DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF CALIFORNIA, DAVIS
ECS152 WEIRAN GUO(912916431) DECEMBER 8, 2017

Project2 Report

1 PART 1

a).
if the number of packets in the system less than B, then the packet loss won't happens.

for markov chain, the maximum packet in the system is B, hence,

$$\lambda\pi_0 = \mu\pi_1 \tag{1.1}$$
$$\lambda\pi_1 = \mu\pi_2 \tag{1.2}$$
$$... \tag{1.3}$$
$$\lambda\pi_B = \mu\pi_{B+1} \tag{1.4}$$
$$\tag{1.5}$$

therefore,

$$1 = \sum_{i=0}^{B+1} \pi_i \tag{1.6}$$
$$1 = \sum_{i=0}^{B+1} \pi_0(\frac{\lambda}{\mu}^i) \tag{1.7}$$
$$\pi_0 = \frac{1-\frac{\lambda}{\mu}}{1-\frac{\lambda}{\mu}^{B+1}} \tag{1.8}$$
$$\pi_{B+1} = \frac{\frac{\lambda}{\mu}^{B+1}(1-\frac{\lambda}{\mu})}{1-\frac{\lambda}{\mu}^{B+2}} \tag{1.9}$$
$$\tag{1.10}$$

hence, the loss rate is

$$P_d = \frac{\frac{\lambda}{\mu}^{B+1}(1-\frac{\lambda}{\mu})}{1-\frac{\lambda}{\mu}^{B+2}} \tag{1.11}$$

b).simulation code

```
# This is a simpy based  simulation of a M/M/1 queue system
# Now the buffer packet size is limited to B

import random
import simpy
import math

RANDOM_SEED = 29
SIM_TIME = 1000000
MU = 1



""" Queue system  """
class server_queue:
        def __init__(self, env, arrival_rate, Packet_Delay,
        Server_Idle_Periods, Buffer_Size):
                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.drop = 0
                self.buffer_size = Buffer_Size
                self.arrival_rate = arrival_rate
                self.Packet_Delay = Packet_Delay
                self.Server_Idle_Periods = Server_Idle_Periods

        def drop_rate(self):#calculate the drop rate, drop/
            total_packets
                return self.drop/self.packet_number

        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_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 < self.buffer_size:#
                   ######drop packet
                        self.queue_len += 1
                        env.process(self.process_packet(
                           env, new_packet))
                else:
                        self.drop += 1


""" Packet class """
```

```python
class Packet:
        def __init__(self, identifier, arrival_time):
                self.identifier = identifier
                self.arrival_time = arrival_time


class StatObject:
    def __init__(self):
        self.dataset =[]

    def addNumber(self,x):
        self.dataset.append(x)
    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 cal_packet_loss(arrival_rate, buffer_size):#######the theoretical loss rate
    pd = 1 - (1 - (arrival_rate/MU)**buffer_size)/(1 - (arrival_rate/MU)**(buffer_size+1))
    #print(Sum)###########each step print out the calculated probability that the packet of system is <= buffer size

    return pd

def main():
    random.seed(RANDOM_SEED)
    for buffer_size in [10, 50]:
        print("Simple queue system model:mu = {0}, buffer size = {1}".format(MU, buffer_size))
        print ("{0:<9} {1:<9} {2:<9} {3:<9} {4:<9} {5:<9} {6:<9} {7:<9} {8:<9} {9:<9}".format(
        "Lambda", "Count", "Min", "Max", "Mean", "Median", "Sd", "Utilization", "Loss Rate", "Theoretical"))
        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, buffer_size)
            env.process(router.packets_arrival(env))
            env.run(until=SIM_TIME)
            #expected_delay = 1/MU/(1-(arrival_rate/MU))
            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:<9.3f} {9:<9.3f}".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),
             round(router.drop_rate(),3),
             round(cal_packet_loss(arrival_rate, buffer_size),3)))
```

```
if __name__ == '__main__': main()
```

c).tables

Simple queue system model:mu = 1

Lambda	Count	Min	Max	Mean	Median	Sd	Utilization	Theoretical
0.200	200377	0.000	15.023	1.251	0.867	1.254	0.200	1.250
0.400	400070	0.000	18.180	1.658	1.146	1.660	0.399	1.667
0.600	601173	0.000	30.204	2.529	1.749	2.539	0.603	2.500
0.800	799713	0.000	54.270	4.996	3.452	4.988	0.800	5.000
0.900	898679	0.000	90.886	9.558	6.698	9.370	0.897	10.000
0.990	991142	0.000	419.359	86.938	64.409	76.626	0.990	100.000

Simple queue system model:mu = 1, buffer size = 10

Lambda	Count	Min	Max	Mean	Median	Sd	Utilization	Loss Rate	Theoretical
0.200	200377	0.000	15.023	1.251	0.867	1.254	0.200	0.00000	0.00000
0.400	399774	0.000	19.706	1.655	1.144	1.658	0.399	0.00003	0.00003
0.600	600324	0.000	26.514	2.495	1.737	2.448	0.602	0.00174	0.00145
0.800	785589	0.000	31.911	3.966	3.068	3.386	0.786	0.01811	0.01845
0.900	860391	0.000	34.938	4.969	4.194	3.796	0.860	0.04418	0.04373
0.990	912681	0.000	30.296	5.897	5.389	3.989	0.912	0.07926	0.07880

