

# ECE 358: Computer Networks

## Lab 1

# Queue Simulation Part 1

Presented by  
Bernie Roehl

# Lab General Information

- **Lab Instructor:** Bernie Roehl (*broehl@uwaterloo.ca*).
- There will be 3 labs that will be done by students individually:
  - LAB1: Development of a simulator of a single transmission system (weight 7%)
  - LAB 2: Development of a simulator for retransmission protocols (Weight 11%)
  - LAB 3: Encapsulation and network utility tools. Weight (7%)
- The total weight for the lab is 25% of the total course mark.

# Lab General Information

- Submission due dates:
  - Lab 1: Feb 2<sup>nd</sup> by 11:59 pm
  - Lab 2: March 16<sup>th</sup> by 11:59 pm
  - Lab 3: April 2<sup>nd</sup> by 11:59 pm.
- All submissions will be to the drop boxes on LEARN.
- Late lab reports will be scored a zero

| Lab TAs                   | email  |
|---------------------------|--|
| Omar Alhussein            | <a href="mailto:oalhusse@uwaterloo.ca">oalhusse@uwaterloo.ca</a> |
| Abdurhman Albasir         | <a href="mailto:aalbasir@uwaterloo.ca">aalbasir@uwaterloo.ca</a> |
| Yigit Ozcan               | <a href="mailto:yozcan@uwaterloo.ca">yozcan@uwaterloo.ca</a>     |
| Thirupathaiah<br>Vasantam | <a href="mailto:tvasanta@uwaterloo.ca">tvasanta@uwaterloo.ca</a> |

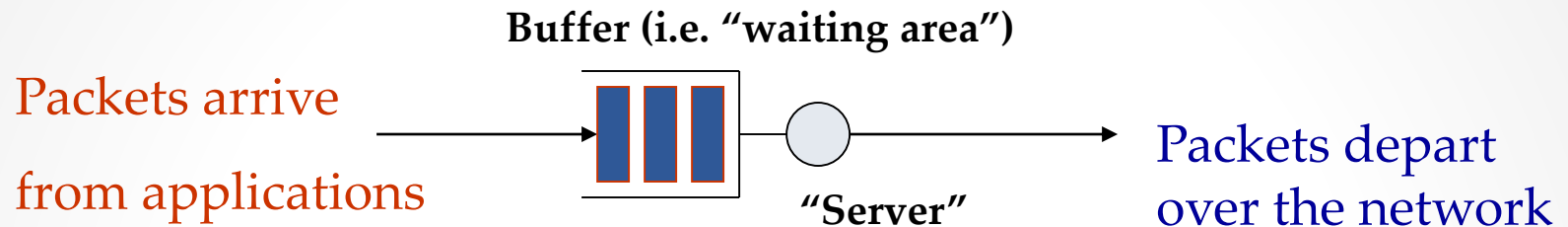
# Introduction

- On any given *node* in a network, there will be one or more applications that generate a *stream* of data
- The stream is broken up into individual *packets*, which are sent out over the lower-level network interface
- The low-level interface can only send packets at a certain maximum rate
- Applications tend to generate data in bursts
- What to do when packets arrive faster than they can be sent out?

# Buffering

- If packets are generated faster than they can be sent, they must be held in a *buffer*
- In practice, all buffers have a limited size (though in modern systems, this can be very large)
- If a buffer fills up, packets are discarded and must later be retransmitted
- Buffers are typically implemented as *queues* of packets

# Queue Model



Single Server Queuing Model

- ❑ The “server” is usually the transmission hardware
- ❑ Analogous to a bank teller or movie-theatre cashier
- ❑ There can be multiple tellers or clerks, or just one
- ❑ The arrival times and the service times are random

# Movie Theatre Analogy

- People (packets) don't arrive at the theatre (network node) all at once, but with random delays in between
- There is a line-up (buffer) to get to the cashier (server)
- There may be one or more cashiers available
- The length of time someone is at the cashier varies randomly depending on whether they want popcorn, drinks, candies, etc
- When someone has gotten what they need, they exit (are transmitted) and the next person (packet) from the line-up (buffer) comes forward
- If the line-up is too long, people leave (i.e. packets are dropped) and may try again later

# Main features of a queuing system

- ❑ **Arrival Process: Stochastic** process formed by packet arrivals (e.g., Poisson, Bernoulli, etc.).
- ❑ **Service Time: Random** variable denoting how long a packet will remain in service. Directly related to the length of a packet (which is why we treat as a random variable, since packets vary in size).
- ❑ **Number of Servers.**
- ❑ **Buffer Size.**
- ❑ **Service Discipline:** First in First Out (FIFO); Last in First Out (LIFO); Static priority, etc



# Kendall Notation

- ❑ Queues are typically described using a notation of the form:  
**A/S/C/K/D**
- ❑ Here **A** corresponds to the *arrival process* of the packets (more precisely the distribution of the inter-arrival time between packets).  $\lambda$  is the average number of packets per unit time. **A** can be M (Markovian), D (deterministic) or G (General).
- ❑ **S** corresponds to the *server process* for the packets. **S** can again be M, D or G.
- ❑ **C** corresponds to the *number of servers* (or network interfaces).
- ❑ **K** corresponds to the *buffer size*, i.e. the number of packets that can be accommodated in the “waiting room”. If omitted, the buffer is assumed to be infinitely large.
- ❑ **D** corresponds to the *queueing discipline*, which is almost always FIFO and is therefore often omitted from the notation.

