Computer Engineering 4DK4

Lab 3: Call Blocking in Circuit Switched Networks

Instructor: Prof. T. Todd

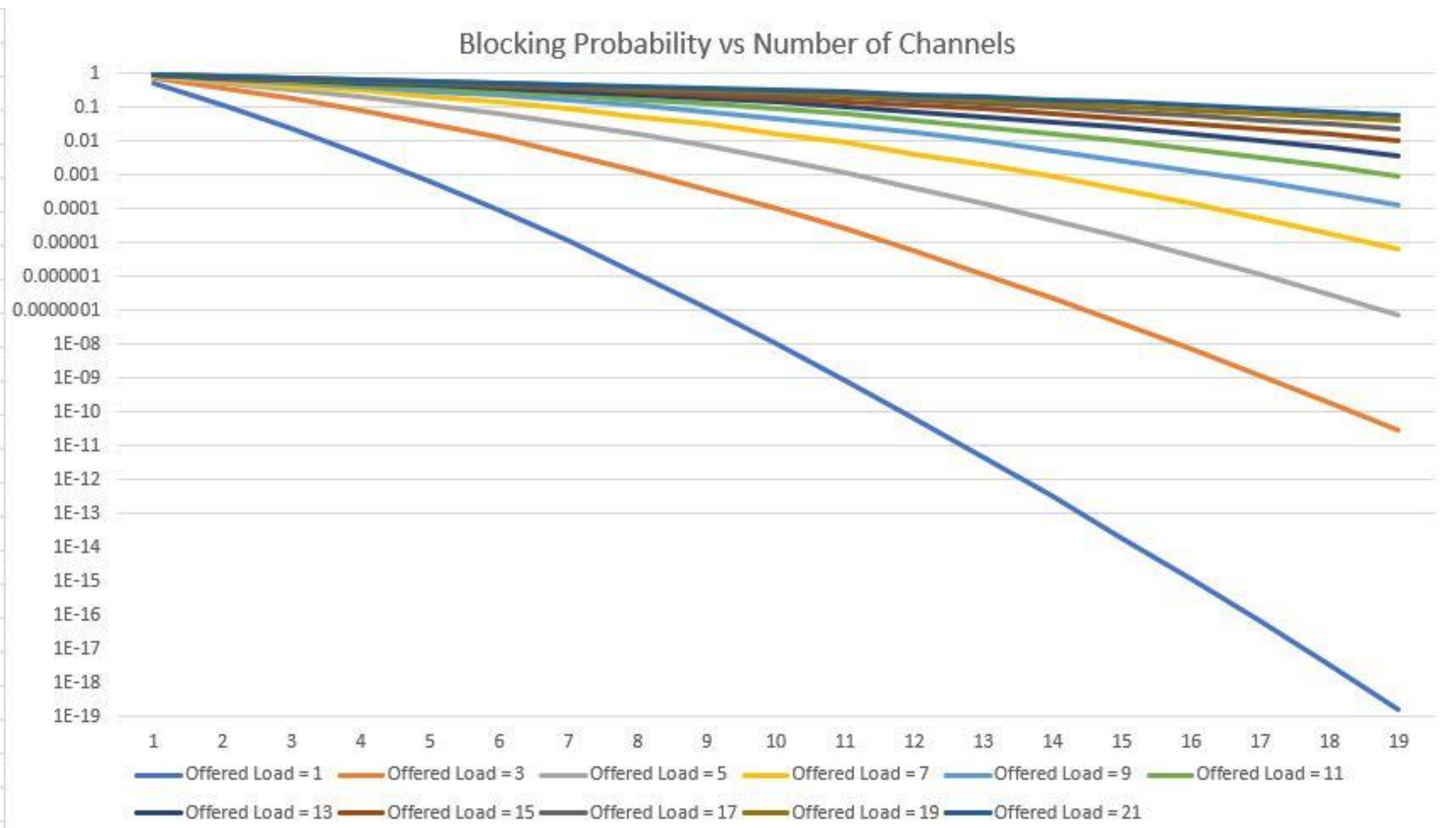
