Answer 1

a) **Problem Summary**

In this discrete event simulation study, the operational capacity of a post office was explored. Here, the packages classified under 2 categories – regular and priority are received under a Poisson process. The rate of arrival of packages is 4 packages per unit time. A package's probability of being classified as a regular is 0.9, and a priority is 0.1. The post office employs 9 workers to handle incoming packages with a rate of 0.5 packages per unit of time. The handling of such packages follows an exponential distribution with a mean of 2. If all the workers are busy, the packages are placed into 2 queues: a regular queue and a priority queue, depending on the package type. The workers always process packages in the priority queue first and then regular queue.

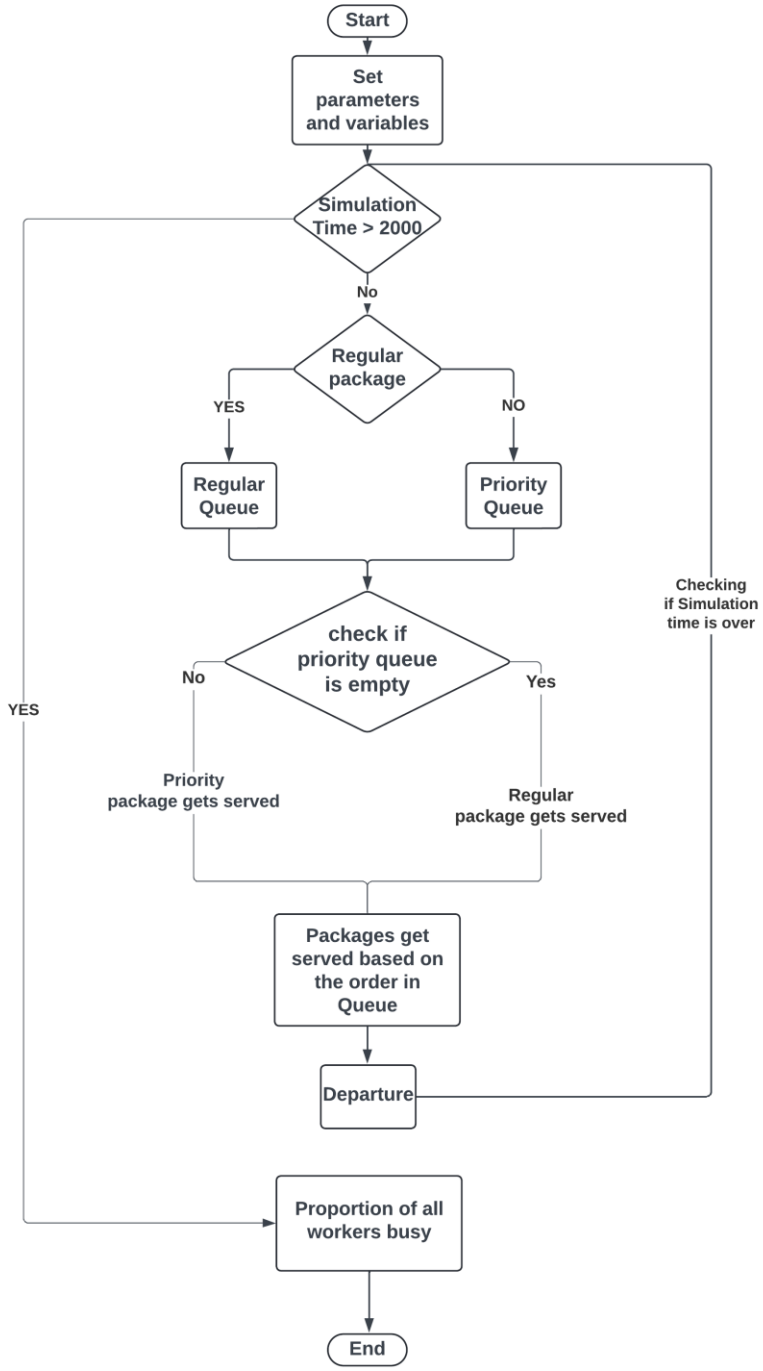
Project Objective

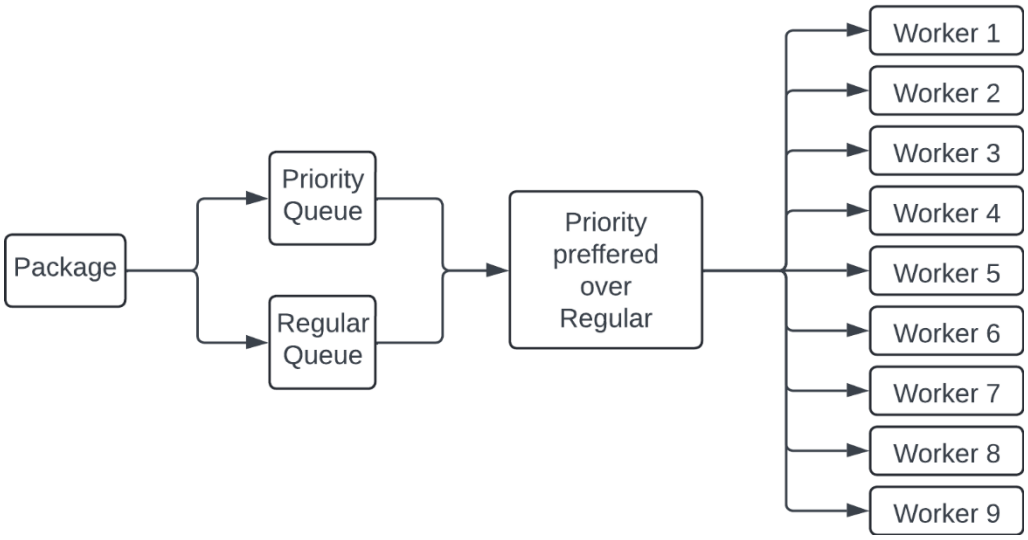
The study's objective is to evaluate the performance of the post office workers. The simulation study wants to assess the proportion of time that all the workers are busy given a time period of 2000 units of time. Such a study aims to delve deep and provide information about the effectiveness of the workers in the post office. The study's results aim to provide a list of future enhancements that can be used to optimise the functioning of the post office. The result will present a 95% confidence interval when all the workers were busy handling the packages.

