Project 2

Name: Richard Xie

ID: 915505564

Part 1

1.

Table compares the mean delay of simulated value with the theoretical value. ()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| λ(pkts/s) | 0.2 | 0.4 | 0.6 | 0.8 | 0.9 | 0.99 |
| Simulated Value (s) | 1.251 | 1.658 | 2.529 | 4.996 | 9.558 | 86.938 |
| Theoretical Value (s) | 1.250 | 1.667 | 2.500 | 5.000 | 10.000 | 100.000 |
| Percent Difference | 0.080% | 0.540% | 1.160% | 0.080% | 4.420% | 13.062% |

2.

, n = 0, 1, 2, … , N – 1

 for n = 0, 1, 2, … , N





 for n = 0, 1, 2, … , N

, 

3. code and output

# Richard Xie 915505564

# This is a simpy based simulation of a M/M/1 queue system

import random

import simpy

import math

RANDOM\_SEED = 29

SIM\_TIME = 1000000

MU = 1

B = 10 # modify buffer size for different output

""" Queue system """

class server\_queue:

def \_\_init\_\_(self, env, arrival\_rate, Packet\_Delay, Server\_Idle\_Periods):

self.server = simpy.Resource(env, capacity = 1)

self.env = env

self.queue\_len = 0

self.flag\_processing = 0

self.packet\_number = 0

self.sum\_time\_length = 0

self.start\_idle\_time = 0

self.arrival\_rate = arrival\_rate

self.Packet\_Delay = Packet\_Delay

self.Server\_Idle\_Periods = Server\_Idle\_Periods

def process\_packet(self, env, packet):

with self.server.request() as req:

start = env.now

yield req

yield env.timeout(random.expovariate(MU))

latency = env.now - packet.arrival\_time

self.Packet\_Delay.addNumber(latency)

#print("Packet number {0} with arrival time {1} latency {2}".format(packet.identifier, packet.arrival\_time, latency))

self.queue\_len -= 1

if self.queue\_len == 0:

self.flag\_processing = 0

self.start\_idle\_time = env.now

def packets\_arrival(self, env):

# packet arrivals

while True:

# Infinite loop for generating packets

yield env.timeout(random.expovariate(self.arrival\_rate))

# arrival time of one packet

self.Packet\_Delay.addC()

self.packet\_number += 1

# packet id

arrival\_time = env.now

#print(self.num\_pkt\_total, "packet arrival")

new\_packet = Packet(self.packet\_number,arrival\_time)

if self.flag\_processing == 0:

self.flag\_processing = 1

idle\_period = env.now - self.start\_idle\_time

self.Server\_Idle\_Periods.addNumber(idle\_period)

#print("Idle period of length {0} ended".format(idle\_period))

if self.queue\_len < B :

self.queue\_len += 1

else :

continue

env.process(self.process\_packet(env, new\_packet))

""" Packet class """

class Packet:

def \_\_init\_\_(self, identifier, arrival\_time):

self.identifier = identifier

self.arrival\_time = arrival\_time

class StatObject:

def \_\_init\_\_(self):

self.dataset =[]

self.total = 0

def addNumber(self,x):

self.dataset.append(x)

def addC(self) :

self.total += 1

def totalC(self) :

return self.total

def sum(self):

n = len(self.dataset)

sum = 0

for i in self.dataset:

sum = sum + i

return sum

def mean(self):

n = len(self.dataset)

sum = 0

for i in self.dataset:

sum = sum + i

return sum/n

def maximum(self):

return max(self.dataset)

def minimum(self):

return min(self.dataset)

def count(self):

return len(self.dataset)

def median(self):

self.dataset.sort()

n = len(self.dataset)

if n//2 != 0: # get the middle number

return self.dataset[n//2]

else: # find the average of the middle two numbers

return ((self.dataset[n//2] + self.dataset[n//2 + 1])/2)

def standarddeviation(self):

temp = self.mean()

sum = 0

for i in self.dataset:

sum = sum + (i - temp)\*\*2

sum = sum/(len(self.dataset) - 1)

return math.sqrt(sum)

def main():

print("Simple queue system model:mu = {0}".format(MU))

print ("{0:<9} {1:<9} {2:<9} {3:<9} {4:<9} {5:<9} {6:<9} {7:<9} {8:<9}".format(

"Lambda", "Count", "Min", "Max", "Mean", "Median", "Sd", "Utilization", "Pd"))

random.seed(RANDOM\_SEED)

for arrival\_rate in [0.2, 0.4, 0.6, 0.8, 0.9, 0.99]:

env = simpy.Environment()

Packet\_Delay = StatObject()

