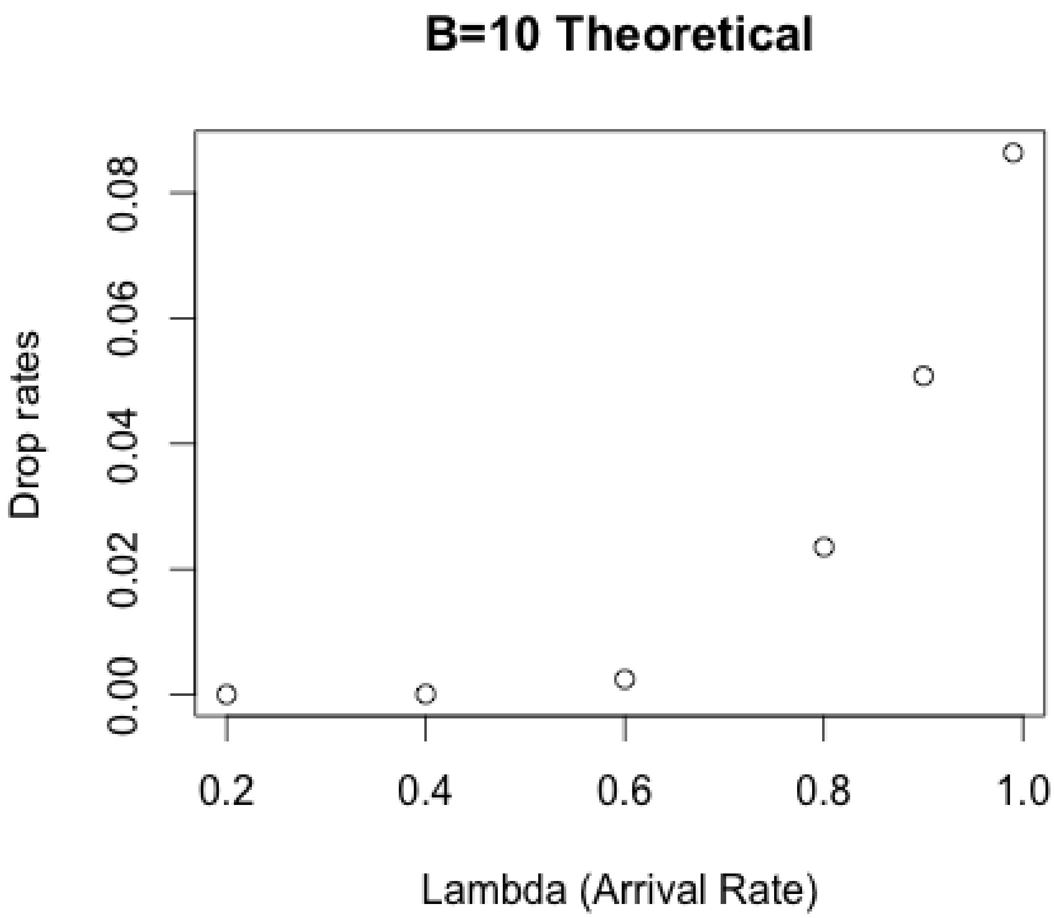
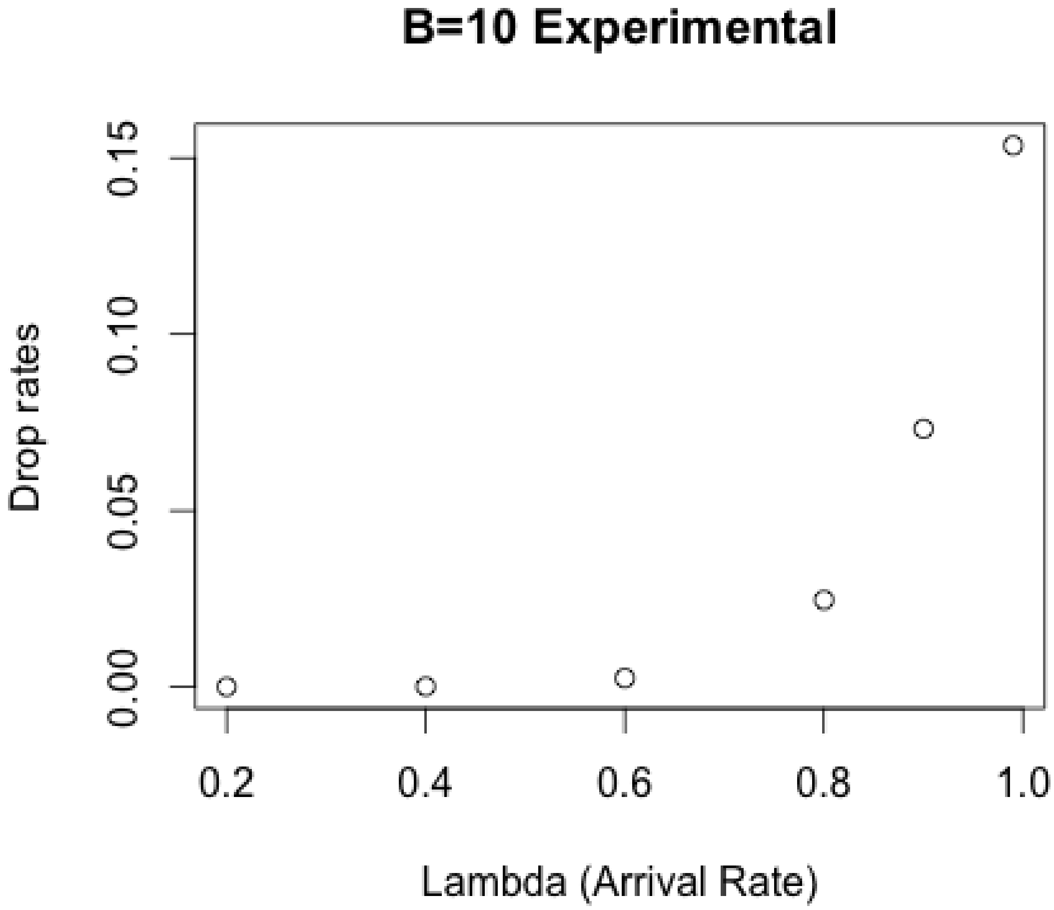
