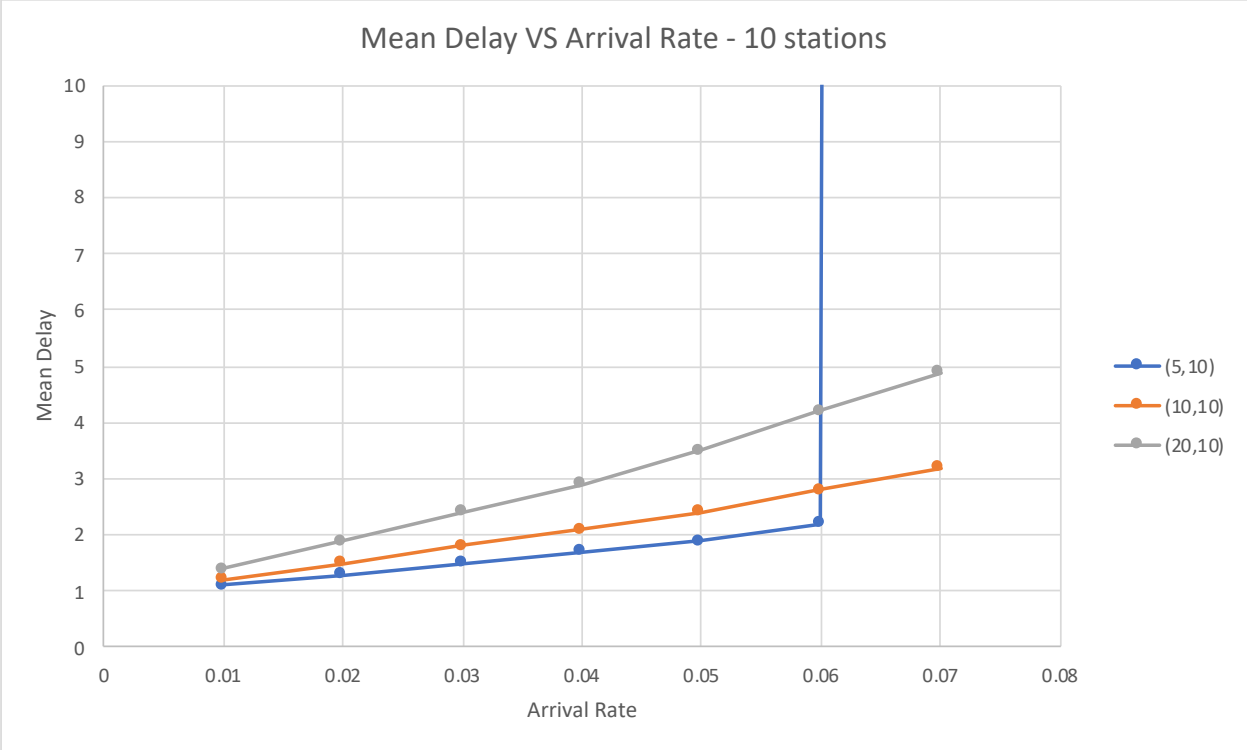
COMPENG 4DK4 LAB4

Richard Qiu – 400318681 – Group 66

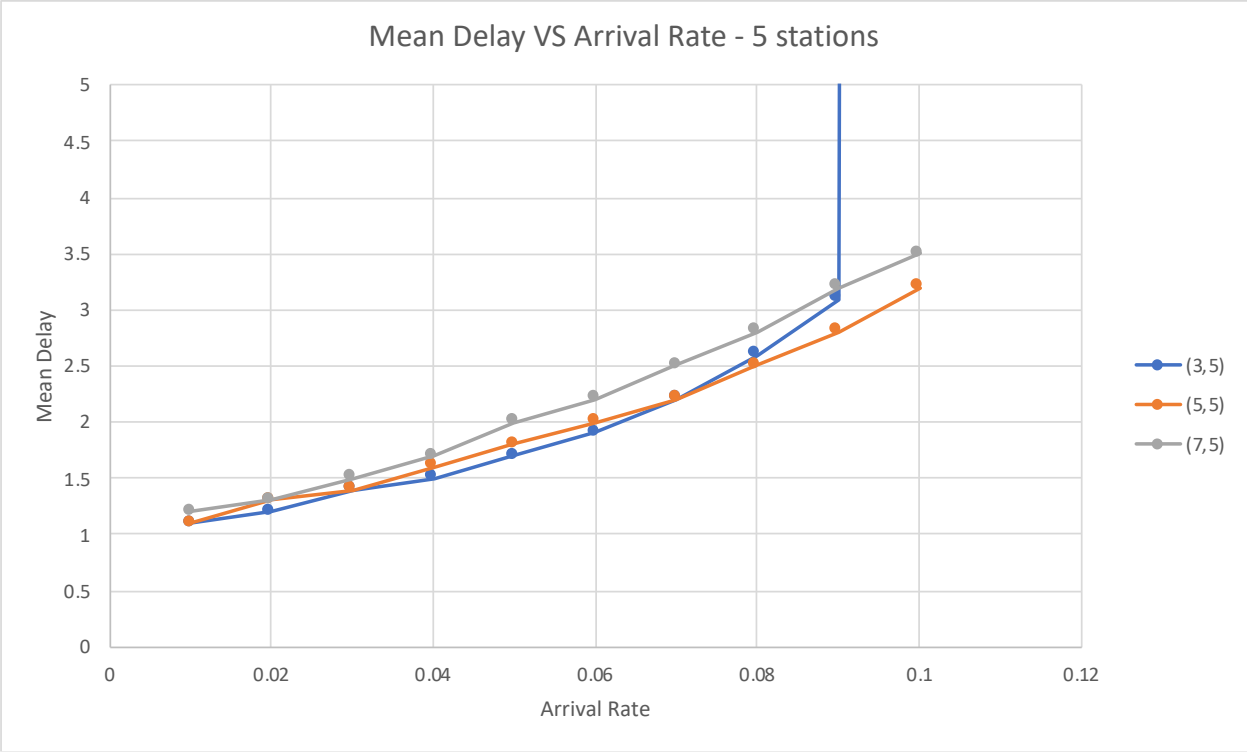
Experiment

2.

The obtained graph is shown below when the number of stations is 10 and the mean back off duration values are 5,10 and 20. The below shows the mean delay vs arrival rate between 0.1 and 0.7 incrementing by 0.01. As the arrival rate approaches 0.07, the mean delay when the back off duration value is 5 starting to become very large (infinite).



The obtained graph is shown below when the number of stations is 5 and the mean back off duration values are 5,10 and 20. The below shows the mean delay vs arrival rate between 0.1 and 0.7 incrementing by 0.01. As the arrival rate approaches 0.09, the mean delay when the back off duration value is 5 starting to become very large (infinite).



In conclusion from both graphs, we can see when the value of back off duration is lower. The mean delay will be lower. However, when the arrival rate increases till some points, the mean delay of the lower back off duration time will start becoming very large(infinite) which means the system is insufficient to support the retransmission with that back off duration time.