Moshiur Howlader - 1316948

Ankur Bargotra – 1306226

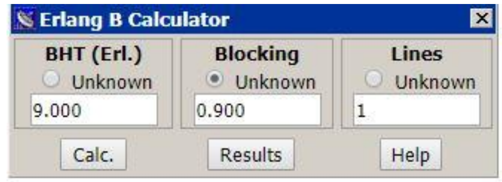
All Code In Appendix Section

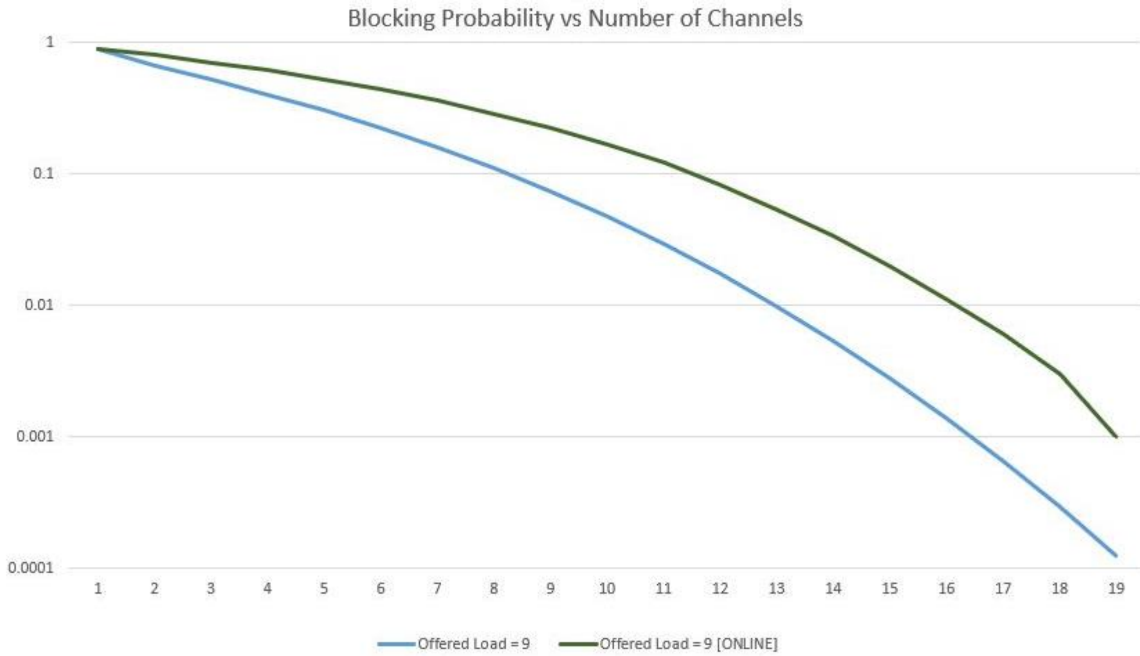
2)