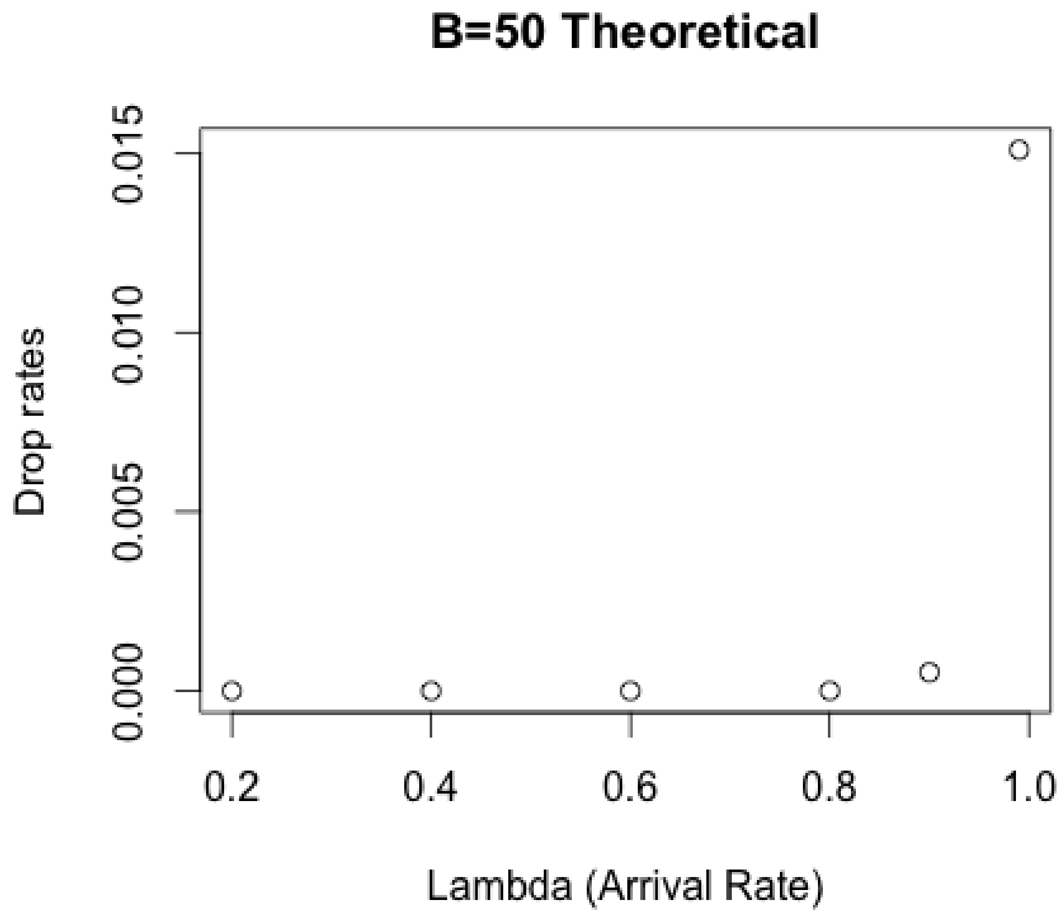
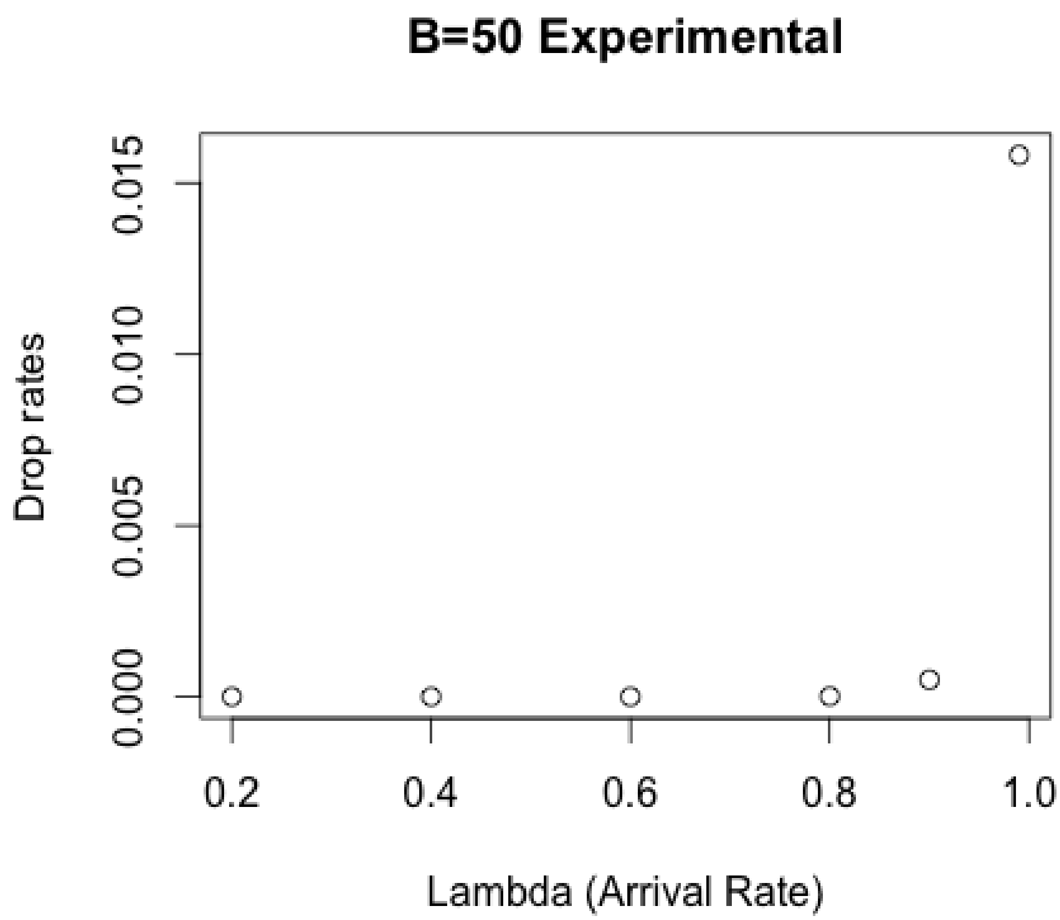
**ECS 152A Project Part 1 – Kevin Huang, Mark Fan**




