

Project – Simulation Analysis of an Ethernet Based Local Area Network (LAN)

1 Introduction

In this project we will build a discrete event simulation model to study the behavior of an Ethernet-based Local Area Network (LAN). The following references and resources will be used in this project.

- Handouts and class lectures
- W. Stallings, Data and Computer Communications, 7th ed., Prentice Hall, 2000. Chapter 16.

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

- It will be a team project with at most 2 members per team.
- The project will be divided into two phases. Phase 1 will be the implementation of a single server queue while Phase 2 will be the modeling and analysis of an Ethernet based LAN. Successful completion of Phase 1 is important and will significantly help in quickly completing Phase 2.
- A submission guideline for Phase 1 is described in section 3.8 in this document regarding submission and grading.
- **Due date of Phase 1 is April 29th, 2004 by 1pm.**

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

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.

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 what is the average number of packets in the buffer, the average waiting time in the system, the average waiting time in the queue, 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.

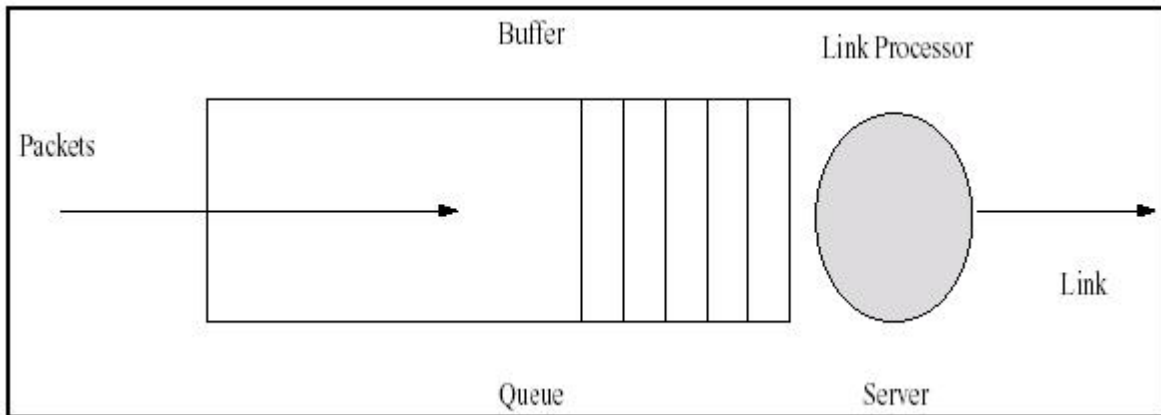


Figure 1: Model of a simple transmitter.

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.

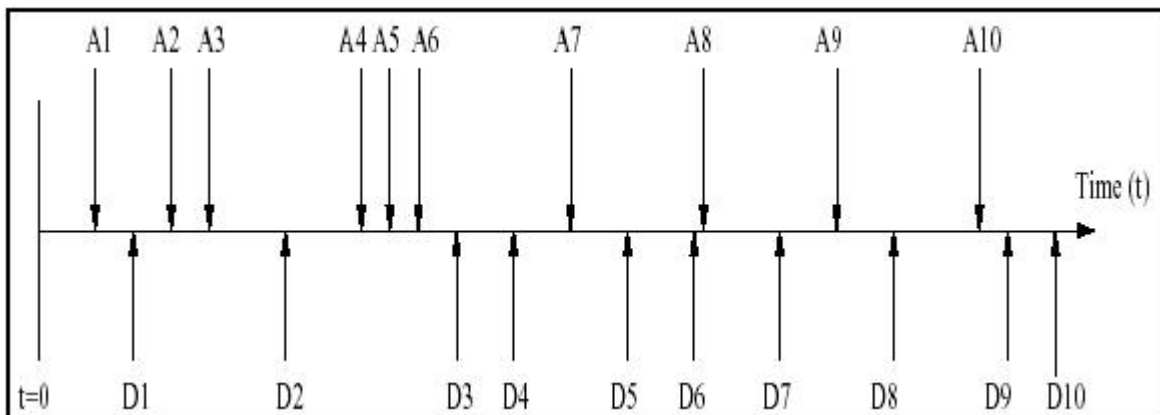


Figure 2: An example sequence of arrivals and departures.

