COMPENG 4DK4 LAB3

Richard Qiu – 400318681 – Group 66

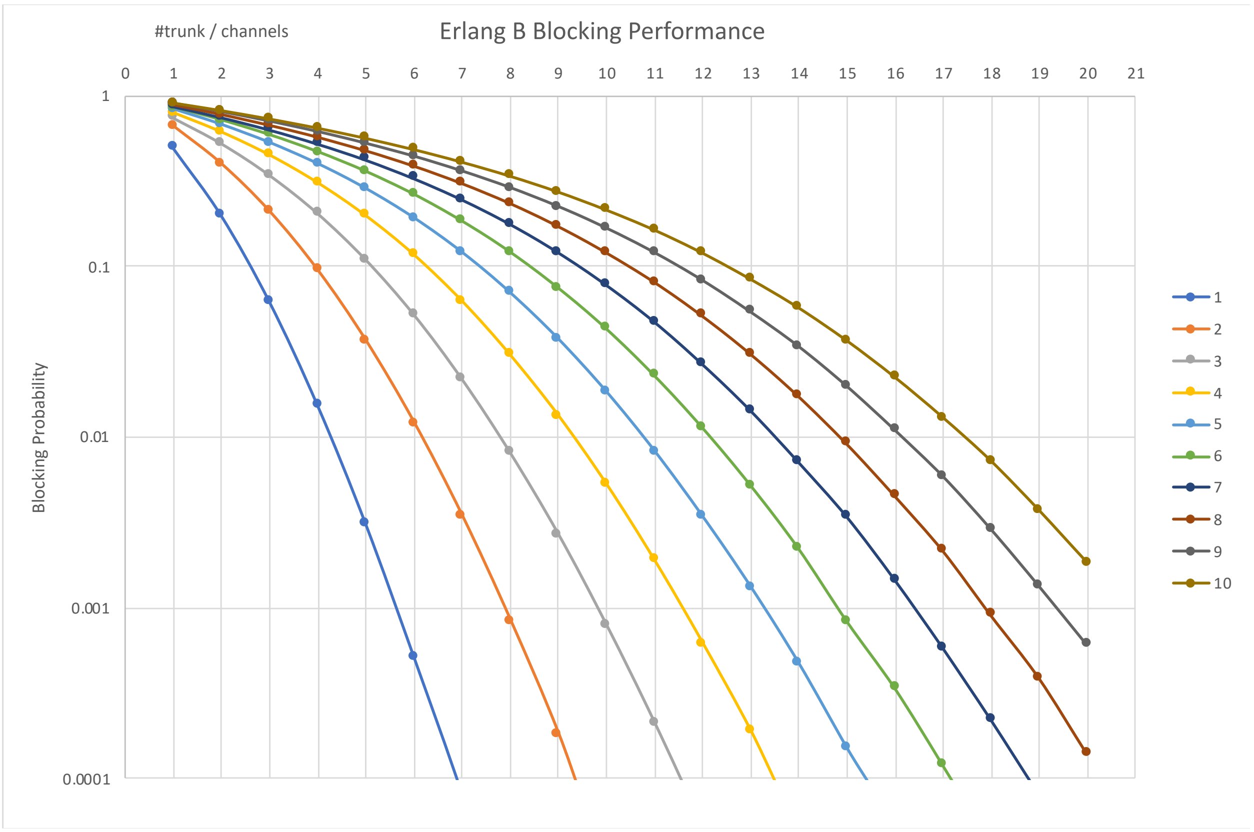
Experiment

2.

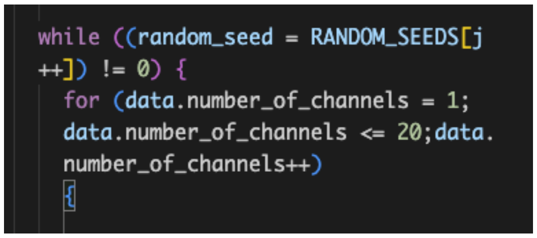
The obtained table and graph is shown below.