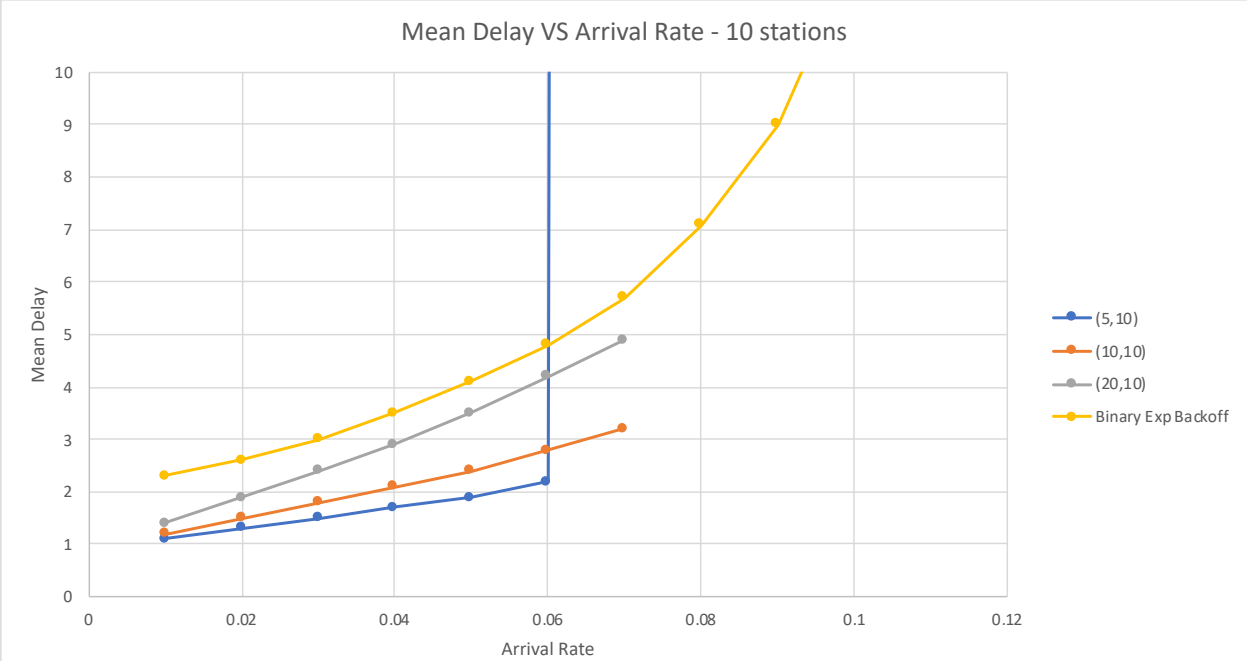
And when the number of stations decreases, the marginal arrival rate compare to higher number of stations where the mean delay starts becoming infinity is larger. This is reasonable as more stations will have more retransmissions happen when the arrival rate starts increasing.

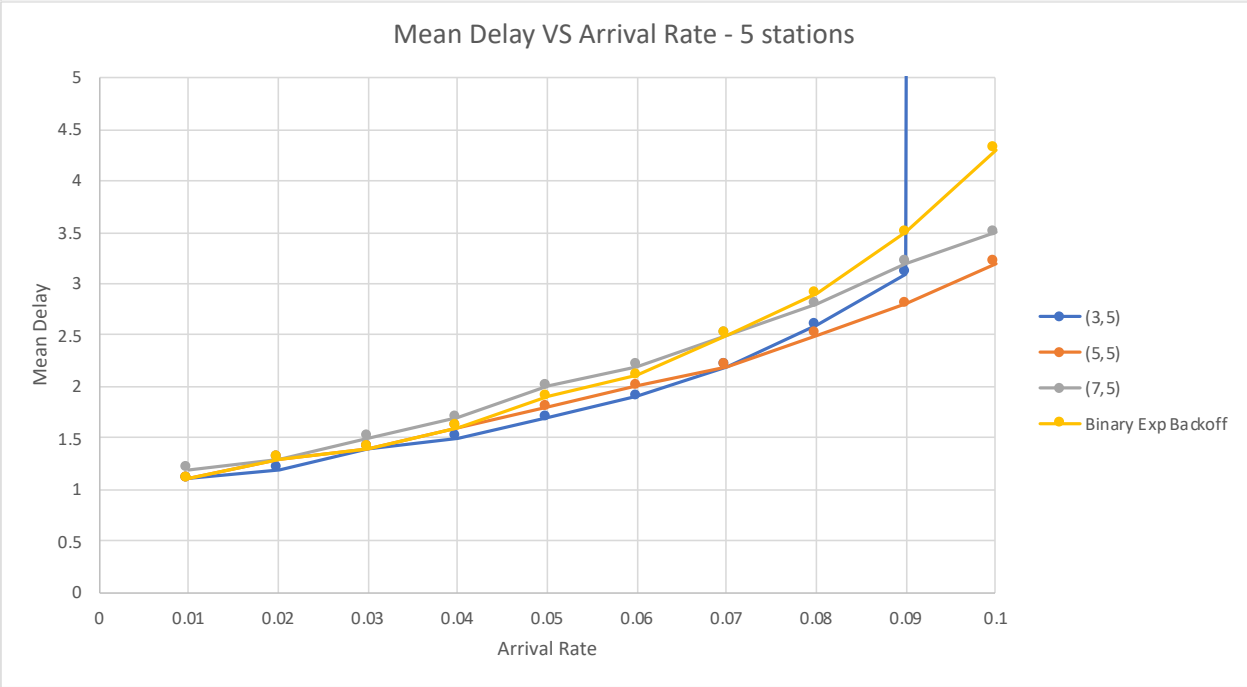
3.

