# M/M/1 queue example

Thomas R. Henderson, Sumit Roy, Collin Brady
University of Washington Seattle, WA

June 16, 2019

## I. Background

When multiple packets arrive at a server, packets wait in queue according to some discipline - typically first-come, first served (FCFS) - to process the arrivals. Queue operations are denoted by Kendall's notation that captures three factors: a) the input or arrival process, b) the server or departure process and c) the # of servers. The most canonical queue studied is the M/M/1/C queue, where: ”M” indicates a Markovian arrival process, i.e. a Poisson process whereby packet inter-arrival times are exponentially distributed with mean $\frac{1}{\lambda}$ (or equiv. the arrival *rate* in packets/sec is $\lambda$). The second ”M” indicates packet service time (the duration from beginning to end of service), that is also distributed exponentially with mean service time $\mu$; the service time is indep. of the inter-arrival process. The ”1” indicates that there is only one server and ”C” denotes the buffer capacity (i.e. a maximum of $C$ packets may wait in queue and any further arrivals are denied entry/lost). An M/M/1/C queue abstraction is shown in Figure 1.

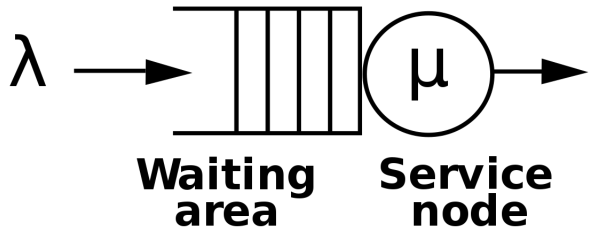

Fig. 1. Example of M/M/1 queue

Clearly, the queue size (# customers waiting for service) varies over time, as a function of the arrival and service process. Time intervals for which the queue is empty implies that the server is (temporarily) idle, whereas the periods when the queue is full (i.e. # in queue equals the capacity $C$) lead to packet drops for any subsequent arrivals. Both of these can be undesirable in a communications system (one would like to minimize the probability of both occurrences) and are thus the object of queue management policies. As we will see, attempting to minimize server idle fraction negatively impacts the other - thereby setting off a trade-off. The steady-state queue distribution for M/M/1/C queue is obtained by consideration the analysis of M/M/1/$\infty$ (i.e. infinite size queue, and hence nothing is dropped) first and then simply re-normalizing the results for finite ($C$) size queue. From the queue distribution, all statistics of interest - such as average packet latency, # of packets in queue, blocking probability as function of finite ($C$) queue size etc. can be obtained.

M/M/1/$\infty$ Queue

Denote by $\rho = \frac{\lambda}{\mu}$, the ratio of the (packet) arrival to server rate. It can be intuitively asserted that such a queue is stable only if $\rho < 1$, i.e. if the arrival rate is less than the service rate, as will be backed up by results. Let $\pi_k = P\{X = n\}$ denote the p.m.f for the steady-state queue size distribution $X$ (i.e. if you send a large # of packets through the system, gather data on resulting queue size in steady-state and compute histogram), then the following well-known results can be found in any basic Queueing theory text:

i. $P\{X = n\} = \rho^n (1 - \rho),\ n = 0, 1, 2.....$ Hence the fraction-of-time that the server is idle $\pi_0 = 1 - \rho$ and corr. busy equals $\rho$ (hence the notion of 'utilization'). The resulting expected value of queue-size $E[X] = \frac{\rho}{1-\rho}$.

ii. The average latency ($W$) in the system experienced by any packet (this is the sum of the wait-time plus the service time) is given by $E[W] = \frac{1}{\mu-\lambda} =$. Clearly as $\lambda \to \mu$, $E[W] \to \infty$ implying the potential for unbounded queue sizes (and hence instability).

The above result can be deduced from a much broader law that is applicable to a very wide variety of queues (beyond just M/M/1) known as Little's Theorem, that relates to averages for the queuing systems:

Little's Theorem

Let $N$ denote the *average* number of customers (packets) in the system (in our notation, $N = E[X]+1$, i.e. the $X$ packets in queue plus the one being served), and let $T$ be the *average* time-in-system per packet ('time-in-system' equals the sum of wait time (W) and the service tiem), then by Little's Theorem $N = \lambda T$. Equivalently, if we just focus on the number of queued packets, then the foll. also holds: $E[X] = E[W]$.