b) The variables used for this simulation study –

- **T:** Total simulation time for which the simulation will be run, and metrics will be calculated. It is set to 2000 for this study.
- **arrival_rate:** The rate at which the packages will arrive at the post office. For the simulation study, the incoming packages follow a Poisson process where λ is set to an average of 4 packages per unit of time.
- **processing_rate:** The mean rate at which the post office workers process incoming packages. The processing rate follows an exponential distribution, with a mean of 2, meaning 0.5 packages are processed every unit of time.
- **regular_probability:** The probability of receiving a regular package is 0.9

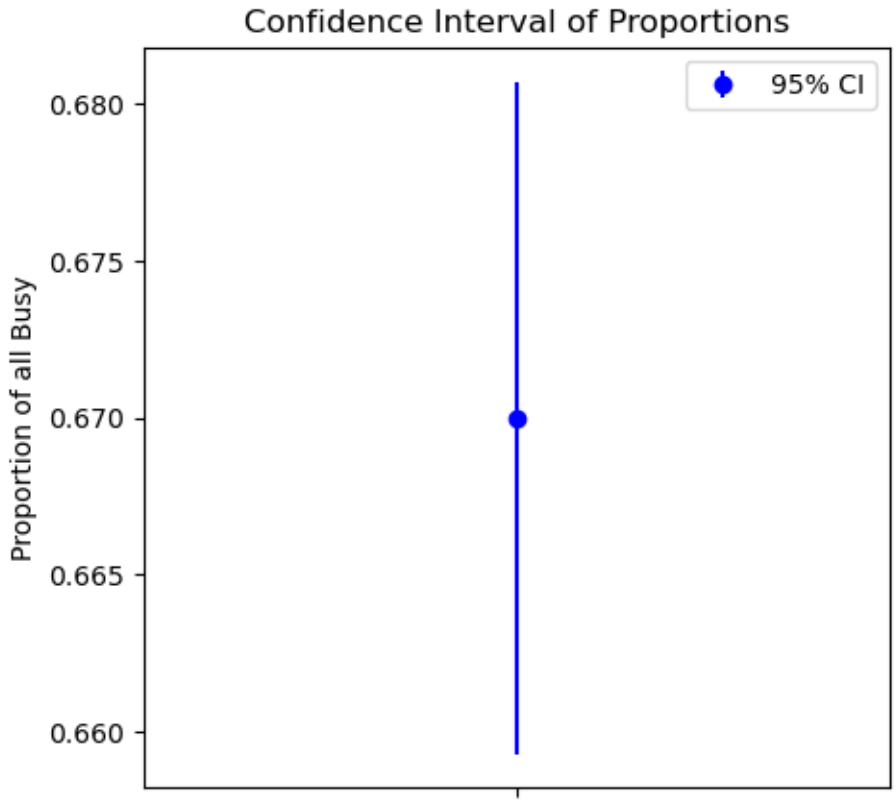
- **workers:** The number of workers employed by the post office to process the packages. The number of workers in this study is set to 9.
- **regular_queue:** The queue used to manage the arrival of regular packages that arrive with a probability of 0.9.
- **priority_queue:** The queue used to manage the arrival of priority packages that arrive with a probability of 0.1.

