CZ4015: Simulation and Modelling
Assignment 1
Part II

Wong Yuh Sheng Reuben
U1820016H

9 April 2021

Contents

1 Introduction

1.1 Summary of Problem Statement

In this assignment, we have been given a description of a 40km highway equipped with 20 cell towers. Each of these cells are capable of handling 10 concurrent calls. Our objective, is then to study the number of calls blocked or dropped, given 2 channel allocation schemes.

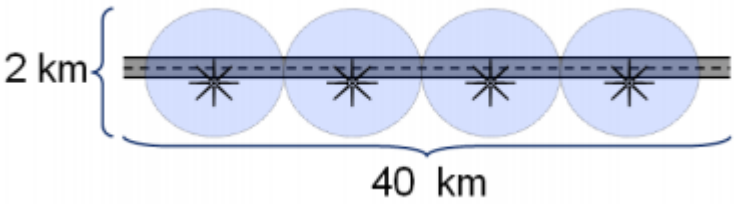


Figure 1: Visual Depiction of Problem Environment

1.2 Relationship Between Events with Event Graph

With reference to Figure 2 below, we can see the interaction between our 3 events. The first Call Initiation event is independent of all other events, and hence, is denoted as having an incoming zig-zag arrow. Based on the given information, we generate the random inter-arrival times of the calls to be initiated. This then implies that the Call Initiation event will trigger another Call Initiation event at a later time, and is demarcated with the self-pointing bold arrow.

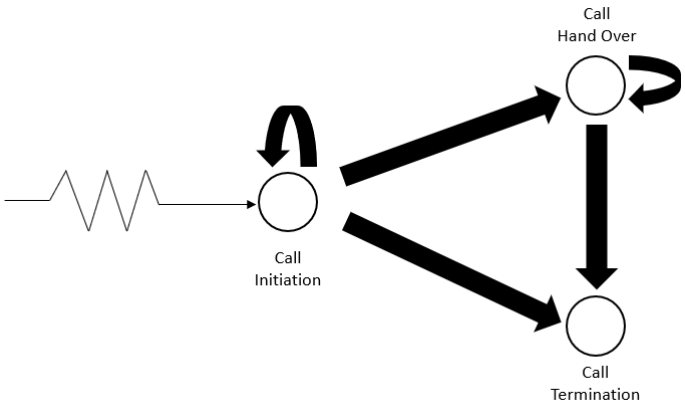


Figure 2: Event Graph

Having initiated a call, we could then end up with 2 possible events. We will first address call terminations, and the situations in which Call Termination would occur:

- Call Initiation that leads straight to Call Termination
  - Caller initiates a call, and then terminates the call within the same cell
  - Caller initiates a call in the last cell, and then leaves the highway
- Call Hand Over that leads to Call Termination
  - The call had been handed over to neighbouring cells, which then terminate within the cell, or when the caller has reached the end of the highway.

Next, we will discuss the circumstances under which a Call Hand Over would take place:

- Some time after (in simulation time) the Call Initiation, when the driver exits the cell in which the call was initiated and the call is handed over to the adjacent cell.
- The current call being referred to is one that has been handed over. Should the driver not terminate the call when moving over to the next cell, the call is handed over again. This implies that self-referring arrow in Call Hand Over, where one hand over triggers another at a later time.

Lastly, I believe it is important to mention that there is **no queuing system** in place for this problem setting. This means that calls that are dropped or blocked are not added to any queues, and that they are considered failed, and will **not reappear** again in this problem. A consequence of this would be that Call Terminations do not trigger a subsequent Call Initiation, since their call requests are not waiting to be accepted.

2 Objectives

Our objectives in this simulation would be to gauge if the company can meet its Quality of Service (QoS) requirements in terms of percentages of **(1) dropped** and **(2) blocked calls**. We then use the simulation to determine which of the following fixed channel allocation schemes results in the best performance:

- No channel reservation
- 9 channels are allocated to each cell for new calls and handovers and 1 channel is reserved for handovers when the other 9 channels are not available

3 Simulation

As mentioned above, we have 2 channel reservation schemes to investigate. This distinction will be implemented within the Cell Station object (discussed below in Section 3.1). In the no reservation scheme, the Cell Station object would have the number of available channels set to 10 as an attribute. Whereas in the channel reservation scheme, we would first specify the number of reserve channels. An attribute would be dedicated to hold the number of available reserve channels. Conversely, the number of normal channels, demarcated as available channels, would then be `10 - reserve_channels`. In our reservation scheme with 1 reserve channel, the number of available channels would then be set to 9.
As a consequence of this, there are separate sets of pseudocode catering to each scheme for the handover handling routine.

