# Computer Engineering 4DK4

## Lab 4: ALOHA MAC & Packet Reservation

Instructor: Prof. T. Todd

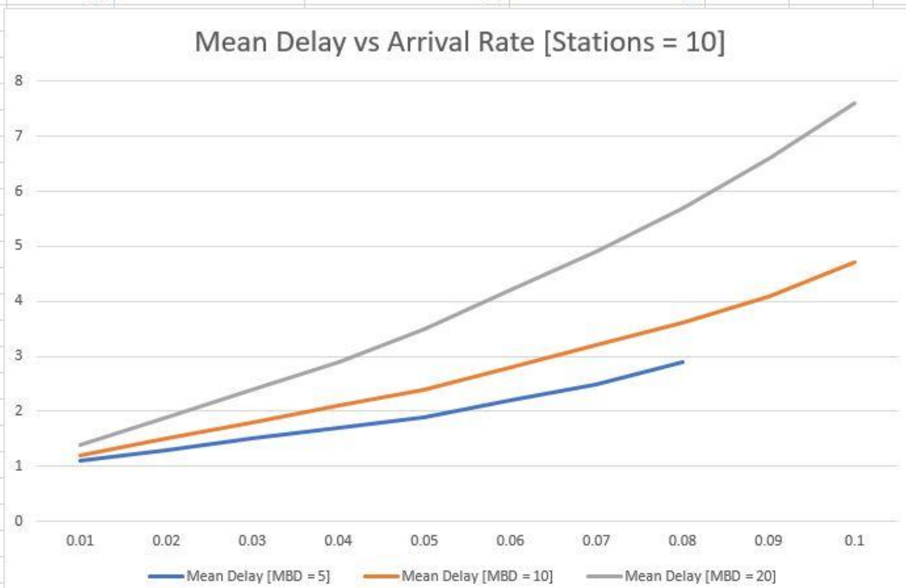
Moshiur Howlader - 1316948

Ankur Bargotra – 1306226

2)

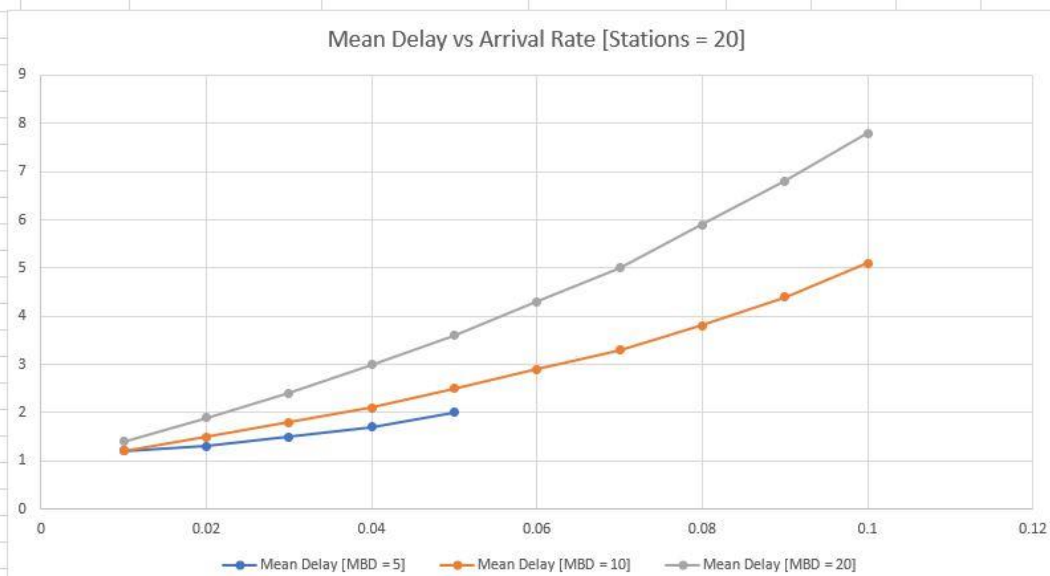
**Mean Delay vs Arrival Rate [Stations = 10]**

| Arrival Rate | Mean Delay [MBD = 5] | Mean Delay [MBD = 10] | Mean Delay [MBD = 20] | |
|---|---|---|---|---|
| 0.01 | 1.1 | 1.2 | 1.4 | Mean Packet Duration: 1 |
| 0.02 | 1.3 | 1.5 | 1.9 | Runlength: 7000000 |
| 0.03 | 1.5 | 1.8 | 2.4 | Bliprate: 100000 |
| 0.04 | 1.7 | 2.1 | 2.9 | Random Seed: 1306226 |
| 0.05 | 1.9 | 2.4 | 3.5 | |
| 0.06 | 2.2 | 2.8 | 4.2 | |
| 0.07 | 2.5 | 3.2 | 4.9 | |
| 0.08 | 2.9 | 3.6 | 5.7 | |
| 0.09 | | 4.1 | 6.6 | |
| 0.1 | | 4.7 | 7.6 | |




