Xi'an Jiaotong-Liverpool University
西交利物浦大学

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE413 DATA COMMUNICATION AND COMMUNICATIONS NETWORKS

COURSEWORK 1

Student Name Zhiyu.Xu
Student ID 1926905

November 2019

Contents

1st Result and code..2
2nd Explanation..10

1st Result and code

The results are recorded in "mm1.out" as follows:

```
5.0000E+00  1.0511E-02
1.0000E+01  1.1095E-02
1.5000E+01  1.1725E-02
2.0000E+01  1.2452E-02
2.5000E+01  1.3263E-02
3.0000E+01  1.4141E-02
3.5000E+01  1.5148E-02
4.0000E+01  1.6342E-02
4.5000E+01  1.7738E-02
5.0000E+01  1.9411E-02
5.5000E+01  2.1465E-02
6.0000E+01  2.4243E-02
6.5000E+01  2.7676E-02
7.0000E+01  3.2535E-02
7.5000E+01  3.9691E-02
8.0000E+01  4.8997E-02
8.5000E+01  6.4200E-02
9.0000E+01  9.7699E-02
9.5000E+01  2.0845E-01
```

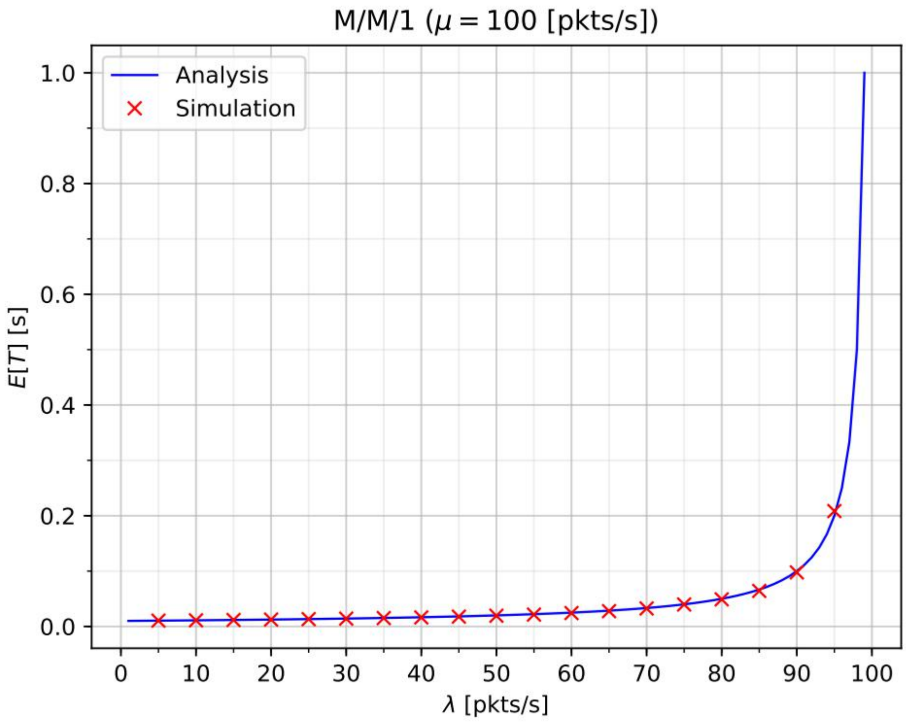

Figure 1 M/M/1

The results are recorded in “mm2.out” as follows:

```
5.0000E+00  2.0092E-02
1.0000E+01  2.0258E-02
1.5000E+01  2.0511E-02
2.0000E+01  2.0871E-02
2.5000E+01  2.1329E-02
3.0000E+01  2.1893E-02
3.5000E+01  2.2615E-02
4.0000E+01  2.3504E-02
4.5000E+01  2.4626E-02
5.0000E+01  2.6058E-02
5.5000E+01  2.7842E-02
6.0000E+01  3.0358E-02
6.5000E+01  3.3583E-02
7.0000E+01  3.8385E-02
7.5000E+01  4.5473E-02
8.0000E+01  5.4823E-02
8.5000E+01  7.0142E-02
9.0000E+01  1.0392E-01
9.5000E+01  2.1546E-01
```