Number of Channels vs Offered Load	1	3	5	7	9	11	13	15	17	19	21
1	0.5	0.75	0.833333	0.875	0.9	0.916667	0.928571	0.9375	0.944444	0.95	0.954545
2	0.111111	0.36	0.510204	0.604938	0.669421	0.715976	0.751111	0.778547	0.800554	0.818594	0.833648
3	0.023256	0.176471	0.326371	0.439181	0.522581	0.58557	0.634421	0.67325	0.704777	0.730847	0.752743
4	0.004219	0.080597	0.201548	0.312752	0.403282	0.475403	0.533203	0.580156	0.618871	0.65125	0.678682
5	0.000662	0.033596	0.118036	0.215593	0.304629	0.380231	0.443238	0.495765	0.539868	0.577248	0.609239
6	9.07E-05	0.012706	0.065024	0.142982	0.22429	0.298726	0.363682	0.41949	0.467339	0.508514	0.544157
7	1.10E-05	0.00436	0.033526	0.09081	0.160428	0.23001	0.294077	0.351074	0.401127	0.444946	0.483369
8	1.20E-06	0.001361	0.016129	0.055014	0.111123	0.173174	0.233969	0.290277	0.341099	0.386467	0.426825
9	1.17E-07	0.000389	0.007231	0.031685	0.074309	0.12719	0.182841	0.236827	0.287103	0.332988	0.374473
10	1.05E-08	0.000102	0.003021	0.017305	0.047832	0.09091	0.140094	0.190404	0.238964	0.284408	0.326252
11	8.67E-10	2.47E-05	0.001178	0.008947	0.029561	0.063083	0.105043	0.150634	0.196471	0.240606	0.282089
12	6.60E-11	5.55E-06	0.00043	0.004376	0.017501	0.042397	0.076923	0.117088	0.159382	0.201441	0.241899
13	4.68E-12	1.16E-06	0.000147	0.002024	0.009909	0.027538	0.054908	0.089278	0.127413	0.166746	0.205582
14	3.10E-13	2.29E-07	4.72E-05	0.000886	0.005359	0.017255	0.038129	0.066667	0.100244	0.13633	0.173024
15	1.92E-14	4.22E-08	1.43E-05	0.000368	0.002768	0.010413	0.025712	0.048672	0.077513	0.109972	0.144088
16	1.12E-15	7.34E-09	4.10E-06	0.000145	0.001365	0.006045	0.01681	0.034685	0.058824	0.087426	0.118621
17	6.22E-17	1.21E-09	1.11E-06	5.42E-05	0.000643	0.003373	0.010639	0.024089	0.043749	0.068413	0.096447
18	3.26E-18	1.89E-10	2.87E-07	1.93E-05	0.000289	0.001809	0.006512	0.016281	0.031842	0.052632	0.077371
19	1.62E-19	2.81E-11	7.05E-08	6.57E-06	0.000125	0.000932	0.003851	0.010695	0.02265	0.039758	0.061174



Compared Using: http://www.erlang.com/calculator/erlb/





For our tests we computed results for 19 channels.

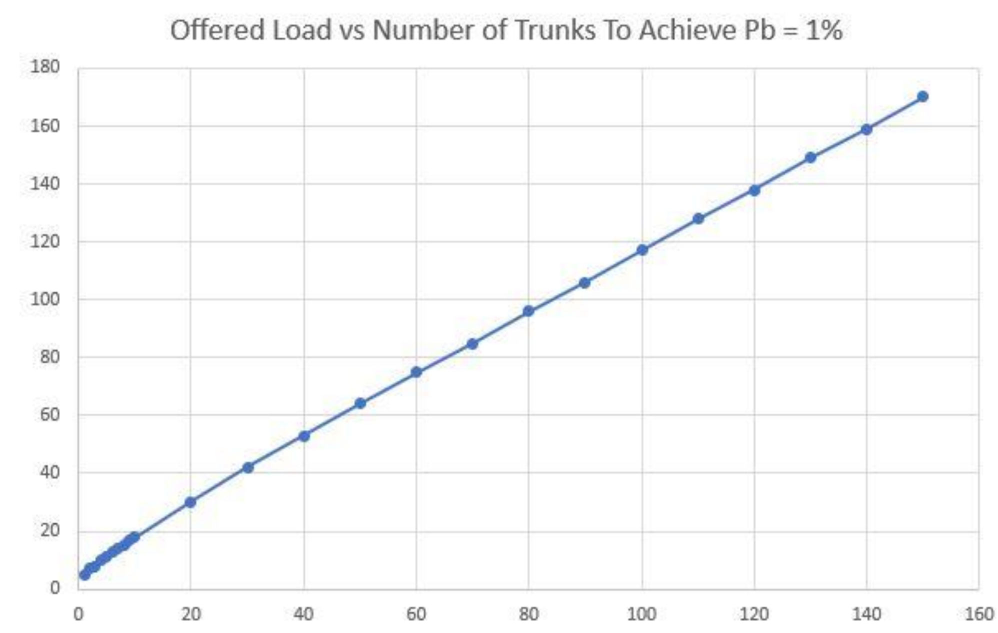
As seen from the graphs above, as the number of channels increases, the blocking probability, P_B, decreases. For any given number of channels/trunks as the offered load increases the blocking probability increases.

Also, observable is multiplexing gain, for a given number of channels, at a given P_B, the system becomes more efficient in utilizing the trunks with increasing system size (offered load).

When comparing our results with an online calculator yield similar results.