This has a very intuitive interpretation that can be verified by simulation: the (long-term) average queue size equals the product of the arrival rate $\lambda$ and the mean wait time ($E[W]$) for a packet in queue.

### A. *ns-3 implementation*

An ns-3 program implementing the M/M/1 queue as a stand-alone queue process (i.e. not integrated with a network device, just existing in isolation with a packet arrival and departure process) can be found in `contrib/training-2019/examples/mm1-queue.cc`.

```
$ ./waf --run 'mm1-queue --PrintHelp'
Program Options:
    --lambda:          arrival rate (packets/sec) [1]
    --mu:              departure rate (packets/sec) [2]
    --initialPackets:  initial packets in the queue [0]
    --numPackets:      number of packets to enqueue [0]
    --queueLimit:      size of queue (number of packets) [100]
    --quiet:           whether to suppress all output [false]
```

Note that lambda and mu can be set by command-line argument, and the number of packets (arrivals) to send through the system must be set to a non-zero value for the program to do anything useful.

Let's run it once by sending 10 packets:

```
$ ./waf --run 'mm1-queue --numPackets=10'
Program Options:
    --lambda:          arrival rate (packets/sec) [1]
    --mu:              departure rate (packets/sec) [2]
    --initialPackets:  initial packets in the queue [0]
    --numPackets:      number of packets to enqueue [0]
    --queueLimit:      size of queue (number of packets) [100]
    --quiet:           whether to suppress all output [false]
```

This produces a data file `mm1queue.dat`. This file has the following content:

```
0.352958 + 1
0.712375 + 2
0.722229 - 1
1.18865 + 2
1.29859 + 3
1.34304 - 2
1.72518 - 1
1.97162 + 2
2.64096 - 1
3.06024 - 0
3.85077 + 1
3.92556 - 0
5.84833 + 1
```

```
5.85479 - 0
7.42087 + 1
7.42253 - 0
9.34972 + 1
9.37861 - 0
11.3829 + 1
11.6735 - 0
```

Every '+' operation is an enqueue operation. Every '-' operation is a dequeue operation. The first column is the timestamp in seconds. The last column is the queue length at the end of the operation. The simulation ends when the last packet to send has been enqueued and dequeued (or dropped).

The time that the queue is busy can be deduced from this file: the total time of the simulation (11.6735 seconds) minus the sum of any time intervals for which a dequeue leads to a zero-length queue (e.g. from time 7.42253 to time 9.34972). Time zero until the first enqueue must also be considered. To answer the below questions, you will want to write some kind of script to parse this file, or else modify the C++ program to calculate idle times and delays.

Try rerunning the program with a different random number seed; do the results change much?

```
$ ./waf --run 'mm1-queue --numPackets=10 --RngRun=2'
```

Drops are traced as well. For instance, let's set the queue length to an (unreasonably) low value of 1 packet, and repeat.

```
$ ./waf --run 'mm1-queue --numPackets=10 --queueLimit=1'
```

```
0.352958 + 1
0.712375 d 1
0.722229 - 0
1.18865 + 1
1.29859 d 1
1.34304 - 0
1.97162 + 1
2.64096 - 0
3.85077 + 1
3.92556 - 0
5.84833 + 1
5.85479 - 0
7.42087 + 1
7.42253 - 0
9.34972 + 1
9.37861 - 0
11.3829 + 1
11.6735 - 0
```

## B. *Questions*

1) For an M/M/1 queuing system, simulate with $\mu = 10$ and $\lambda =$ 1,3,5,7, and 9, and plot the queue idle proportion, mean queue length, and average packet delay, all as a function of the ratio $\lambda/\mu$. For each trial, send 100,000 packets through the system. Set the queue limit to a large value to avoid drops.

2) Experiment with the buffer overflow outcome of a 100 packet M/M/1 queue by setting $\mu = 10$ and $\lambda =$ 9.5, 9.7, and 9.9. Starting from an empty queue, run a number of trials to measure how long it takes for the queue to overflow. Provide your estimate of the average time until overflow, sample standard deviation, and number of trials conducted for each $lambda$.

Two ways to accomplish the above are 1) to write a script to repeatedly run with the desired configuration but with different RngRun number values, and for each run, find the time of the first drop, or 2) run the program for a very long number of packets and use the entry into an idle queue as the start time to measure until the next drop.