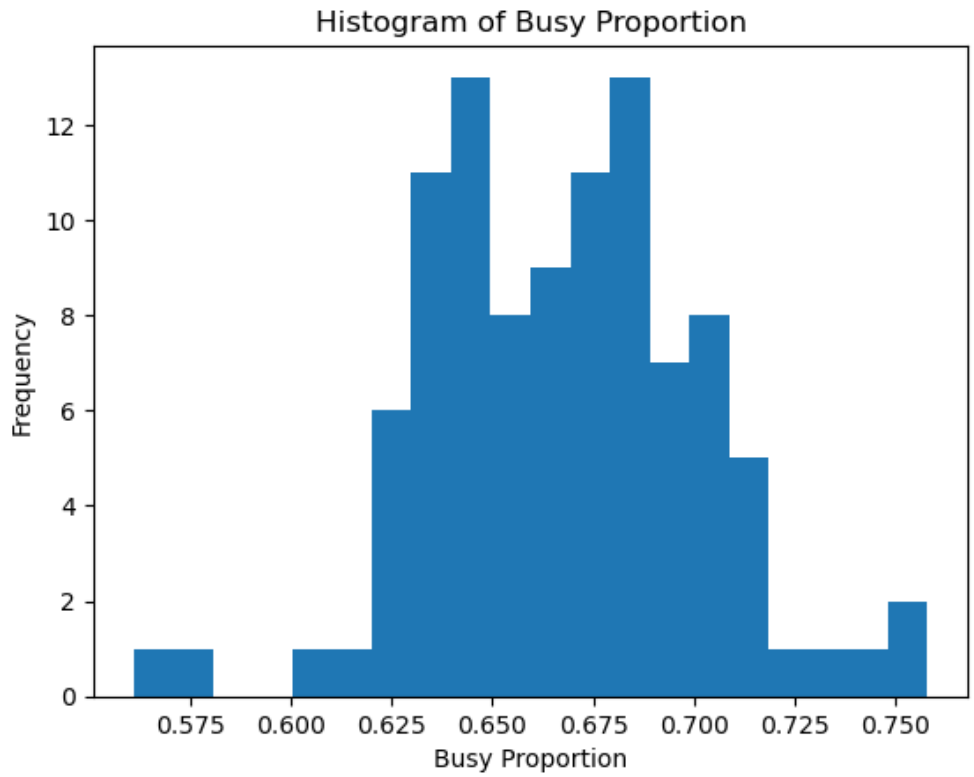




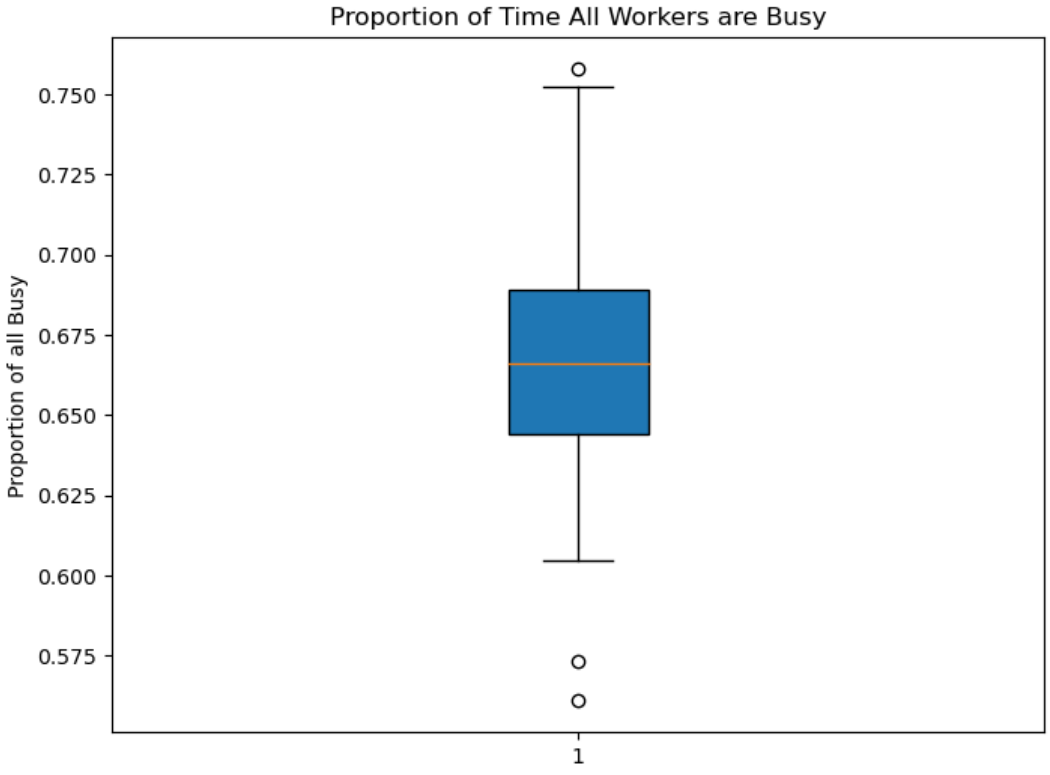
c) Metrics –

Metric	Value
Max proportion	0.7697329797475084
Mean proportion	0.6764814159077787
Minimum proportion	0.5847243416719641
Confidence Interval	[0.6687363725607226, 0.6842264592548347]