**Mean Delay vs Arrival Rate [Stations = 20]**

| Arrival Rate | Mean Delay [MBD = 5] | Mean Delay [MBD = 10] | Mean Delay [MBD = 20] | |
|---|---|---|---|---|
| 0.01 | 1.2 | 1.2 | 1.4 | Mean Packet Duration: 1 |
| 0.02 | 1.3 | 1.5 | 1.9 | Runlength: 7000000 |
| 0.03 | 1.5 | 1.8 | 2.4 | Bliprate: 100000 |
| 0.04 | 1.7 | 2.1 | 3 | Random Seed: 1306226 |
| 0.05 | 2 | 2.5 | 3.6 | |
| 0.06 | | 2.9 | 4.3 | |
| 0.07 | | 3.3 | 5 | |
| 0.08 | | 3.8 | 5.9 | |
| 0.09 | | 4.4 | 6.8 | |
| 0.1 | | 5.1 | 7.8 | |



According to the graphs and data acquired, there is a correlation between the parameters.

Decreasing the number of stations, allows an increased arrival rate until we reach an asymptotic value for mean delay, this is because with less stations there are less collisions.

Increasing the mean back duration, allows an increased arrival rate until we reach an asymptotic value for mean delay.

Also observed, for a fixed mean back off duration, as the number of stations is increased leads to higher mean delay values for the given arrival rate due to more collisions.

Thus, to conclude, if you add more stations you also need to decrease the mean back duration, to lower the collision rate, to give reasonable values for mean delay.

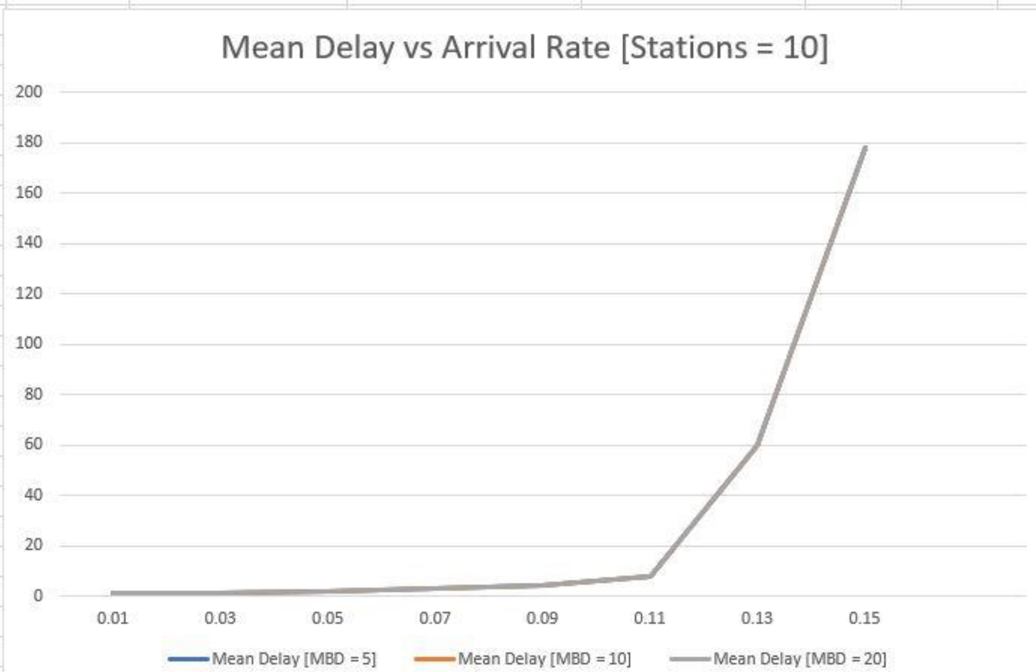
3)

Code Changes In 'packet_transmission.c':

```
backoff_duration = uniform_generator() * pow(2.0, (double)this_packet->collision_count);
```

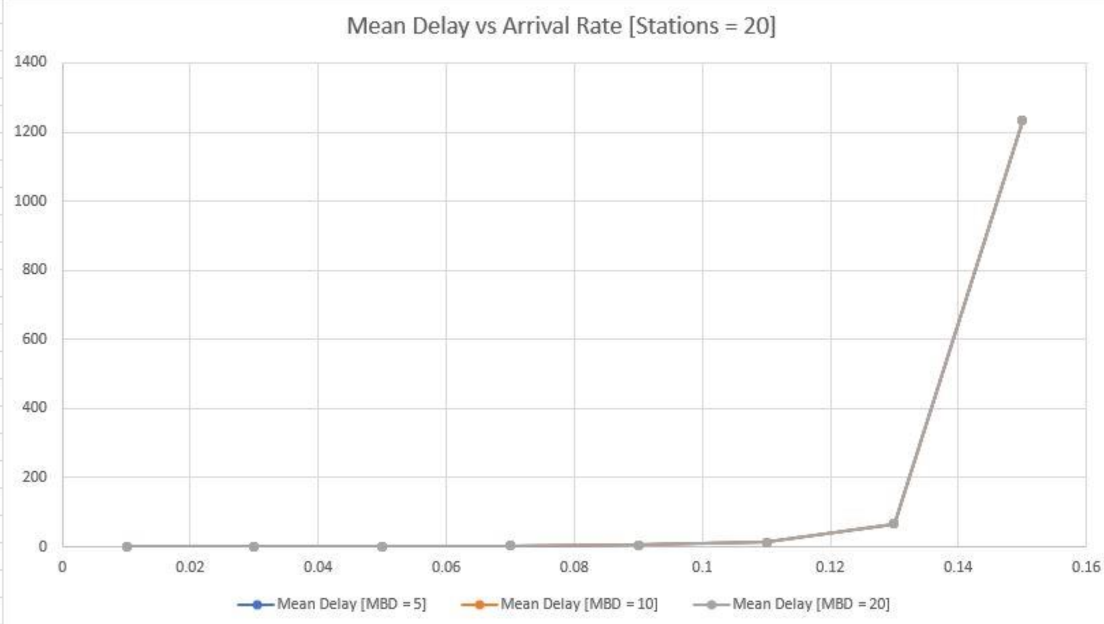
**Mean Delay vs Arrival Rate [Stations = 10]**

| Arrival Rate | Mean Delay [MBD = 5] | Mean Delay [MBD = 10] | Mean Delay [MBD = 20] | |
|---|---|---|---|---|
| 0.01 | 1.1 | 1.1 | 1.1 | Mean Packet Duration: 1 |
| 0.03 | 1.5 | 1.5 | 1.5 | Runlength: 7000000 |
| 0.05 | 2 | 2 | 2 | Bliprate: 100000 |
| 0.07 | 2.9 | 2.9 | 2.9 | Random Seed: 1306226 |
| 0.09 | 4.4 | 4.4 | 4.4 | |
| 0.11 | 8.1 | 8.1 | 8.1 | |
| 0.13 | 60 | 60 | 60 | |
| 0.15 | 178 | 178 | 178 | |



**Mean Delay vs Arrival Rate [Stations = 20]**

| Arrival Rate | Mean Delay [MBD = 5] | Mean Delay [MBD = 10] | Mean Delay [MBD = 20] | |
|---|---|---|---|---|
| 0.01 | 1.1 | 1.1 | 1.1 | Mean Packet Duration: 1 |
| 0.03 | 1.5 | 1.5 | 1.5 | Runlength: 7000000 |
| 0.05 | 2.1 | 2.1 | 2.1 | Bliprate: 100000 |
| 0.07 | 3.1 | 3.1 | 3.1 | Random Seed: 1306226 |
| 0.09 | 4.9 | 4.9 | 4.9 | |
| 0.11 | 12.9 | 12.9 | 12.9 | |
| 0.13 | 65.9 | 65.9 | 65.9 | |
| 0.15 | 1231.9 | 1231.9 | 1231.9 | |




Observed, for a fixed number of stations, for any mean back off duration the mean delay remains the same.

When we used a fixed mean back off in question 2, the results depended directly on the packet arrival rate and mean packet duration.

With a random exponential back off, results are no longer dependent on any of the parameters, just the Poisson process.

Also observed, similar to question 2, as the number of stations increases, the asymptote for mean delay moves closer to 0 (arrival rate and number of stations are inversely correlated).

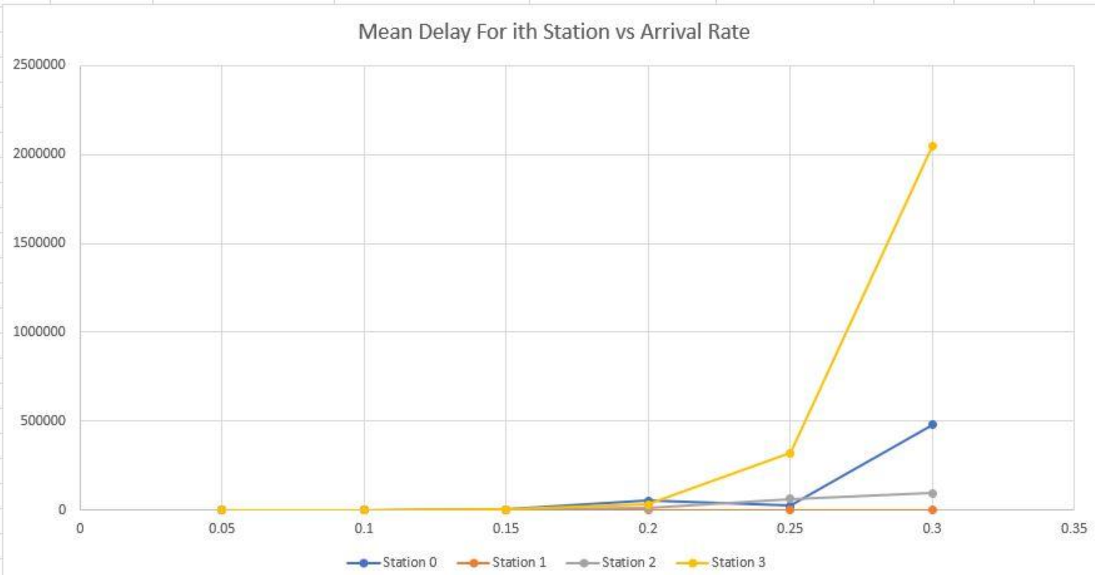
4)

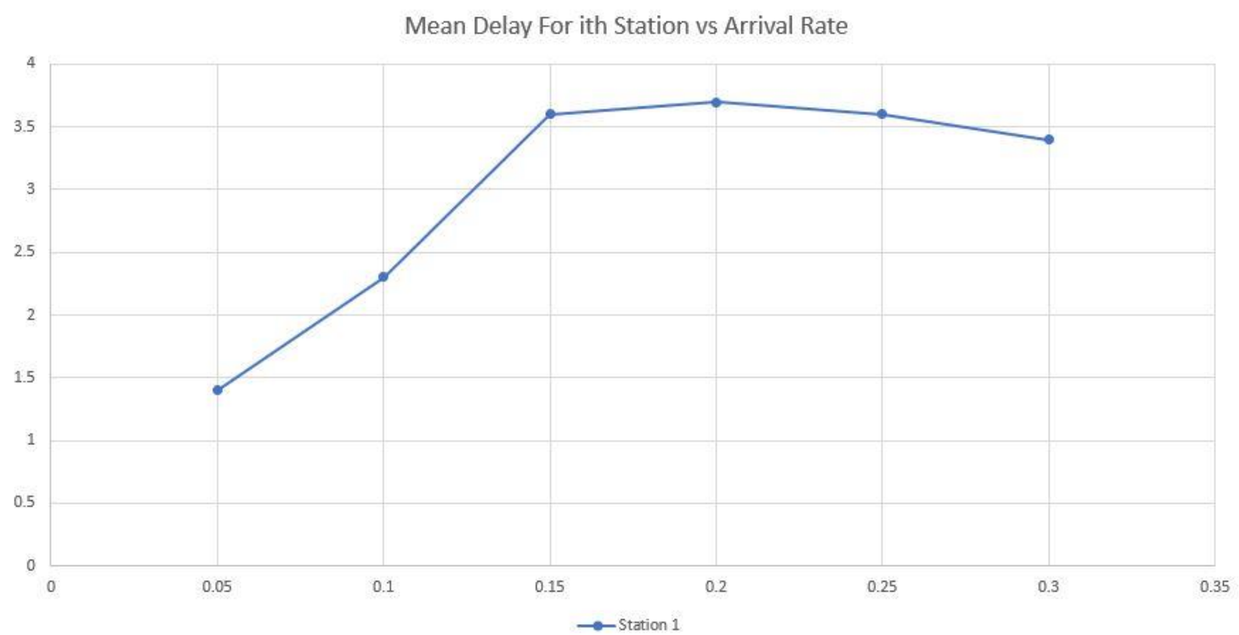
Code Changes In ‘packet_transmission.c’:

```
if(this_packet->station_id == 1){
    schedule_transmission_start_event(simulation_run, now,(void *) this_packet);
}
else{
    backoff_duration = uniform_generator() * pow(2.0, (double)this_packet->collision_count);
    schedule_transmission_start_event(simulation_run, now + backoff_duration, (void *) this_packet);
}
```

**Mean Delay vs Arrival Rate**

| Arrival Rate | Station 0 | Station 1 | Station 2 | Station 3 | |
|---|---|---|---|---|---|
| 0.05 | 2.1 | 1.4 | 2.1 | 2.1 | Mean Packet Duration: 1 |
| 0.1 | 6.8 | 2.3 | 6.5 | 6.8 | Runlength: 7000000 |
| 0.15 | 926.8 | 3.6 | 1142.4 | 1198.3 | Bliprate: 100000 |
| 0.2 | 53110.3 | 3.7 | 8663 | 30248.7 | Random Seed: 1306226 |
| 0.25 | 23369.9 | 3.6 | 61394 | 320589.9 | Number Of Stations: 20 |
| 0.3 | 478633.7 | 3.4 | 92710.2 | 2046170.4 | Mean Backoff Duration: 10 |




In code, we set station 1 not to have a back off duration. Compared to other stations its mean delay is *considerably* less for any arrival rate.

5)

Code Changes In 'simparameters.h':

```
#define NUMBER_OF_STATIONS 5
#define MEAN_PACKET_DURATION 1      /* normalized packet Tx time */
#define PACKET_ARRIVAL_RATE 0.01  /* packets per Tx time */
#define MEAN_BACKOFF_DURATION 10   /* in units of packet transmit time, Tx */
#define RUNLENGTH 7000000
#define BLIPRATE 100000
#define GUARD_TIME 0.01
```

Code Changes In ‘packet_arrival.c’:

```
schedule_transmission_start_event(simulation_run, now+GUARD_TIME, (void *) new_packet);
```

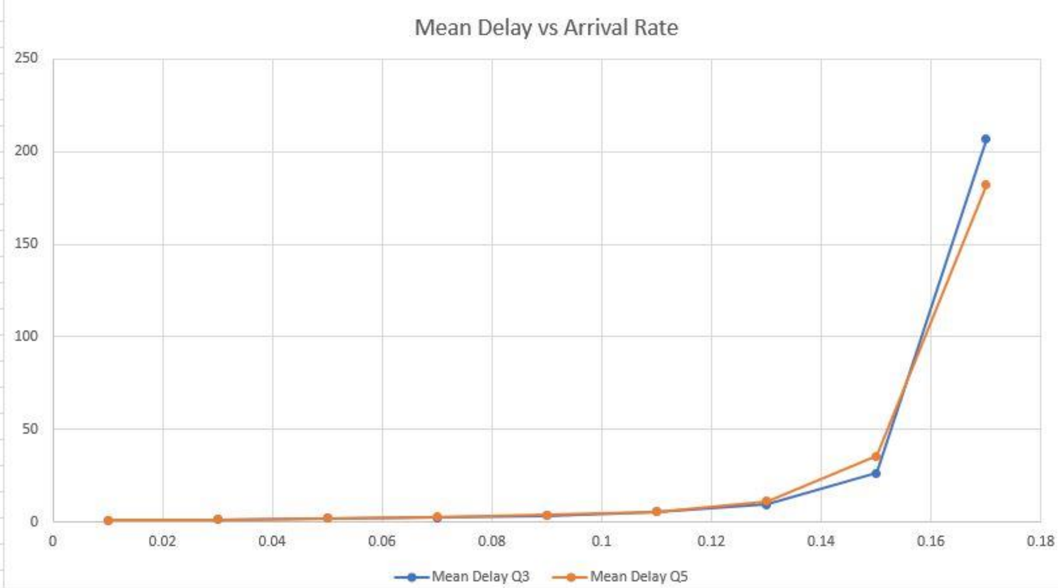
Code Changes In ‘packet_transmission.c’:

```
schedule_transmission_start_event(simulation_run,
                                  now+GUARD_TIME,
                                  (void*) next_packet);
```

**Mean Delay vs Arrival Rate**

| Arrival Rate | Mean Delay Q3 | Mean Delay Q5 |
|---|---|---|
| 0.01 | 1.1 | 1.1 |
| 0.03 | 1.4 | 1.4 |
| 0.05 | 1.9 | 1.9 |
| 0.07 | 2.5 | 2.6 |
| 0.09 | 3.5 | 3.6 |
| 0.11 | 5.5 | 5.4 |
| 0.13 | 9.4 | 11 |
| 0.15 | 26.5 | 35.6 |
| 0.17 | 206.9 | 181.9 |

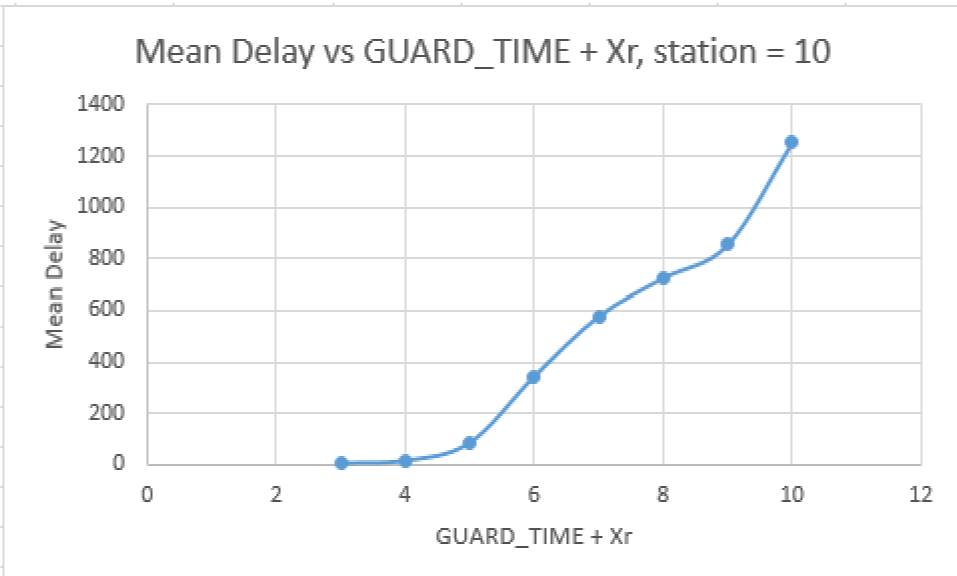
Mean Packet Duration: 1
Runlength: 7000000
Bliprate: 100000
Random Seed: 1306226
Number Of Stations: 5
Mean Backoff Duration: 10
Guard Time: 0.01