3)

Trunks	Load
1	5
2	7
3	8
4	10
5	11
6	13
7	14
8	15
9	17
10	18
20	30
30	42
40	53
50	64
60	75
70	85
80	96
90	106
100	117
110	128
120	138
130	149
140	159
150	170



There is a linearly proportional relationship between the number of trunks and the load for a given probability. In this case, to achieve a 1% P_B.

4) The equation we need to use to verify our simulation is:

$$W(t) = 1 - (Pw) * \mathrm{e}^{-(\mathrm{N}-\mathrm{A})*\frac{t}{h}} \;\; (eqn\, 1),$$

$$where\ t\ is\ in\ seconds$$

Also average caller waiting time is

$$Tw = \frac{Pw * h}{N * \left(1 - \frac{A}{N}\right)}\ (eqn\ 2)$$

$$or\ rearranging\ for\ Pw = \frac{Tw * N * (1 - \frac{A}{N})}{h}\ (eqn\ 3)$$

We substitute eqn 3 into eqn 1 to get:

$$W(t) = \frac{Tw * N * \left(1 - \frac{A}{N}\right)}{h} * \mathrm{e}^{-(\mathrm{N}-\mathrm{A})*\frac{t}{h}}\ (eqn\ 4)$$

Eqn 4 allows us to verify our results from simulation since Tw can be easily computer from simulation as (sim_data->accumulated_wait_time)/(sim_data->blocked_call_count). Divide accumulated wait time by the number of blocked call count to find the average wait time Tw. Eqn 1 allows easier verification of the theoretical result.

Using Eqn 1, we find that theoretically W(t) is 47.176 %, Tw = 0.809, t = 0.809*60 when N=11, A=9, h=3, lambda =3.

We obtain 45% for our W(t), and Tw as 0.82.

```
100% Call Count = 5000000
random seed = 1306226
call arrival count = 5685012
blocked call count = 685003
Blocking probability = 0.12049 (Service fraction = 0.87951)
BlockedQueue Size is 0.0000
Average Call Time is 2.9981
Total Waiting Time (min) is 0.8213
Probability of waiting less than t seconds 0.4525

100% Call Count = 5000000
random seed = 1316948
call arrival count = 5686683
blocked call count = 686673
Blocking probability = 0.12075 (Service fraction = 0.87925)
BlockedQueue Size is 0.0000
Average Call Time is 2.9974
Total Waiting Time (min) is 0.8188
Probability of waiting less than t seconds 0.4541
```

When we change our parameter to N=5, A=4, h=1, lambda =4, we find theoretically that W(t) is 68.156 %, Tw = 0.5543, t = 0. 5543*60.

We obtain 62.5% for our W(t), and Tw as 0.56.

```
100% Call Count = 5000000
random seed = 1306226
call arrival count = 5685012
blocked call count = 685003
Blocking probability = 0.12049 (Service fraction = 0.87951)
BlockedQueue Size is 0.0000
Average Call Time is 2.9981
Total Waiting Time (min) is 0.5616
Probability of waiting less than t seconds 0.6256

100% Call Count = 5000000
random seed = 1316948
call arrival count = 5686683
blocked call count = 686673
Blocking probability = 0.12075 (Service fraction = 0.87925)
BlockedQueue Size is 0.0000
Average Call Time is 2.9974
Total Waiting Time (min) is 0.5603
Probability of waiting less than t seconds 0.6265
```

5) In simparameter.h, we change the simulation parameters to see how the system behaves.

```
#ifndef _SIMPARAMETERS_H_
#define _SIMPARAMETERS_H_

/*******************************************************************

#define Call_ARRIVALRATE 2   /* calls/minute */
#define MEAN_CALL_DURATION 1 /* minutes */
#define RUNLENGTH 5e6 /* number of successful calls */
#define BLIPRATE 1e3
#define NUMBER_OF_CHANNELS 11 //N, number of taxis
#define WAIT_TIME 3 //W, after this time, taxi arrives to customer
#define GIVEUP_TIME 1 //G, increment frustrated customer
#define DELIVERY_TIME 5 //D, after this time, clear one taxi channel

/* Comma separated list of random seeds to run. */
#define RANDOM_SEED_LIST 333, 4444, 1316948
```

When N = 11, W = 3, G = 1, and D = 5 we get (roughly same for different seeds):

```
100% Call Count = 5000000
random seed = 1316948
call arrival count = 5000725
blocked call count = 722
Blocking probability = 0.00014 (Service fraction = 0.99986)
Calls processed = 5000000
Number of frustrated customers (taxi came too late) = 3747637
Total customers processed = 5000000
Percentage of frustrated customers (taxi came too late) = 0.749527
```

Since wait time is longer than give up time, the percentage of frustrated customers (or customer that left after call was processed) is high. However, the number of calls that gets blocked by the taxi is low since there is sufficient number of taxi to handle the customers.