To achieve the binary exponential backoff, we can simply change the following line as shown below.

```
backoff_duration = uniform_generator() * pow(2,this_packet->collision_count);
```

It will set the backoff duration uniformly in the range [0,2^Nc) where Nc is the number of collisions that the packet has suffered.
And we can compare the results from part 2. The new plots are shown below.

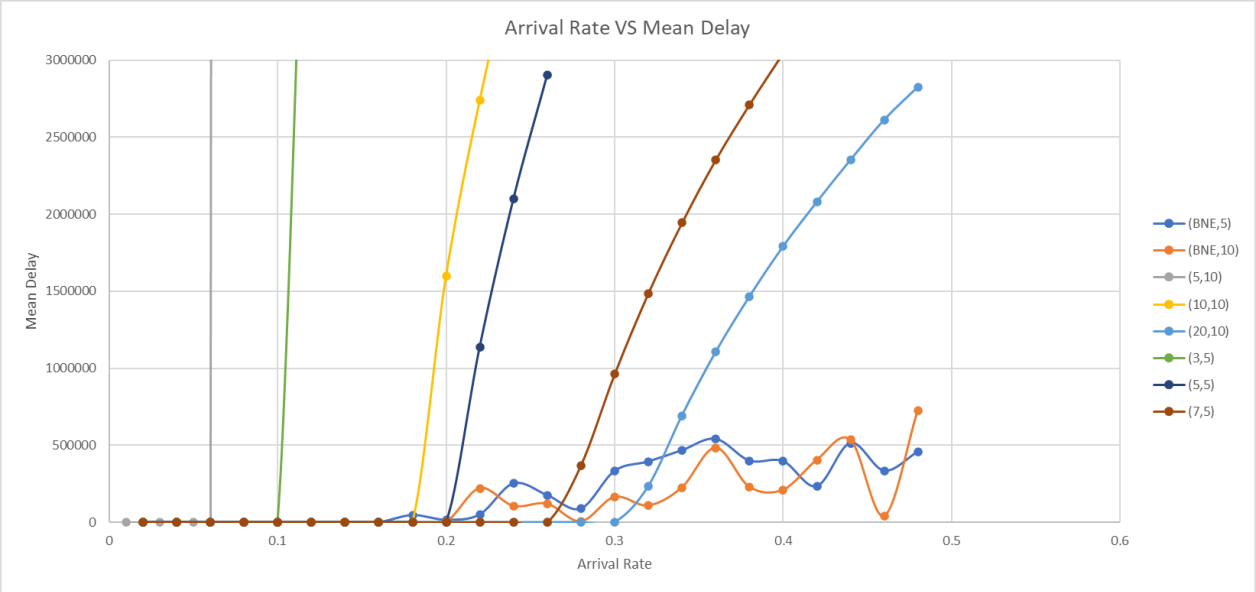




As we can see from both graphs, by using the binary exponential backoff will not provide us the minimum average mean delay. However, by using the binary exponential allow us to adapt dynamically based on the number of collisions which it will significantly reduce the chance of repeated collisions. Compare to the fixed mean backoff, there is not a threshold arrival rate for the binary exponential backoff which may cause the system result an infinite mean delay.

By extending the throughput input(arrival rate) to 0.5, we can further prove our conclusions, as we can see below, the fixed mean backoff duration will reach a threshold value and starting

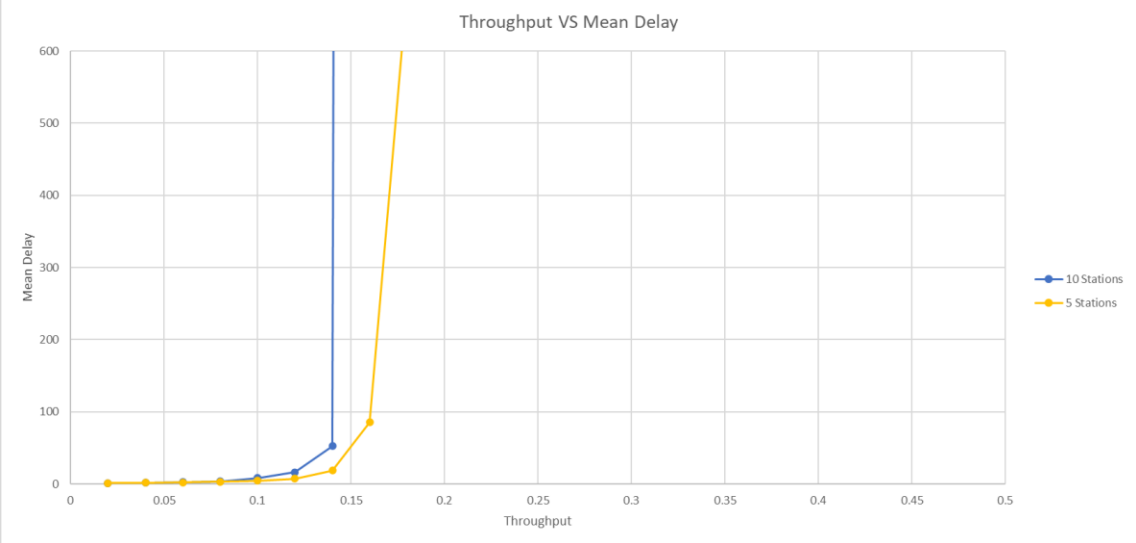
increasing the mean delay value extraordinary while the binary exponential backoff will try to adjust itself to decrease the mean values.



By investigating further, we can also generate the mean delay vs throughput curve.
In order to get the throughput, in the code, we can add this line at the end of the main.c file which we will use the number of successful transmission packets / the current time to get the throughput.

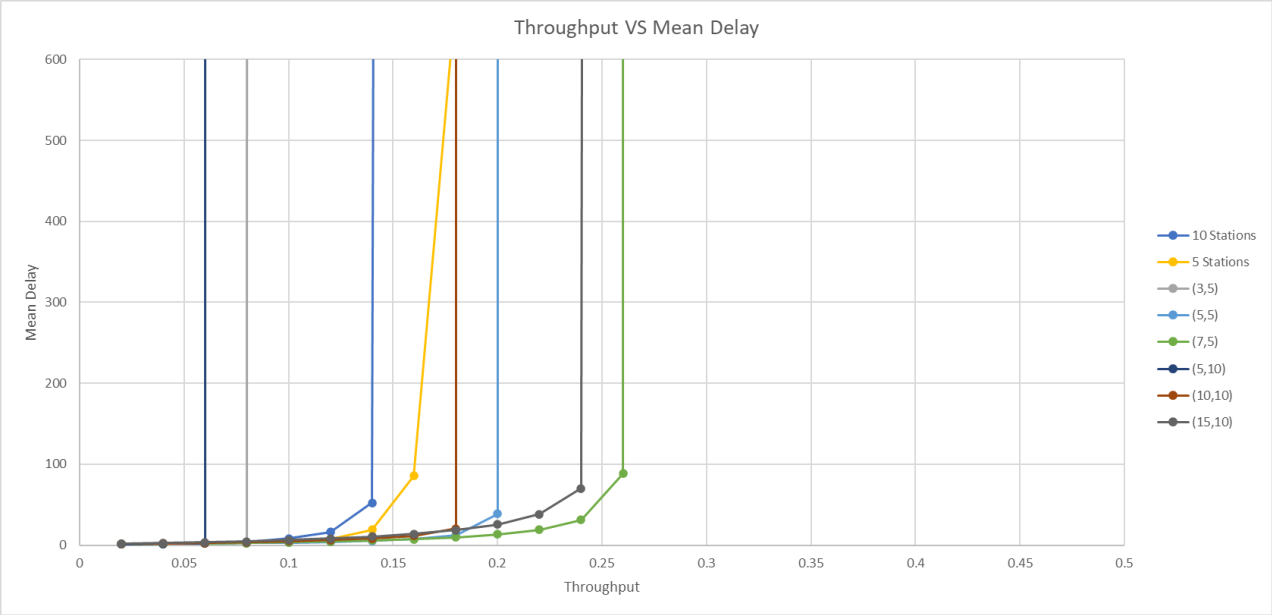
```
printf("numof packet is = %10ld, the S is = %f \n",data.number_of_packets_processed, data.number_of_packets_processed / simulation_run->clock->time);
```

And by collecting all the data, we can get the 10 and 5 stations using binary exponential backoff curve as shown below.



As we can seen from this graph, fewer number stations will result in a better throughput. This is reasonable as the number of stations increase, the possibility of collisions also increases which it will reduce the overall throughput of the channel. We can also observe the result is close to the theoretical aloha max throughput (18.4%), but it won't exceed this theoretical value with a low mean delay.

I also run throughput simulations from part2, the result is shown below.



When we are using the fixed mean backoffs, we can find the max throughput is actually over the theoretical aloha max throughput (18.4%), this is because the assumption for the aloha max throughput is made when the arrivals and retransmissions are Poisson process with rate G. But when the backoff duration is a fixed value, the retransmit will not be a Poisson anymore, which is also why when the arrival rate reaches some threshold, the mean delay is going to be infinity.

4.

To modify the simulation from Part3 so that only one particular station will always retransmit an unsuccessful packet in the very next slot. We can modify the code as shown below. In this code, I set that particular station equals to station number 3. And when it detects the channel transmission is unsuccessful, it will then detect where this unsuccessful packet is coming from which if the packet is coming from the station number 3, it will schedule the retransmission immediately and there will be no backoff at all.

```
if(this_packet->station_id == 3){
  // backoff_duration = uniform_generator() * pow(2,this_packet->collision_count);

  schedule_transmission_start_event(simulation_run,
                now,
                (void *) this_packet);
}else{
  backoff_duration = uniform_generator() * pow(2,this_packet->collision_count);

  schedule_transmission_start_event(simulation_run,
                now + backoff_duration,
                (void *) this_packet);
}
```

After run several simulations, I found that for that particular station channel 3, the mean delay is relevant low compare to other stations mean delay, this is reasonable since we have assigned the channel 3 will retransmit immediately if it fails.

One of the example is showing below when the arrival rate is 0.1 and the number of stations is equal to 10 and 5.

Station #	Mean Delay
0	5.6
1	5.8
2	6
3	2.1
4	6.3
5	5.7
6	9.9
7	5.8
8	6.9
9	5.6

Table 1 Number of stations = 10, packet arrival rate = 0.1

Station #	Mean Delay
0	4.4
1	4.2
2	4.2
3	1.8
4	4.2

Table 2 Number of stations = 5, packet arrival rate = 0.1

5.

In order to implement an aloha, we basically need to generate slots on the channel and determine which slot the packet should at when it arrives. And also the guard time Epsilon is set to 0.01 in this part to secure the connection.

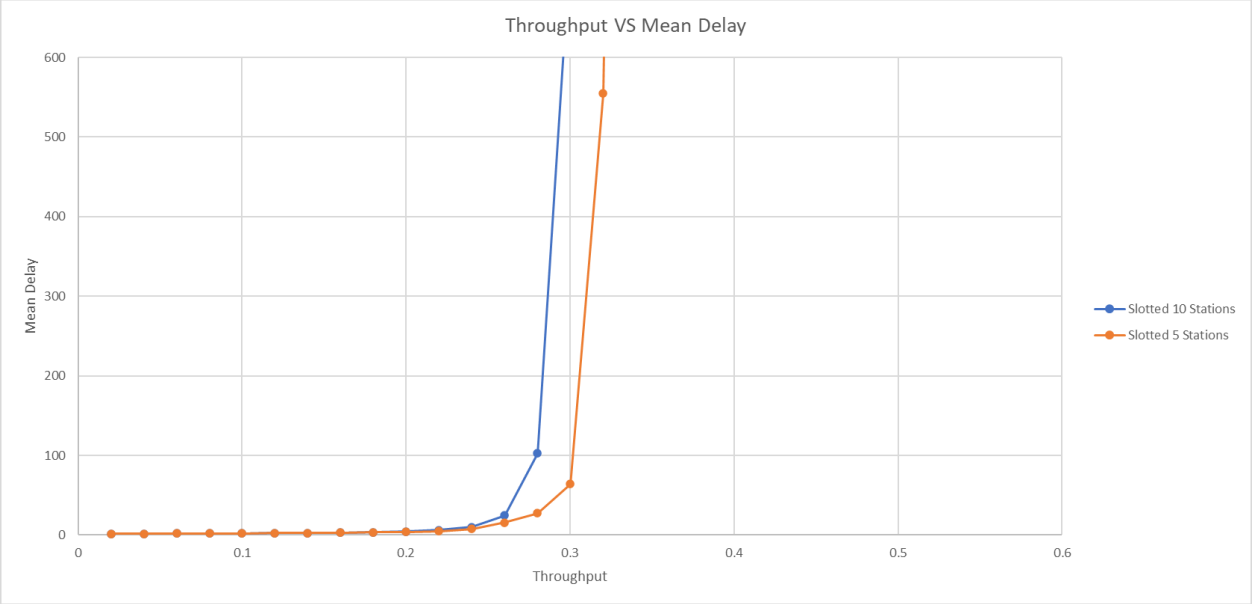
Below is the function schedule what the slot time the packet should start in the channel.

```
//event time is a double type
double find_Next_slot_time(Time curr_time){
  double slot_time = MEAN_PACKET_DURATION + 2 * Epsilon;
  double remain = curr_time / slot_time;
  double next_slot_time;
  double slot_index;
  if(remain > 0){
    if (remain - (int)remain > 0){
      slot_index = (int)remain + 1;
    }else{
      slot_index = (int)remain;
    }

  }else{
    slot_index = (int)remain;
  }
  next_slot_time = slot_time * slot_index + Epsilon;
  return next_slot_time;
}
```

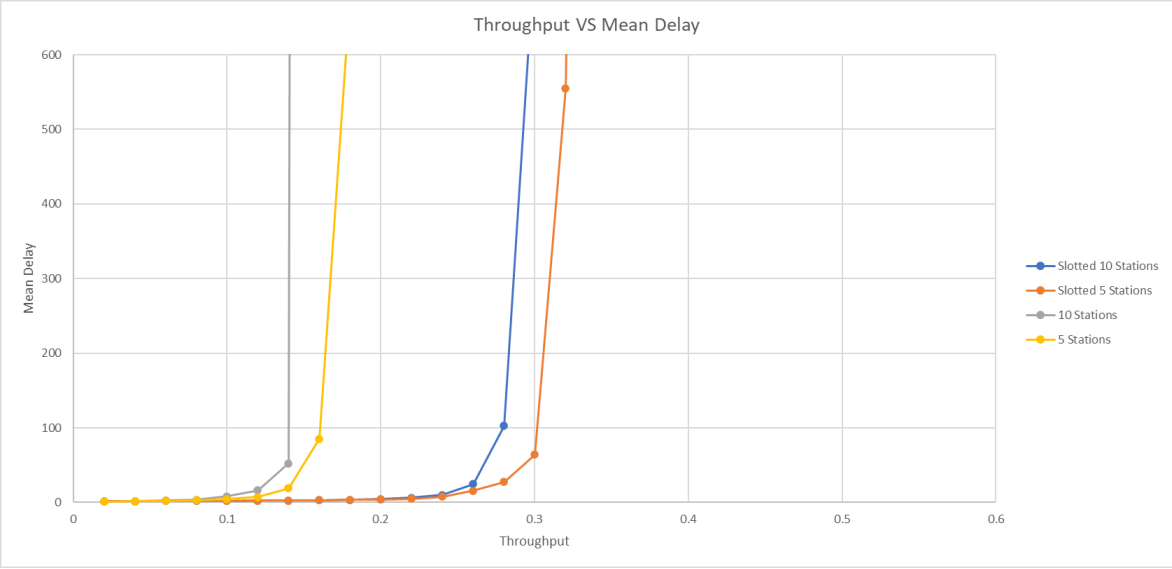
It first assigns the time for each slot which is the packet duration plus the start and end epsilons. Then I am finding the remainder by dividing the current time with the slotted time, and based on the C math rule, we firstly check if the remain is greater than zero, if it is not, which means the packet just started, it will set the slot index equal to zero which is the type casting here convert to int. If the remain is zero, it will check if there are any decimal points in this double value. If they have any digits, we need to increase the slot index by 1, else if the remain has no digits in the decimal point, which means it is an integer so we can directly use that slot index for that packet. And finally we will multiple this slot index with the slot time and plus the start epsilon to get the final packet start time. The modified function call should appear below.

```
long int
schedule_transmission_start_event(Simulation_Run_Ptr simulation_run,
                                  Time event_time,
                                  void * packet)
{
  Event event;

  event.description = "Start Of Packet";
  event.function = transmission_start_event;
  event.attachment = packet;

  return simulation_run_schedule_event(simulation_run, event,
  find_Next_slot_time(event_time));

}
```

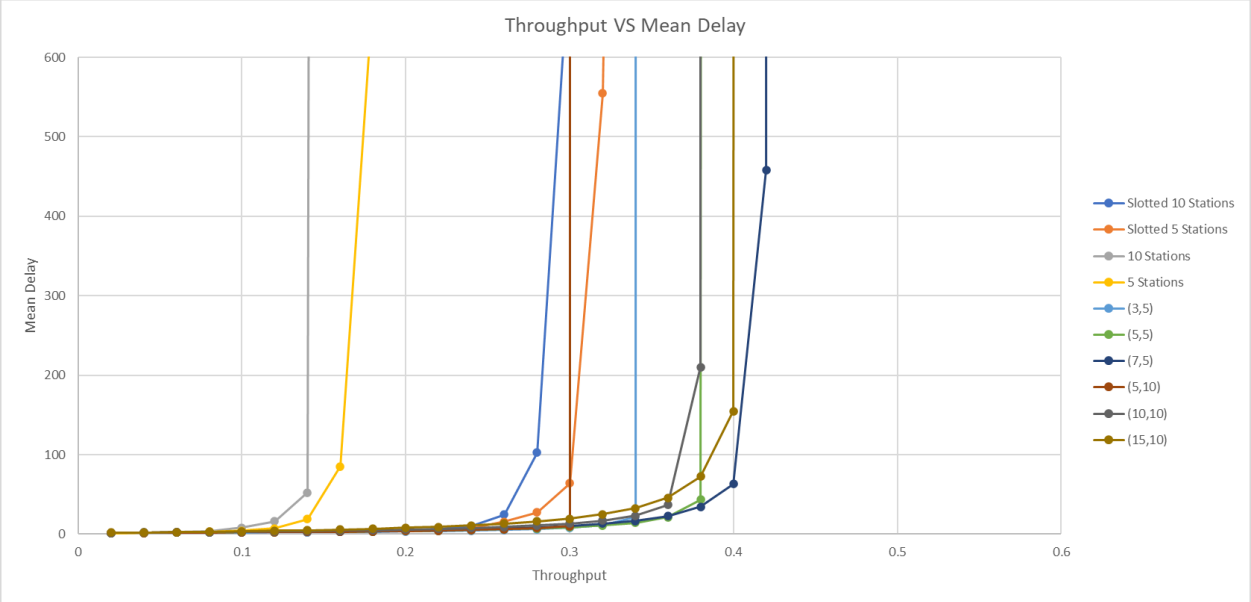
By running the simulation and collecting the data, the throughput vs mean delay curve is shown below.



As we can seen from the graph, fewer number of stations still have the better throughput. This proves our theory from part 3 which more number of stations will cause the increase of the possibility of the collision which will reduce the throughput. And by comparing the part3 with the part5 curve below. We can see by using the slotted aloha, the throughput is improved nearly twice. And the throughput of the slotted aloha is approaching the maximum slotted aloha throughput value (36.8%).



And I also run the simulations with the fixed mean backoff, the graph is shown below.



By comparing this graph with the graph from the part3, we can find all the throughputs have improved which proves the design of this slotted aloha will improve the channel efficiency on processing the data packets.