# Kendall Notation

- Usually when a queue has infinite buffers, we drop the buffer size notation and service discipline and simply write: **A/S/C**
- Examples:
  - $M/M/1/10$  means Poisson arrivals, exponential service time, single server, 10-packet queue
  - $M/D/1$  means Poisson arrivals, Deterministic service time, single server, infinite queue
- In this lab, we'll be looking at  $M/M/1$  and  $M/M/1/K$  queues

# Lab 1 Objectives

- ❑ Become familiar with the basic elements of **discrete event simulation**. In particular, the notion of an *event scheduler* for simulating discrete event systems
- ❑ The generation of **random variables** for a given distribution (e.g. exponential).
- ❑ The behavior of a **single buffer queue** with different parameters, which is a key element of a computer network.
- ❑ The need to be careful about **when a system should be observed**, in order to obtain valid performance metrics.

# Why Simulation?

- Modern operating systems hide much of the low-level complexity of networking, and therefore make it difficult to study it at a low level
- Large networks may involve hundreds or thousands of nodes, which makes it cost-prohibitive to build entire systems just for testing and analysis
- Simulators allow us to easily adjust and tune various system parameters and observe the results

# Simulation Basics

- ❑ In general, a simulation model consists of three elements:
  - Input variables (according to some statistical distribution)
  - Simulator (mathematical/logical relationship between input and output)
  - Output variables (used to estimate the performance measures of the system)

# Simulation basics

- ❑ When simulating a system, you will often need to generate input variables with different distributions from a uniform random variable
- ❑ If you need to generate a random variable  $X$  with a distribution function  $F(x)$ , you need to do the following:
  1. Generate  $U \sim U(0,1)$  (Uniform random variable)
  2. Return  $X = F^{-1}(U)$  where  $F^{-1}(U)$  is the inverse of the distribution function of the desired random variable.
- ❑ In this lab you need to generate an exponentially distributed random variables that will be used to determine when the next packet will arrive from the application or when an observation should be made.
- ❑ Recall that an exponential distribution is  $F(x) = 1 - e^{-\lambda x}$

# Simulator design

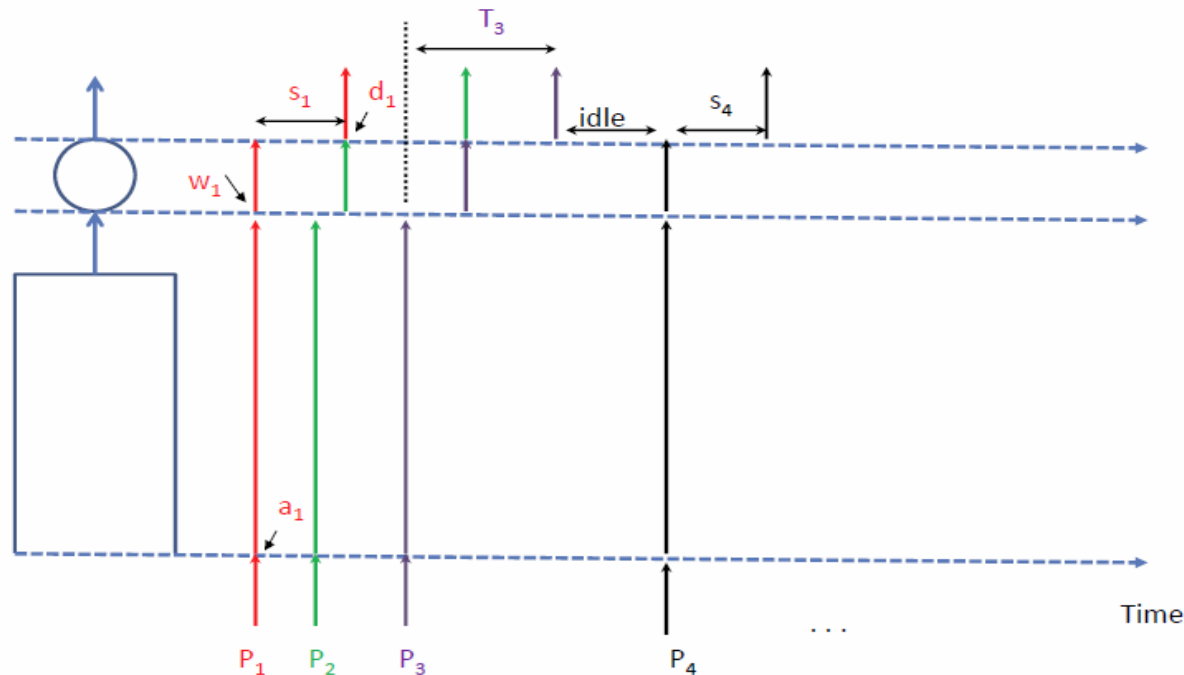
## ❑ Discrete event simulator (DES)

- one of the basic paradigms for simulator design
- used in problems where the system evolves in time, and the *state* of the system changes at discrete points in time when some *events* happen.
- we don't need to simulate every tick of the system, since by definition nothing happens between events

## ❑ DES is suitable for the simulation of queues

- **State:** e.g. number of packets in the system
- **Typical Events:**
  - Packet arrival
  - Packet departure

# The dynamics of an infinite single server queue



- For the  $n^{\text{th}}$  packet  $P_n$ ,  $a_n$  is its arrival time,  $s_n$  is its service time,  $d_n$  is its departure time and  $T_n$  is its sojourn time (i.e., the total time spent in the system (buffer and server)).

