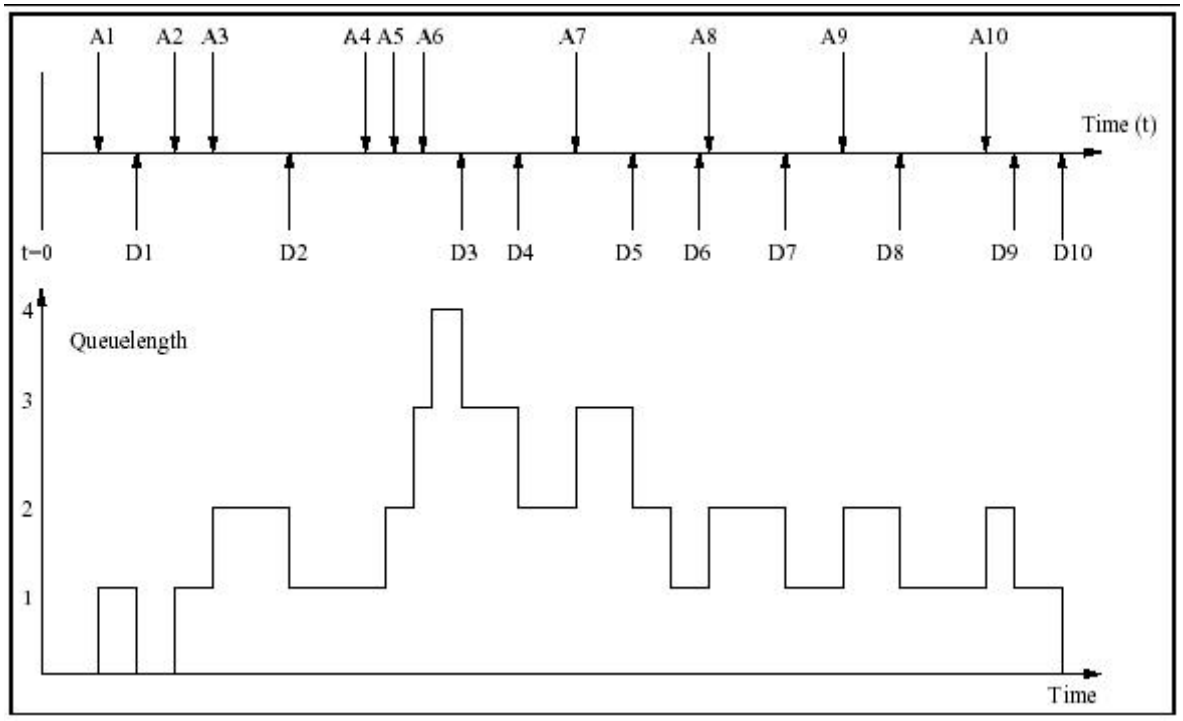

Figure 3: The instantaneous queue-length as a function of time for the example sequence.

3. Project I: Simulation Model of a Single Server Queue

In this Project we will develop a discrete event simulation of the transmitter shown in Figure 1. In the following discussion, the link processor will be referred to as the server and the buffer as the queue. We will assume that the queue and the server have the following characteristics:

- The queue can hold *a maximum* number of packets denoted by MAXBUFFER. (Make sure that you can simulate the case when the buffer size is infinite (i.e., no packets are dropped.)

- The arrivals to the queue follow these characteristics:
    1. There is only one arrival at a particular instant of time.
    2. The inter-arrival time between packets follows a negative exponential distribution with rate λ packets/second.

- The server transmits the packets one at a time in a FIFO (first-in first-out) order.

- The length of the packets varies and hence the transmission time also varies. The transmissions time is also negative exponentially distributed with rate μ packets/second.

3.1 Overview

As mentioned before, in discrete event simulation, we only consider the points in time when the events in consideration occur. In our case, there are two events, 1) the arrival event and 2) the departure event. The departure events depend on the arrival events, since packets may get processed only upon arrival. So the key idea is that we will generate arrival events and create appropriate departure events and monitor the state of the queue and the server to determine the average queue-length and the mean server utilization.

In order to implement the key idea, we will use the following main data structures:

1. **Event:** The key components of an event are the following.
    - Event-time: This is the time when the event occurs. For an arrival event it is the time the packet arrives at the transmitter and for a departure event it is the time when the server is finished transmitting the packet.
    - Type of the event which can one of two types 1) an arrival event or 2) a departure event.
    - A pointer to the next event.
    - A pointer to the previous event.

2. **Global Event List (GEL)**: This will maintain all the events sorted in increasing order of time. The operations on the GEL will be: (1) insert an event and (2) remove the first event. We can implement the GEL using a double linked list since we will be inserting at random points.

3. **First-In First-Out Queue:** This will model the buffer and buffer packets that are waiting to be processed. The key operations in the queue will be 1) inserting a packet at the end of the queue; 2) removing a packet from the front of the queue and 3) determining whether the incoming packet needs to be dropped.

It is important to remember that we will be maintaining our own clock. We will not be using the system clock. The main code will essentially look like this:

```
Initialize;
```

```
for (i = 0; i < 100000; i++){
      1.  get the first event from the GEL;
      2.  If the event is an arrival then process-arrival-event;
      3. Otherwise it must be a departure event and hence process-
         service-completion;
}
output-statistics;
```

3.2 Initialization

The initialization procedure will initialize the data structures and other key variables.