When N = 11, W = 1, G = 3, and D = 5 we get (roughly same for different seeds):

```
100% Call Count = 5000000
random seed = 1316948
call arrival count = 5025581
blocked call count = 25577
Blocking probability = 0.00509 (Service fraction = 0.99491)
Calls processed = 5000000
Number of frustrated customers (taxi came too late) = 1250131
Total customers processed = 5000000
Percentage of frustrated customers (taxi came too late) = 0.250026
```

Since wait time on average is shorter than give up time, the percentage of frustrated customers is significantly lower. Blocking probability in general does not change very much since N is still 11.

When we lower N = 3 (Taxis), keep rest constant W = 3, G = 1, and D = 5 we get (roughly same for different seeds):

```
100% Call Count = 5000000
random seed = 1316948
call arrival count = 10054520
blocked call count = 5054518
Blocking probability = 0.50271 (Service fraction = 0.49729)
Calls processed = 5000000
Number of frustrated customers (taxi came too late) = 1248367
Total customers processed = 5000000
Percentage of frustrated customers (taxi came too late) = 0.249673
```

We get higher blocking probability of the taxi services.

When we increase the D from D = 5 to D = 10, and, keep rest constant N = 3 (Taxis), W = 3, G = 1, we get higher percentage of frustrated customers leaving before the taxi arrives. Since taxis spend more time delivering current clients that, less time to serve other customers. Therefore, instead of 24.97% frustrated customers, we get 42.85% frustrated customers. Also the blocking probability increases 2 or 3 percent more because of a higher D value.

```
100% Call Count = 5000000
random seed = 1316948
call arrival count = 10782384
blocked call count = 5782382
Blocking probability = 0.53628 (Service fraction = 0.46372)
Calls processed = 5000000
Number of frustrated customers (taxi came too late) = 2142677
Total customers processed = 5000000
Percentage of frustrated customers (taxi came too late) = 0.428535
```

The implemented simulation models the simulation correctly based on intuition. When Wait time for taxi to pick up customer is higher than the give up time of the customer to wait for that wait time, then naturally more customers will leave. Otherwise, more patience from the customers, so lower probability of customers leaving before taxi arrives. The number of taxi influences the blocking probability of the taxi service. More taxi means more service available. Longer the taxi spend time delivering customers, less time for taxis to serve other customers. This leads to more customers leaving before the taxi can make it to the customer.

Appendix

2)

We used Matlab to calculate blocking probability. A double loop was implemented to calculate the numerator and denominator, to compute P_B.

```
offered_load = 21; denum = 0;

for n=1:19
    for i=0:n
        denum = denum + ((offered_load^i)/factorial(i));
    end
  Pb(n) = ((offered_load^n)/factorial(n))/denum;
end
```

3) No code change, just trial and error to obtain desired value.

4) main.c

Initialize some sim_data variables to help find the probability of waiting time less than t seconds.

```
typedef struct _simulation_run_data_
{
  Channel_Ptr * channels;
  long int blip_counter;
  long int call_arrival_count;
  long int calls_processed;
  long int blocked_call_count;
  long int number_of_calls_processed;
  double accumulated_call_time;
  unsigned random_seed;
  Fifoqueue_Ptr blockedQueue; //store all blocked calls inside this queue
  long int blockedSize;
  double accumulated_wait_time;
} Simulation_Run_Data, * Simulation_Run_Data_Ptr;
```