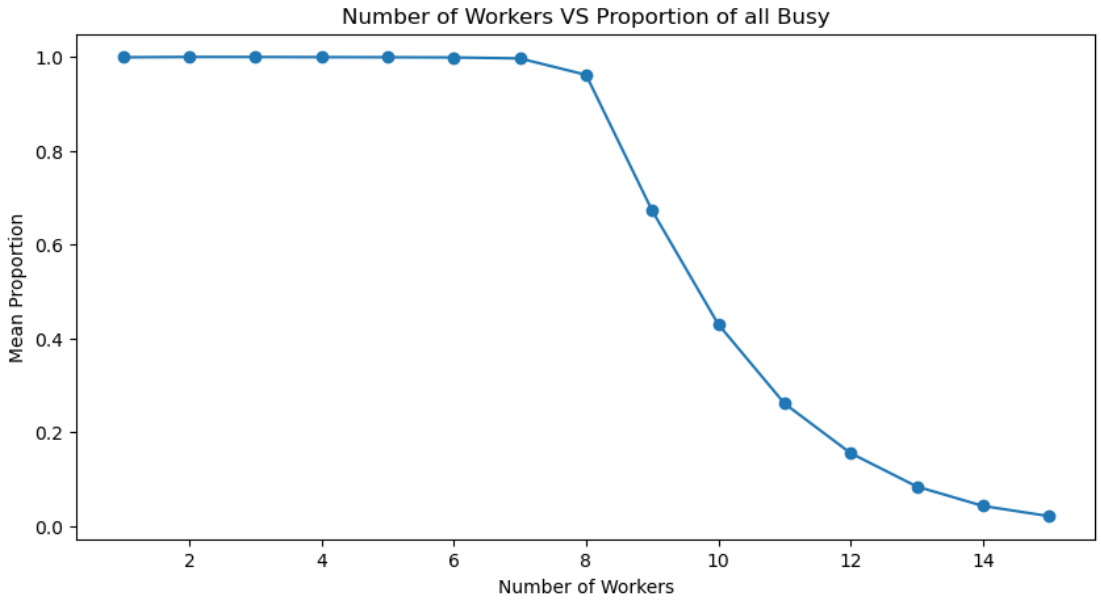




The results centre between the 0.625 and 0.7 values. Based on this run, the histogram shows a bimodal distribution. This suggests that all workers were busy processing packages 60% to 70% of the time.



From the box plot, we can infer that there were some results where the proportion lies outside the whiskers of the plot. These results are to be considered as the outliers of the system.



From the graph, we can see that the proportion of time when all the workers are busy is very high when the number of workers is low. As the number of workers hit 8, the proportion of time when all the workers are busy decreases sharply. This was done on a mean of 50 simulation runs.

d) From this, we can conclude that the proportion of time when all the users are busy hovers around a mean of 0.67. This means that 67% of the time, all the workers will be busy processing the packages. If the number of workers were to decrease, the proportion of time when all the workers were busy would increase sharply. Based on the confidence interval of [0.668, 0.68], we can infer that given the slim range, the mean estimate will be highly precise. This means that the mean represents the performance of the system. This proportion of business means there is a scope for improvement, where the number of workers could be reduced to increase the proportion. Through this, the post office can improve its revenue by saving on one worker's salary while still managing to serve the processing requirements, as the proportion will increase to 96%, providing better utilisation. With 9 workers, the system works well to serve the requirements of the system and maintains a good proportion of total time busy, where workers have opportunities for rest, but their utilisation is down.

e) (Code attached in appendix)

For the purpose of the study, I have implemented the study in a simpy simulation. The code consists of 3 basic blocks of data.
In the first block, the necessary libraries are called.

[1]:
```
import simpy
import numpy as np
```

In the second block of code, a function that runs a simulated environment is set up. Resource managers like workers, priority queues, and regular queues are defined here.