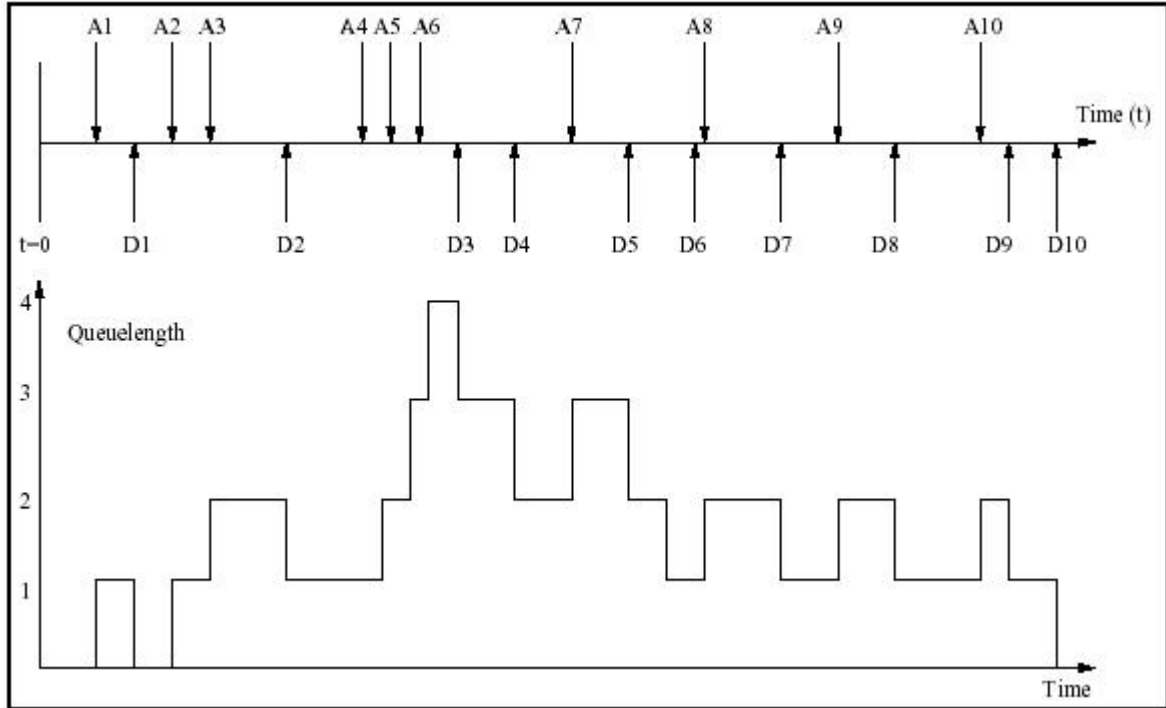


Figure 3: The instantaneous queue-length as a function of time for the example sequence.

3 Phase I: Simulation Model of a Single Server Queue

In this phase 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 *infinite* or *finite*¹ number of packets (specified by user).
- The arrivals to the queue follow these characteristics:
 1. There is only one arrival at a time.
 2. The interval arrival time between packets follows a negative exponential distribution² 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

¹ We will specify the maximum number of packets the buffer can hold later in class.

² We will define this later.

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. The key components of an event are:
 - 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 important³ 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 0⁴.
- Set the service rate and the arrival rate of the packets.

³ This will be discussed again later.

⁴ We maintain our own time; we do not use the system (real time) clock.

- 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:
 - 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.
 - If the server is busy, i.e., if ($length > 0$):
 - If the queue is not full, i.e. if ($length-1 < MAXBUFFER^5$), put the packet into the queue.
 - 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 measurements of the system behavior.

3.4 Processing a Departure Event

- Set current time equal to the event time.
- Update statistics which maintain the measurements of the system behavior.
- 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:

⁵ 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.

