Lecture 17: Queueing Theory and Discrete-Event Simulation

Note: The QT slides are due to Harry Perros who has good taste in ideas but bad taste in slide background colors….
Queue ADT (Review from CS 112!)

The **Queue ADT** is a simple variant of a stack which makes a simple change which in fact changes everything: instead of moving items in and out of the same “end” of the list, as in a stack:

![Diagram of Push and Pop operations for a stack](image1)

Instead you use different ends of the list:

![Diagram of Enqueue and Dequeue operations for a queue](image2)
Queue ADT (Review from CS 112!)

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office!), I’ll only give a brief example:

Enqueue  →  Queue  →  Dequeue
Queue ADT

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I’ll only give a brief example:

enqueue(5);
Queue ADT

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I’ll only give a brief example:

enqueue(5);
enqueue(7);

Enqueue 7 5 Dequeue
Queue ADT

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I’ll only give a brief example:

enqueue(5);
enqueue(7);
enqueue(2);
Queue ADT

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I’ll only give a brief example:

enqueue(5);
enqueue(7);
enqueue(2);
int k = dequeue();

enqueue

2 7
dequeue

k = 5
Queue ADT

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I’ll only give a brief example:

enqueue(5);
enqueue(7);
enqueue(2);
int k = dequeue();
enqueue(8);

Enqueue 8 2 7 Dequeue

k = 5
Queue ADT

This means that instead of reversing the order of the items, as with a stack, they remain in the same order; since you have stood in lines many times at Starbucks (or outside my office), I’ll only give a brief example:

enqueue(5);
enqueue(7);
enqueue(2);
int k = dequeue();
enqueue(8);
enqueue( dequeue() )

Enqueue 7 8 2 Dequeue

k = 5
Queue ADT

Queues occur all the time, in real life:

And in computer systems (CPUs and Networks):

In fact, anywhere where one service is desired by many, and must be fairly distributed... there is a whole branch of math called “queueing theory” which you will learn about in CS 237 and CS 350.....
• There are also queues that we cannot see (unless we use a software/hardware system), such as:
  – *Streaming a video*: Video is delivered to the computer in the form of packets, which go through a number of routers. At each router they have to waiting to be transmitted out
  – *Web services*: A request issued by a user has to be executed by various software components. At each component there is a queue of such requests.
  – *On hold at a call center*
Notation - single queueing systems

- Queue → Single Server
- Queue → Multiple Servers
- Multi-Queue → Single Server
- Multi-Queue → Multiple Servers
Notation - Networks of queues

Tandem queues

Arbitrary topology of queues
Parameters of interest

You define a queueing system by specifying the following:

- **Service discipline**: How is the queue organized, i.e., FIFO, Priority Queue, etc. (typically FIFO queue).
- **How many servers?** (typically 1)
- **How many queues?** (typically 1)
- **Distribution of arrivals**: Poisson (with exponential inter-arrival times) or general (any distribution) with some mean and standard deviation.
- **Distribution of service times** (how long does each task need the server): Typically Exponential with some mean.
Measures of interest

You measure the performance of a queueing system using metrics such as the following:

- **Wait time**: How long does a task wait in the queue?
- **Mean wait time (per task)**.
- **Mean queue length** (\(=\) average number of tasks waiting).
- **Server utilization**: What percentage of time is server busy?
- **System throughput**: How many tasks complete per unit time?

One can also characterize these in terms of distribution, e.g., distribution of the queue length.
The single server queue

Calling population: finite or infinite

Queue: Finite or infinite capacity

Service discipline: FIFO
Queue formation

- A queue is formed when customers arrive faster than they can get served.

- Examples:
  - Service time = 10 minutes, a customer arrives every 15 minutes ---> No queue will ever be formed!
  - Service time = 15 minutes, a customer arrives every 10 minutes ---> Queue will grow for ever (bad for business!)
• Service times and inter-arrival times are rarely constant.
• From real data we can construct a histogram of the service time and the inter-arrival time.
• If real data is not available, then we assume a theoretical distribution.
• A commonly used theoretical distribution in queueing theory is the exponential distribution.
The M/M/1 queue

- M implies the exponential distribution (Markovian)
- The M/M/1 notation implies:
  - a single server queue
  - exponentially distributed inter-arrival times
  - exponentially distributed service times.
  - Infinite population of potential customers
  - FIFO service discipline
Stability condition

- A queue is stable, when it does not grow to become infinite over time.
- The single-server queue is stable if on the average, the service time is less than the inter-arrival time, i.e.
  
  $\text{mean service time} < \text{mean inter-arrival time}$
Behavior of a stable queue
Mean service time < mean inter-arrival time

When the queue is stable, we will observe busy and idle periods continuously alternating
Behavior of an unstable queue
Mean service time > mean inter-arrival time

Queue continuously increases..
This is the case when a car accident occurs on the highway
Arrival and service rates: definitions

- **Arrival rate is the mean number of arrivals per unit time** = \( \frac{1}{(\text{mean inter-arrival time})} \)
  - If the mean inter-arrival = 5 minutes, then the arrival rate is \( \frac{1}{5} \) per minute, i.e. 0.2 per minute, or 12 per hour.

- **Service rate is the mean number of customers served per unit time** = \( \frac{1}{(\text{mean service time})} \)
  - If the mean service time = 10 minutes, then the service rate is \( \frac{1}{10} \) per minute, i.e. 0.1 per minute, or 6 per hour.
Throughput

- This is average number of completed jobs per unit.
- Example:
  - The throughput of a production system is the average number of finished products per unit time.
- Often, we use the *maximum throughput* as a measure of performance of a system.
Throughput of a single server queue

• This is the average number of jobs that depart from the queue per unit time (after they have been serviced)

• Example: The mean service time = 10 mins.
  – What is the maximum throughput (per hour)?
  – What is the throughput (per hour) if the mean inter-arrival time is:
    • 5 minutes?
    • 20 minutes?
Throughput vs the mean inter-arrival time.
Service rate = 6

Throughput

\[ \text{Throughput} \]

Arrival rate -->

6

Max. throughput

Stable queue: whatever comes in goes out!

Unstable queue: More comes in than goes out!
Server Utilization =
Percent of time server is busy =
(arrival rate) x (mean service time)

- Example:
  - Mean inter-arrival = 5 mins, or arrival rate is $1/5 = 0.2$ per min. Mean service time is 2 minutes
  - Server Utilization = Percent of time the server is busy:
    $0.2 \times 2 = 0.4$ or 40% of the time.
  - Percent of time server is idle?
  - Percent of time no one is in the system (either waiting or being served)?
Little’s Law

Denote the mean number of customers in the system as \( L \) and the mean waiting time in the system as \( W \). Then:

\[ \lambda W = L \]