[2]:
```
# Simulating Post Office
def postOffice(env, workers, regular_probability, arrival_rate, processing_rate, T):

    # Creating Workers and queues to handle packages
    workers = simpy.Resource(env, capacity=workers)
    regular_queue = simpy.Store(env)
    priority_queue = simpy.Store(env)
```

This data block also contains the functions required for handling the arrival and processing of the packages. The function handling the arrival of the data sorts the packages into regular or priority queues. If some of the workers are free, packages in the priority queue are handled first, and if the priority queue is empty, the workers process the packages in the regular queue.

```
    # Function to handle incoming packages
    def package_arrival(env):
        while True:
            # New package arrival
            yield env.timeout(np.random.exponential(1 / arrival_rate))
            package_type = "regular" if np.random.choice([0.9, 0.1], 1, p=[0.9, 0.1])[0] ==

            # Place packages into queue
            if(package_type == "priority"):
                priority_queue.put(package_type)
            else:
                regular_queue.put(package_type)

            # Process priority packages before regular packages
            if(workers.count < workers.capacity):
                if(len(priority_queue.items) > 0):
                    yield priority_queue.get()
                    env.process(package_processing(env, "priority"))
                else:
                    yield regular_queue.get()
                    env.process(package_processing(env, "regular"))
```

After arrival, the processing phase starts, where the checks are added to calculate the time all the workers are busy.

```
# Function to process the packages in the queues
def package_processing(env, package_type):
    with workers.request() as request:
        yield request

        # Updating time when all the workers are busy
        nonlocal last_update, all_busy_time
        current_time = env.now
        if(workers.count == workers.capacity):
            all_busy_time = all_busy_time + (current_time - last_update)
        last_update = current_time

        # package processing
        yield env.timeout(np.random.exponential(1 / processing_rate))
        next_package_type = None
        if(len(priority_queue.items) > 0):
            next_package_type = "priority"
            yield priority_queue.get()
        elif(len(regular_queue.items) > 0):
            next_package_type = "regular"
            yield regular_queue.get()

        if(next_package_type is not None):
            env.process(package_processing(env, next_package_type))
```

The last block of code sets the simulation parameters. Based on the collected results, the mean, standard deviation, and 95% confidence interval are calculated.

[3]:
```
# Simulation Parameters
workers = 9  #Number of workers in the post office
regular_probability = 0.9  #Probability of receiving regular package
arrival_rate = 4  #Packages are arriving at a Poisson process rate of lambda = 4
processing_rate = 0.5  #Packages are processed at an exponential rate of mu = 2
T = 2000  #Total Simulation run time
number_of_runs = 100  #Number of times running the simulation

# Collecting Results
results = []
for i in range(0, number_of_runs):
    results.append(postOffice(simpy.Environment(), workers, regular_probability, arrival_ra
```

[4]:
```
# Calculating results and confidence interval
max = np.max(results)
min = np.min(results)
mean = np.mean(results)
stddev = np.std(results)
lower, upper = mean - (1.96 * (stddev / np.sqrt(len(results)))), mean + (1.96 * (stddev / n

print(f"Max Proportion of Time All Workers are Busy: {max}")
print(f"Mean Proportion of Time All Workers are Busy: {mean}")
print(f"Min Proportion of Time All Workers are Busy: {min}")
print(f"95% Confidence Interval: [{lower}, {upper}]")
```

```
Max Proportion of Time All Workers are Busy: 0.811026184265119
Mean Proportion of Time All Workers are Busy: 0.6620354414217118
Min Proportion of Time All Workers are Busy: 0.5636412305766644
95% Confidence Interval: [0.6544723898222953, 0.6695984930211283]
```

Importing Libraries

In [9]:

```
import simpy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Simulating Post Office

In [2]:

```
# Simulating Post Office
def postOffice(env, workers, regular_probability, arrival_rate, processing_rate, T):

    # Creating Workers and queues to handle packages
    workers = simpy.Resource(env, capacity=workers)
    regular_queue = simpy.Store(env)
    priority_queue = simpy.Store(env)

    # Function to process the packages in the queues
    def package_processing(env, package_type):
        with workers.request() as request:
            yield request

            # Updating time when all the workers are busy
            nonlocal last_update, all_busy_time
            current_time = env.now
            if(workers.count == workers.capacity):
                all_busy_time = all_busy_time + (current_time - last_update)
            last_update = current_time

            # package processing
            yield env.timeout(np.random.exponential(1 / processing_rate))
            next_package_type = None
            if(len(priority_queue.items) > 0):
                next_package_type = "priority"
                yield priority_queue.get()
            elif(len(regular_queue.items) > 0):
                next_package_type = "regular"
                yield regular_queue.get()

            if(next_package_type is not None):
                env.process(package_processing(env, next_package_type))


    # Function to handle incoming packages
    def package_arrival(env):
        while True:
            # New package arrival
            yield env.timeout(np.random.exponential(1 / arrival_rate))
            package_type = "regular" if np.random.choice([0.9, 0.1], 1, p=[0.9, 0.1])[0]
== regular_probability else "priority"

            # Place packages into queue
            if(package_type == "priority"):
                priority_queue.put(package_type)
            else:
                regular_queue.put(package_type)

            # Process priority packages before regular packages
            if(workers.count < workers.capacity):
                if(len(priority_queue.items) > 0):
                    yield priority_queue.get()
```