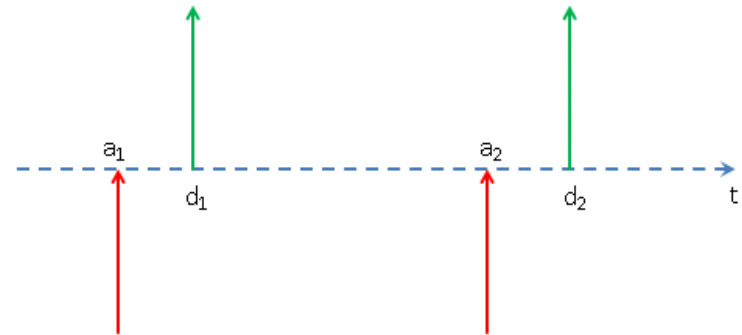


# Performance Measures

- ❑ We are interested in measuring the following parameters:
  - The time-average number of packets in the queue,  $E[N]$ ,
  - The proportion of time the server is idle (i.e., the queue is empty),  $P_{IDLE}$
  - In the case of a finite queue, the probability that a packet will be dropped due to the buffer being full when it arrives.
  - When should we record the state of the system?
  - The measurement and recording of the state should have **statistically representative** information on the performance that we are interested in.

# The need for another event

- Consider a D/D/1



- What do we see if we look at the system at only arrival and departure times  $(a_1, d_1, a_2, d_2, \dots)$ ?
- Need for an observation event: an event that happens at random times, *independent* of other events
- Here we use a Poisson observer
- The simulation will now contain *three* types of events: packet arrival, packet departure, and observer arrival.

# DES for a simple queue M/M/1

- ❑ Discrete event scheduler (DES) or in short Event Scheduler (ES):
  - The *ES* contains a set of time-ordered events  $E_i$ ,  $i=1, \dots, N$ , such that  $t(E_i) < t(E_{i+1})$  for all  $i$  along with the time of their occurrences.

|           |              |
|-----------|--------------|
| $E_i$     | $t(E_i)$     |
| $E_{i+1}$ | $t(E_{i+1})$ |
| $E_{i+2}$ | $t(E_{i+2})$ |
| $E_{i+3}$ | $t(E_{i+3})$ |

- Note that in this first lab, ES for M/M/1 queue can be created completely ahead of time. This is not always possible.

# DES Simulation Initialization

- ❑ In order to create a DES for a simple M/M/1 queue with an infinite buffer, you should do the following:
  - Choose a duration  $T$  for your simulation (see later how to choose  $T$ ).
  - Generate a set of random observation times according to a Poisson distribution (i.e., exponential inter-arrival times) with parameter  $\alpha$  and record the observer event times in the ES.

Discrete Event Simulator (DES)

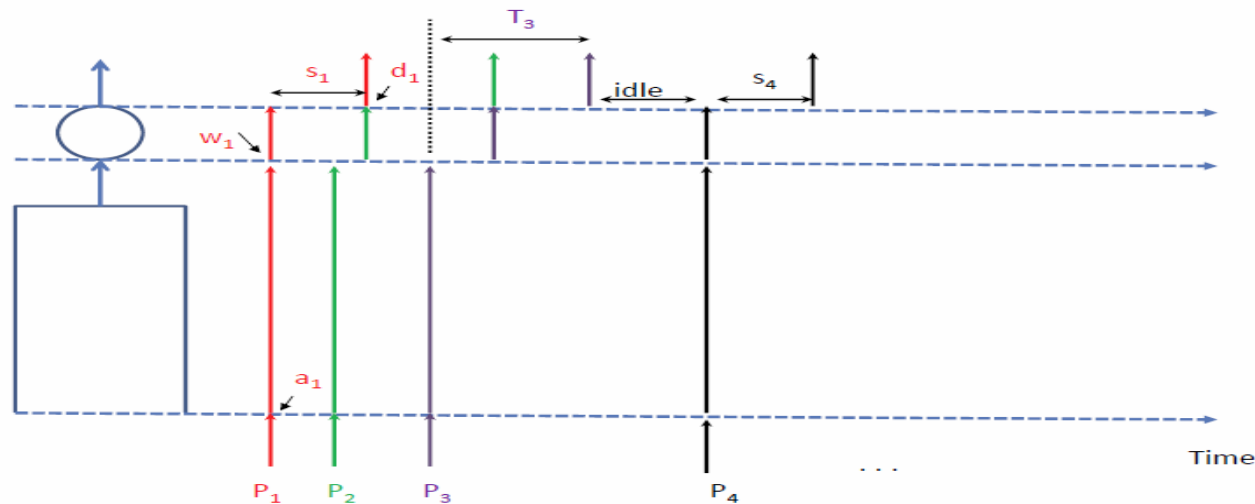
| Event    | Time   |
|----------|--------|
| Observer | 0.0104 |
| Observer | 0.016  |
| Observer | 0.0442 |
| Observer | 0.0816 |
| Observer | 0.0946 |
| Observer | 0.1069 |
| Observer | 0.1211 |
| Observer | 0.1245 |