Offered load / # Channels	1	2	3	4	5	6	7	8	9	10
1	0.50009	0.66671	0.7503	0.79992	0.83331	0.85726	0.87509	0.88895	0.90004	0.9091
2	0.20023	0.39998	0.52919	0.61528	0.67566	0.72007	0.75397	0.78059	0.80207	0.81981
3	0.06256	0.21043	0.34573	0.4508	0.5296	0.59041	0.63773	0.67538	0.7064	0.73221
4	0.01539	0.09515	0.20586	0.31049	0.39826	0.46959	0.52768	0.57461	0.61373	0.64683
5	0.00308	0.03665	0.11012	0.19889	0.28509	0.36014	0.42485	0.47864	0.52501	0.56368
6	0.00051	0.01212	0.05233	0.11723	0.1919	0.26475	0.33085	0.38948	0.44092	0.48431
7	0.00008	0.00344	0.02184	0.06276	0.12057	0.18446	0.24868	0.30809	0.36131	0.4087
8	0.00001	0.00084	0.00809	0.03032	0.06999	0.12158	0.17858	0.23539	0.28915	0.33823
9	0	0.00018	0.0027	0.01341	0.03748	0.07495	0.12199	0.1728	0.22444	0.27309
10	0	0.00003	0.00079	0.0053	0.01837	0.0432	0.07863	0.12116	0.16781	0.21447
11	0	0.00001	0.00021	0.0019	0.00826	0.02301	0.04775	0.08134	0.12073	0.16341
12	0	0	0.00005	0.00062	0.00342	0.01139	0.02697	0.05158	0.08303	0.11975
13	0	0	0.00001	0.00019	0.00132	0.00524	0.0144	0.03089	0.05425	0.08407
14	0	0	0	0.00005	0.00047	0.00223	0.00716	0.01741	0.03407	0.05713
15	0	0	0	0.00002	0.00015	0.00084	0.00343	0.00919	0.01997	0.03666
16	0	0	0	0	0.00005	0.00034	0.00146	0.00455	0.01105	0.02239
17	0	0	0	0	0.00001	0.00012	0.00058	0.00217	0.00593	0.013
18	0	0	0	0	0.00001	0.00004	0.00022	0.00091	0.00291	0.00723
19	0	0	0	0	0	0.00001	0.00008	0.00039	0.00135	0.00371
20	0	0	0	0	0	0.00001	0.00003	0.00014	0.00061	0.00183

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.



```
while ((random_seed = RANDOM_SEEDS[j
++]) != 0) {
  for (data.number_of_channels = 1;
  data.number_of_channels <= 20;data.
  number_of_channels++)
  {
```

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.