3.1 Entities and System Variables

In this problem, we have the callers and the cell stations as entities.
As for system variables, we have the following:

- Statistical Counters
  - Total number of calls
  - Number of dropped calls
  - Number of blocked calls
  - Clock
- State Variables
  - Cell station object, each with the number of available channels
  - Caller object, each with time of initiated call, speed, station the call was initiated, position of the caller, call duration and travelling direction.

3.2 Main Program

Algorithm 1: Main Program Logic

```
Function Init():
    Set: clock = 0
    Require: Global FEL = empty ordered list
    Require: Global Stations_Array[20]
    for i ← 0 to 19 do
        Stations_Array.append(CellStation(num_channels=10, station_id=i,
          num_reserve=0))                                  // Instantiate 20 CellStations
    end
    // Instantiate initial event list
    Random inter-arrival time, A_1 = random_time()
    First caller time, t_1 = 0 + A_1
    speed, station, position, duration, direction = Generate_Args()
    Instantiate first call initiation event, call_1 = CallInit(time=t_1, speed, station,
      position, duration, direction)
    // Insert the instantiated event into time-ordered FEL
    FEL.insert(call_1)
    // Zero stat counters
    Total number of calls = Number of dropped calls = Number of blocked calls = 0
    warmed_up = False
Function Main():
    Init()
    for step ← 0 to total_steps do
        if step > warmup_steps and not warmed_up then
            reset simulator              // zeroes stat counters upon steady state
            warmed_up = True
        end
        Event = FEL.dequeue()
        clock = Event.time
        switch Event Type do
            case Call Initialisation do
                CallInitHandler()
            end
            case Call Handover do
                CallHandoverHandler()
            end
            case Call Termination do
                CallTerminationHandler()
            end
        end
    end
    // Compile statistics at the end of the run
    return simulator.blocked / simulator.total, simulator.dropped / simulator.total
```

3.3 Helper Functions

The functionalities encapsulated in the functions below are necessary for all 3 event handling routines. Hence, they have been abstracted as functions here to reduce the complexities of the other algorithms.

Algorithm 2: Helper functions for updating state and checking bounds

Function `Generate_Args()`**:**

```
    // Quickly generate random values for event creation
    speed = random_speed()
    station = random_station()
    position = random_pos(station=st_1)
    duration = random_dur()
    direction = random_dir()
    return speed, station, position, duration, direction
```

Function `FreePrevStation(`*current_station, direction, used_reserved=False*`)`**:**

```
    prev_station_id = current_station.id - direction
    prev_station = Stations_Array[prev_station_id]
    if used_reserved then
        prev_station.reserve_available = True
    else
        prev_station.available_channels++
    end
```

Function `ComputeNextEvent(`*speed, position, duration, direction, current_station*`)`**:**

```
    total_distance = speed * duration
    absolute_position = position * current_station.range
    // Direction takes on values -1 for left and 1 for right
    end_position = absolute_position + direction * total_distance
    // Getting index of next station to retrieve from array
    next_station_id = current_station.id + direction
    // Note that stations share bounds at boundary
    boundary_index = max(0, direction)
    boundary_position = current_station.range[boundary_index]
    Distance between current position and next cell boundary, d =
     abs(absolute_position - boundary_position)
    time_to_next = d / speed
    remaining_duration = current_init.duration - time_to_next
    next_event_time = clock + time_to_next
    // NOTE: This has been changed to occur after time checking
    if end_position in current_station.range or next_station_id < 0
    or next_station_id > 19 or remaining_duration ≤ 0 then
        termination_time = clock + duration
        // null returns for consistency
        return ("Termination", termination_time, current_station, null, null, null)
    end
    return ("Handover", next_event_time, next_station_id, speed, remaining_duration,
     direction)
```

3.4 Call Intialisation Event Handler

Algorithm 3 below applies for both channel reservation schemes (with and without). Note that calls can only be blocked during call intialisations.