```
                    env.process(package_processing(env, "priority"))
                else:
                    yield regular_queue.get()
                    env.process(package_processing(env, "regular"))

    # Tracking total time all workers were busy
    all_busy_time = 0
    last_update = 0

    # Main Simulation run
    env.process(package_arrival(env))
    env.run(until=T)

    # Returning the proportion of time when all the workers were busy
    return all_busy_time / T
```

Results

In [3]:

```
# Simulation Parameters
workers = 9  #Number of workers in the post office
regular_probability = 0.9  #Probability of receiving regular package
arrival_rate = 4  #Packages are arriving at a Poisson process rate of lambda = 4
processing_rate = 0.5  #Packages are processed at an exponential rate of mu = 2
T = 2000  #Total Simulation run time
number_of_runs = 100  #Number of times running the simulation

# Collecting Results
results = []
for i in range(0, number_of_runs):
    results.append(postOffice(simpy.Environment(), workers, regular_probability, arrival
_rate, processing_rate, T))
```

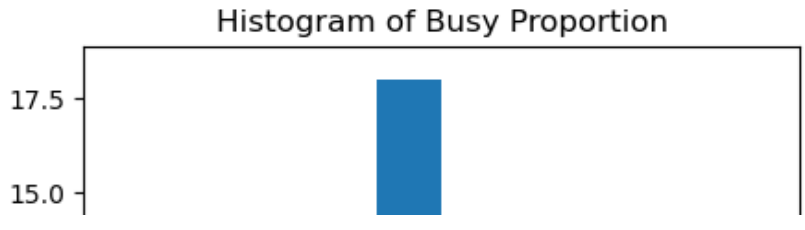
In [4]:

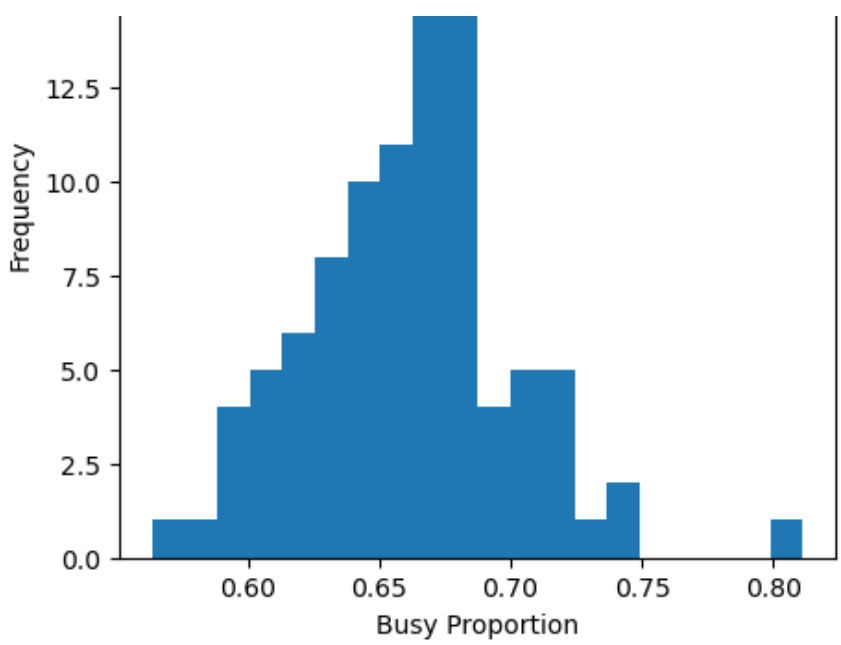
```
# Calculating results and confidence interval
max = np.max(results)
min = np.min(results)
mean = np.mean(results)
stddev = np.std(results)
lower, upper = mean - (1.96 * (stddev / np.sqrt(len(results)))), mean + (1.96 * (stddev
/ np.sqrt(len(results))))

print(f"Max Proportion of Time All Workers are Busy: {max}")
print(f"Mean Proportion of Time All Workers are Busy: {mean}")
print(f"Min Proportion of Time All Workers are Busy: {min}")
print(f"95% Confidence Interval: [{lower}, {upper}]")
```