- Initialize all the data structures. Initialize all the counters for maintaining the statistics. Let *length* denote the number of packets in the queue (including, if any, being transmitted by the server). We will initialize length to be 0. Also, let say we use the variable *time* to denote the current time. We initialize time to 0.
- Set the service rate and the arrival rate of the packets.
- Create the first arrival event and then insert it into the GEL. The event time of the first arrival event is obtained by adding a randomly generated inter-arrival time to the current time, which is 0.

3.3 Processing an Arrival Event

When we process an arrival event, we need to do the following tasks:

- Set current time to be the event time.
- Since we generate one arrival at a time, we first schedule the next arrival event. This is done as follows:
    1. Find the time of the next arrival, which is the current time (which is maintained by the time variable) plus a randomly generated time drawn from a negative exponentially distributed random variable with rate λ.
    2. Create a new packet and determine its service time which is a randomly generated time drawn from a negative exponentially distributed random variable with rate μ.
    3. Create the new arrival event.
    4. Insert the event into the event list. Note that there can be other events in the event list. The newly created event must be placed in the right place so that the events are ordered in time.
- Process the arrival event. In particular we do the following:
    a) If the server is free i.e., if (length == 0), the packet can be immediately scheduled for transmission. Since we know how long it will take to transmit the packet, we know when (relative to the current time) the packet will depart. We schedule a departure event for that time. In summary we do the following:
        1. Get the service time of the packet.

2. Create a departure event at time which is equal to the current time plus the service time of the packet.
3. Insert the event into the GEL. Again, we need to make sure that we insert the event at the right place so that GEL is sorted in time.

b) If the server is busy, i.e., if (length > 0):

- If the queue is not full, i.e. if (length-1 < MAXBUFFER), put the packet into the queue. (Remember that *MAXBUFFER* is the maximum number of packets the buffer can hold (this may be finite or infinite); *length* is the total number of packets in the buffer plus the one that is being processed, if any.
- If the queue is full, then drop the packet; record a packet drop.
- Since this is a new arrival event, we increment the length.
- Update statistics which maintain the mean queue-length and the server busy time.

3.4 Processing a Departure Event

• Set current time equal to the event time.
• Update statistics which maintain the mean queue-length and the server busy time.
• Since this is a packet departure, we decrement the length.
• If the queue is empty, i.e., if (length == 0), do nothing.
• If queue is not empty, i.e., if (length > 0), then we do the following:

1. Dequeue the first packet from the buffer;
2. Create a new departure event for a time which is the current time plus the time to transmit the packet.
3. Insert the event at the right place in the GEL.

3.5 Collecting Statistics

We will be interested in the following performance measures:

• Utilization: What fraction of the time is the server busy? To determine this, keep a running count of the time the server is busy. When the simulation terminates, the time for which the server is busy divided by the total time will give the mean server utilization.

• Mean queue length: What is the mean number of packets in the queue as seen by a new arriving packet? As mentioned before, to do this we maintain the sum of the area under the curve and when the simulation terminates, the area divided by the total time will give the mean queue length. *Think a simple way of doing this.*

• Number of packets dropped: What is the total number of packets dropped with different λ values? To determine this, keep a running count of the number of packets dropped. Notice that you have to determine whether the packet needs to be dropped when it arrives at the buffer.

3.6 Generating Time Intervals in Negative Exponential Distribution

As mentioned before, both the inter-arrival time and the transmit time follow the negative exponential distribution. Let x be a random variable that follows a negative exponential distribution with rate α. x can be generated using the following equation: $x = (-1/\alpha) \log_e(1- u)$, where u is a uniformly distributed random variable between 0 and 1. To generate the inter-arrival time we use λ for the rate parameter α. For generating the transmission time, we use μ for the rate parameter α. The code for generating these random variables is given below.

```
double negative-exponentially-distributed-time (double rate)
{
   double u;
   u = drand48();
   return ((-1/rate)*log(1-u));
}
```

3.7 Project I Experiments

1. Assume that μ = *1 packet/second*. Plot the queue-length and the server utilization as a function of λ for λ = 0.1, 0.2, 0.4, 0.5, 0.6, 0.80, 0.90 packets/second when the buffer size is infinite.
2. Mathematically compute the mean queue lengths and the server utilization and compare with the simulation results (The mathematical formulation will be discussed in class).
3. Assume that μ = *1 packet/second*. Plot the total number of dropped packets as a function of λ for λ = 0.2, 0.4, 0.5, 0.6, 0.8, 0.9 packets/second for MAXBUFFER = 1, 20, and 30.

3.8 Extra Credit

This part is not required for regular submission. You can do the implementation for extra credit. This part requires some prior research before implementation. You can discuss with the TA for available resources.

In our regular implementation, we assume packet arrival time is negative exponentially distributed. However, it is observed that Internet traffic follows some sort of self-similarity phenomenon (not negative exponential distribution). The self-similarity means that the traffic had similar statistical properties at a range of timescales: milliseconds, seconds, minutes, hours, days, weeks, etc. Self-similar traffic can be better represented by heavy-tailed distributions, e.g., Pareto distribution. To generate self-similar traffic, the packet arrival rate (λ) should follow the Pareto distribution (instead of negative exponential distribution as in Sec. 3.6). Therefore, to get extra credit, you must implement packet arrival rate (λ) which follows Pareto distribution. Service/transmission time will follow negative exponential distribution as before. After the implementation, rerun the experiments of Sec. 3.7, and compare the results with the previously-obtained results.