Simple queue system model:mu = 1, buffer size = 50

Lambda	Count	Min	Max	Mean	Median	Sd	Utilization	Loss Rate	Theoretical
0.200	200093	0.000	14.758	1.241	0.859	1.244	0.199	0.00000	0.00000
0.400	400168	0.000	26.712	1.666	1.154	1.668	0.399	0.00000	0.00000
0.600	599717	0.000	30.476	2.491	1.734	2.474	0.599	0.00000	0.00000
0.800	801115	0.000	48.397	4.981	3.458	4.937	0.801	0.00000	0.00000
0.900	897903	0.000	77.005	9.555	6.717	9.152	0.897	0.00029	0.00047
0.990	974599	0.000	80.541	24.022	22.694	15.528	0.976	0.01552	0.01472

2 Part 2

a). Sample execution

```
    """Binary Exponetial and Linear Backoff Algorithm"""

import random
import simpy
import math

RANDOM_SEED = 29
SIM_TIME = 1000000
MU = 1
TS = 1 #time slot
N_HOST = 3 #number of hosts

""" Host class """
class Host:
    def __init__(self, env, arrival_rate):
        self.env = env
        self.arrival_rate = arrival_rate
        self.L = 0
        self.S = 0
```

```
        #self.arrives = 0
        self.N = 0 #times retransmitted
        self.server = simpy.Resource(env, capacity = 1)
        env.process(self.packets_arrival(env))

    def packets_arrival(self, env):
        while True:
            yield env.timeout(random.expovariate(self.
                arrival_rate))
            #self.arrives += 1
            if (self.L == 0): #no packet in the queue
                self.S = math.floor(env.now) + 1
                self.N = 0
            self.L += 1

    def process_packet(self, env):
        self.L -= 1
        if (self.L > 0):
            self.N = 0
            self.S = math.floor(env.now) + 1

    def exp_backoff(self, env):
        r = min(self.N, N_HOST)
        self.S = math.floor(env.now + random.randint(0,2**r)*TS)
            + 1
        self.N += 1

    def linear_backoff(self, env):
        K = min(self.N, 1024)
        self.S = math.floor(env.now + random.randint(0, K)*TS) +
            1
        self.N += 1

""" Queue system  """
class Ethernet:
    def __init__(self, env, arrival_rate, Type):
        self.env = env
        self.hosts = [Host(env, arrival_rate) for i in range(
            N_HOST)]
        self.success = 0
        self.collide = 0
        self.blank = 0
        self.Type = Type
        self.arrival_rate = arrival_rate
```

```
    def sim(self, env):
        while True:
            hostLst = []
            for host in self.hosts:
                if (host.L > 0 and (host.S == math.floor(env.now
                    ))):#host is requesting now
                    hostLst.append(host)
            #print(len(hostLst))
            if (len(hostLst) == 1):#succeed
                hostLst[0].process_packet(env)
                self.success += 1
            elif (len(hostLst) > 1):#collide
                self.collide += 1
                for host in hostLst:
                    if (self.Type == 0):#exponetial
                        host.exp_backoff(env)
                    else:
                        host.linear_backoff(env)
            else:
                self.blank += 1
            yield self.env.timeout(TS)


def main():
        print("Ethernet system model:mu = {0}, Ts = {1}, N = {2}
            ".format(MU, TS, N_HOST))

        random.seed(RANDOM_SEED)
        for Type in [0,1]:
                if (Type == 0):
                        print ("For Exponetial Backoff Algorithm
                            ")
                else:
                        print ("For Linear Backoff Algorithm")
                print ("{0:<9} {1:<9} {2:<9} {3:<9} {4:<9}".
                    format(
                        "Lambda", "Success", "Failed","Blank", "
                            Throughput"))
                for arrival_rate in [0.01, 0.02, 0.03, 0.04,
                    0.05, 0.06, 0.07, 0.08, 0.09]:
                        env = simpy.Environment()
                        ethernet = Ethernet(env, arrival_rate,
                            Type)
```

```
        env.process(ethernet.sim(env))
        env.run(until=SIM_TIME)
        print ("{0:<9.3f} {1:<9} {2:<9}{3:<9} 
            {4:<9.5f}".format(
                arrival_rate,
                ethernet.success,
                ethernet.collide,
                ethernet.blank,
                ethernet.success/SIM_TIME))

if __name__ == '__main__': main()
```

b).Simulation results of the 10-host system

```
Ethernet system model:mu = 1, Ts = 1, N = 10
For Exponetial Backoff Algorithm
```

Lambda	Success	Failed	Blank	Throughput
0.010	99876	9259	890865	0.09988
0.020	200186	47015	752799	0.20019
0.030	300151	137241	562608	0.30015
0.040	399742	193964	406294	0.39974
0.050	499488	172628	327884	0.49949
0.060	598620	131757	269623	0.59862
0.070	698978	93424	207598	0.69898
0.080	796404	61827	141769	0.79640
0.090	892039	36764	71197	0.89204

for exponential backoff algorithm, as λ increases, the Throughput increases linearly.

```
For Linear Backoff Algorithm
```

Lambda	Success	Failed	Blank	Throughput
0.010	99829	15284	884887	0.09983
0.020	200483	85326	714191	0.20048
0.030	293187	592115	114698	0.29319
0.040	289192	633604	77204	0.28919
0.050	290455	632573	76972	0.29046
0.060	289794	633053	77153	0.28979
0.070	291135	632170	76695	0.29113
0.080	289783	633348	76869	0.28978
0.090	288847	633688	77465	0.28885

for linear backoff algorithm, as λ increases, the Throughput increases at first, but reaches the ceiling throughput around 0.29.

Analysis: as λ increases, for exponential backoff algorithm, the success slots keep in-

creases until the throughput approaches 1, the blank time slots keep decreasing which means the algorithm is effective.
However, the throughput of linear backoff algorithm reaches its ceiling aroung 0.29, which is not effective, that means the blank time slots does not decrease after λ reaches certain level, while the throughput is 0.29 that far less than 1.