```
Max Proportion of Time All Workers are Busy: 0.811026184265119
Mean Proportion of Time All Workers are Busy: 0.6620354414217118
Min Proportion of Time All Workers are Busy: 0.5636412305766644
95% Confidence Interval: [0.6544723898222953, 0.6695984930211283]
```

In [5]:

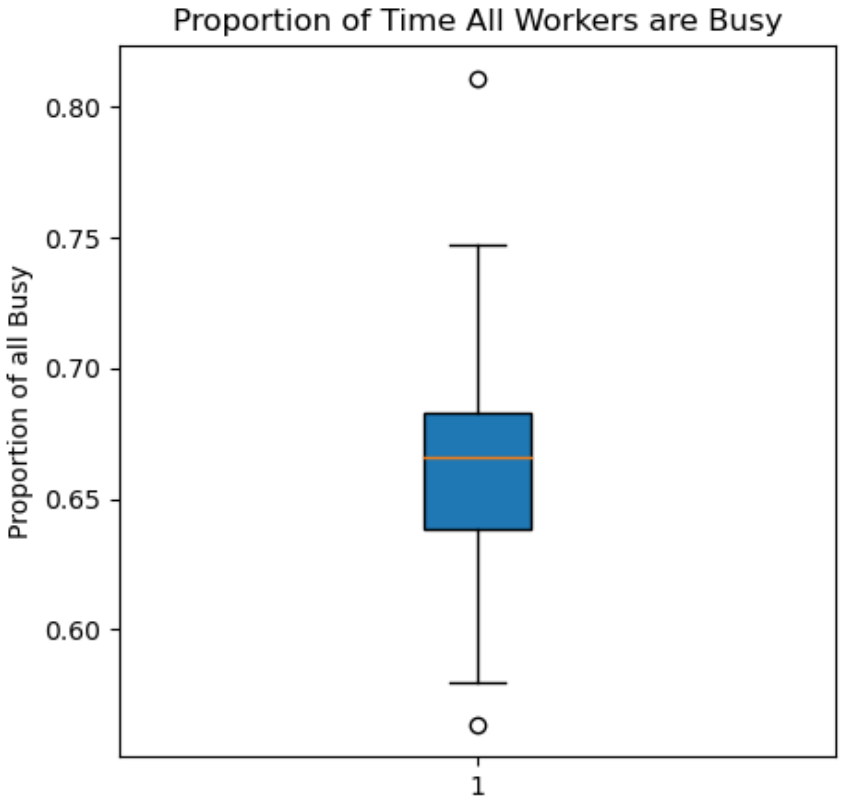
```
plt.figure(figsize=(5, 5))
plt.hist(results, bins=20)
plt.title("Histogram of Busy Proportion")
plt.xlabel("Busy Proportion")
plt.ylabel("Frequency")
plt.show()
```





In [7]:

```
plt.figure(figsize=(5, 5))
plt.boxplot(results, patch_artist=True)
plt.title("Proportion of Time All Workers are Busy")
plt.ylabel("Proportion of all Busy")
plt.show()
```



In [11]:

```
df = pd.DataFrame(columns=["workers", "mean"])

for i in range(1, 16):
    workers = i
    regular_probability = 0.9
    arrival_rate = 4
    processing_rate = 0.5
    T = 2000
    number_of_runs = 50

    results = []
    for i in range(0, number_of_runs):
```

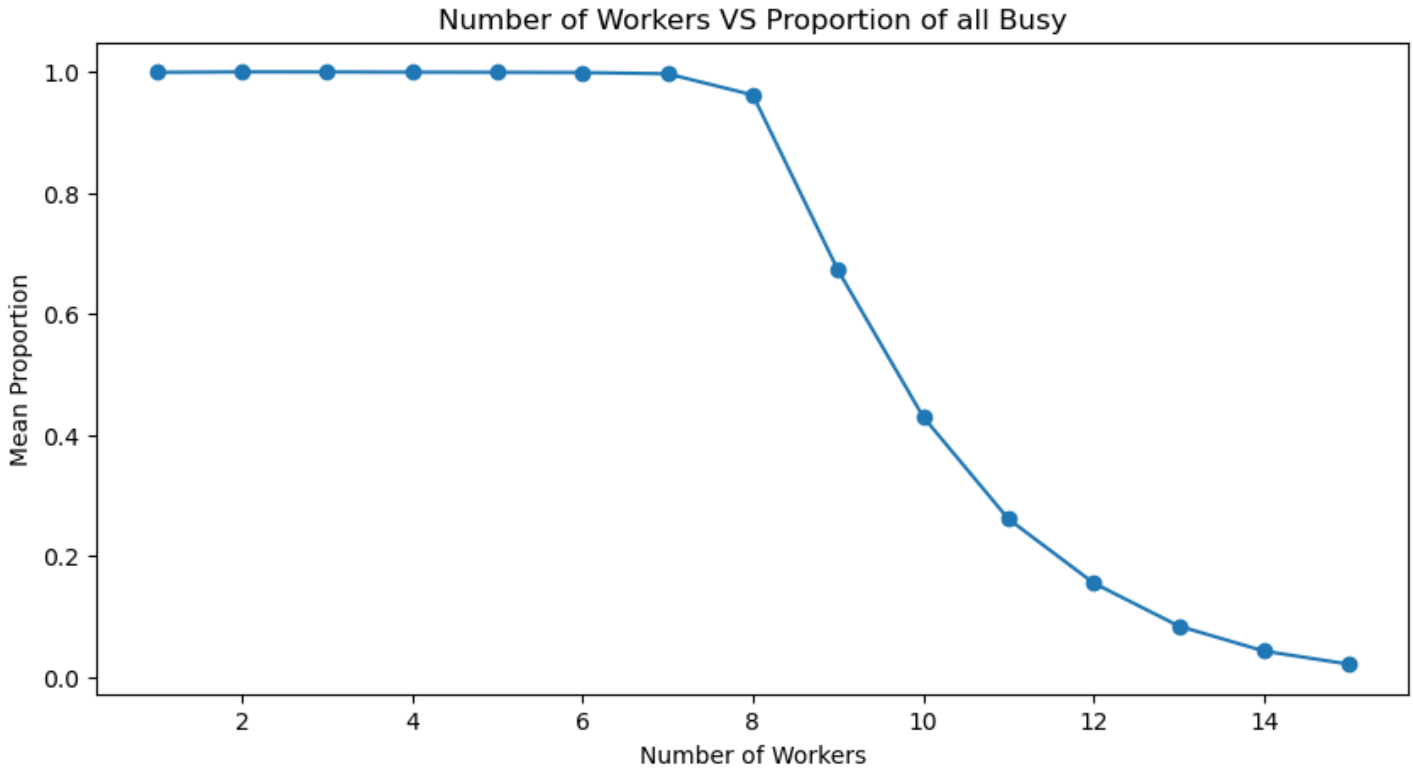
```
        results.append(postOffice(simpy.Environment(), workers, regular_probability, arr
ival_rate, processing_rate, T))