# DES Simulation Initialization

- Generate a set of packet arrivals (according to a Poisson distribution with parameter  $\lambda$ ) and their corresponding length (according to an exponential distribution with parameter  $1/L$ ), and the corresponding service time.
- Calculate their departure times based on the state of the system (the departure time of a packet of length  $L_p$  depends on how much it has to wait and on its service time  $L_p/C$  where  $C$  is the link rate).

| Event   | Time   | Length (bits) | Service Time | Departure |
|---------|--------|---------------|--------------|-----------|
| Arrival | 0.0135 | 1614          | 0.0016       | 0.015071  |
| Arrival | 0.0499 | 30981         | 0.031        | 0.08085   |
| Arrival | 0.0685 | 4487          | 0.0045       | 0.085337  |
| Arrival | 0.0815 | 6693          | 0.0067       | 0.088157  |
| Arrival | 0.0925 | 16592         | 0.0166       | 0.109135  |
| Arrival | 0.1325 | 56.81         | 6E-05        | 0.132512  |
| Arrival | 0.1537 | 21037         | 0.021        | 0.174745  |
| Arrival | 0.1597 | 10904         | 0.0109       | 0.185649  |

# DES Simulation Initialization



❑ If the server is empty, then

The departure time of a packet = its arrival time + ?

❑ If the server is busy , then

The departure time of a packet = ? + Service time

# DES Simulation Initialization

- ❑ Populate the Discrete Event Scheduler (DES) with the observation, arrival and departure times and then sort the DES according to the timing.
- ❑ You can use arrays or linked lists or any other data structure you like.
- ❑ If you use arrays, it should look like this:

Before  
Sorting

|           |             |
|-----------|-------------|
| Arrival   | 0.065633411 |
| Arrival   | 0.087665212 |
| Arrival   | 0.120001212 |
| Arrival   | 0.145761228 |
| Arrival   | 0.187517327 |
| Arrival   | 0.20225649  |
| Arrival   | 0.251937076 |
| Arrival   | 0.264276402 |
| Departure | 0.092050945 |
| Departure | 0.09386122  |
| Departure | 0.124291617 |
| Departure | 0.155661949 |
| Departure | 0.201689653 |
| Departure | 0.227061173 |
| Departure | 0.258413235 |
| Departure | 0.296843715 |
| Observer  | 0.006593814 |
| Observer  | 0.007847576 |
| Observer  | 0.019231011 |
| Observer  | 0.024443005 |
| Observer  | 0.034910443 |
| Observer  | 0.059539353 |
| Observer  | 0.074816364 |

After  
Sorting

|           |             |
|-----------|-------------|
| Observer  | 0.034910443 |
| Observer  | 0.059539353 |
| Arrival   | 0.065633411 |
| Observer  | 0.074816364 |
| Arrival   | 0.087665212 |
| Departure | 0.092050945 |
| Departure | 0.09386122  |
| Observer  | 0.095019652 |
| Observer  | 0.101452866 |
| Observer  | 0.108010377 |
| Observer  | 0.11618913  |
| Observer  | 0.117902974 |
| Arrival   | 0.120001212 |
| Departure | 0.124291617 |
| Observer  | 0.143199194 |
| Arrival   | 0.145761228 |
| Observer  | 0.149535313 |
| Departure | 0.155661949 |
| Observer  | 0.157701614 |
| Observer  | 0.166385588 |
| Observer  | 0.175330675 |
| Arrival   | 0.187517327 |
| Departure | 0.201689653 |

# DES Simulation Initialization

- Initialize the following variables to zero:  $N_a$  = number of packet arrivals,  $N_d$  = number of packets departures,  $N_o$  = number of observations and idle counter

$N_a$   
number of arrivals

Numbers  
2679

$N_d$   
number of departures

Numbers  
2679

$N_o$   
number of observations

Numbers  
2679

Idle counter

Numbers  
2679

Initialize all counters to Zero



# DES Simulation (processing events)

- Dequeue one event at a time from your event scheduler (i.e., read one event at a time from the ES, starting with the first in the list), and call the corresponding event *procedure*. Each event procedure (one per type of event) will consist of updating some variables.

