# COMPENG 4DK4 Lab 3 Report

Aaron Pinto
pintoa9

Raeed Hassan
hassam41

November 2, 2022

## Random Number Generator Seeds

For the experiments in this lab, we used the same set of 18 random number seeds for all experiments. Experiment 2 instructs us to include runs with our *McMaster Student ID numbers* as our seeds. We used our *McMaster IDs* and shifted them by one digit at a time to create 9 different seeds from each our IDs, for a total of 18 different seeds. All the random number generator seeds can be seen in Table 1. In the C code used for the experiments, leading zeroes are removed.

| | |
|---|---|
| 400188200 | 400190637 |
| 001882004 | 001906374 |
| 018820040 | 019063740 |
| 188200400 | 190637400 |
| 882004001 | 906374001 |
| 820040018 | 063740019 |
| 200400188 | 637400190 |
| 004001882 | 374001906 |
| 040018820 | 740019063 |

Table 1: Random Number Generator Seeds

## Experiment 2

The set of curves that show the tradeoffs between blocking probability, offered load (in Erlangs) and the number of channels can be seen in Figure 1. We can observe that the blocking probability decreases as the number of channels increase, and increasing the offered load will increase the blocking probability at a given number of channels.

When plotting these results with the Erlang B formula in MATLAB, we observe the same results with essentially identical curves as seen in Figure 2. The MATLAB code used to calculate the Erlang B formula can be seen in Listing 1. We compared several of our results of from our program with an online Erlang B calculator and had matching results. For example, for $A = 5, N = 10$, both calculated $P_B = 0.184$.

Listing 1: Erlang B Formula

```
3 syms k
4
5 for A = 1:20
6         for N = 1:20
7                 PB(A,N) = ((A^N)/factorial(N))/symsum(A^k/
                      factorial(k),k,0,N);
8         end
9 end
```