    df = df._append({"workers": workers, "mean": np.mean(results)}, ignore_index=True)
df
```

Out[11]:

	workers	mean
0	1.0	0.998690
1	2.0	0.999426
2	3.0	0.999378
3	4.0	0.999017
4	5.0	0.998814
5	6.0	0.998378
6	7.0	0.996353
7	8.0	0.961336
8	9.0	0.671707
9	10.0	0.429491
10	11.0	0.260808
11	12.0	0.155009
12	13.0	0.083874
13	14.0	0.042534
14	15.0	0.020765

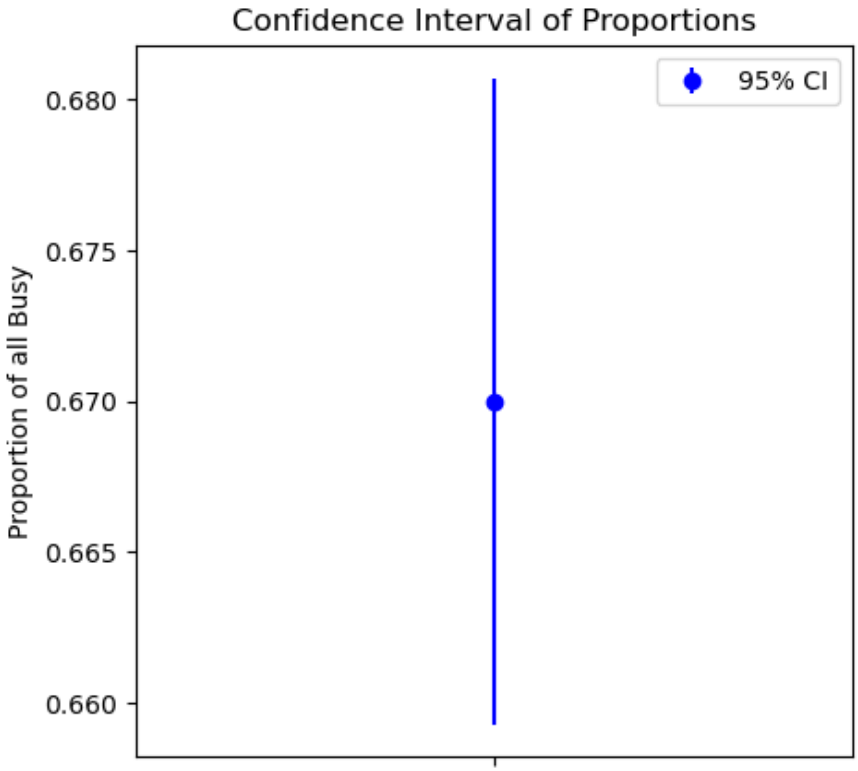
In [15]:

```
plt.figure(figsize=(10, 5))
plt.plot(df["workers"], df["mean"], marker="o")
plt.title("Number of Workers VS Proportion of all Busy")
plt.xlabel("Number of Workers")
plt.ylabel("Mean Proportion")
plt.show()
```



In [19]:

```
plt.figure(figsize=(5, 5))
plt.errorbar(x=[""], y=[mean], yerr=[1.96 * (stddev / np.sqrt(len(results)))], fmt="o",
color="blue", label="95% CI")
plt.title("Confidence Interval of Proportions")
plt.ylabel("Proportion of all Busy")
plt.legend()
plt.show()
```



In [ ]: