COMPENG 4DK4 LAB3

Richard Qiu – 400318681 – Group 66

# Experiment

## 2.

The obtained table and graph is shown below.



While the data inside this table is the blocking probability.



The offered load is calculated by the product of the Call\_ARRIVALRATE and MEAN\_CALL\_DURATION, in this experiment. I perform 10 different simulations with every time increase the Erlang load by 1 by increasing the Call\_ARRIVALRATE in the simparameters.h file. And I made a for loop to increase the number of channels from 1 to 20 as shown below.

A computer code with white text

Description automatically generated

Every time it will store the blocking probability in a text file with the corresponding number of channels it used. My student number is used as the random seed here.

Below is the python program I wrote to compute the Erlang B formula.

offerd\_load = 10

num\_channel = 1

def factorial\_iterative(n):

result = 1

for i in range(1, n + 1):

result \*= i

return result

def sigma(first,last,const):

sum =0.0

for i in range(first,last+1):

# print(i)

sum += (float(pow(const,i))/float(factorial\_iterative(i)))

return sum

with open('example.txt', 'w') as file:

for i in range(num\_channel,21):

Pb = (float(pow(offerd\_load,i))/float(factorial\_iterative(i)))/float(sigma(0,i,offerd\_load))

# print(Pb, "num\_channel is ", i, "\n")

print(f"{Pb:.10f} num\_channel is {i:.10f}\n")

file.write(f"{Pb:.10f} num\_channel is {i:.10f}\n")

The computation result table is shown below.



This result is very similar to the simulation result and we can also prove the computation by using the online Erlang B calculator using this link. {<http://www.site2241.net/erlang.htm>}

## 3.



By utilizing the experiment2 graph, we can set the maximum probability equal to 0.01 and find the corresponding required #channels. By observing this graph, we can conclude the following result.



The graph is generated below.



As we can seen from this graph, by adding the red trend line to show the relation between the erlang load and number of channels, we can find the slope is around 1.5, which is greater than the linear increase of 1. That proves when we want to improve the blocking probability, we can increase the erlang load by increasing the erlang load. This is also called trunking efficiency or multiplexing gain.

## 4.

To simulate the cellular system where assume that callers never hang up when channels are found busy, instead they will wait until their call can be served. We can add a queue to the part 2 code and will check if this queue is empty for each transmission.

I use the provided simlib file to generate a queue as shown below.

data.buffer = fifoqueue\_new();

And I made the below changes when the caller finds there are no available channels, they will be put on the queue and wait for connection later. While the call\_count is used to count how many callers don’t need to wait less than t seconds to get an available channel, so when the channel is found free in the first transmission, it will automatically add one.

if((free\_channel = get\_free\_channel(simulation\_run)) != NULL) {

**/\* Yes, we found one. Allocate some memory and start the call. \*/**

sim\_data->call\_count++;

**/\* Place the call in the free channel and schedule its**

**departure. \*/**

server\_put(free\_channel, (void\*) new\_call);

new\_call->channel = free\_channel;

schedule\_end\_call\_on\_channel\_event(simulation\_run,

now + new\_call->call\_duration,

(void \*) free\_channel);

} else {

**/\* No free channel was found. The call is blocked. \*/**

**//change this so now it will add to the queue and wait until their call can be served**

**// sim\_data->blocked\_call\_count++;**

fifoqueue\_put(sim\_data->buffer, (void \*)new\_call);

}

And in the end\_call\_on\_channel\_event function, I add the code below.

if(fifoqueue\_size(sim\_data->buffer) > 0){

next\_call = (Call\_Ptr)fifoqueue\_get(sim\_data->buffer);

if (now - next\_call->arrive\_time < t\_thres)

{

**// printf("the delay is %f\n",now - this\_call->arrive\_time);**

sim\_data->call\_count++;

}

server\_put(channel, (void \*)next\_call);

next\_call->channel = channel;

schedule\_end\_call\_on\_channel\_event(simulation\_run,

now + next\_call->call\_duration,

(void \*) channel);

}

This will check if there are any callers waiting in the queue and it will schedule those callers end time. It will also check the delay of this wait call to determine if it is over the t seconds threshold. The full code is attached in the file.

By running these simulations different times with different system parameters, we want to find the probability that a call has to wait less than t seconds by comparing the computation results.

Below is the experimental simulation table, as we can seen below, A is the offered load, h is the average call holding time and l is the call arrival rate.



We can see from this graph, when the A is the same, when the average call holding time is less, the probability that a call has to wait less than t seconds is greater. This is reasonable as we can think caller will take less time to occupy the channels, so more channels will be available for the following callers to use. And we can also observe that when the t is smaller, the W(t) will become smaller because it will decrease our condition which could result more callers will have to wait more time.