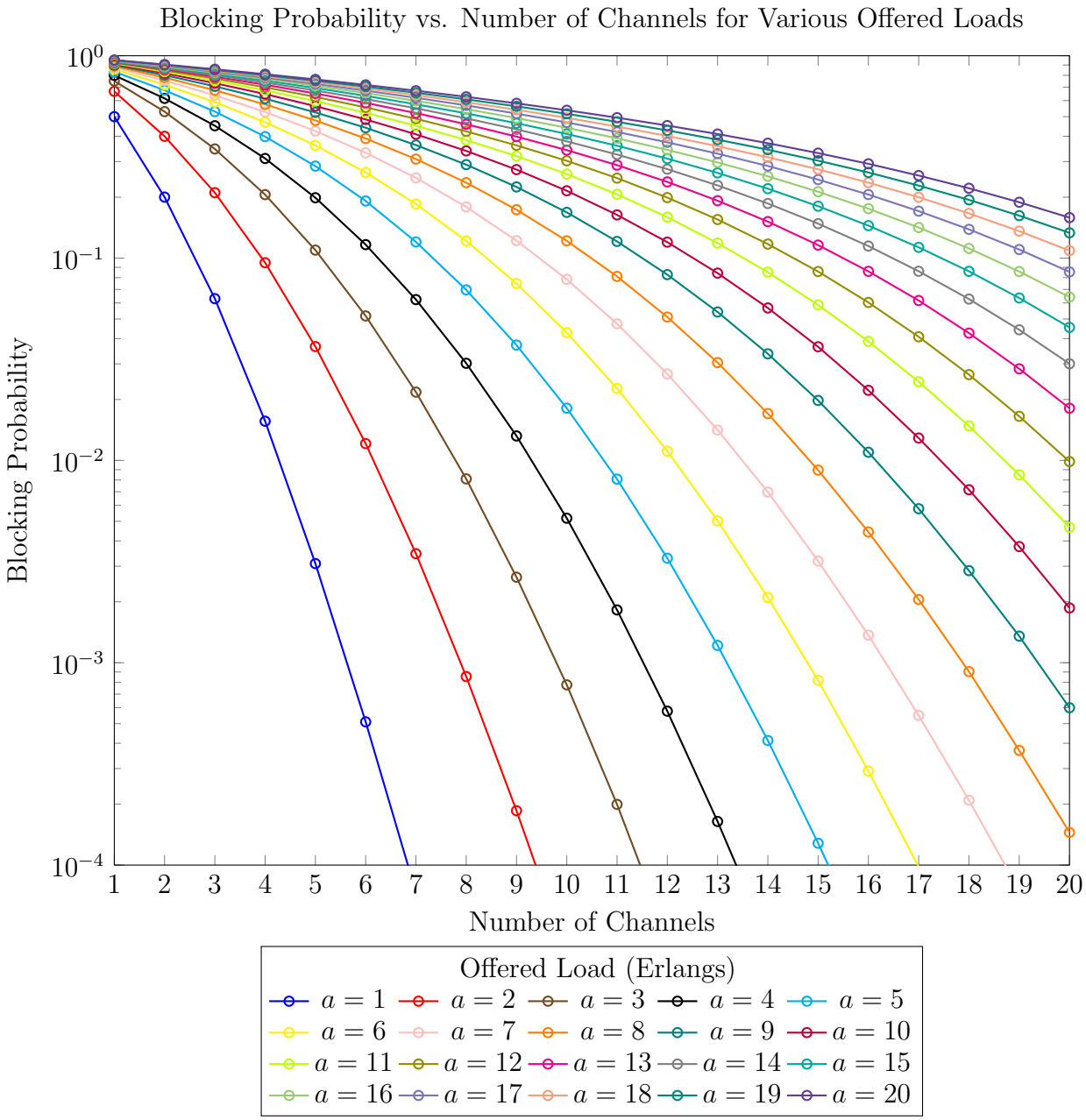


Figure 1: Experiment 2: Simulation

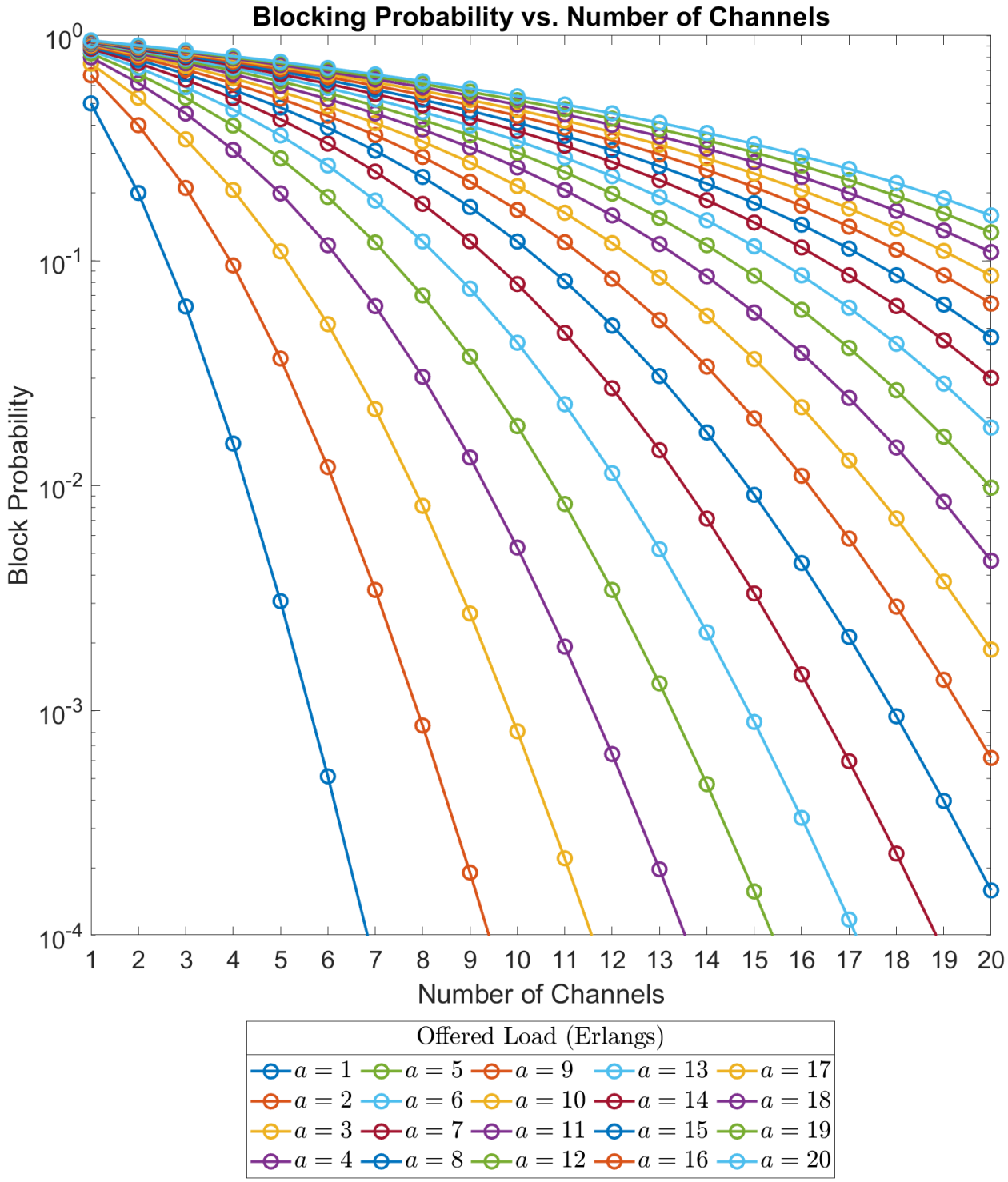


Figure 2: Experiment 2: MATLAB

## Experiment 3

The maximum offered loading (in Erlangs) versus the number of cellular channels needed to achieve a 1% blocking probability is graphed on Figure 3. The minimum number of cellular channels required to have an "acceptable" cellular network with less than 1% blocking probability is 5 cellular channels, which remains acceptable for a maximum offered load of 1 Erlang. The increase in the maximum offered loading follows demonstrates mostly linear growth.