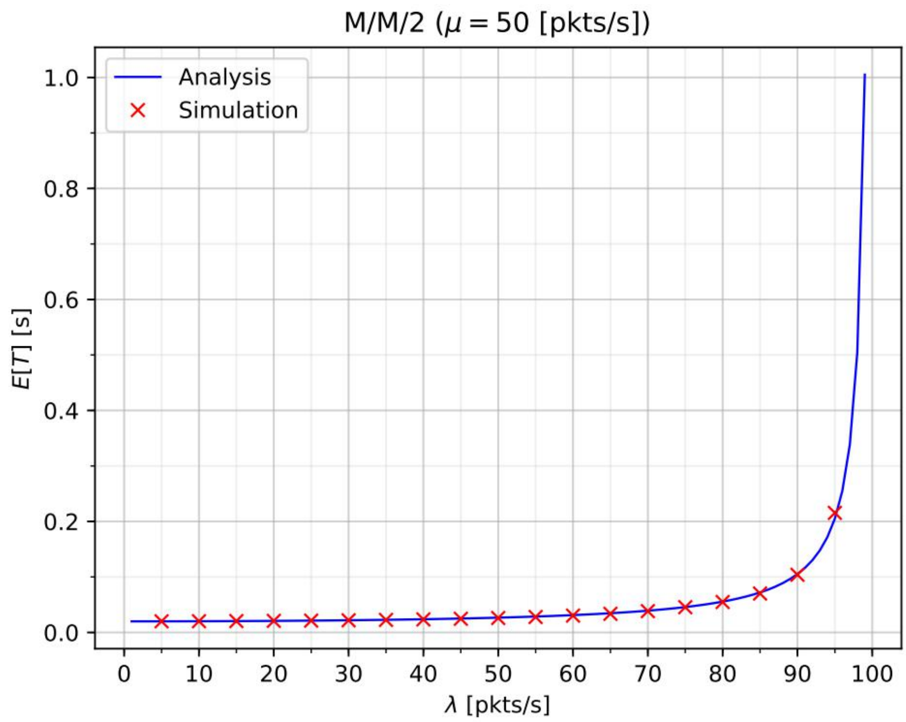

Figure 2 M/M/2

The results are recorded in “mm5.out” as follows:

```
5.0000E+00  5.0127E-02
1.0000E+01  5.0127E-02
```

```
1.5000E+01  5.0132E-02
2.0000E+01  5.0149E-02
2.5000E+01  5.0197E-02
3.0000E+01  5.0312E-02
3.5000E+01  5.0520E-02
4.0000E+01  5.0868E-02
4.5000E+01  5.1402E-02
5.0000E+01  5.2227E-02
5.5000E+01  5.3460E-02
6.0000E+01  5.5348E-02
6.5000E+01  5.8098E-02
7.0000E+01  6.2226E-02
7.5000E+01  6.8487E-02
8.0000E+01  7.7166E-02
8.5000E+01  9.2385E-02
9.0000E+01  1.2569E-01
9.5000E+01  2.3718E-01
```