And to compare with the computation result, I write a python program as shown below.

import math

offerd\_load = 2

num\_channel = 3

t = 60

h=30

def factorial\_iterative(n):

result = 1

for i in range(1, n + 1):

result \*= i

return result

def sigma(first,last,const):

sum =0.0

for i in range(first,last):

**# print(i)**

sum += (float(pow(const,i))/float(factorial\_iterative(i)))

return sum

with open('example.txt', 'w') as file:

for i in range(num\_channel,21):

a = float(pow(offerd\_load,i))/float(factorial\_iterative(i))

p=offerd\_load/num\_channel

sig\_ma = float(sigma(0,i,offerd\_load))

**# print(p, sig\_ma, a)**

Pb = (a)/(a+(1-p)\*sig\_ma)

**# print(Pb, "num\_channel is ", i, "\n")**

exp = math.exp(-(num\_channel-offerd\_load)\*t/h)

W\_t = 1-Pb\*exp

print("W\_t is", W\_t, "the t is:", t)

print(f"{Pb:.10f} num\_channel is {i:.10f}\n")

file.write(f"W\_t is {W\_t:.10f} num\_channel is {i:.10f} \n")

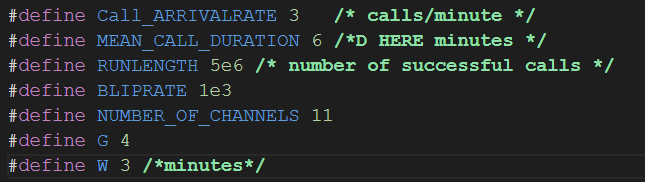
This will get the result of the W(t) formula. The result is shown below.

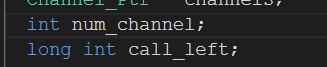


And by comparing both tables, we can prove our simulation is correct.

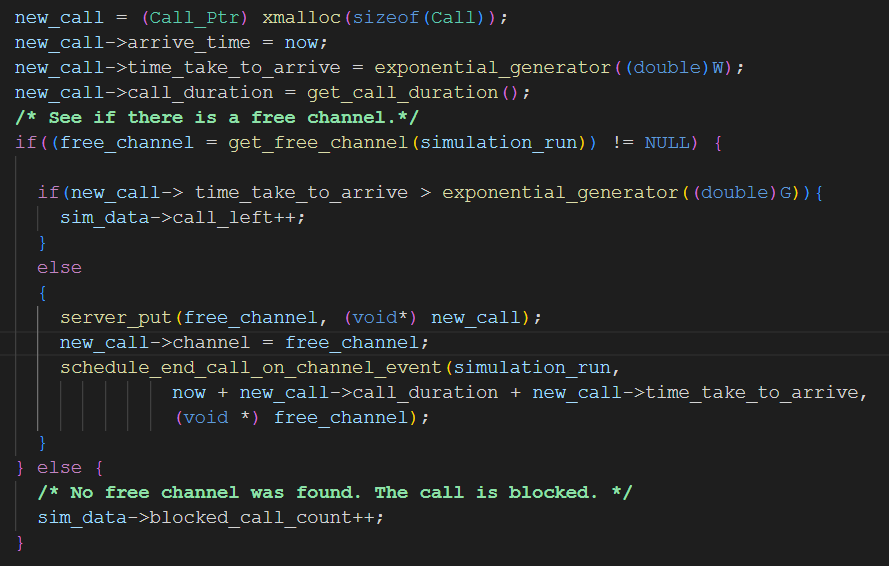
## 5.

To simulate this experiment, we can use the code from part 2 while the num of channel will be the N taxis, the customers calls will be the same as the schedule call arrival event, we can define a new W minute in the simparam file which is the time it takes to arrive at the customer, the D will be just the Mean\_call\_duration which is the time to transport the customer. And we will also define a new G which is the customer gives up waiting time. A new call\_left object will be created in the simulation\_run\_data to record the number of give up customers. The code is shown below.





And inside the call\_arrival\_event function, the following code is modified. It will use the exponential\_generator to get the time it takes to arrive the customer location and also if the channel is free to server a customer, it will compare the take to arrive customer time with the customer give up time. If it takes too long (greater than give up time) to arrive to the customer. The customer will leave and increment the call\_left counter. And also we need to add the time it takes to arrive as a parameter to the schedule end event function because it will be a whole cycle to server one customer.



After running several simulations. Below is the initial result.



As we can see the blocking probability will be decrease when the number of taxis increase. But the probability of a taxi arriving to find that a customer has left has no effects with the increase of the taxis.

Then I increase the G which is the give up time of a customer, as we can seen from the below table, the blocking P still show the same trend as decreasing and also by comparing this table with the first table. We can know the taxi P is also decreasing which is reasonable since we increase the give up time of the customer.



Then I set G back to 4 and decrease the W (times takes to arrive to customer). By comparing the table below with the first table, we can observe the probability of a taxi arriving to find that a customer has left is decreasing as well which is also reasonably as we can think the taxis will take less time to arrive the customer so the customer won’t leave.



Finally, I set back the W to 3 and increase the D(time to transport) to 6. By comparing the below table with the first one, we can see the customer giving up probability stays the same, but the blocking probability with D=6 increases. This is still reasonably because since the taxis will take more time to server the customer, the number of available taxis will decrease when a customer is called which could increase the overall blocking probability.