B=10 Experimental
Drop rates
0.00
0.05
0.10
0.15
0.2
0.4
0.6
0.8
1.0
Lambda (Arrival Rate)


B=50 Theoretical
Drop rates
0.000
0.005
0.010
0.015
0.2
0.4
0.6
0.8
1.0
Lambda (Arrival Rate)




**Python Code for Simulation1.py:**

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

from decimal import *
getcontext().prec = 10
import random
import simpy
import math

RANDOM_SEED = 34 #old one was 33
SIM_TIME = 100000000 #one hundred million
MU = 1
buffer_size = 50
```

```
dropped_pkts = Decimal(0)
drop_rate = Decimal(0)

""" 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
            #self.packet_count = 0

        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
              global dropped_pkts # need this else get the local variable unbound error
               global drop_rate # need this else get the local variable unbound error
            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 < buffer_size: #if buffer still has space
                                    self.queue_len += 1
                                    env.process(self.process_packet(env, new_packet))
                        else: #buffer_size is greater than queue length
                                    dropped_pkts += 1
```

```
""" 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 =[]

    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 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}".format(
       "Lambda", "Count", "Min", "Max", "Mean", "Median", "Sd", "Utilization"))
        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}".format(
                round(arrival_rate, 3),
                int(Packet_Delay.count()),
                round(Packet_Delay.minimum(), 8),
                round(Packet_Delay.maximum(), 8),
                round(Packet_Delay.mean(), 3),
                round(Packet_Delay.median(), 3),
                round(Packet_Delay.standarddeviation(), 3),
                round(1-Server_Idle_Periods.sum()/SIM_TIME, 3)))
        drop_rate = (Decimal(dropped_pkts)) / (Decimal(router.packet_number)) #calculate drop
rate
        print("The drop rate is: ")
        print('{0:.24f}'.format(drop_rate))
if __name__ == '__main__': main()
```

**R Code for generating the Plots:**

```
lambda <- c(0.2, 0.4, 0.6, 0.8, 0.9, 0.99)

b10_theoretical <- c(8.19E-08, 6.29E-05, 2.43E-03, 2.35E-02, 5.08E-02, 8.64E-02)

b10_experimental <- c(1.50028E-07, 0.000105007, 0.002455453, 0.024608602, 0.073165069,
0.153582507)

b50_theoretical <- c(9.01E-36, 7.61E-21, 3.23E-12, 2.85E-06, 5.20E-04, 1.51E-02)

b50_experimental <- c(0, 0, 0, 1.25E-05, 0.00049516, 0.015815203)

plot(lambda, b10_theoretical, main="B=10 Theoretical", xlab="Lambda (Arrival Rate)", ylab=("Drop
rates"))

plot(lambda, b50_theoretical, main="B=50 Theoretical", xlab="Lambda (Arrival Rate)", ylab=("Drop
rates"))

plot(lambda, b10_experimental, main="B=10 Experimental", xlab="Lambda (Arrival Rate)",
ylab=("Drop rates"))

plot(lambda, b50_experimental, main="B=50 Experimental", xlab="Lambda (Arrival Rate)",
ylab=("Drop rates"))
```

**Table**

| | A | B | C | D |
|---|---|---|---|---|
| 1 | Lambda | Theoretical | Experimental | Percent Difference ([Experimental - Theoretical] / Theoretical) |
| 2 | Buffer = 10 | | | |
| 3 | 0.2 | 8.19E-08 | 1.50028E-07 | 8.31E-01 |
| 4 | 0.4 | 6.29E-05 | 0.000105007 | 6.69E-01 |
| 5 | 0.6 | 2.43E-03 | 0.002455453 | 1.17E-02 |
| 6 | 0.8 | 2.35E-02 | 0.024608602 | 4.72E-02 |
| 7 | 0.9 | 5.08E-02 | 0.073165069 | 4.40E-01 |
| 8 | 0.99 | 8.64E-02 | 0.153582507 | 7.78E-01 |
| 9 | Buffer = 50 | | | |
| 10 | 0.2 | 9.01E-36 | 0 | 1.00E+00 |
| 11 | 0.4 | 7.61E-21 | 0 | 1.00E+00 |
| 12 | 0.6 | 3.23E-12 | 0 | 1.00E+00 |
| 13 | 0.8 | 2.85E-06 | 1.25E-05 | 3.38E+00 |
| 14 | 0.9 | 5.20E-04 | 0.00049516 | 4.78E-02 |
| 15 | 0.99 | 1.51E-02 | 0.015815203 | 4.88E-02 |
| 16 | | | | |

Simulation was done with seed = 33 and 34. SIM_TIME = 100,000 for B = 10, lambda = 0.4, 0.6, 0.8, 0.99 and B = 50, lambda = 0.8, 0.9, 0.99. SIM_TIME = 100,000,000 for B = 10, lambda = 0.2. For B = 50, lambda = 0.2, 0.4, 0.6, even with a simulation time of 100,000,000 and running it several times, the results were still 0.0…0 (such that the number of trailing 0's is 38). This is because the drop rate is incredibly low the aforementioned lambda and buffer combinations. If we increased SIM_TIME to let's say 1,000,000,000 or more, we would have a greater chance of obtaining a non-zero drop rate.