call_arrival.c

We implemented a FIFO queue to store blocked calls. We calculate the accumulated wait time by taking the difference between when the blocked call is processed minus when the call was blocked.

```
/* See if there is a free channel.*/
if((free_channel = get_free_channel(simulation_run)) != NULL) {

    if(fifoqueue_size(sim_data->blockedQueue)>0){ //sim_data->blockedSize>0
        sim_data->blockedSize--;
        sim_data->accumulated_wait_time += now;
        new_call =(Call_Ptr) fifoqueue_get(sim_data->blockedQueue); //remove the front call
        //memory already allocated for contents in queue
        new_call->arrive_time = now;
        new_call->call_duration = get_call_duration();
        /* 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{
      /* Yes, we found one. Allocate some memory and start the call. */
      new_call = (Call_Ptr) xmalloc(sizeof(Call));
      new_call->arrive_time = now;
      new_call->call_duration = get_call_duration();

      /* Place the call in the free channel and schedule its
         departure. */
      server_put(free_channel, (void*) new_call);
      new_call->channel = free_channel;
```

```
    }else{
      /* Yes, we found one. Allocate some memory and start the call. */
      new_call = (Call_Ptr) xmalloc(sizeof(Call));
      new_call->arrive_time = now;
      new_call->call_duration = get_call_duration();

      /* 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 {
    sim_data->blocked_call_count++;
    sim_data->blockedSize++;
    //No free channel, so store the "blocked" call to a blockedQueue
    //to be processed later
    new_call = (Call_Ptr) xmalloc(sizeof(Call));
    sim_data->accumulated_wait_time -= now;//new_call->arrive_time = now;
    //subtract here, add above to find the difference to know the wait time
    //therefore will accumulate total wait time


    fifoqueue_put((sim_data->blockedQueue), (void*) new_call);
    //waitTime = simulation_run_get_time(simulation_run);
}
```

call_departure.c

No changes in call_departure.c

Output.c

```
void output_results(Simulation_Run_Ptr this_simulation_run)
{
    double xmtted_fraction;
    double Tw;
    double A;
    int N;
    int h;
    double Wt_prob;
    Simulation_Run_Data_Ptr sim_data;
```

```
printf("BlockedQueue Size is %.4f\n", sim_data->blockedSize);

printf("Average Call Time is %.4f\n", ((sim_data->accumulated_call_time))/(sim_data->number_of_calls_processed));
Tw = (double) sim_data->accumulated_wait_time/(2*(sim_data->blocked_call_count));
printf("Total Waiting Time (min) is %.4f\n", Tw);
A = (double) Call_ARRIVALRATE*MEAN_CALL_DURATION;
N = NUMBER_OF_CHANNELS;
h = MEAN_CALL_DURATION;
double my_exp;
my_exp = (double) exp(-1*(N-A)*Tw*60/h);
Wt_prob = (double) 1 - my_exp*(Tw*N*(1-A/N))/h;

printf("Probability of waiting less than t seconds %.4f\n", Wt_prob);
printf("\n");
```

5) main.c

Initialize few more sim_data variables to help find blocking probability, and probability of a taxi arriving to find that a customer left.

```
/* Initialize our simulation_run data variables. */
data.blip_counter = 0;
data.call_arrival_count = 0;
data.calls_processed = 0;
data.blocked_call_count = 0;
data.number_of_calls_processed = 0;
data.accumulated_call_time = 0.0;
data.random_seed = random_seed;
data.num_frustrated_customers = 0;

/* Create the channels. */
data.channels = (Channel_Ptr *) xcalloc((int) NUMBER_OF_CHANNELS,
                                         sizeof(Channel_Ptr));
```

main.h

call struct was changed to contain the 2 channels for simulation. The wait_channel and the delivery_channel. Few more parameter were included to help simulate this taxi arrival events, wait_time (W), delivery_time (D), and giveup_time (G).

```c
typedef struct _call_
{
  double arrive_time;
  double call_duration;
  double wait_time;
  double delivery_time;
  double giveup_time;
  Channel_Ptr wait_channel;
  Channel_Ptr delivery_channel;
} Call, * Call_Ptr;

typedef struct _simulation_run_data_
{
  Channel_Ptr * channels;
  long int blip_counter;
  long int call_arrival_count;
  long int calls_processed;
  long int blocked_call_count;
  long int number_of_calls_processed;
  double accumulated_call_time;
  unsigned random_seed;
  long int num_frustrated_customers;
} Simulation_Run_Data, * Simulation_Run_Data_Ptr;
```