|           |             |
|-----------|-------------|
| Observer  | 0.034910443 |
| Observer  | 0.039539353 |
| Arrival   | 0.065633411 |
| Observer  | 0.074816364 |
| Arrival   | 0.087665212 |
| Departure | 0.092050945 |
| Departure | 0.09386122  |
| Observer  | 0.095019652 |
| Observer  | 0.101452866 |
| Observer  | 0.108010377 |
| Observer  | 0.11618913  |
| Observer  | 0.117902974 |
| Arrival   | 0.120001212 |
| Departure | 0.124291617 |
| Observer  | 0.143199194 |
| Arrival   | 0.145761228 |
| Observer  | 0.149535313 |
| Departure | 0.155661949 |
| Observer  | 0.157701614 |
| Observer  | 0.166385588 |
| Observer  | 0.175330675 |
| Arrival   | 0.187517327 |
| Departure | 0.201689653 |

Dequeue the first event in the DES

# DES Simulation (processing events)

❑ If the event is Arrival

$N_a$   
number of arrivals

Numbers  
2679

$N_d$   
number of departures

Numbers  
2679

$N_o$   
number of observations

Numbers  
2679

Which counter should be  
updated?

# DES Simulation (processing events)

❑ If the event is Departure

$N_a$   
number of arrivals

Numbers  
2679

$N_d$   
number of departures

Numbers  
2679

$N_o$   
number of observations

Numbers  
2679

Which counter should be  
updated?

# DES Simulation (processing events)

❑ If the event is Observer

$N_a$   
number of arrivals

Numbers  
2679

$N_d$   
number of departures

Numbers  
2679

$N_o$   
number of observations

Numbers  
2679

Which counter should be  
updated?

# DES Simulation (processing events)

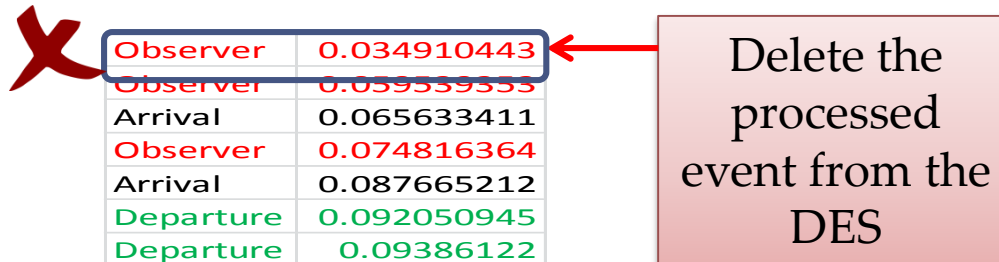
- ❑ If the event is Observer
- ❑ Record the values of your performance metrics.
  - The number of packets in the queue. **How can we calculate it?**
  - If the queue is empty (**how can we know that?**), increment the idle counter

Idle counter



# DES Simulation (processing events)

- ❑ After processing the event, delete the event from the DES



Delete the processed event from the DES

|           |             |
|-----------|-------------|
| Observer  | 0.034910443 |
| Observer  | 0.059539353 |
| Arrival   | 0.065633411 |
| Observer  | 0.074816364 |
| Arrival   | 0.087665212 |
| Departure | 0.092050945 |
| Departure | 0.09386122  |
| Observer  | 0.095019652 |
| Observer  | 0.101452866 |
| Observer  | 0.108010377 |
| Observer  | 0.11618913  |
| Observer  | 0.117902974 |
| Arrival   | 0.120001212 |
| Departure | 0.124291617 |
| Observer  | 0.143199194 |
| Arrival   | 0.145761228 |
| Observer  | 0.149535313 |
| Departure | 0.155661949 |
| Observer  | 0.157701614 |
| Observer  | 0.166385588 |
| Observer  | 0.175330675 |
| Arrival   | 0.187517327 |
| Departure | 0.201689653 |

- ❑ Process the next event and repeat the same procedures as in the previous slides until you finish all the events in the DES

# DES Simulation (calculating parameters)

- After the simulation is finished, calculate...
  - The time-average number of packets in the queue,  $E[N]$ .  
How?
  - The proportion of time the server is idle (i.e., the system is empty),  $P_{IDLE}$ . How?

# DES Simulation

## Initialization for M/M/1/N

- ❑ Note that an ES for M/M/1/N queue **cannot** be created completely ahead of time.
- ❑ In order to create a DES for a simple M/M/1/N queue with a finite buffer, you should do the following:
  - Choose a duration T for your simulation (check for consistent results).
  - Generate a set of random observation times according to a Poisson distribution (i.e., exponential inter-arrival times) with parameter  $\alpha$  and record the observer event times in the ES.