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.*
- Average waiting time in system: What is the average waiting time in the system for each packet? To measure this quantity, accumulate the waiting time in system for all packets, then divide it by the total number of packets being processed.
- Average waiting time in queue: What is the average waiting time in the queue for each packet? To measure this quantity, accumulate the waiting time in queue for all packets, then divide it by the total number of packets being processed.
- 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 for a 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 Seeding the Number Generator

You are **required** to seed the number generator with the current system time at the beginning of each program run (perhaps in your initialization step). This will help randomize the results between each program execution. Points will be deducted if you do not do this.

➤ If using Java:

This is done for you automatically by the Java Runtime Environment when you create an object of type 'Random.' For help with this, see the Java API 1.4 documentation (<http://java.sun.com/j2se/1.4.1/docs/api/>)

➤ If using C/C++:

`drand48()` and/or `rand()` have separate seed functions so you may need to seed both. If you use random functions other than these, it is up to you to find the correct method to seed them with the current time. **Remember to include the appropriate header files.** See the function's man pages for more information. Code to seed both `drand48()` and `rand()` follows:

```
// Seed number generator
{
    struct timeval t;
    if( gettimeofday(&t, NULL) != 0) {
        cerr << "gettimeofday() - Couldn't get time of day" << endl;
    }
    else {
        // Seed drand48()
        srand48( (t.tv_sec + t.tv_usec) );

        // Seed rand()
        srand( (unsigned int)(t.tv_sec + t.tv_usec) );
    }
}
```

➤ If using some other language:

Your responsibility.

3.8 Phase I Experiments

- With an infinite buffer:
 - Assume that $\mu = 2$ packet/second. Plot (1) the server utilization, (2) the mean queue length, (3) the average waiting time in system, and (4) the average waiting time in queue as a function of λ for $\lambda = 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8$ packets/second.
- With a finite buffer size of 5 and 10:
 - Assume that $\mu = 2$ packet/second. Plot the number of packets dropped as a function of λ for $\lambda = 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8$ packets/second.

Submission Guideline for Phase 1 (Due on 4/29/2004 by 1pm)

- Hardcopy (to be turned into the HW box)
 - Please submit all the source codes. If you use STL in your project, you *DO NOT* need to submit those. If you use other templates, please only submit the .h files. Please try to print 2 pages of materials on 1 sheet to save paper. (i.e. using `enscript -2r <filename>`)
 - With an infinite buffer, assume that $\mu = 2$ packet/second and λ varies from 0.2 to 1.8 packets/second with increments of 0.2:
 1. Server Utilization – Please provide a table with the experimental results versus different values of λ . Plot a graph of *server utilization vs. lambda*.
 2. Mean Queue Length – Please provide a table with the experimental results versus different values of λ . Plot the *mean queue length vs. lambda*.
 3. Average Waiting Time in System – Please provide a table with the experimental results versus different values of λ . Plot the *average waiting time in system vs. lambda*.
 4. Average Waiting Time in Queue – Please provide a table with the experimental results versus different values of λ . Plot the *average waiting time in queue vs. lambda*.
 - With a buffer size of 5 and 10, assume that $\mu = 2$ packet/second and λ varies from 0.2 to 1.8 packets/second with increments of 0.2:
 1. Number of Dropped Packets – Please provide a table with the experimental results versus different values of λ for buffer sizes of 5 and 10. Plot the *number of packets dropped vs. lambda* for buffer sizes of 5 and 10.
 - Please mathematically derive the formula for Mean-Queue-Length using the Markov Chain queuing model and Balance Equations. More details will be discussed in lectures and/or discussion sections later.
 - Write a 1-2 page report to discuss the experimental results. In particular, your writeup should address the following issues:
 - Relationships between the experimental results, λ , and μ .
 - Compare the experimental results and theoretical values.
 - What constitutes the behavior (i.e. the results) of the system?
 - Any other important observations relating to the project you found.
- ❖ Writeup format:
 - Do the writeup in sections (i.e. one section for server utilization; one section for mean queue length; etc.)
 - Be clear, brief, precise, and consistent with the results obtained from your simulation.

Late works will not be accepted. Please follow the submission guideline for turning in Phase 1. Any submissions not following the guideline may result in severe points deduction.