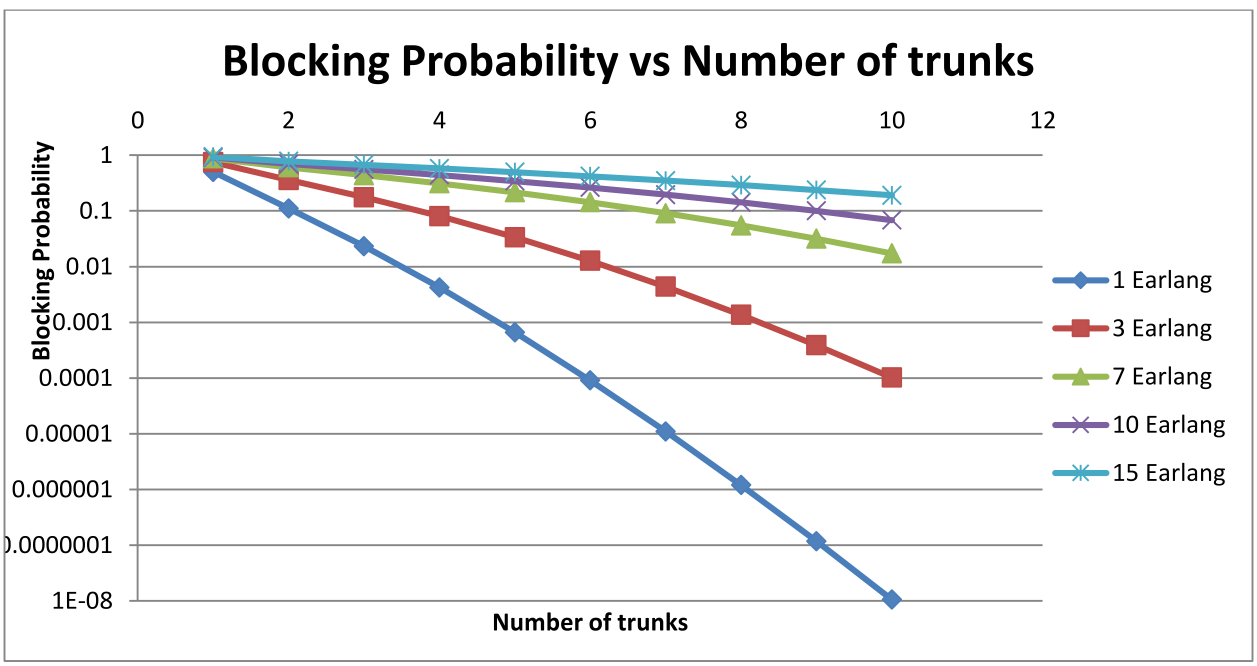
COE4DK4 – LAB 3

Vishal Kharker 1140837
Khalid Asad 1147299
08-Nov-15

EXPERIMENT 2

During this experiment it was expected to code an Erlang B formula using to realize the tadeoffs between blocking probability, offered load and number of channels. In the table below it is shown that in order to keep 1% blocking probability performance, the required number of lines increases as the load increases. The relationship between the required number of lines and offered load is not a linear one but more of a logarithmic relationship. As it is seen, with an offered load of 1 erlang a total of 5 lines are required to maintain the 1% performance ratio. On the other hand when the offered load increases to 3 erlangs, a total of 8 lines are required. This result is caused because as the load on the system is increasing, so is the efficiency of the system to manage number of lines.

	OFFERED LOAD (A)				
Number of Lines	1	3	7	10	15
1	0.5	0.75	0.875	0.90909091	0.9375
2	0.111111111	0.36	0.60493827	0.69444444	0.77854671
3	0.023255814	0.17647059	0.43918054	0.55617353	0.67324955
4	0.004219409	0.08059701	0.31275238	0.44138418	0.58015608
5	0.000661813	0.03359602	0.21559321	0.34411562	0.49576458
6	9.07194E-05	0.01270588	0.14298215	0.26263815	0.4194902
7	1.10058E-05	0.0043596	0.09080987	0.19569441	0.35107428
8	1.19547E-06	0.00136121	0.0550136	0.14196879	0.29027739
9	1.17442E-07	0.00038853	0.03168491	0.10000363	0.23682716
10	1.0525E-08	0.0001019	0.01730502	0.06821351	0.19040392



To confirm our results from experiment 2, we used an online Erlang B calculator and the results are shows below in a chart.

Calculator: http://www.erlang.com/calculator/erlb/

Erlang B Results Table

Here are the results (max 20) of the Erlang B Calculator. The unknown figures are shown in red.

B.H.T.	Blocking	Lines
1.000	0.010	5
3.000	0.010	8
7.000	0.010	14
10.000	0.010	18
15.000	0.010	24

Results displayed - 07/11/2015, 22:44:22

MATLAB CODE:

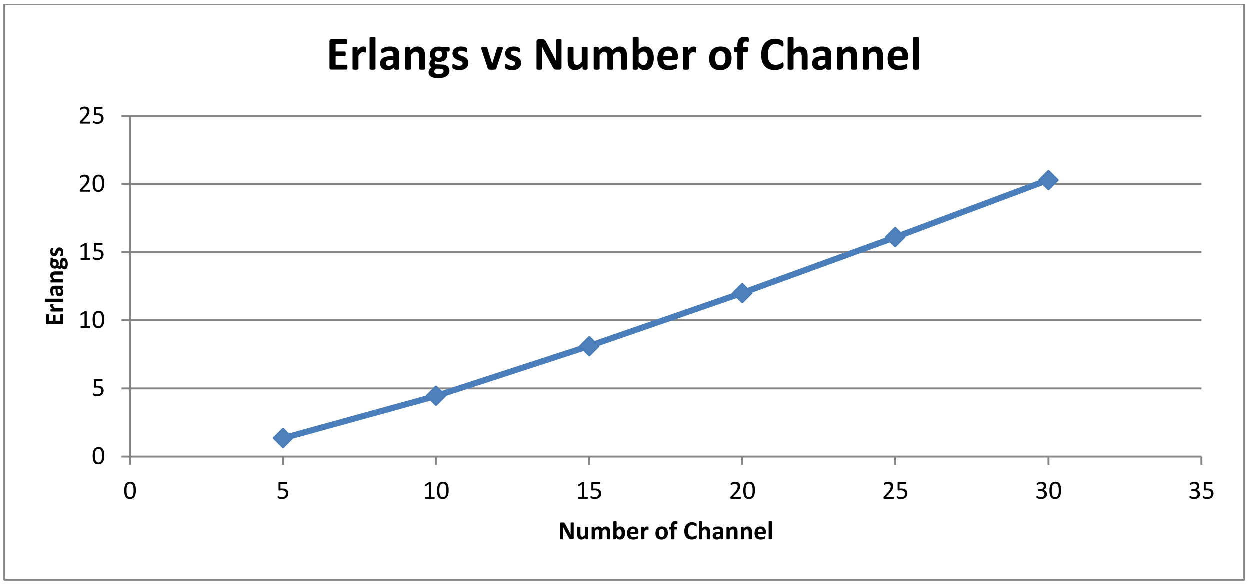
```
denom = 0;
for C = 1:20
    for k = 0:C
        denom = denom + ((9^k)/factorial(k));
    end
    Pb (C)= (9^C/factorial(C))/denom
end

xlswrite ('HelloWorld.xls',Pb)
```

EXPERIMENT 3:

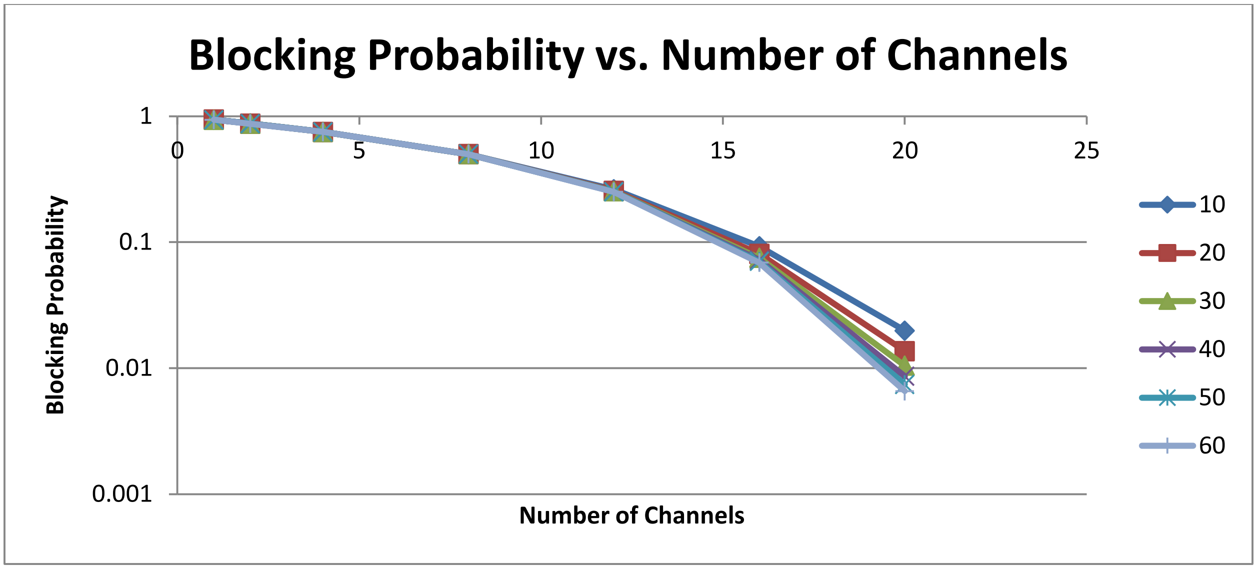
The chart below highlights the relationship between number of channels and the load it can handle. It is a direct relationship between the channels needed to handle a certain load while achieving the top performances.

B.H.T.	Blocking	Lines
1.350	0.010	5
4.450	0.010	10
8.100	0.010	15
12.000	0.010	20
16.100	0.010	25
20.300	0.010	30



EXPERIMENT 4

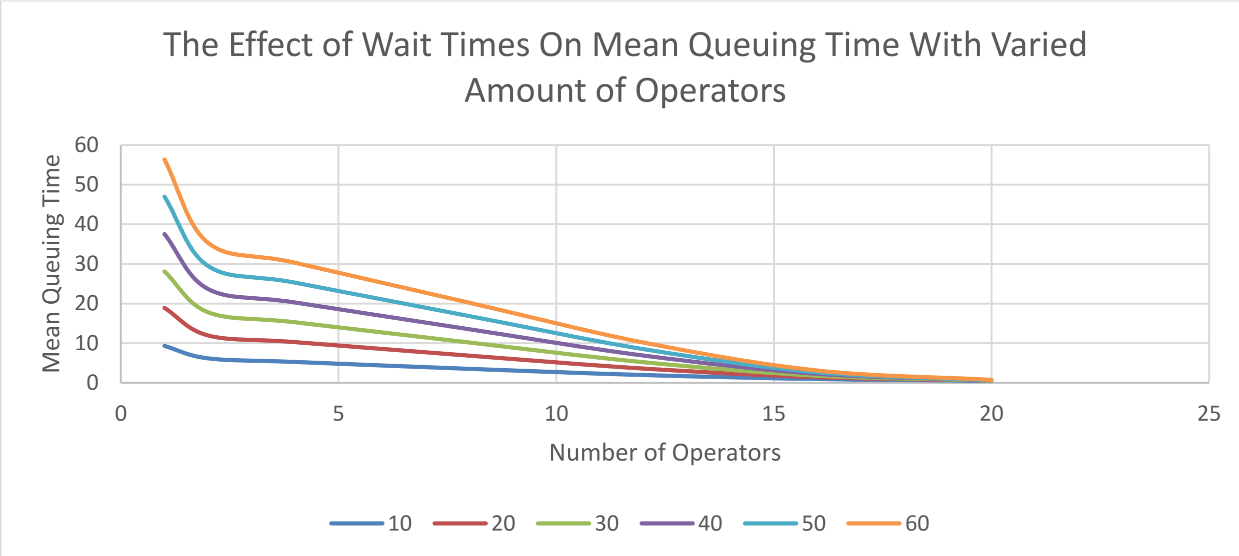
	Number of Channels						
	1	**2**	**4**	**8**	**12**	**16**	**20**
10	0.93811	0.87556	0.749975	0.499685	0.2626225	0.0927275	0.0198525
20	0.93857	0.874875	0.750075	0.50002	0.25601	0.0807825	0.0136625
30	0.93799	0.87234	0.7500675	0.50003	0.253205	0.075175	0.0104475
40	0.93781	0.87266	0.7500425	0.5000775	0.2520575	0.0723825	0.0086175
50	0.93765	0.872348	0.7499825	0.5000425	0.251275	0.0705175	0.00742
60	0.93756	0.87232	0.7499725	0.4999825	0.2502825	0.06883	0.006565



After modifying the simulation to act as a call center and graphing the results, some conclusions can be made about this system:

- A lower number of operators results in an increase in the chance of dropping a call due to huge wait times and vice versa.
- As the amount of operators decreases, small wait times have little impact on blocking probability and vice versa.

	Number of Operators						
	1	**2**	**4**	**8**	**12**	**16**	**20**
10	9.32033	6.17123	5.248132	3.500072	1.928528	0.906407	0.461302
20	18.8981	11.94624	10.24807	6.835122	3.573943	1.38135	0.559497
30	28.03477	17.7845	15.24705	10.16594	5.212583	1.813472	0.609827
40	37.51453	23.66031	20.24435	13.49878	6.86205	2.239555	0.64346
50	46.92135	29.45831	25.24229	16.83158	8.503698	2.659643	0.674583
60	56.27163	35.28402	30.23905	20.16225	10.1223	3.052553	0.701198



Some more conclusions can be made about this system:

- A lower number of operators results in an increase in mean queuing time with the values of verging closer to the amount of wait time
- As the amount of operators increases, the mean queuing times verge closer to 0 around roughly 20 operators

Code:

<u>Call_departure.c</u>

```
void
end_call_on_channel_event(Simulation_Run_Ptr simulation_run, void * c_ptr)
{

…

  /*
  * See if there are calls waiting in the buffer. If so, take the next one
immediately.
  */

   if ((channel = get_free_channel(simulation_run)) != NULL) {
          if (fifoqueue_size(sim_data->buffer) > 0) {
                 for (q_count = 0; q_count < fifoqueue_size(sim_data->buffer);
q_count++) {
                        next_call = (Call_Ptr)fifoqueue_get(sim_data->buffer);
                        call_time = simulation_run_get_time(simulation_run) -
this_call->arrive_time;
                        sim_data->queue_wait_time += call_time;
                        if (call_time >= MAX_WAIT_TIME) {
                               sim_data->blocked_call_count++;
                               TRACE(printf("Call Blocked"););
                         }
                        else {
                               next_call->arrive_time =
simulation_run_get_time(simulation_run);
                               next_call->call_duration = get_call_duration();
```

```c
                                        server_put(channel, (void*)next_call);
                                        next_call->channel = channel;

               schedule_end_call_on_channel_event(simulation_run,
simulation_run_get_time(simulation_run) + get_call_duration(), (void *)channel);
                                        TRACE(printf("Waiting Call Completed"););
                                        break;
                                   }
                              }
                         }
                    }
```

Call_arrival.c

```c
void
call_arrival_event(Simulation_Run_Ptr simulation_run, void * ptr)
{
…
if((free_channel = get_free_channel(simulation_run)) != NULL) {
…
} else {
    new_call = (Call_Ptr)xmalloc(sizeof(Call));
    new_call->arrive_time = now;
    fifoqueue_put(sim_data->buffer, (void*)new_call);
    sim_data->calls_in_queue++;
    TRACE(printf("Call placed in Queue"););
}
```

main.h

```c
typedef struct _call_
{
  double arrive_time;
  double call_duration;
  Channel_Ptr channel;
  Call_Status status;
} Call, * Call_Ptr;

typedef struct _simulation_run_data_
{
  double queue_wait_time;
} Simulation_Run_Data, * Simulation_Run_Data_Ptr;
```

Simparameters.h

```c
#define MAX_WAIT_TIME 5
```

Output.c

```c
void output_results(Simulation_Run_Ptr this_simulation_run)
{
  double xmtted_fraction;
  double avgQTime;
       …
  avgQTime = (sim_data->queue_wait_time) / (fifoqueue_size(sim_data->buffer));
    …
  printf("Average Wait Time of the Queue = %.5f\n", avgQTime);
}
```

Main.c

```c
int main(void)
```

```
{
…
while ((random_seed = RANDOM_SEEDS[j++]) != 0) {
…
    data.calls_in_queue = 0;
    data.queue_wait_time = 0.0;
    …
    data.buffer = fifoqueue_new();
…
}
…
}
```

EXPERIMENT 5

We selected our system parameters to be A = 7*7 or 49 with our N = 63, that gives us W(t) = 98.16% success rate and 1.84% failure rate and if we compare to our results, we get W(t) as (98.2 + 98.12 + 98.09 + 98.15 + 98.12)/5 = 98.136%.

Code changes:

Call_departure.c

```
void
end_call_on_channel_event(Simulation_Run_Ptr simulation_run, void * c_ptr)
{

…

  /*
  * See if there are calls waiting in the buffer. If so, take the next one
immediately.
  */

    if ((channel = get_free_channel(simulation_run)) != NULL) {
            if (fifoqueue_size(sim_data->buffer) > 0) {
                    for (q_count = 0; q_count < fifoqueue_size(sim_data->buffer);
q_count++) {
                            next_call = (Call_Ptr)fifoqueue_get(sim_data->buffer);
                            call_time = simulation_run_get_time(simulation_run) -
this_call->arrive_time;
                            sim_data->queue_wait_time += call_time;
                            if (call_time >= MAX_WAIT_TIME) {
                             }
                            else {
                                    next_call->arrive_time =
simulation_run_get_time(simulation_run);
                                    next_call->call_duration = get_call_duration();
                                    server_put(channel, (void*)next_call);
                                    next_call->channel = channel;

       schedule_end_call_on_channel_event(simulation_run,
simulation_run_get_time(simulation_run) + get_call_duration(), (void *)channel);
                                    TRACE(printf("Waiting Call Completed"););
                                    break;
                            }}}}
```