EEE414-DATA COMMUNICATION-LAB1

Author: Zhipeng Ye

Student ID: 1926908

Faculty: MSc Multimedia Telecommunications

Date: 12th December 2019

Introduction

In the class, we have studied M/M/N queuing which is a theory about balancing efficiency and economy. As we all know, the more number of servers become, the more efficient the system will be. However the cost of queuing system will raise with the increase of number of servers. Therefore, scientists and engineers set up M/M/N queuing theory to research and draw up solution of this domain.

Firstly, the first M represents that the arrival rate of consumer tends to Markov process (memoryless) and the second M means that service time submits to Markov process (memoryless). Furthermore, N represents the number of service.

Suppose N means steady-state number of customers, λ means steady-state arrival rate, T means steady-state customer delay.

Then $N = \lambda \cdot T$.

The possibility of the number of costumer is n in given time t. tends to poisson process. Here is the reason.

$$\begin{aligned} P_0(t, t+\Delta t) &= 1 - \lambda \cdot \Delta t + o(\Delta t) \\ P_1(t, t+\Delta t) &= \lambda \cdot \Delta t + o(\Delta t) \\ P_{n\geq 2}(t, t+\Delta t) &= o(\Delta t) \end{aligned}$$

We can use these formula to demonstrate that $P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$. Then the interval time submits to exponential distribution. This is because that $P\{no\ costumber\ arrived\ at\ [0,t)\} = P_0(t) = 1 - e^{-\lambda t}$. The function of possibility density is $f(t) = \lambda e^{-\lambda t}, t > 0$.

Thus, the number of costumers who leaved tend to possian process and the interval time of costumer leaving tend to the exponential distribution.

For M/M/1 system, the possibility of numbers of costumers are at system, $p_n = p_0 \cdot \rho^n,\ \rho = \frac{\lambda}{u}, p < 1$. Moreover, $p_0 = 1 - \rho, p_n = (1 - \rho) \cdot p^n$.

Then, the M/M/1 has below properties.

M/M/1 Queueing System: Main Results

- Average number of customers in the system

$$\overline{N} = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda}$$

- Average delay

$$\overline{T} = \frac{\overline{N}}{\lambda} = \frac{1}{\mu - \lambda}$$

- Average waiting time in the queue

$$\overline{W} = \frac{1}{\mu - \lambda} - \frac{1}{\mu} = \frac{\rho}{\mu - \lambda}$$

- Average number of customers in the queue

$$\overline{N_Q} = \lambda \overline{W} = \frac{\rho^2}{1-\rho}$$

And the average delay will submit this curve.

M/M/1 Queueing System: Main Results

- Average number of customers in the system

$$\overline{N} = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}$$

- Average delay

$$\overline{T} = \frac{\overline{N}}{\lambda} = \frac{1}{\mu-\lambda}$$

- Average waiting time in the queue

$$\overline{W} = \frac{1}{\mu-\lambda} - \frac{1}{\mu} = \frac{\rho}{\mu-\lambda}$$

- Average number of customers in the queue

$$\overline{N_Q} = \lambda\overline{W} = \frac{\rho^2}{1-\rho}$$

Simulation

Simulate 3 case , as mentioned in the given task instruction.

$$E[T] = \frac{1}{\mu} + \frac{E[N_Q]}{\lambda} = \frac{1}{\mu} + \frac{\rho^2}{\lambda(1-\rho)}.$$

Our experimental value may submit to this formula.

Step

Simulate

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
##
# @file     new_mmN.py
# @author   Kyeong Soo (Joseph) Kim <kyeongsoo.kim@gmail.com>
# @date     2019-10-29
#
# @brief    Simulate M/M/1 queueing system
#
# @remarks  Copyright (C) 2019 Kyeong Soo (Joseph) Kim. All rights reserved.
#
# @remarks  This software is written and distributed under the GNU General
```

```
#           Public License Version 2 (http://www.gnu.org/licenses/gpl-
2.0.html).
#           You must not remove this notice, or any other, from this
software.
#

import argparse
import random

import numpy as np
import simpy


def source(env, mean_ia_time, mean_srv_time, server, delays, number, trace):
    """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=1000, random_seed=1234, trace=True):
    """Runs a simulation and returns statistics."""
    if trace:
        print('M/M/' + str(num_servers) + ' 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)


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 1000",
        default=1000,
        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))
```

Here is my command line code, because I use Mac OS, I choose shell code to run simulation batchly.

```
for i in $(seq 5 5 95);
do echo $i ;
python new_mm1.py -M 1 -A $i -S 100 --no-trace >> mm1.out
done
```

Result

```
5.0000E+00  1.0640E-02
1.0000E+01  1.1237E-02
1.5000E+01  1.1913E-02
2.0000E+01  1.2733E-02
2.5000E+01  1.3536E-02
3.0000E+01  1.4300E-02
3.5000E+01  1.5230E-02
4.0000E+01  1.6420E-02
4.5000E+01  1.7755E-02
5.0000E+01  1.9363E-02
5.5000E+01  2.1401E-02
6.0000E+01  2.3922E-02
6.5000E+01  2.6818E-02
7.0000E+01  3.0910E-02
7.5000E+01  3.6378E-02
8.0000E+01  4.3777E-02
8.5000E+01  5.3441E-02
9.0000E+01  6.8108E-02
9.5000E+01  8.9153E-02
```

Plot diagram

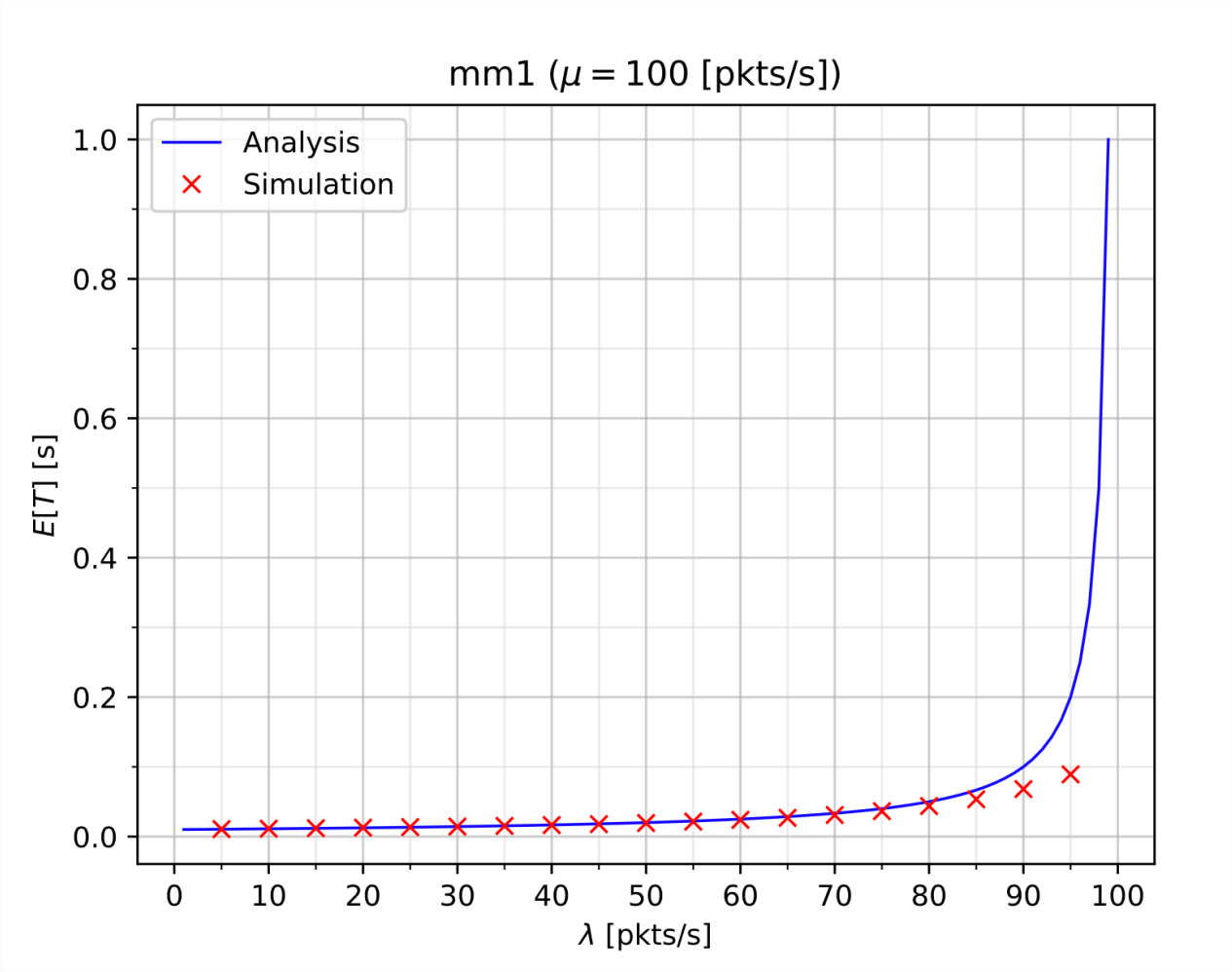
```
import argparse

import numpy as np
import matplotlib.pyplot as plt

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-F",
        "--file_path",
        help="file_path",
        default='',
        type=str)
    args = parser.parse_args()
```

```
    file_path = args.file_path

    file_prefix = file_path.split('.')[0]

    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)
    y1 = 1 / (100 - x1)
    plt.plot(x1, y1, 'b-', label="Analysis", linewidth=1)

    # load and plot simulation results
    # change delimiter '/t' into ','
    x2, y2 = np.loadtxt(file_path, 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'' + file_prefix + ' ($\mu=100$ [pkts/s])')

    plt.savefig(file_prefix + '.pdf')
    plt.show()
```

Shell code

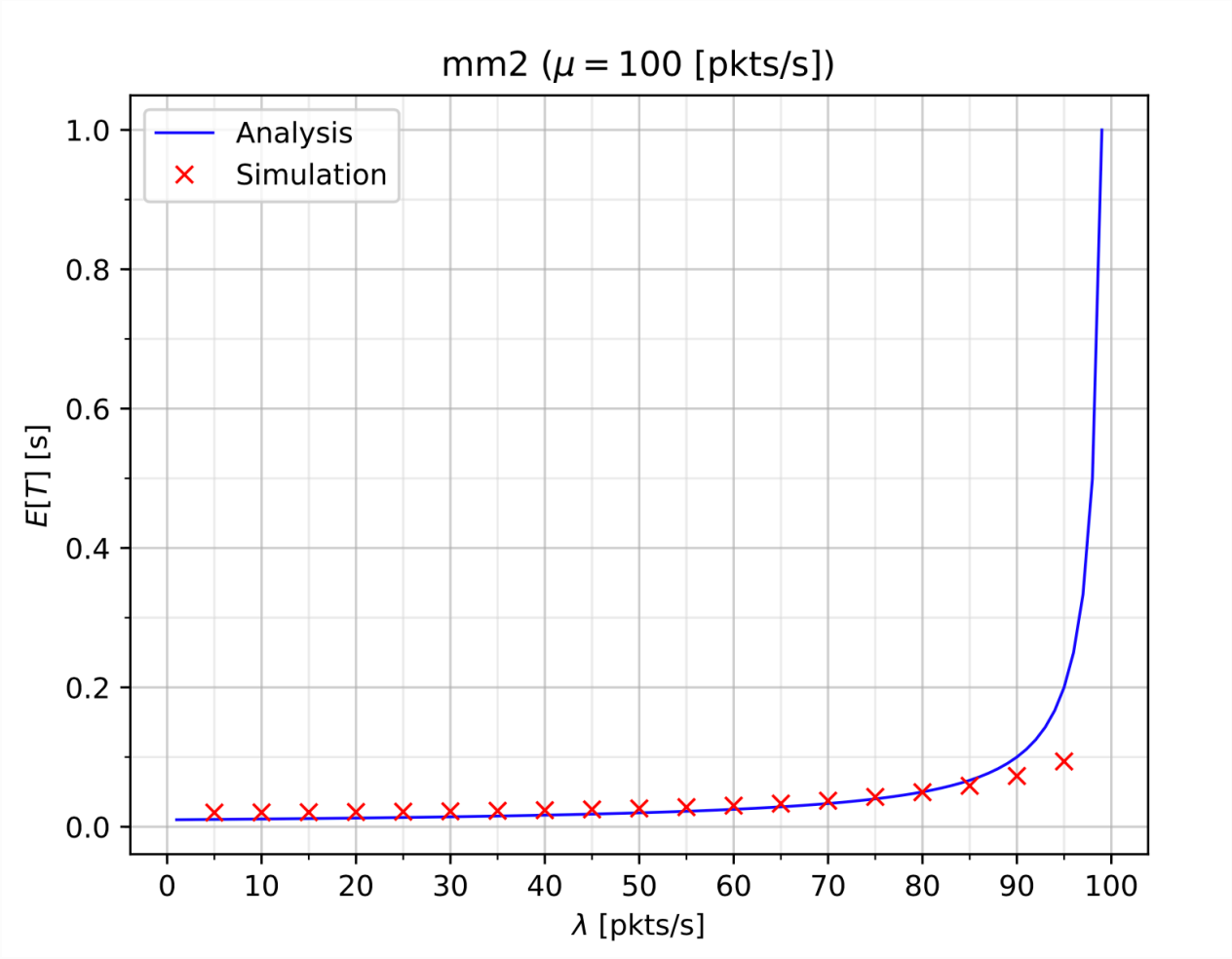
```
python plot_mmN.py -F 'mm1.out'
```

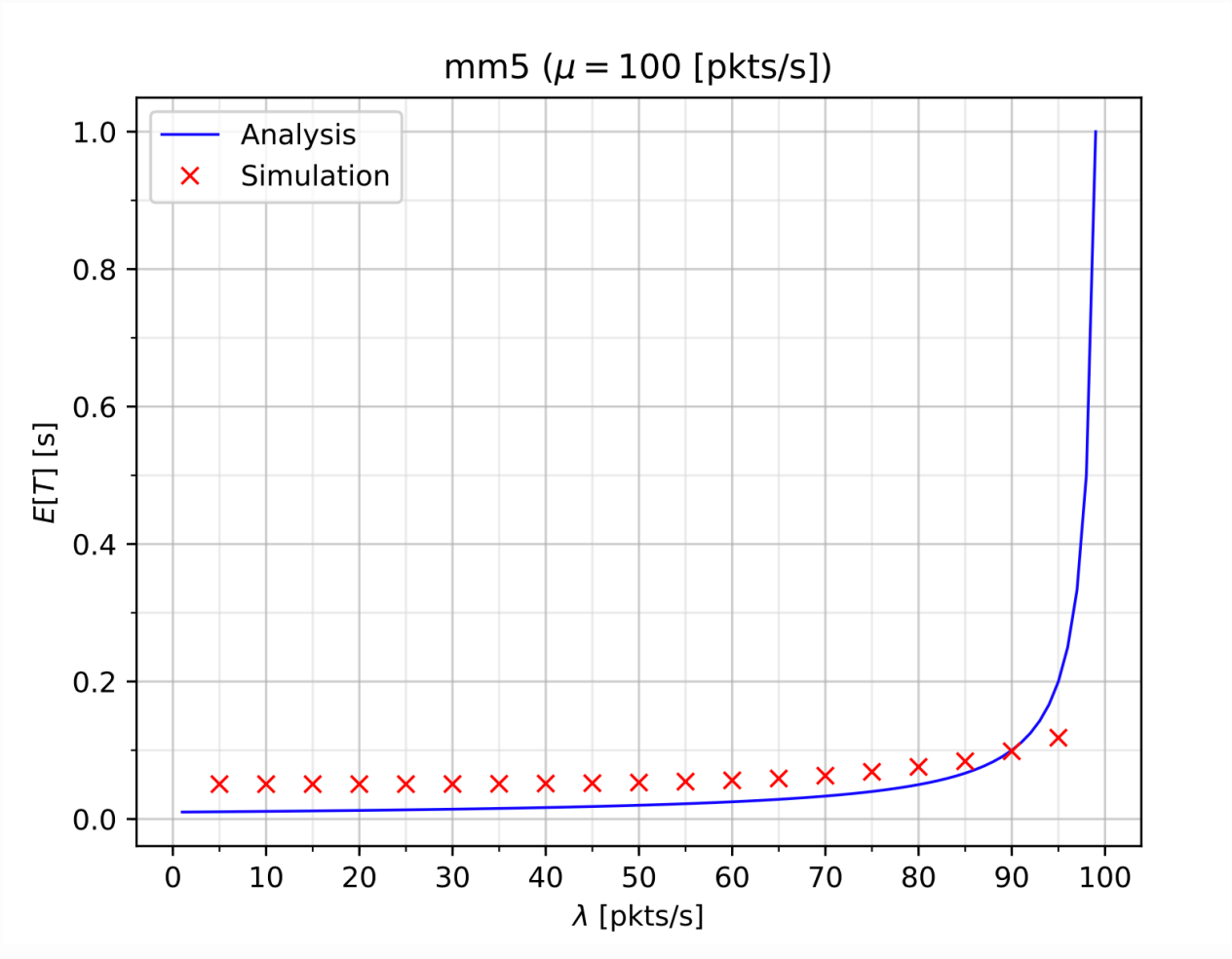
Result



Analysis

As we can see from result, the simulation is consistent to theoretical curve. Refer to previous process, here is my result of M/M/2 and M/M/5.


mm2 ($\mu = 100$ [pkts/s])
Analysis
Simulation
1.0
0.8
0.6
0.4
0.2
0.0
$E[T]$ [s]
0
10
20
30
40
50
60
70
80
90
100
λ [pkts/s]


mm5 ($\mu = 100$ [pkts/s])
Analysis
Simulation
1.0
0.8
0.6
0.4
0.2
0.0
$E[T]$ [s]
0
10
20
30
40
50
60
70
80
90
100
λ [pkts/s]

Comparision and conclusion

As we can see from the diagram, the results are consistent with this formula.

$$E(T) = \frac{1}{u} + \frac{(\frac{\lambda}{u})^2}{\lambda(1-\frac{\lambda}{u})}$$

Therefore, average delay is associated with arrival time and service time.