Discrete Event Simulator (DES)

| Event    | Time   |
|----------|--------|
| Observer | 0.0104 |
| Observer | 0.016  |
| Observer | 0.0442 |
| Observer | 0.0816 |
| Observer | 0.0946 |
| Observer | 0.1069 |
| Observer | 0.1211 |
| Observer | 0.1245 |



# DES Simulation Initialization

- Generate a set of packet **arrivals** (according to a **Poisson distribution** with parameter  $\lambda$ )
- Note that you **cannot** generate the departure time based on the arrival time only.
- You will have to generate the departure time *on the fly* during the simulation

| Event   | Time     |
|---------|----------|
| Arrival | 0.065633 |
| Arrival | 0.087665 |
| Arrival | 0.120001 |
| Arrival | 0.145761 |
| Arrival | 0.187517 |
| Arrival | 0.202256 |
| Arrival | 0.251937 |
| Arrival | 0.264276 |

# DES Simulation Initialization

- ❑ Populate the Discrete Event Scheduler (DES) with the observation and arrival times and then sort the DES according to the timing.
- ❑ It should look like that

Before  
Sorting

| Event    | Time     |
|----------|----------|
| Arrival  | 0.065633 |
| Arrival  | 0.087665 |
| Arrival  | 0.120001 |
| Arrival  | 0.145761 |
| Arrival  | 0.187517 |
| Arrival  | 0.202256 |
| Arrival  | 0.251937 |
| Arrival  | 0.264276 |
| Observer | 0.006594 |
| Observer | 0.007848 |
| Observer | 0.019231 |
| Observer | 0.024443 |
| Observer | 0.03491  |
| Observer | 0.059539 |
| Observer | 0.074816 |
| Observer | 0.09502  |

After  
Sorting

|          |          |
|----------|----------|
| Observer | 0.03491  |
| Observer | 0.059539 |
| Arrival  | 0.065633 |
| Observer | 0.074816 |
| Arrival  | 0.087665 |
| Observer | 0.09502  |
| Observer | 0.101453 |
| Observer | 0.10801  |
| Observer | 0.116189 |
| Observer | 0.117903 |
| Arrival  | 0.120001 |
| Observer | 0.143199 |
| Arrival  | 0.145761 |
| Observer | 0.149535 |
| Observer | 0.157702 |

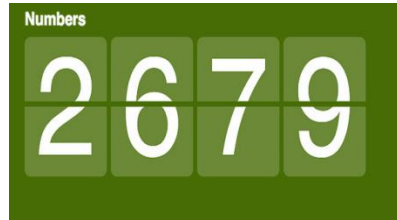
# DES Simulation Initialization

❑ Initialize the following counters

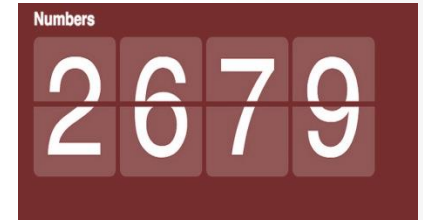
$N_a$   
number of arrivals



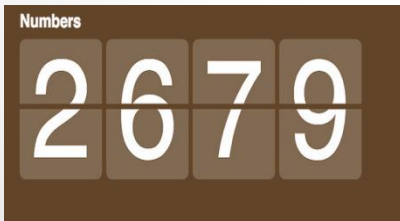
$N_d$   
number of departures



$N_o$   
number of observations



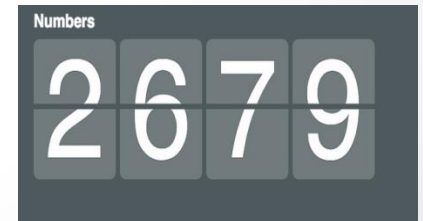
Idle counter



Number of dropped  
packets



Number of generated  
packets



Initialize all counters to Zero

# DES Simulation (processing events)

- Dequeue one event at a time from your event scheduler (i.e., read one event at a time from the ES, starting with the first in the list), and call the corresponding *event procedure*. Each event procedure (one per type of event) will consist of updating some of variables.

|  |          |          |
|--|----------|----------|
|  | Observer | 0.03491  |
|  | Observer | 0.059539 |
|  | Arrival  | 0.065633 |
|  | Observer | 0.074816 |
|  | Arrival  | 0.087665 |
|  | Observer | 0.09502  |
|  | Observer | 0.101453 |
|  | Observer | 0.10801  |
|  | Observer | 0.116189 |
|  | Observer | 0.117903 |
|  | Arrival  | 0.120001 |
|  | Observer | 0.143199 |
|  | Arrival  | 0.145761 |
|  | Observer | 0.149535 |
|  | Observer | 0.157702 |

Dequeue the first event in the DES

# DES Simulation

## (processing events (Arrival))

- ❑ If the event is Arrival
- ❑ Check how many packets are in the queue:
  - If the buffer is full, then
    - ❑ Drop the packet and increment the corresponding counter(s). (**Which one(s)?**)

$N_a$   
number of arrivals

Numbers  
2 6 7 9

$N_d$   
number of departures

Numbers  
2 6 7 9

$N_o$   
number of observations

Numbers  
2 6 7 9

Idle counter

Numbers  
2 6 7 9

Number of dropped packets

Numbers  
2 6 7 9

Number of generated packets

Numbers  
2 6 7 9

Which counter(s) should be updated?