Algorithm 3: Call Initialisation Handler

```
Set: current_init ← Current Dequeued Call Initialisation Event
// Schedule next call to be initiated
Random inter-arrival time, A_t = random_time()
Next call time, t = clock + A_t
speed, station_id, position, duration, direction = Generate_Args()
Instantiate next Call Initialisation Event, call_init = CallInit(time=t, speed,
  station_id, position, duration, direction)
// Insert next call initialisation into FEL
FEL.insert(call_init)
current_station = Stations_Array[current_init.station]    // Acquire station from array
// NOTE: This is still valid under reservation scheme.  Since the number of available
   channels is set to 10 - the number of channels we use for reserved
if current_station.available_channels == 0 then
    Number of blocked calls++
else
    current_station.available_channels−−
    // Get relevant parameters from event object
    speed, position, duration, direction = current_init.get_params()
    event_type, event_time, station, speed, remaining_duration, direction =
       ComputeNextEvent(speed, position, duration, direction, current_station)
       // Proceed to Terminate
    if event_type == "Terminate" then
        Insantiate Call Termination Event, termination_event =
         CallTerminate(time=event_time, station=station)
        FEL.insert(termination_event)
    // Proceed to Handover
    else
        Instantiate Hand Over Event, handover_event =
         CallHandover(time=event_time, speed=speed, station=station,
         duration=remaining_duration, direction=direction)
        FEL.insert(handover_event)
    end
end
Total number of calls++
```

3.5 Call Handover Event Handler

While both algorithms 4 and 5 share many similarities, there are some core differences with how cell station channels are freed and checked.

For example, in algorithm 4, freeing a previous station simply increases the number of available channels. Meanwhile, in algorithm 5, we would check if a reserved channel was used or not. Conditional statements perform this check and pass the necessary flags into the event object, such that the routine will be able to make the appropriate changes in the state variables.

Note that calls can only be dropped during call handovers.

Algorithm 4: Call Handover Handler without Channel Reservation

```
Set: current_handover ← Current Dequeued Call Handover Event
// Extract information from Event object
speed, station_id, position, duration, direction = current_handover.get_params()
current_station = Stations_Array[station_id]          // Acquire station from array
FreePrevStation(current_station, direction)
if current_station.available_channels == 0 then
    Number of dropped calls++
else
    current_station.available_channels−−
    // Get relevant parameters from event object
    speed, position, duration, direction = current_handover.get_params()
    event_type, event_time, station, speed, remaining_duration, direction =
        ComputeNextEvent(speed, position, duration, direction, current_station)
        // Proceed to Terminate
    if event_type == "Terminate" then
        Insantiate Call Termination Event, termination_event =
         CallTerminate(time=event_time, station=station)
        FEL.insert(termination_event)
    // Proceed to Handover
    else
        Instantiate Hand Over Event, handover_event =
         CallHandover(time=event_time, speed=speed, station=station,
         duration=remaining_duration, direction=direction)
        FEL.insert(handover_event)
    end
end
```

Algorithm 5: Modified Call Handover Handler with Channel Reservation

```
Set: current_handover ← Current Dequeued Call Handover Event
// Extract information from Event object
speed, station_id, position, duration, direction, used_reserved =
 current_handover.get_params()
current_station = Stations_Array[station_id]          // Acquire station from array
// Frees a normal or reserved channel appropriately
FreePrevStation(current_station, direction, used_reserved)
if current_station.available_channels == 0 and not current_station.reserved_available()
 then
    Number of dropped calls++
// One of the types of channels are available, find out which
else
    if current_station.available_channels == 0 then
        // NOTE: Decrements number of available reserve channels
        current_station.free_reserved−−
        using_reserved = True
    else
        current_station.available_channels−−              // Use normal channel
        using_reserved = False
    end
    // Get relevant parameters from event object
    speed, position, duration, direction = current_handover.get_params()
    event_type, event_time, station, speed, remaining_duration, direction =
       ComputeNextEvent(speed, position, duration, direction, current_station)
       // Proceed to Terminate
    if event_type == "Terminate" then
        Insantiate Call Termination Event, termination_event =
         CallTerminate(time=event_time, station=station,
         used_reserved=using_reserved)
        FEL.insert(termination_event)
    // Proceed to Handover
    else
        Instantiate Hand Over Event, handover_event =
         CallHandover(time=event_time, speed=speed, station=station,
         duration=remaining_duration, direction=direction,
         used_reserved=using_reserved)
        FEL.insert(handover_event)
    end
end
```

3.6 Call Termination Event Handler

The only role of the termination event handler is to clean up the channel used by a previous event, and release the resources of the cell station. The distinction between the handling for the channel reservation schemes can be more closely inspected within the **FreePrevStation** function. Thus, the execution of Call Termination under both schemes share the same

algorithm below (Algorithm 6).

Algorithm 6: Call Termination Handler

Set: current_terminate ← Current Dequeued Call Termination Event

```
// Note that used_reserved is by default false in the No Reservation Case
```

time, station_id, used_reserved = current_terminate.get_params()
current_station = Stations_Array[station_id]
`FreePrevStation(`*current_station, direction, used_reserved*`)`

4 Input Analysis

Random variates drive our simulation. In order for our simulation to be used to study the real system, our generated random values must be distributed approximately the same way as such values in the real system. Hence, in this section, we will **(1)** study the collected real world data, **(2)** propose theoretical distributions that fit the data, and finally **(3)** test the goodness of fit of these distributions to the data.
All codes for input analysis can be found in `data_exploration.ipynb`.