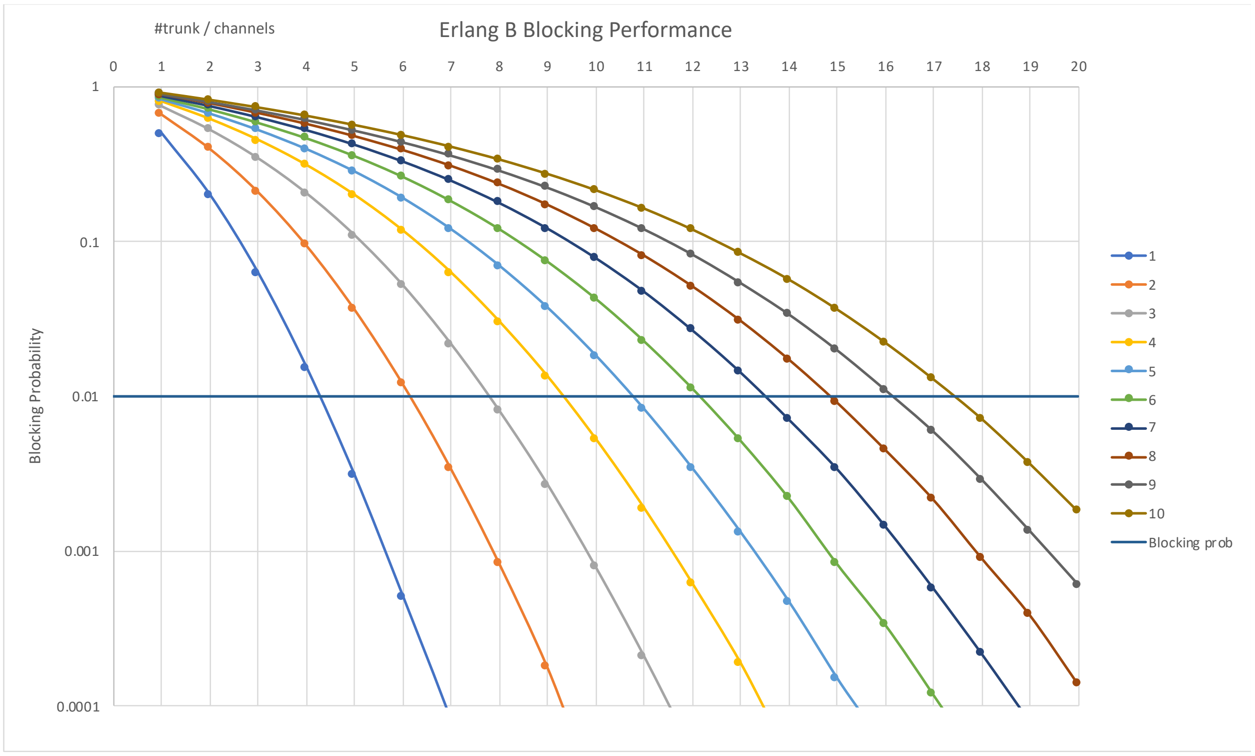
```
offerd_load = 10
num_channel = 1

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.

Offered load / # Channels	1	2	3	4	5	6	7	8	9	10
1	0.50000	0.66667	0.75000	0.80000	0.83333	0.85714	0.87500	0.88889	0.90000	0.90909
2	0.20000	0.40000	0.52941	0.61538	0.67568	0.72000	0.75385	0.78049	0.80198	0.81967
3	0.06250	0.21053	0.34615	0.45070	0.52966	0.59016	0.63755	0.67546	0.70640	0.73206
4	0.01538	0.09524	0.20611	0.31068	0.39834	0.46957	0.52734	0.57464	0.61381	0.64666
5	0.00307	0.03670	0.11005	0.19907	0.28487	0.36040	0.42472	0.47901	0.52491	0.56395
6	0.00051	0.01208	0.05216	0.11716	0.19185	0.26492	0.33133	0.38975	0.44052	0.48451
7	0.00007	0.00344	0.02186	0.06275	0.12052	0.18505	0.24887	0.30816	0.36158	0.40904
8	0.00001	0.00086	0.00813	0.03042	0.07005	0.12188	0.17882	0.23557	0.28916	0.33832
9	0.00000	0.00019	0.00270	0.01334	0.03746	0.07514	0.12210	0.17314	0.22430	0.27321
10	0.00000	0.00004	0.00081	0.00531	0.01838	0.04314	0.07874	0.12166	0.16796	0.21458
11	0.00000	0.00001	0.00022	0.00193	0.00829	0.02299	0.04772	0.08129	0.12082	0.16323
12	0.00000	0.00000	0.00006	0.00064	0.00344	0.01136	0.02708	0.05141	0.08309	0.11974
13	0.00000	0.00000	0.00001	0.00020	0.00132	0.00522	0.01437	0.03066	0.05439	0.08434
14	0.00000	0.00000	0.00000	0.00006	0.00047	0.00223	0.00713	0.01722	0.03379	0.05682
15	0.00000	0.00000	0.00000	0.00002	0.00016	0.00089	0.00332	0.00910	0.01987	0.03650
16	0.00000	0.00000	0.00000	0.00000	0.00005	0.00033	0.00145	0.00453	0.01105	0.02230
17	0.00000	0.00000	0.00000	0.00000	0.00001	0.00012	0.00060	0.00213	0.00582	0.01295
18	0.00000	0.00000	0.00000	0.00000	0.00000	0.00004	0.00023	0.00094	0.00290	0.00714
19	0.00000	0.00000	0.00000	0.00000	0.00000	0.00001	0.00009	0.00040	0.00137	0.00375
20	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00003	0.00016	0.00062	0.00187

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}

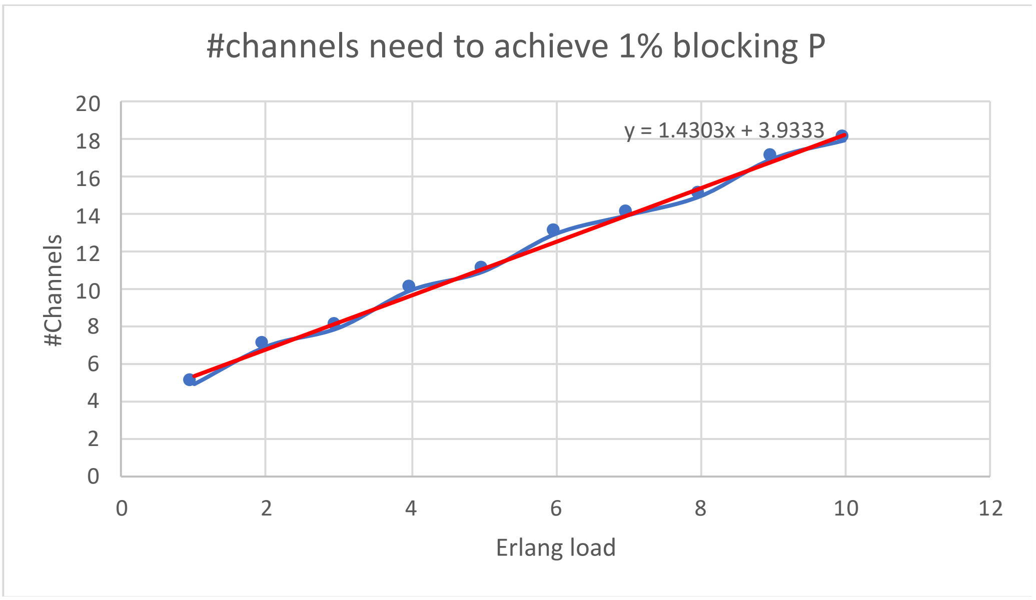
3.



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.

Erlang load	#channels need to achieve 1% blocking P
1	5
2	7
3	8
4	10
5	11
6	13
7	14
8	15
9	17
10	18

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.

	A=1, h =1, l=1	A=1, h =1, l=1	A=2,h=2,l=1	A=2,h=1,l=1	A=2,h=0.5,l=1
	1	1	2	2	4
N #num of channels	W(60)	W(30)	W(60)	W(60)	W(60)
1					
2	0.877263188	0.797504783			
3	0.987736404	0.966663003	0.73017639	0.8364206	0.93984443
4	0.998985827	0.99548018	0.93597162	0.97641021	0.996788204
5	0.999931216	0.999486625	0.98667002	0.9970538	0.999863803
6	0.999994814	0.999948204	0.99758762	0.99966961	0.999991179
7	1	0.99999541	0.99960762	0.99996358	1
8	1	0.999999821	0.99994183	0.99999559	1
9	1	1	0.999991	1	1
10	1	1	0.99999958	1	1
11	1	1	1	1	1
12	1	1	1	1	1
13	1	1	1	1	1
14	1	1	1	1	1
15	1	1	1	1	1
16	1	1	1	1	1
17	1	1	1	1	1
18	1	1	1	1	1
19	1	1	1	1	1
20	1	1	1	1	1

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.