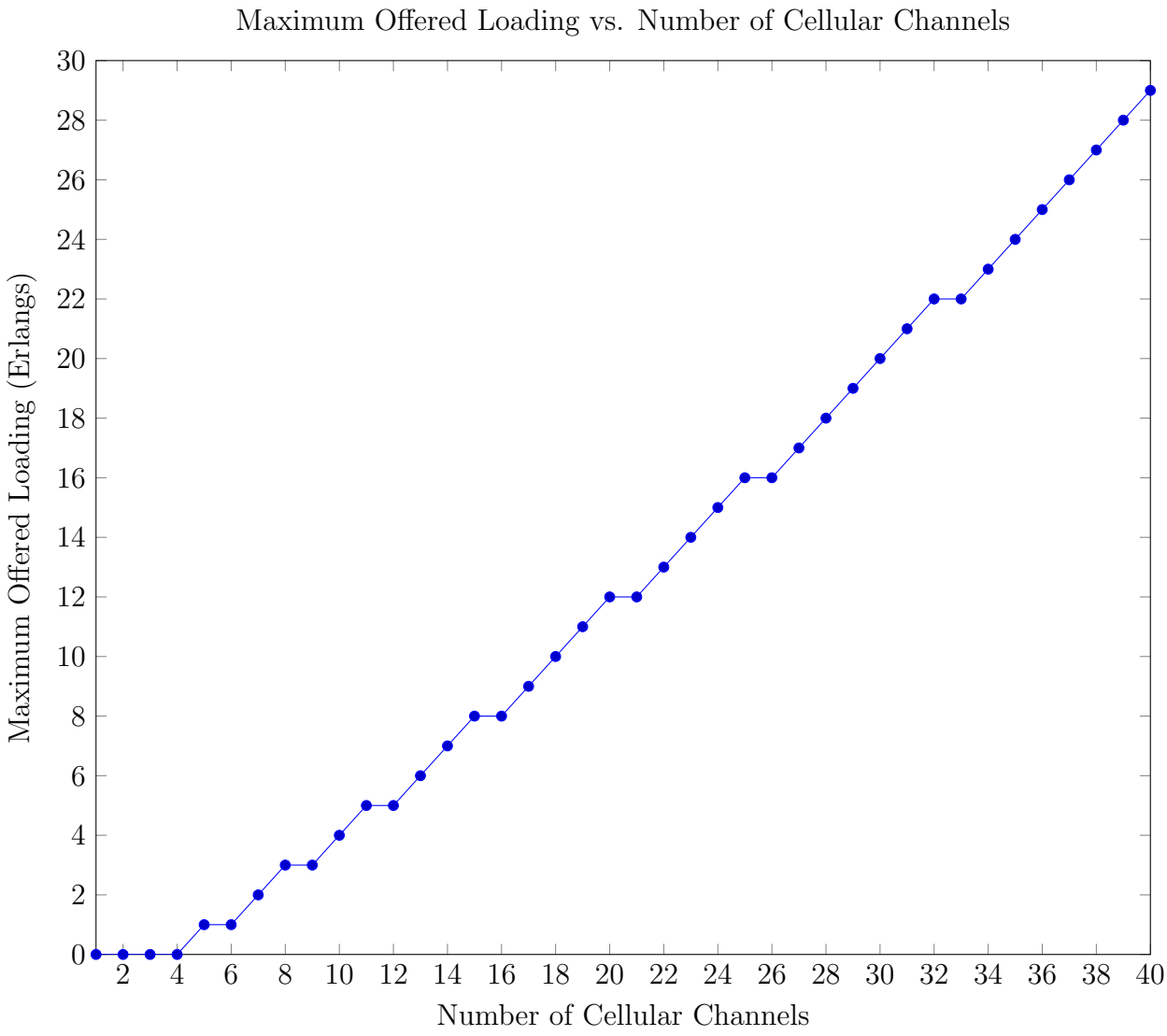


Figure 3: Experiment 3

## Experiment 4

To simulate this case, the code was modified to add a FIFO queue where calls would be placed on call arrival if all channels are full. This queue would be checked on call departure

(the only event that can free up a channel) and place a call from the queue that hasn't hung up into any empty channel. The modifications for call arrivals can be seen in Listing 2 and modifications for call departures can be seen in Listing 3.

The probability of not being served versus offered loading for different values of $w$ and $N$ is shown in Figure 4. We can see that the waiting probability for a given offered load decreases when the number of channels, $N$, increase, and does not demonstrate a significant change when the mean hang-up time, $w$, changes.

The mean time that customers have to wait on the queue versus offered loading for different values of $w$ and $N$ is shown in Figure 5. We can see that the mean waiting time for 1 Erlangs of offered load depends on the number of channels, $N$, and will decrease as $N$ increases. As the mean hang-up time, $w$, increases the mean queue wait time increases.