4.1 Inter-Arrivals

The data provided us with the arrival times of the calls. Hence, some processing had to be done to determine the inter-arrival times instead, by subtracting the arrival time of the preceding call from the current call. We can then plot the distributions of the inter-arrival times, as in Figure 3.

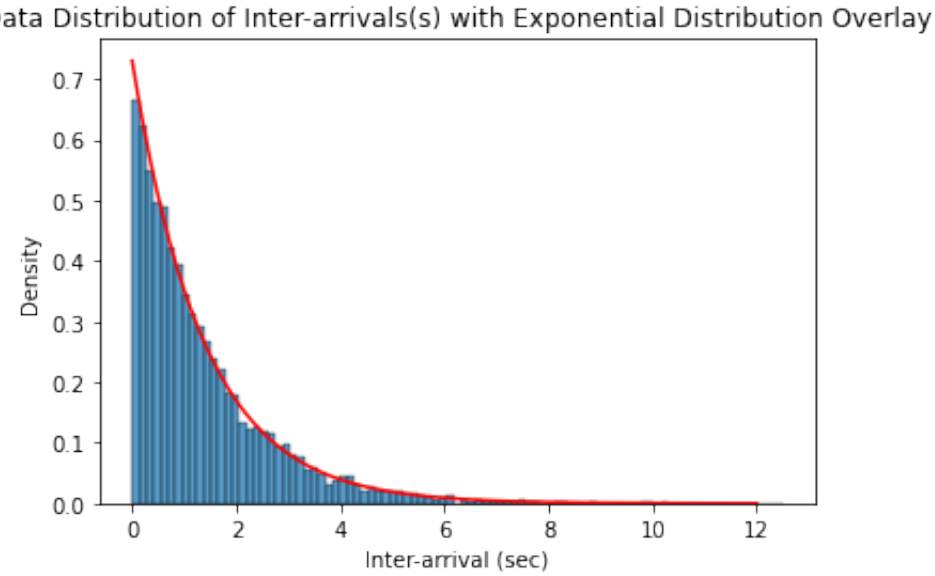


Figure 3: Distribution of inter-arrival times of calls

As we observe the data to seemingly be distribution in an exponential manner, we then theorise that it is distributed as such. We have derived in the lectures that for an exponential distribution parameterised as in Equation (1),

$$f_\beta(x) = \frac{1}{\beta} e^{-\frac{x}{\beta}}, \text{ where } x \geq 0 \tag{1}$$

the value of β, also known as the scale, that maximises the likelihood of our observed data, our Maximum Likelihood Estimator, is simply the sample mean. Hence, we hypothesise that our data is drawn from an exponential distribution parameterised by $\beta = 1.369817$, i.e. $InterArrivals \sim Exponential(1.369817)$.

We then overlay this distribution over our plot as in Figure 3, and it seems to be a good fit to our data. However, it is insufficient to simply "eye-ball" a fit.
Hence, we employ the Kolmogorov–Smirnov test (KS-test) to test goodness of fit. The KS-test quantifies the distance between 2 cumulative distribution functions (CDFs) by looking at the maximal distance between the 2 distributions.
We perform the KS-test with the help of the `scipy.stats` module. The results are shown in Table 1.

D-value	p-value
0.010015330445135917	0.26664034156645433

Table 1: Results of KS-test for Inter-Arrival

We set our significance level to be 0.05 and our null hypothesis, H_0, to be that our theoretical distribution fits the data. As we can see from Table 1, our p-value is greater than the level of significance, and thus, we do not reject our null hypothesis. We then conclude that our proposed distribution fits the data well.

4.2 Velocity

In order for units to be consistent with other variables in our simulation, we first convert all values from km/h to km/s. We then follow a procedure similar to section 4.1, by first plotting the data distribution, hypothesising a fit to the data, and testing the goodness of fit.