```python
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
```

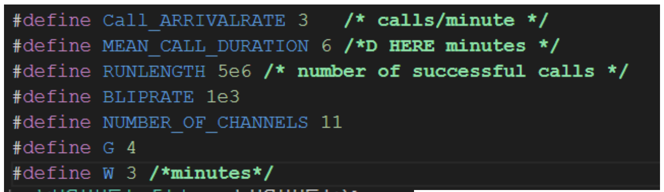
```
  return result

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.

	1	1	2	3	4
N #num of channels	W(60)	W(30)	W(60)	W(60)	W(60)
1					
2	0.87737352	0.79782311			
3	0.95672007	0.92864345	0.73043082	0.83649803	0.93985099
4	0.98885214	0.98162028	0.85443264	0.91170893	0.96751953
5	0.99774997	0.99629033	0.93779173	0.96226878	0.98611946
6	0.99962423	0.99938046	0.97852989	0.98697772	0.99520937
7	0.9999463	0.99991146	0.99378183	0.99622849	0.99861254
8	0.99999329	0.99998893	0.99843879	0.99905308	0.99965165
9	0.99999925	0.99999877	0.99965267	0.99978933	0.9999225
10	0.99999993	0.99999988	0.99993051	0.99995786	0.9999845
11	0.99999999	0.99999999	0.99998737	0.99999234	0.99999718
12	1	1	0.99999789	0.99999872	0.99999953
13	1	1	0.99999968	0.9999998	0.99999993
14	1	1	0.99999995	0.99999997	0.99999999
15	1	1	0.99999999	1	1
16	1	1	1	1	1
17	1	1	1	1	1
18	1	1	1	1	1
19	1	1	1	1	1
20	1	1	1	1	1

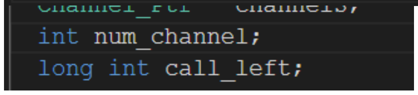
And by comparing both tables, we can prove our simulation is correct.