call_arrival.c

Initialize new parameters for the new_call. Store the free_channel content in the call's wait server, and schedule the call as long as there is a free channel of taxi available.

```c
/* See if there is a free channel.*/
if((free_channel = get_free_channel(simulation_run)) != NULL) {

  /* Yes, we found one. Allocate some memory and start the call. */
  new_call = (Call_Ptr) xmalloc(sizeof(Call));
  new_call->arrive_time = now;
  new_call->call_duration = get_call_duration();
  new_call->wait_time = exponential_generator((double) WAIT_TIME);
  new_call->giveup_time = exponential_generator((double) GIVEUP_TIME);

  /* Place the call in the wait channel and schedule its
     departure. */
  server_put(free_channel, (void*) new_call);
  new_call->wait_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. */
  sim_data->blocked_call_count++;
}
```

call_departure.h

```
void
end_call_on_channel_event(Simulation_Run_Ptr ThisSimulation_Run, void*);

long int
schedule_end_call_on_channel_event(Simulation_Run_Ptr, double, void*);

void
end_delivery_on_channel_event(Simulation_Run_Ptr ThisSimulation_Run, void*);

long int
schedule_end_delivery_on_channel_event(Simulation_Run_Ptr, double, void*);
```

call_departure.c

For call_departure, we first check the scenario of whether the arriving taxi comes later than the giveup time of the customer. Depending on whether the customer has given up, we end the call, or reschedule the call

```
void
end_call_on_channel_event(Simulation_Run_Ptr simulation_run, void * c_ptr)
{
  Call_Ptr this_call;
  Channel_Ptr channel;
  Simulation_Run_Data_Ptr sim_data;
  double now;

  channel = (Channel_Ptr) c_ptr;

  now = simulation_run_get_time(simulation_run);
  sim_data = simulation_run_data(simulation_run);

  /* Remove the call from the channel.*/
  this_call = (Call_Ptr) server_get(channel);
  if(this_call->wait_time > this_call->giveup_time){
    TRACE(printf("End Of Call.\n"););

    /* Collect statistics. */
    sim_data->number_of_calls_processed++;
    sim_data->accumulated_call_time += now - this_call->arrive_time;
    sim_data->num_frustrated_customers++;

    output_progress_msg_to_screen(simulation_run);

    /* This call is done. Free up its allocated memory.*/
    xfree((void*) this_call);
  }else{
```

Below is the case when we need to reschedule the event since the taxi has arrived, and the customer will be delivered to their destination. Remove customer from wait channel and store in delivery channel.

```
}else{
  //reschedule, and update call duration
  //store in delivery channel
  //this_call = (Call_Ptr) server_get(channel);
    server_put(channel, (void*) this_call);
    this_call->wait_channel = NULL;
    this_call->delivery_channel = channel;
    this_call->call_duration += this_call->wait_time; //makes impact in blocked call count
  schedule_end_delivery_on_channel_event(simulation_run,
                                         now + this_call->call_duration,
                                         (void *) channel); //double pointer effectively
}
```

The end delivery channel event occurs below. Finish the delivery, and end the "call" of the customer.

```
void
end_delivery_on_channel_event(Simulation_Run_Ptr simulation_run, void * c_ptr)
{
  //finish delivery
  Call_Ptr this_call;
  Channel_Ptr channel;
  Simulation_Run_Data_Ptr sim_data;
  double now;

  channel = (Channel_Ptr) c_ptr;

  now = simulation_run_get_time(simulation_run);
  sim_data = simulation_run_data(simulation_run);

  /* Remove the call from the channel.*/
  this_call = (Call_Ptr) server_get(channel);
    TRACE(printf("End Of Call.\n"););

    /* Collect statistics. */
    sim_data->number_of_calls_processed++;
    sim_data->accumulated_call_time += now - this_call->arrive_time;

    output_progress_msg_to_screen(simulation_run);

    /* This call is done. Free up its allocated memory.*/
    xfree((void*) this_call);

}
```

Output.c

Added probability of taxi arriving to find that the customer left.

```
printf("Calls processed = %ld \n", sim_data->number_of_calls_processed);
printf("Percentage of frustrated customers (taxi came too late) = %f \n", (double) (sim_data->num_frustrated_customers)/(sim_data->number_of_calls_proc
```