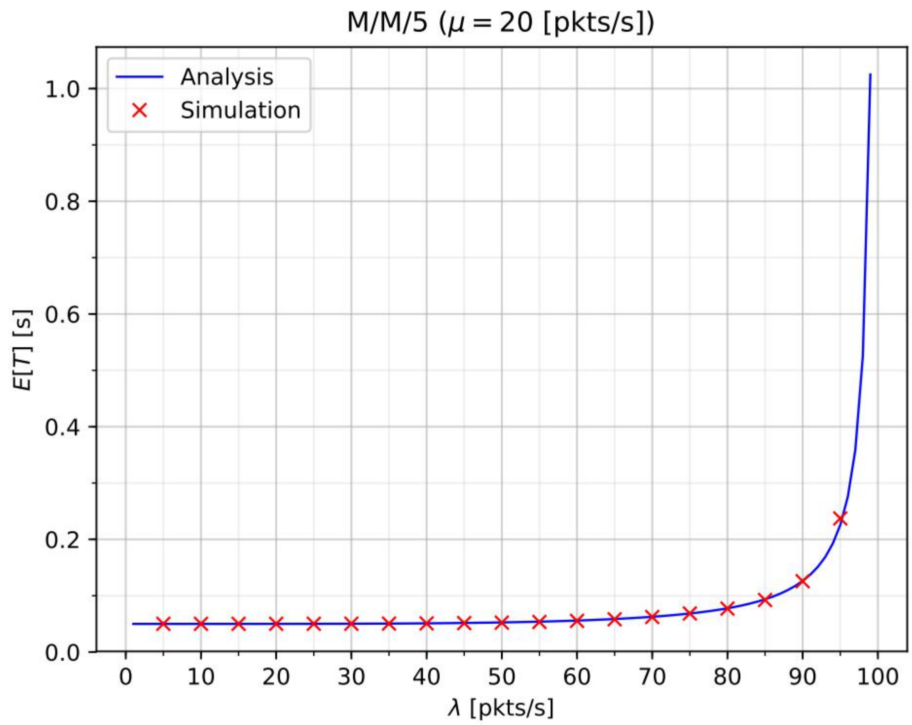

Figure 3 M/M/5

```python
import argparse
import numpy as np
import random
import simpy
import sys

def source(env, mean_ia_time, mean_srv_time, server, delays, number, trace):
```

```python
    """Generates packets with exponential interarrival time."""
    for i in range(number):
        ia_time = random.expovariate(1.0 / mean_ia_time)
        srv_time = random.expovariate(1.0 / mean_srv_time)
        pkt = packet(env, 'Packet-%d' % i, server, srv_time, delays, trace)
        env.process(pkt)
        yield env.timeout(ia_time)

def packet(env, name, server, service_time, delays, trace):
    """Requests a server, is served for a given service_time, and leaves the server."""
    arrv_time = env.now
    if trace:
        print("t={0:.4E}s: {1:s} arrived".format(arrv_time, name))

    with server.request() as request:
        yield request
        yield env.timeout(service_time)
        delay = env.now - arrv_time
        delays.append(delay)
        if trace:
            print("t={0:.4E}s: {1:s} served for {2:.4E}s".format(env.now, name, service_time))
            print("t={0:.4E}s: {1:s} delayed for {2:.4E}s".format(env.now, name, delay))

def run_simulation(num_servers, mean_ia_time, mean_srv_time, num_packets=10000, random_seed=1234, trace=True):
    """Runs a simulation and returns statistics."""
    if trace:
        print('M/M/1 queue\n')
    random.seed(random_seed)
    env = simpy.Environment()

    # start processes and run
    server = simpy.Resource(env, capacity=num_servers)
    delays = []
    env.process(source(env, mean_ia_time,
                       mean_srv_time, server, delays, number=num_packets, trace=trace))
    env.run()

    # return mean delay
    return np.mean(delays)
```

```python
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-M",
        "--num_servers",
        help="number of servers; default is 1",
        default=1,
        type=int)                # for extension to M/M/m
    parser.add_argument(
        "-A",
        "--arrival_rate",
        help="packet arrival rate [packets/s]; default is 1.0",
        default=1.0,
        type=float)
    parser.add_argument(
        "-S",
        "--service_rate",
        help="packet service rate [packets/s]; default is 10.0",
        default=0.1,
        type=float)
    parser.add_argument(
        "-N",
        "--num_packets",
        help="number of packets to generate; default is 10000",
        default=10000,
        type=int)
    parser.add_argument(
        "-R",
        "--random_seed",
        help="seed for random number generation; default is 1234",
        default=1234,
        type=int)
    parser.add_argument('--trace', dest='trace', action='store_true')
    parser.add_argument('--no-trace', dest='trace', action='store_false')
    parser.set_defaults(trace=True)
    args = parser.parse_args()

    # set variables using command-line arguments
    num_servers = args.num_servers # for extension to M/M/m
    mean_ia_time = 1.0/args.arrival_rate
    mean_srv_time = 1.0/args.service_rate
    num_packets = args.num_packets
    random_seed = args.random_seed
```

```
    trace = args.trace

    # run a simulation
    mean_delay = run_simulation(num_servers, mean_ia_time,
mean_srv_time,

num_packets, random_seed,
                                                    trace)

    # print arrival rate and mean delay
    print("{0:.4E}\t{1:.4E}".format(args.arrival_rate,
mean_delay))
```