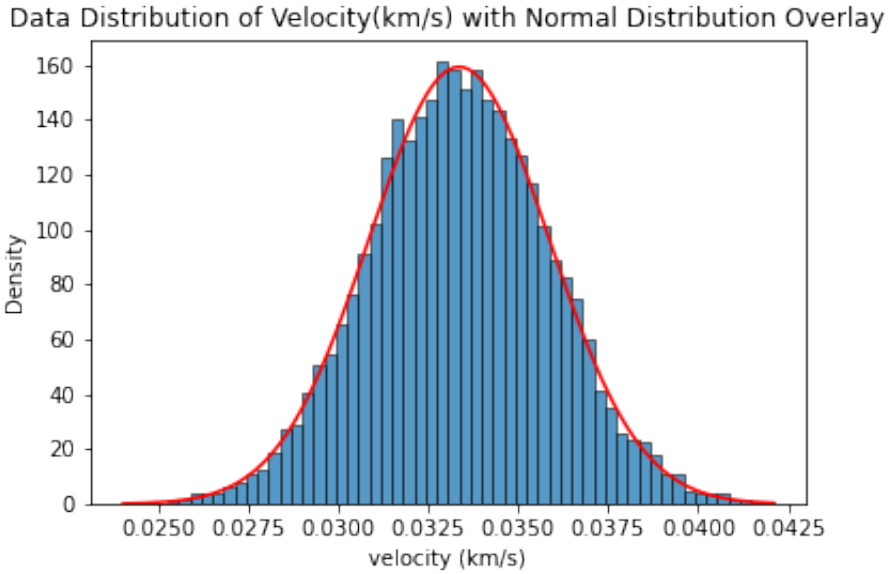


Figure 4: Distribution of velocity

We plot the distribution of velocity as in Figure 4, and it appears to take on a normal distribution, parameterised by μ and σ as in Equation (2).

$$f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{2}$$

From my tutorial presentation, we found that MLEs of a normal distribution are the sample mean, and population variance. Those parameters are then used to obtain the normal distribution shown by the red curve in Figure 4.
Lastly, we employ the KS-test once again to test goodness of fit, where our null hypothesis H_0 is that our normal distribution fits the observed data. The results are shown in Table 2.

D-value	p-value
0.006439789372512217	0.7987384412525099

Table 2: Results of KS-test for Velocity

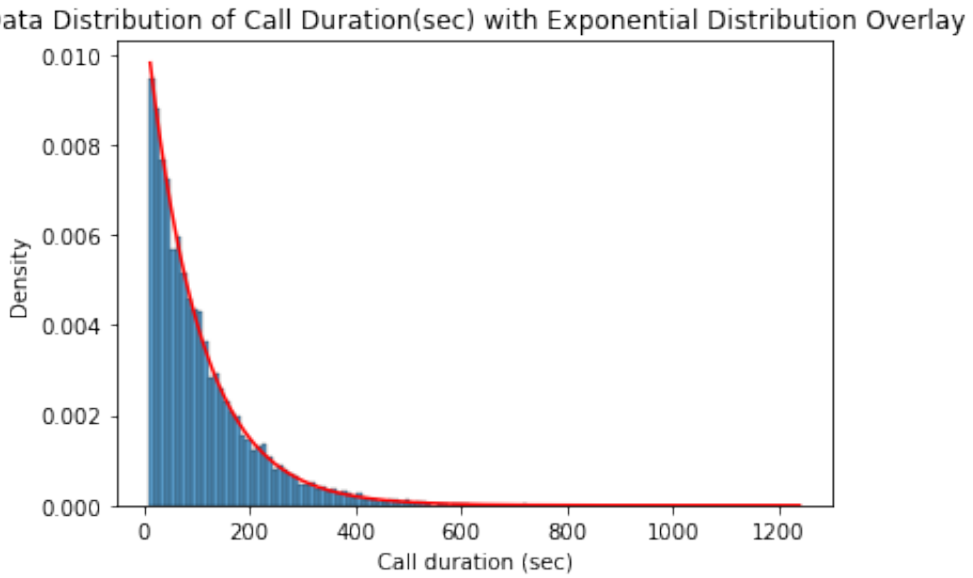


Figure 5: Distribution of call duration

The p-value obtained is greater than our level of significance of 0.05, and thus, we do not reject our null hypothesis and our distribution fits the data well.

4.3 Call Duration

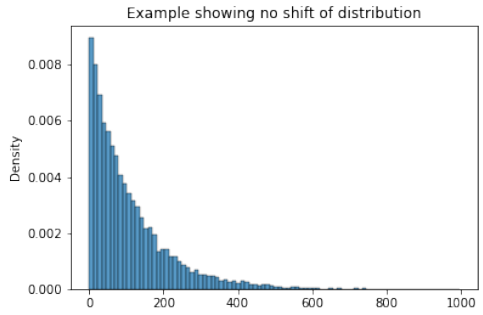
We begin by plotting the distributions of call duration in Figure 5.
It is important to note that in the given data, the minimum value of call duration is not 0, and the distribution is in fact shifted to the right by around 10. Hence, directly using the sample mean would be parameterising the distribution wrongly, and we would need to subtract the mean by this offset before utilising it. For convenience, we can use `scipy.stats` to fit the data to an exponential distribution. This would parameterise the exponential distribution with the appropriate scale, β, as well as shift the distribution to the right by the required offset. This distribution is shown in red in Figure 5.
We then perform the KS-test, with results summarised in Table 3.