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.

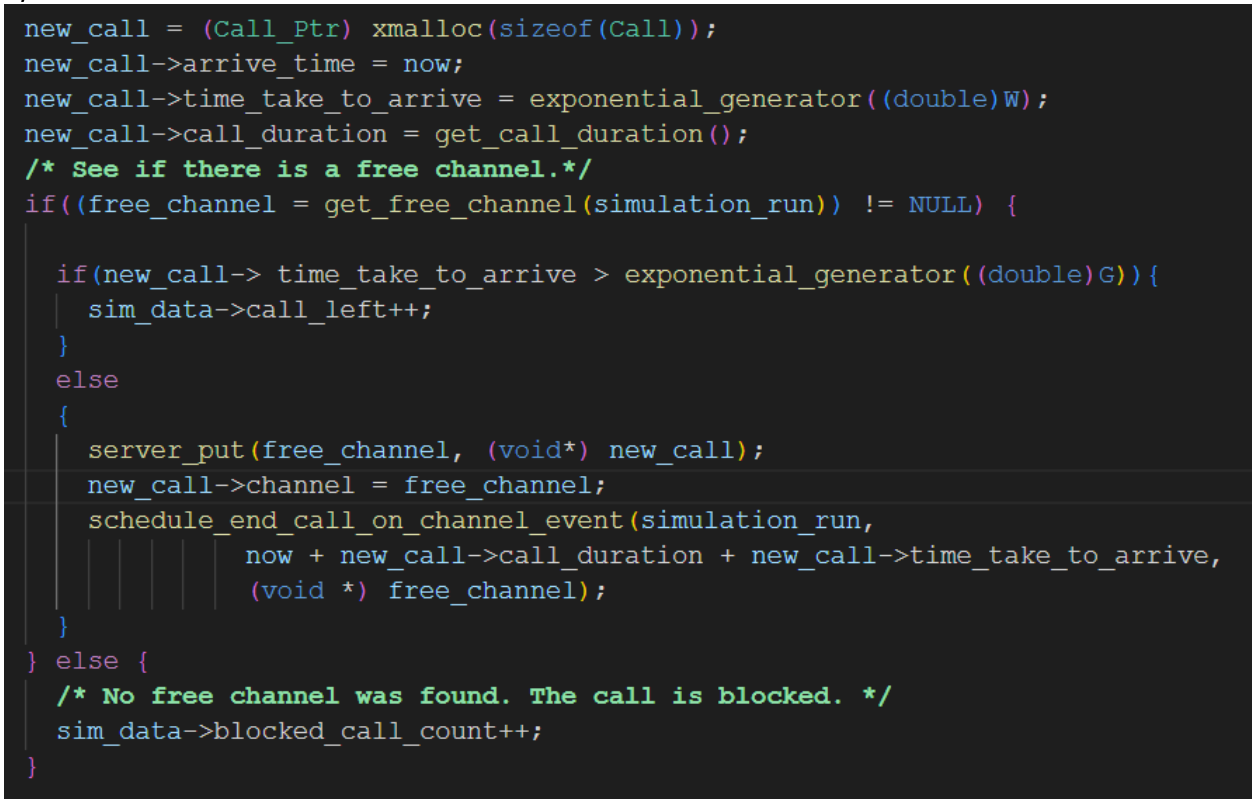


```
#define Call_ARRIVALRATE 3   /* calls/minute */
#define MEAN_CALL_DURATION 6 /*D HERE minutes */
#define RUNLENGTH 5e6 /* number of successful calls */
#define BLIPRATE 1e3
#define NUMBER_OF_CHANNELS 11
#define G 4
#define W 3 /*minutes*/
```



```
  int num_channel;
  long int call_left;
```

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.



```
new_call = (Call_Ptr) xmalloc(sizeof(Call));
new_call->arrive_time = now;
new_call->time_take_to_arrive = exponential_generator((double)W);
new_call->call_duration = get_call_duration();
/* See if there is a free channel.*/
if((free_channel = get_free_channel(simulation_run)) != NULL) {

  if(new_call-> time_take_to_arrive > exponential_generator((double)G)){
    sim_data->call_left++;
  }
  else
  {
    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 + new_call->time_take_to_arrive,
              (void *) free_channel);
  }
} else {
  /* No free channel was found. The call is blocked. */
  sim_data->blocked_call_count++;
}
```

After running several simulations. Below is the initial result.