Code 1 code of New_M/M/m.py

```
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

# Major ticks every 10, minor ticks every 5 for x axis
x_major_ticks = np.arange(0, 101, 10)
x_minor_ticks = np.arange(0, 101, 5)

# Major ticks every 0.1, minor ticks every 0.1 for y axis
y_major_ticks = np.arange(0, 1.1, 0.2)
y_minor_ticks = np.arange(0, 1.1, 0.1)

ax.set_xticks(x_major_ticks)
ax.set_xticks(x_minor_ticks, minor=True)
ax.set_yticks(y_major_ticks)
ax.set_yticks(y_minor_ticks, minor=True)

# Or if you want different settings for the grids:
ax.grid(which='minor', alpha=0.2)
ax.grid(which='major', alpha=0.5)

# calculate and plot analytical results
x1 = np.arange(1, 100)
a = x1/100
y1 = (((a*a)/(x1*(1-a))))+1/100
plt.plot(x1, y1,'b-', label="Analysis", linewidth=1)

# load and plot simulation results
x2, y2 = np.loadtxt('mm1.out', delimiter='\t', unpack=True)
plt.plot(x2, y2, 'rx', label="Simulation")

# add labels, legend, and title
plt.xlabel(r'$\lambda$ [pkts/s]')
```

```
plt.ylabel(r'$E[T]$ [s]')
plt.legend()
plt.title(r'M/M/1 ($\mu=100$ [pkts/s])')

plt.savefig('mm1.pdf')
plt.show()
```

Code 2 Plot code of M/M/1

```
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

# Major ticks every 10, minor ticks every 5 for x axis
x_major_ticks = np.arange(0, 101, 10)
x_minor_ticks = np.arange(0, 101, 5)

# Major ticks every 0.1, minor ticks every 0.1 for y axis
y_major_ticks = np.arange(0, 1.1, 0.2)
y_minor_ticks = np.arange(0, 1.1, 0.1)

ax.set_xticks(x_major_ticks)
ax.set_xticks(x_minor_ticks, minor=True)
ax.set_yticks(y_major_ticks)
ax.set_yticks(y_minor_ticks, minor=True)

# Or if you want different settings for the grids:
ax.grid(which='minor', alpha=0.2)
ax.grid(which='major', alpha=0.5)

# calculate and plot analytical results
x1 = np.arange(1, 100)
a = x1/100
p0 = 1/(1+2*a+((((2*a)**2)/2)*1)/(1-a))
pm = (((2*a)**2)/(2*(1-a)))*p0
y1 = ((a/(x1*(1-a)))*pm)+1/50
plt.plot(x1, y1,'b-', label="Analysis", linewidth=1)

# load and plot simulation results
x2, y2 = np.loadtxt('mm2.out', delimiter='\t', unpack=True)
plt.plot(x2, y2, 'rx', label="Simulation")

# add labels, legend, and title
plt.xlabel(r'$\lambda$ [pkts/s]')
plt.ylabel(r'$E[T]$ [s]')
plt.legend()
plt.title(r'M/M/2 ($\mu=50$ [pkts/s])')
```

```python
plt.savefig('mm2.pdf')
plt.show()
```

Code 3 Plot code of M/M/2

```python
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

# Major ticks every 10, minor ticks every 5 for x axis
x_major_ticks = np.arange(0, 101, 10)
x_minor_ticks = np.arange(0, 101, 5)

# Major ticks every 0.1, minor ticks every 0.1 for y axis
y_major_ticks = np.arange(0, 1.1, 0.2)
y_minor_ticks = np.arange(0, 1.1, 0.1)

ax.set_xticks(x_major_ticks)
ax.set_xticks(x_minor_ticks, minor=True)
ax.set_yticks(y_major_ticks)
ax.set_yticks(y_minor_ticks, minor=True)
# Or if you want different settings for the grids:
ax.grid(which='minor', alpha=0.2)
ax.grid(which='major', alpha=0.5)
# calculate and plot analytical results
x1 = np.arange(1, 100)
a = x1/100
p0 =
1/(1+5*a+(((5*a)**2)/2)+(((5*a)**3)/6)+(((5*a)**4)/24)
+((((5*a)**5)/120)*1)/(1-a))
pm = (((5*a)**5)/(120*(1-a)))*p0
y1 = ((a/(x1*(1-a)))*pm)+1/20
plt.plot(x1, y1,'b-', label="Analysis", linewidth=1)

# load and plot simulation results
x2, y2 = np.loadtxt('mm5.out', delimiter='\t', unpack=True)
plt.plot(x2, y2, 'rx', label="Simulation")

# add labels, legend, and title
plt.xlabel(r'$\lambda$ [pkts/s]')
plt.ylabel(r'$E[T]$ [s]')
plt.legend()
plt.title(r'M/M/5 ($\mu=20$ [pkts/s])')

plt.savefig('mm2.pdf')
plt.show()
```