	D-value	p-value
Directly using sample mean	0.08722013653673563	1.2257712860415087e-66
Correctly parameterising	0.005854123916584519	0.8809285916372338

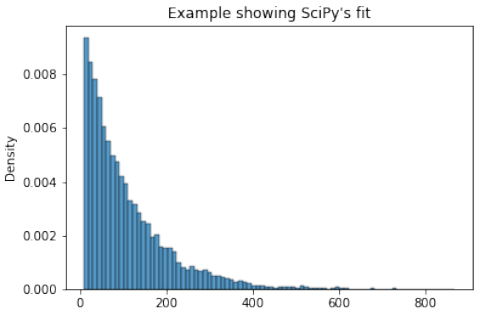
Table 3: Results of KS-test for Call Duration

Thus, we can conclude that at our level of significance of 0.05, we do not reject our null hypothesis and the distribution fits the data well. It is interesting to note, that while directly using the sample mean fails the KS-test, plotting that distribution over the data also seems to employ a good fit, and is actually not easily discernable, just by looking at it, that it fails the KS-test. The reason for this is that the KS-test looks at the maximum distance between CDFs. When we do not shift our hypothesised distribution to the right, in the region of 0 to 10, the observed data has not accumulated any probability in its CDF, while the hypothesised exponential distribution has, leading to a "large distance" between the 2 CDFs.

To further emphasise this point, Figure 6 shows that the 2 distributions look fairly similar. Please see `data_exploration.ipynb` for details on how the data used in the figure were generated.



(a) Exponential distribution directly using sample mean



(b) Shifted distribution fit with scipy

Figure 6: Example showcasing similarity in fits

4.4 Base Stations

We first plot the distribution of base stations for incoming calls in Figure 7.

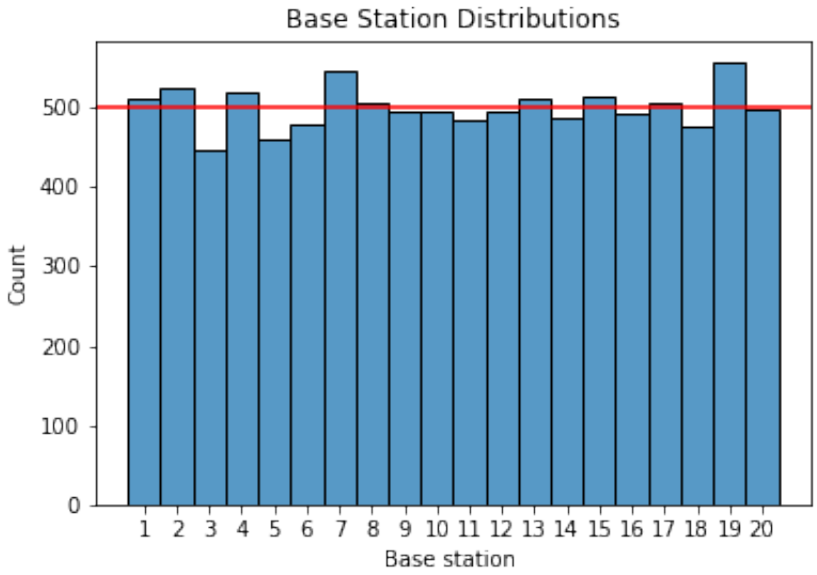


Figure 7: Distribution of base stations

Figure 7 implies that the data is discretely uniformly distributed. If that were the case, the expected number of incoming calls in each station should be 500, since we have 10000 data points. We can easily test the goodness of fit using the Chi-squared test. To perform the test, we need to know the number of calls arriving in each station, as well as the expected number of calls for each station (500 for all stations). With this information, we easily obtain the results in Table 4.

Statistic	**p-value**
25.656000000000002	0.14006290765463547

Table 4: Results of Chi-squared test for Base Stations

Once again, we can see that the p-value obtained is greater than our level of significance of 0.05. Hence, our null hypothesis is not rejected, and the observed data fits our discrete uniform distribution well.

5 Running of Experiments

From this report, you will gather that in analysis, we look at the number of steps, i.e. the number of events dequeued from the FEL instead of the actual simulation clock time. This is done because it simply makes things easier to analyse, when we can count the length of simulation by number of events instead of some arbitrary time.