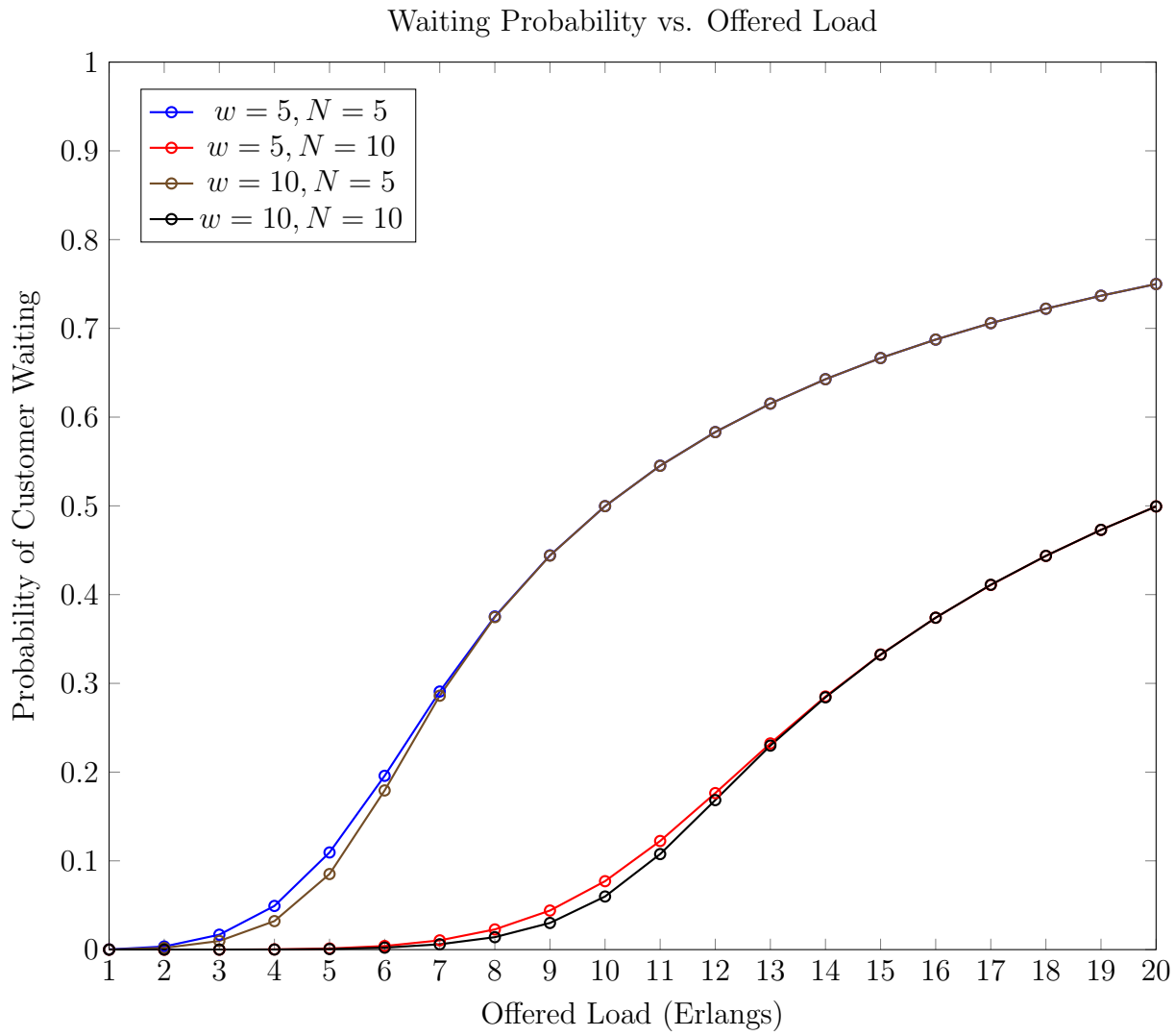


Figure 4: Experiment 4: Waiting Probability

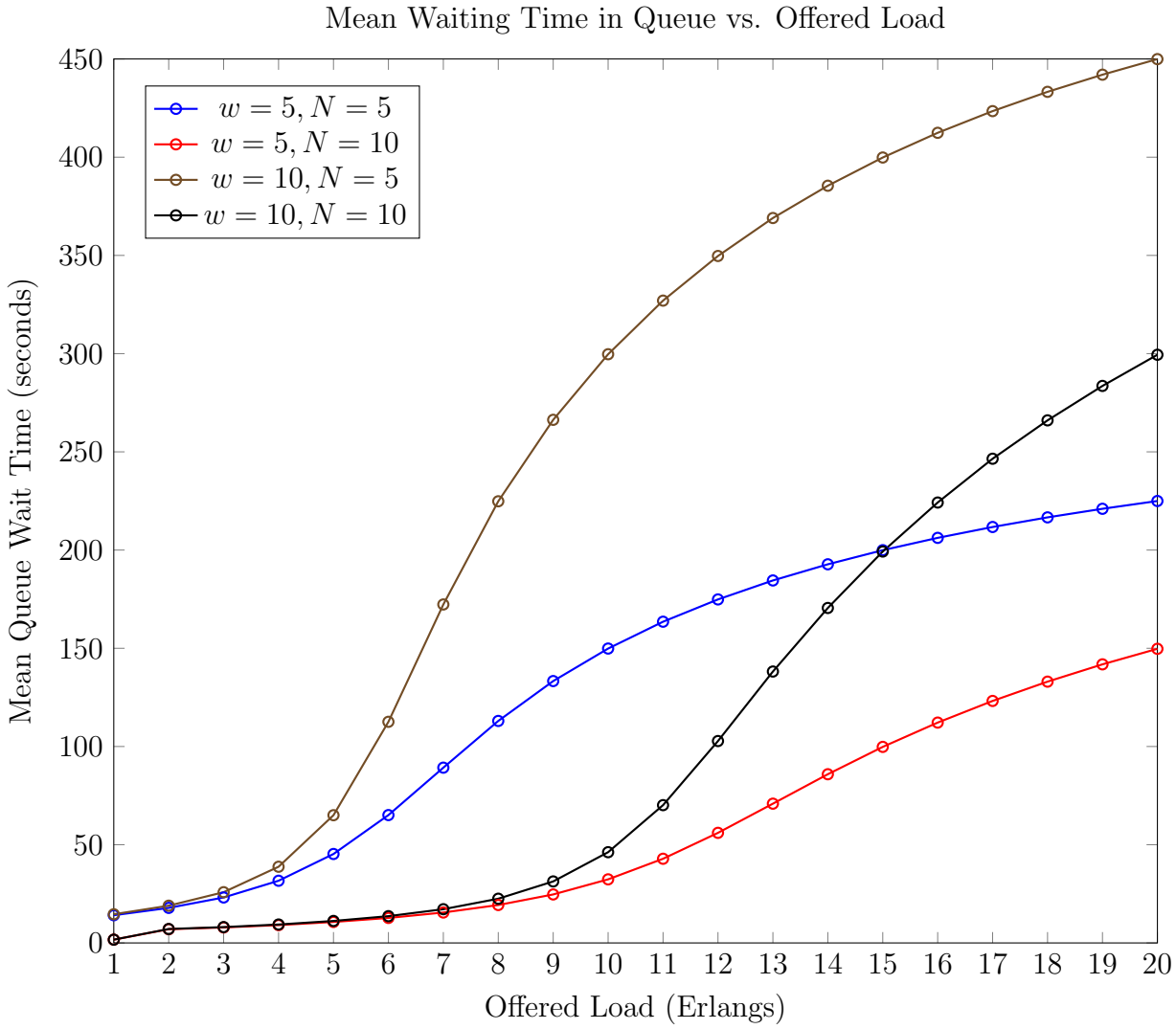


Figure 5: Experiment 4: Mean Delay

Listing 2: Wait on Call Arrival

```
} else {
        /* No free channel was found. The call is placed in queue.
           */
        fifoqueue_put(sim_data->buffer, (void *) new_call);
        sim_data->waited_call_count++;
}
```