Code 4 Plot code of M/M/5

2nd Explanation

To run all the simulations in one batch, type the following:

For case 1:

```
for /L %A in (5, 5, 95) do python new_mm1.py -M 1 -A %A -S 100 --no-trace >> mm1.out
```

For case 2:

```
for /L %A in (5, 5, 95) do python new_mm1.py -M 2 -A %A -S 50   --no-trace >> mm2.out
```

For case 3:

```
for /L %A in (5, 5, 95) do python new_mm1.py -M 5 -A %A -S 20   --no-trace >> mm5.out
```

But if using the provided new_mm1.py directly, it cannot perform the M/M/2 and M/M/5 algorithms, because the number of services is fixed to 1 inside the new_mm1.py code, instead of changing it through external input. On line 31 in code 1 we should change the value of capacity from a fixed 1 num_servers, in addition, since the variable parameters are added, the parameter description should be added in the function header file(On line 26 and 34 in code 1 respectively). In order to make the simulation situation closer to the theoretical situation, the number of packets needs to be set to 10000.

In these three cases, when number of servers is one, the service rate is 100, while the number of servers is two and five, the service rates are 50 and 20. We can find that the products of service numbers and service rate are same, both are 100.

So ρ is a fixed number and is 100.

In case1 we consider M/M/1 theory, can be known from the formula:

Theoretically average delay: $\overline{\mathrm{T}} = \dfrac{\overline{\mathrm{N}}}{\lambda} = \dfrac{1}{\mu - \lambda}$

where, μ is the service rate and λ is arrival rate.

In case2 and case3 we consider M/M/m (m equal 2 and 5), The formula and derivation process are shown below:

Because $\sum_{n=0}^{\infty} p_n = 1$, we obtain:

$$\mathrm{p}_0 = \frac{1}{\sum_{n=0}^{m-1} \frac{(m\rho)^n}{n!} + \frac{(m\rho)^m}{m!} \frac{1}{1-\rho}}$$

Where $\rho = \dfrac{\lambda}{\mathrm{m} \cdot \mu}$

We first obtain p_{m+}, which will be used in deriving main results:

$$p_{m+} \Leftrightarrow P\{N \geq m\} = \sum_{n=m}^{\infty} p_n = \frac{(m\rho)^m}{m!(1-\rho)} p_0$$

p_{m+} is the probability that all servers are busy and the customers has to wait in the queue (also called ErlangC formula).

Theoretically average delay: $\overline{\mathrm{T}} = \overline{\mathrm{W}} + \frac{1}{\mu} = \frac{\rho}{\lambda(1\text{-}\rho)} \mathrm{p}_{\mathrm{m+}} + \frac{1}{\mu}$

For case2 m=2, μ=50, λ=5,10,15,...,95.

For case3 m=5, μ=20, λ=5,10,15,...,95.

The 17th line in code2 code3 and code4 is the theoretical result of M/M/1, M/M/2 and M/M/5. The algorithm in the code uses the above formula. Contrast Figure1 Figure2 and Figure 3, we can find that the analytical results and simulation results completely coincide in three cases, and the overall average delay from M/M/1 to M/M/5 has increased slightly. This is because the increase in the number of services has led to the above-mentioned increase in pm, which eventually led to an increase in average delay.

This shows that the simulated theoretical results of the queuing are consistent with the expected results. And it shows that when the service rate product is fixed, the delay caused by the fewer services will be less.