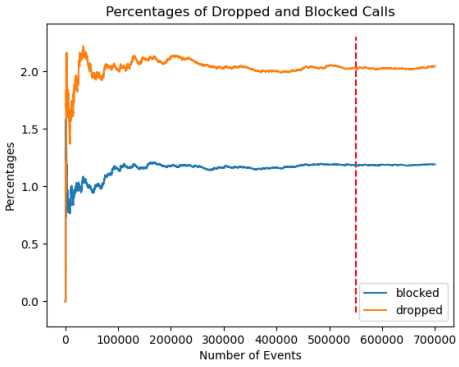
Within the repository, there is a `README.md` file with instructions on running the experiments. Of note, there is a `serial.py` file for running the independent repetitions serially. This is slow, but useful for debugging. To speed up experimentation, `parallel.py` makes use of multiprocessing to run multiple experiments in parallel, before collating the results.

You may also easily analyse the outputs yourself with `analyse.py`. Just use the same arguments that were used to run the experiment above. For detailed instructions, see the **results directory of the github repository here**.

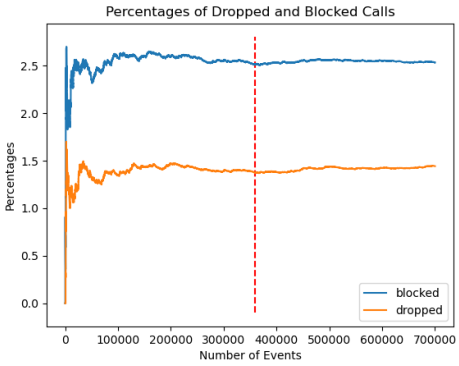
6 Output Analysis

6.1 Warmup Period

As our simulation is a steady-state simulation, we need to "warm up" our simulation, such that the statistical counters collected from the runs are unaffected by the initial conditions. Hence, once our simulation run is warmed up, we then zero our statistical counters and then start to collect the relevant information, discarding the original information collected.



(a) Warmup Period with no Channel Reservation



(b) Warmup Period with 1 Channel Reserved

Figure 8: Warmup Period for Various Reservation Schemes

We "eye-ball" a cut-off for the warmup period of each channel reservation scheme, where the percentages of dropped and blocked calls remain fairly constant. With no channel reservation, we take the number of warmup steps to be 600,000 events, and gather information for another 100,000 steps. This brings us to a total of 700,000 steps. With one channel being reserved, we take the number of warmup steps to be 360,000, and we once again gather information for another 100,000 steps, for a total of 460,000 steps.

6.2 Results

6.2.1 No Channel Reservation

Results	**Repetitions** (n)	**Blocked**	**Dropped**
μ	16	1.1885520469462807	1.9537040602754647
σ^2	16	0.00393447560103555	0.01835203419279192
σ	16	0.06272539837287246	0.1354696799759707
δ	16	0.03342	0.07219

Table 5: Results of Experiment for No Channel Reservation

From Table 5, we can see that the service provider will not meet its QoS requirements within our 95% confidence interval. To further convince ourselves of the results, we attempt to shrink the confidence interval by performing more repetitions. In Table 6, we see the results for a hundred repetitions, where statistical analysis chooses the confidence interval to also be 95%. While the block rate is within the QoS, we see that the drop rate still is unable to meet QoS requirements at the 95% confidence interval. Of note, we did indeed **manage to shrink our confidence interval**.

Results	**Repetitions** (n)	**Blocked**	**Dropped**
μ	100	1.2207688820987297	1.973504185195326
σ^2	100	0.007937715597402818	0.01623915536838635
σ	100	0.0890938583596132	0.12743294459591817
δ	100	0.01767	0.02529

Table 6: Results of Experiment for No Channel Reservation

6.2.2 1 Channel Reserved

Results	**Repetitions** (n)	**Blocked**	**Dropped**
μ	16	2.5663864457680967	1.419992003743566
σ^2	16	0.008933632757054333	0.011791589673627364
σ	16	0.09451789649084628	0.10858908634677503
δ	16	0.05037	0.05786

Table 7: Results of Experiment for 1 Channel Reserved

From Table 7, we see that the service provider is still unable to meet its QoS requirements within our 95% confidence interval. As expected, when reserving 1 channel for handovers, our drop rate decreases while our block rate increases. Similar to Section 6.2.1, we also perform 100 repetitions to narrow our confidence interval. The results are presented in Table 8. As expected, our confidence interval has narrowed. The conclusion drawn remains the same, that while the drop rate has improved, now both block and drop rates are unable to meet QoS requirements when compared to the no channel reservation scheme. These results are well within our expectations, that as we reserve more channels for handovers, calls are less likely to be dropped. However, in doing so, we are reducing the number of channels that are able to accept new calls, and thus would increase the drop rate. These expectations are realised in our empirical findings.

Results	Repetitions (n)	Blocked	Dropped
μ	100	2.568590667723725	1.4290350111574914
σ^2	100	0.01674010944010588	0.010973416602996557
σ	100	0.1293835748466778	0.1047540767846128
δ	100	0.02567	0.02079

Table 8: Results of Experiment for 1 Channel Reserved with 100 Repetitions

6.3 2 Channels Reserved

For a better picture of the behaviour of channel reservation schemes, we also perform a study of a system with 2 channels being reserved for handovers. We first begin by determining the warmup period, presenting in Figure 9. From the analysis, we determine the warmup period to be 350,000 steps, and thereafter, collect results for another 100,000 steps for a total of 450,000 steps.

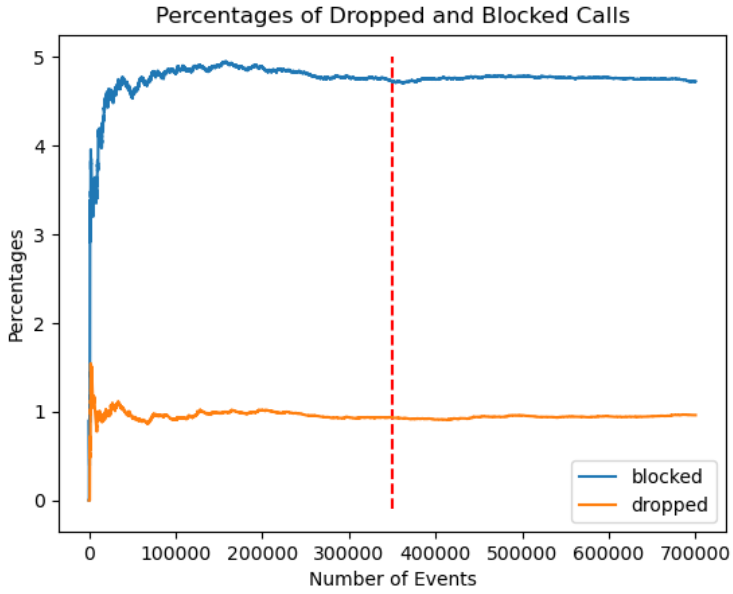


Figure 9: Warmup Analysis for 2 Channels Reserved

We present the results for the simulation study in Table 9. Drop rate has decreased as expected, while block rate has increased substantially, though an increase was also expected. At last, with 2 channels reserved for handovers, at our 95% confidence interval, the drop rate is finally able to meet QoS requirements of $< 1\%$. However, this comes at a cost of severely exceeding our QoS requirements for the block rate of 2%. Ultimately, this scheme is also unable to meet QoS requirements.

Results	Repetitions (n)	Blocked	Dropped
μ	20	4.769406695018381	0.9563043152100997
σ^2	20	0.028584430769381657	0.006588015530390817
σ	20	0.16906930759124097	0.08116659122071604
δ	20	0.07913	0.03799

Table 9: Results of Experiment for 2 Channels Reserved

6.4 Comment on System and Recommendation

From our study, we have thus determined that the current system has no means of meeting the QoS systems just by simply changing channel reservation schemes. The only way then, would be to increase the number of channels in each cell station. Therefore, my recommendation would be to first experiment with increasing the number of channels in each cell station in conjunction with various channel reservation schemes to find the optimal system design.

7 Conclusion

This report has demonstrated the key logic for the main driver function of the simulation, together with important helper functions that encapsulate functionalities that are constantly reused. With strict adherence to the flow of events observed in Figure 2, Call Initialisation trigger subsequent Call Initialisations, as well as other Call Handovers or Call Terminations. Likewise, a call may be repeatedly handed over, but only terminated once.

In order to determine the proportion of dropped and blocked calls accurately, all calls that attempt initialisation are noted of, and each Call Initialisation event unconditionally increases the total number of calls. Successful initialised calls are allocated resources. These resources are then successfully de-allocated (without being blocked or dropped) in the Handover and Termination routines. Calls can only be blocked during initialisations and dropped during handovers.

We expanded this report by analysing the inputs to our simulation, ensuring that our generated random variates come from distributions matching real-world data by conducting the necessary statistical tests. We then described how to run the code, and provide the relevant output analysis, such as how the warmup period was determined, before finally providing the results from the experiments.

Finally, we determine that the system is unable to meet its QoS requirements no matter what just by changing the channel reservation scheme, and make the appropriate recommendation in Section 6.4.

Code for the experiments can be found here.