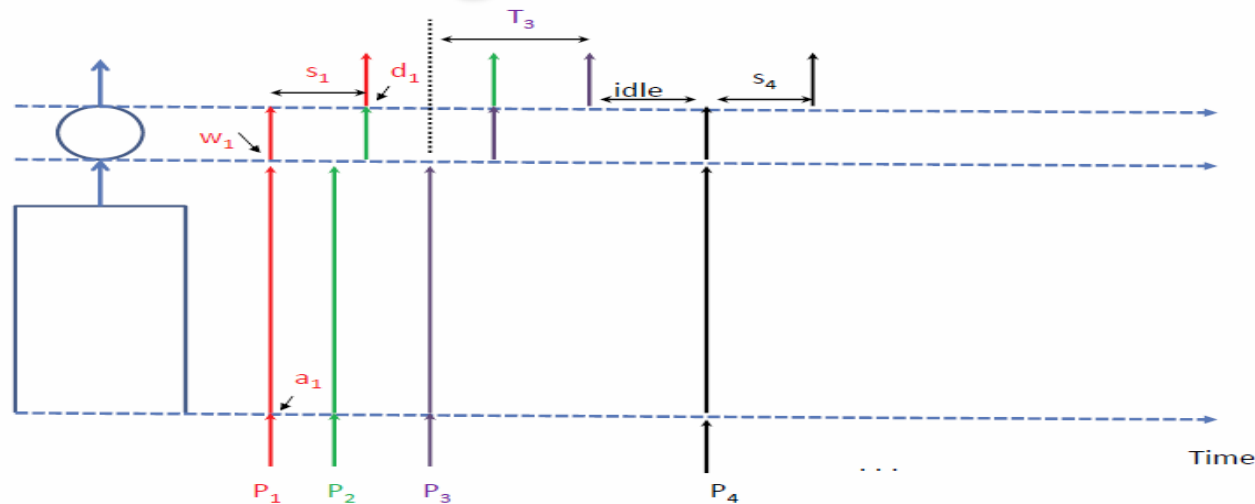
# DES Simulation

## (processing events (Arrival))

- Otherwise (i.e., the queue is either empty or partially filled)
  - ❑ Generate a length for the arriving packet (according to an exponential distribution with parameter  $1/L$ ), and the corresponding service time.
  - ❑ Calculate the departure time (of the packet which just arrived in the system) based on the state of the system (the departure time of a packet of length  $L_p$  depends on how much it has to wait and on its service time  $L_p/C$  where  $C$  is the link rate).

# DES Simulation

## (processing events (**Arrival**))



- ❑ If the server is empty, then

The departure time of a packet = its arrival time + ?

- ❑ If the server is **busy** , then

The departure time of a packet = ? + Service time

- Insert the departure time (of the packet which just arrived in the system) that you calculated in the DES and sort.

# DES Simulation

## (processing events (Arrival))

- ❑ Then, you should increment the corresponding counter(s). (**Which one(s)?**)

$N_a$   
number of arrivals

Numbers

2 6 7 9

$N_d$   
number of departures

Numbers

2 6 7 9

$N_o$   
number of observations

Numbers

2 6 7 9

Idle counter

Numbers

2 6 7 9

Number of dropped packets

Numbers

2 6 7 9

Number of generated packets

Numbers

2 6 7 9

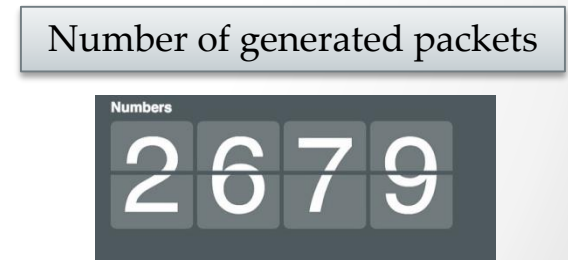
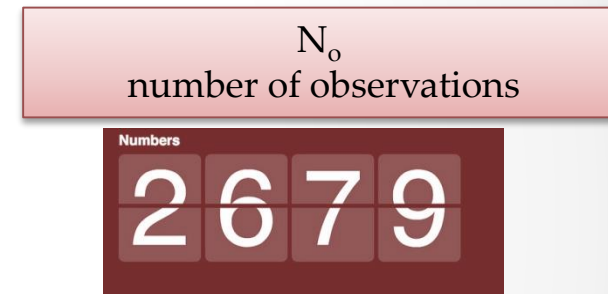
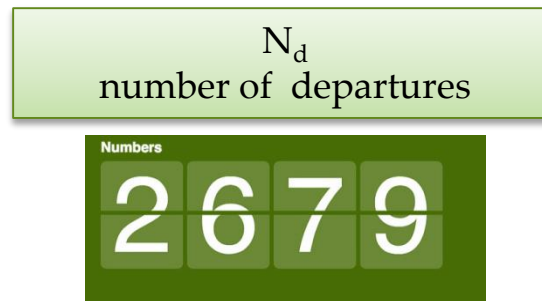
Which counter(s) should be updated?



# DES Simulation

## (processing events (Departure))

❑ If the event is Departure



Which counter(s) should be updated?

# DES Simulation

## (processing events (Observer))

❑ If the event is Observer

$N_a$   
number of arrivals

Numbers  
2 6 7 9

$N_d$   
number of departures

Numbers  
2 6 7 9

$N_o$   
number of observations

Numbers  
2 6 7 9

Idle counter

Numbers  
2 6 7 9

Number of dropped packets

Numbers  
2 6 7 9

Number of generated packets

Numbers  
2 6 7 9

Which counter(s) should be updated?

# DES Simulation

## (processing events (Observer))

### ❑ If the event is Observer

❑ Record the values of your performance metrics.

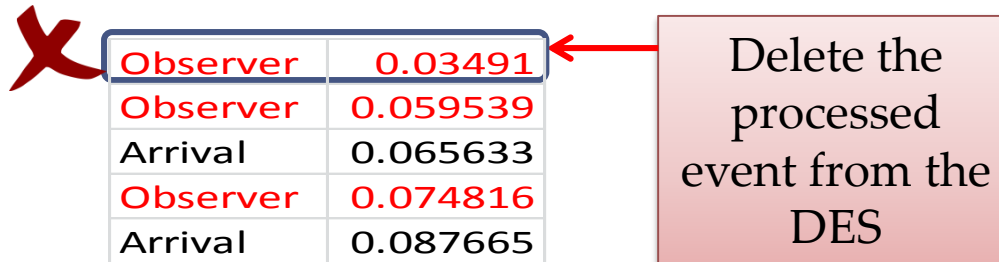
- The number of packets in the queue. How can we calculate it?
- If the queue is empty (how can we know that?), increment the idle counter

Idle counter



# DES Simulation (processing events)

- ❑ After processing the event, delete the event from the DES



Delete the processed event from the DES

|           |          |
|-----------|----------|
| Observer  | 0.03491  |
| Observer  | 0.059539 |
| Arrival   | 0.065633 |
| Observer  | 0.074816 |
| Arrival   | 0.087665 |
| Departure | 0.092051 |
| Observer  | 0.09502  |
| Observer  | 0.101453 |
| Observer  | 0.10801  |
| Observer  | 0.116189 |
| Observer  | 0.117903 |
| Arrival   | 0.120001 |
| Observer  | 0.143199 |
| Arrival   | 0.145761 |
| Observer  | 0.149535 |
| Observer  | 0.157702 |

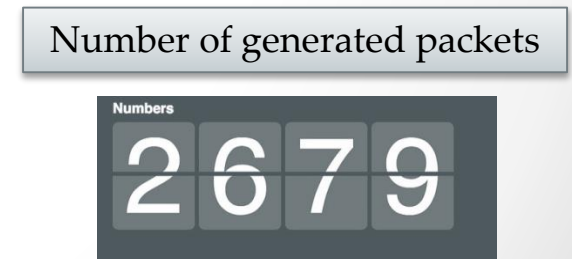
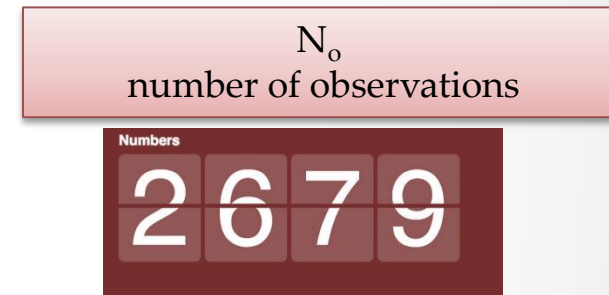
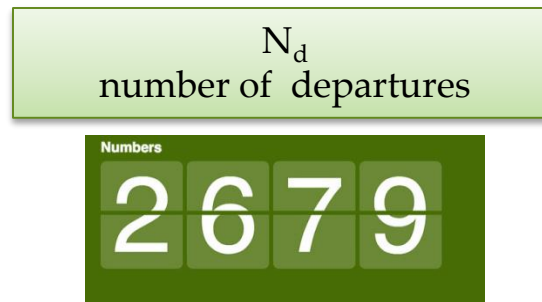
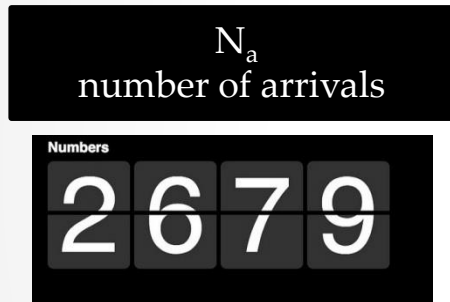
- ❑ Process the next event and repeat the same procedures as in the previous slides until you finish all the events in the DES

# DES Simulation (calculating parameters)

- After the simulation is finished, calculate...
  - The time-average number of packets in the queue,  $E[N]$ . How?
  - The proportion of time the server is idle (i.e., the system is empty),  $P_{IDLE}$ . How?
  - The probability that a packet will be dropped,  $P_{LOSS}$ . How?

# DES Simulation (calculating parameters)

Use the final counter values to compute the various values



# DES Simulation (simulation time)

- Run each experiment in this lab for  $T$  seconds and record the result. Then run the experiment again for  $2T$  seconds and see if the expected values of the output variables are similar to the output from the previous run, i.e., the difference is less than 5%. If the results are not similar, keep increasing  $T$  until we get stable results.
- Note that this should be done for every experiment.
- If you did not indicate this time checking procedure in your report, you will lose marks.

# Notes

- **Programming languages: You can only use C or C++**
- **Your code must compile run on the ecelix servers.**
- **You must submit your code and report on Learn.**