N taxis	Taxi P	Blocking P	Call_ARRIVALRATE	D(Time to transport)	W(Time to arrive to customer)	G(Give up time)
1	0.42851	0.88992	3	3	3	4
2	0.42835	0.78242	3	3	3	4
3	0.42858	0.67815	3	3	3	4
4	0.42818	0.5785	3	3	3	4
5	0.42833	0.48332	3	3	3	4
6	0.42854	0.39451	3	3	3	4
7	0.42855	0.31297	3	3	3	4
8	0.42868	0.24018	3	3	3	4
9	0.4288	0.17739	3	3	3	4
10	0.42844	0.12565	3	3	3	4
11	0.42858	0.08437	3	3	3	4
12	0.42853	0.05358	3	3	3	4
13	0.42882	0.03222	3	3	3	4
14	0.42858	0.01819	3	3	3	4
15	0.4289	0.00979	3	3	3	4
16	0.42853	0.0049	3	3	3	4
17	0.42866	0.00237	3	3	3	4
18	0.42845	0.00105	3	3	3	4
19	0.42866	0.00043	3	3	3	4

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.

N taxis	Taxi P	Blocking P	Call_ARRI	D(Time to	W(Time to	G(Give up
1	0.23064	0.92451	3	3	3	10
2	0.23075	0.84983	3	3	3	10
3	0.23067	0.7763	3	3	3	10
4	0.23074	0.70397	3	3	3	10
5	0.2312	0.63246	3	3	3	10
6	0.23085	0.56369	3	3	3	10
7	0.23081	0.49643	3	3	3	10
8	0.23052	0.43224	3	3	3	10
9	0.23078	0.37036	3	3	3	10
10	0.23084	0.31228	3	3	3	10
11	0.23107	0.25779	3	3	3	10
12	0.23082	0.20819	3	3	3	10
13	0.23062	0.16394	3	3	3	10
14	0.2308	0.12529	3	3	3	10
15	0.23087	0.09277	3	3	3	10
16	0.23073	0.06639	3	3	3	10
17	0.23085	0.04573	3	3	3	10
18	0.23054	0.03039	3	3	3	10
19	0.23054	0.01917	3	3	3	10

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.

N taxis	Taxi P	Blocking P	Call_ARRI\	D(Time to	W(Time tc	G(Give up
1	0.20011	0.90112	3	3	1	4
2	0.20004	0.8042	3	3	1	4
3	0.1999	0.70962	3	3	1	4
4	0.20021	0.61796	3	3	1	4
5	0.19997	0.5301	3	3	1	4
6	0.19984	0.44625	3	3	1	4
7	0.20002	0.36752	3	3	1	4
8	0.20024	0.29542	3	3	1	4
9	0.20009	0.23006	3	3	1	4
10	0.19993	0.17371	3	3	1	4
11	0.20013	0.12543	3	3	1	4
12	0.19982	0.08731	3	3	1	4
13	0.19974	0.05761	3	3	1	4
14	0.20013	0.03619	3	3	1	4
15	0.19999	0.02151	3	3	1	4
16	0.20008	0.01217	3	3	1	4
17	0.19966	0.00661	3	3	1	4
18	0.19983	0.00334	3	3	1	4
19	0.19999	0.0016	3	3	1	4
20	0.2	0.00073	3	3	1	4

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.

N taxis	Taxi P	Blocking P	Call_ARRIVALRATE	D(Time to transport)	W(Time to arrive to customer)	G(Give up time)
1	0.42872	0.92967	3	6	3	4
2	0.42844	0.8601	3	6	3	4
3	0.42852	0.79133	3	6	3	4
4	0.42835	0.72354	3	6	3	4
5	0.42871	0.65717	3	6	3	4
6	0.42848	0.59136	3	6	3	4
7	0.42826	0.52826	3	6	3	4
8	0.42858	0.46578	3	6	3	4
9	0.42863	0.40632	3	6	3	4
10	0.42883	0.34909	3	6	3	4
11	0.42878	0.29585	3	6	3	4
12	0.42869	0.24608	3	6	3	4
13	0.42867	0.19999	3	6	3	4
14	0.42871	0.15875	3	6	3	4
15	0.42859	0.12294	3	6	3	4
16	0.42836	0.09203	3	6	3	4
17	0.42851	0.06713	3	6	3	4
18	0.42881	0.0467	3	6	3	4
19	0.42833	0.03164	3	6	3	4
20	0.42865	0.02038	3	6	3	4