Listing 3: Connect or Hang Up on Call Departure

```
1  if ((free_channel = get_free_channel(simulation_run)) != NULL) {
2          int queue_size;
3          if ((queue_size = fifoqueue_size(sim_data->buffer)) > 0) {
4                  for (int i = 0; i < queue_size; ++i) {
5                          new_call = (Call_Ptr) fifoqueue_get(
                               sim_data->buffer);
6
7                          if (new_call->hang_up_time < now) {
8                                  sim_data->call_hung_up_count++;
9                                  sim_data->
                                       accumulated_queue_wait_time +=
                                       new_call->hang_up_time -
                                       new_call->arrive_time;
10                                 xfree((void *) new_call);
11                                 new_call = NULL;
12                                 continue;
13                         }
14
15                         sim_data->accumulated_queue_wait_time +=
                               now - new_call->arrive_time;
16                         break;
17                 }
18
19                 // If the queue only contained customers that have
                       hung up, we have nothing to do
20                 if (new_call == NULL) {
21                         return;
22                 }
23
24                 new_call->arrive_time = now;
25                 new_call->call_duration = get_call_duration();
26
27                 /* Place the call in the free channel and schedule
                       its departure. */
28                 server_put(free_channel, (void *) new_call);
29                 new_call->channel = free_channel;
30
31                 schedule_end_call_on_channel_event(simulation_run,
                       now + new_call->call_duration, (void *)
                      free_channel);
32         }
33 }
```

## Experiment 5

To simulate this case, the code was modified to add a FIFO queue where calls would be placed on call arrival if all channels are full. This queue would be checked on call departure (the only event that can free up a channel) and place a call from the queue into any empty channel if the queue contains a call. The modifications for call arrivals can be seen in Listing 4 and modifications for call departures can be seen in Listing 5. The values for $P_w$, $T_w$, $W(1\text{min})$ were found in simulation and calculated with the given formulas for the following cases:

1. $\lambda = 2$, $h = 3$, $N = 10$
   Simulation results: $P_w = 0.1006$, $T_w = 0.0746$ min, $W(1\text{min}) = 0.9739$
   Calculated value: $P_w = 0.0974$, $T_w = 0.0730$ min, $W(1\text{min}) = 0.9743$
2. $\lambda = 7$, $h = 2$, $N = 15$
   Simulation results: $P_w = 0.7198$, $T_w = 1.4224$ min, $W(1\text{min}) = 0.5662$
   Calculated value: $P_w = 0.6891$, $T_w = 1.3783$ min, $W(1\text{min}) = 0.5820$
3. $\lambda = 2$, $h = 7$, $N = 15$
   Simulation results: $P_w = 0.7198$, $T_w = 4.9785$ min, $W(1\text{min}) = 0.3774$
   Calculated value: $P_w = 0.6891$, $T_w = 4.8239$ min, $W(1\text{min}) = 0.4026$

The results are largely similar. There are minor discrepancies in the results as not all customers that enter the queue will be served at the end of the simulation, resulting in some undercounting or overcounting of some statistics.

Listing 4: Wait on Call Arrival

```
} else {
        /* No free channel was found. The call is placed in queue.
           */
        fifoqueue_put(sim_data->buffer, (void *) new_call);
        sim_data->waited_call_count++;
}
```

Listing 5: Wait on Call Departure

```
// See if there is are calls waiting in the buffer and there is a
   free channel. If so, take the next one out and connect it
   immediately.
if ((fifoqueue_size(sim_data->buffer) > 0) && ((free_channel =
   get_free_channel(simulation_run)) != NULL)) {
        next_call = (Call_Ptr) fifoqueue_get(sim_data->buffer);

        next_call->call_duration = get_call_duration();
        next_call->waiting_time = now - next_call->arrive_time;

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

```
10          next_call->channel = free_channel;
11
12          schedule_end_call_on_channel_event(simulation_run,
13                                             now + next_call->
                                                   call_duration,
14                                             (void *) free_channel);
15  }
```