Server\_Idle\_Periods = StatObject()

router = server\_queue(env, arrival\_rate, Packet\_Delay, Server\_Idle\_Periods)

env.process(router.packets\_arrival(env))

env.run(until=SIM\_TIME)

print ("{0:<9.3f} {1:<9} {2:<9.3f} {3:<9.3f} {4:<9.3f} {5:<9.3f} {6:<9.3f} {7:<9.3f} {8:<9.9f}".format(

round(arrival\_rate, 3),

int(Packet\_Delay.count()),

round(Packet\_Delay.minimum(), 3),

round(Packet\_Delay.maximum(), 3),

round(Packet\_Delay.mean(), 3),

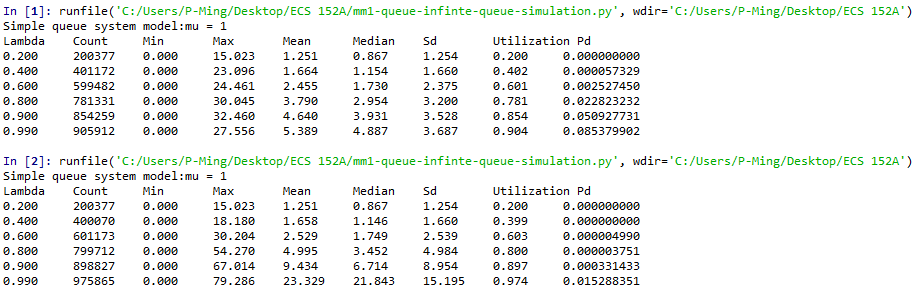
round(Packet\_Delay.median(), 3),

round(Packet\_Delay.standarddeviation(), 3),

round(1-Server\_Idle\_Periods.sum()/SIM\_TIME, 3),

(Packet\_Delay.totalC() - Packet\_Delay.count()) / float(Packet\_Delay.totalC())))

if \_\_name\_\_ == '\_\_main\_\_': main()



4.

Table compares the loss probability Pd of simulated value with the theoretical value.

B = 10

|  |  |  |  |
| --- | --- | --- | --- |
| λ(pkts/s) | Simulated  Value | Theoretical Value | Percent Difference |
| 0.2 | 0.000000000 | 8.192 x 10^-8 | 0 |
| 0.4 | 0.000057329 | 0.000062317 | 8.882% |
| 0.6 | 0.002527450 | 0.002427454 | 4.119% |
| 0.8 | 0.022823232 | 0.023492858 | 2.850% |
| 0.9 | 0.050927731 | 0.050813731 | 0.224% |
| 0.99 | 0.085379902 | 0.086409993 | 1.192% |

B = 50

|  |  |  |  |
| --- | --- | --- | --- |
| λ(pkts/s) | Simulated  Value | Theoretical Value | Percent Difference |
| 0.2 | 0.000000000 | 9.0072 x 10^-36 | 0 |
| 0.4 | 0.000000000 | 7.6059 x 10^-21 | 0 |
| 0.6 | 0.000004990 | 3.2331 x 10^-12 | 1.5434 x 10^8 % |
| 0.8 | 0.000003751 | 0.000002855 | 31.405% |
| 0.9 | 0.000331433 | 0.000517779 | 35.990% |
| 0.99 | 0.015288351 | 0.015085778 | 1.343% |

Part 2

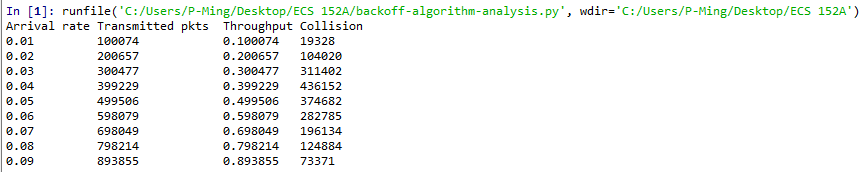
1.1 See the other pdf file.

1.2

Table of throughput with binary exponential backoff algorithm.

|  |  |
| --- | --- |
| λ(pkts/s) | Throughput |
| 0.01 | 0.100074 |
| 0.02 | 0.200657 |
| 0.03 | 0.300477 |
| 0.04 | 0.399229 |
| 0.05 | 0.499506 |
| 0.06 | 0.598079 |
| 0.07 | 0.698049 |
| 0.08 | 0.798214 |
| 0.09 | 0.893855 |

Code output

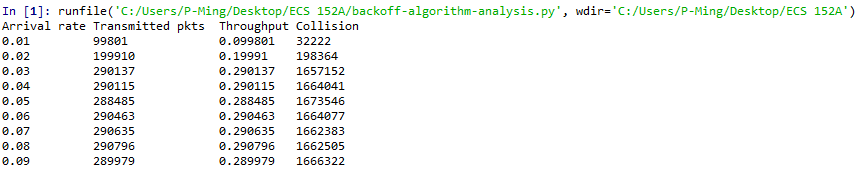


It’s almost like linear grow, which satisfies what we discussed in class.

Table of throughput with linear backoff algorithm.

|  |  |
| --- | --- |
| λ(pkts/s) | Throughput |
| 0.01 | 0.099801 |
| 0.02 | 0.19991 |
| 0.03 | 0.290137 |
| 0.04 | 0.290115 |
| 0.05 | 0.288485 |
| 0.06 | 0.290463 |
| 0.07 | 0.290635 |
| 0.08 | 0.290796 |
| 0.09 | 0.289979 |

Code output



It grows for a while, then stops growing, which satisfies what we discussed in class.