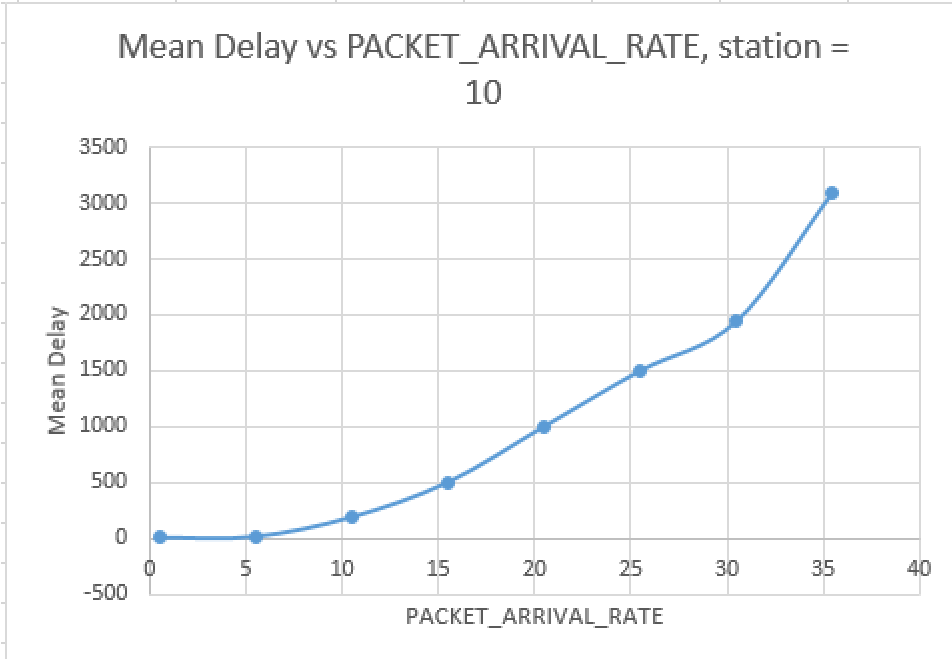

To convert to slotted-ALOHA, guard times were added to the start of a packet transmission in packet_arrival at to the end of a packet transmission (start of the next packet) in packet_transmission. After experimenting with different guard times, 0.01 was small enough that it had negligible effect on overhead.

It is observed that for higher arrival rates, in our case 0.15 the slotted-ALOHA has less mean delay than ALOHA protocols. For lower arrival rates they perform nearly the same.

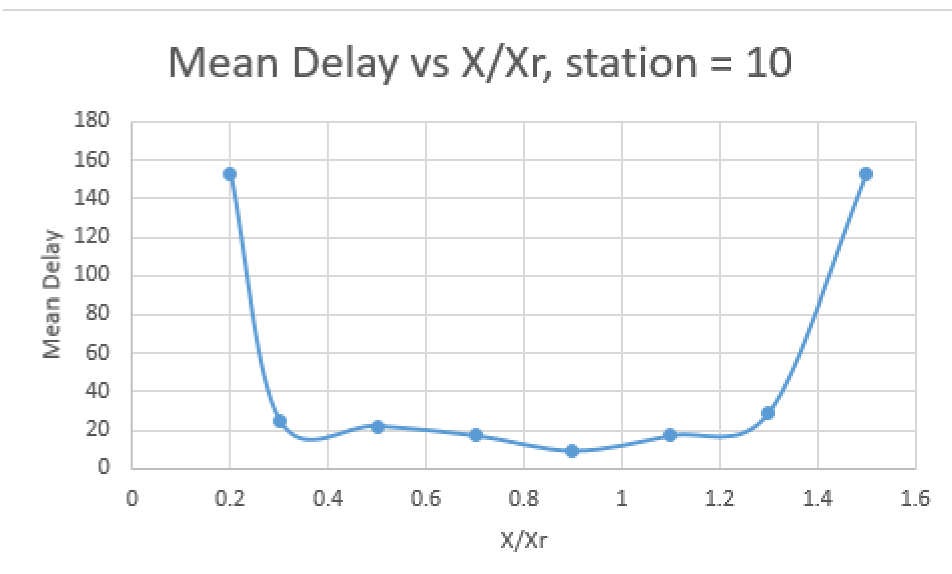
6. It is asked to simulate a scenario where N stations has a shared data channel. The S-Aloha contains continuous reservation mini-slots of duration Xr seconds. Then the packet is processed through a FCFS transmission through the data channel. In other words, there must be an intermediate S-ALOHA channel that must process station packets before being transmitted through the data channel.




The intermediate "buffer" or S-ALOHA channels can cause a delay or a stall in the overall speed of the packet transmission of the system. As we increase the GUARD_TIME and the S-ALOHA channel duration Xr, we notice that the mean delay increases. These guarding, and mini-slot delays cause an overall delay in the transmission of the packets.




When we compare our mean delay to packet arrival rate, we notice that we do obtain a predictable curve for the mean delay as packet arrival rate increases.




The X/Xr ratio indicates the ratio between duration of the main channel to the S-Aloha channel. If the ratio between them is lower than one, the mean delay is relatively high since packets spend more time on the regular channel than the ALOHA channel cause a delay in the system.

Same holds for when the X/Xr ratio is greater than 1, since packets spend more time on the ALOHA channel then than the main channel causing similar delays from the ALOHA end.

roughly the same amount of time for both channels. When the X/Xr ratio is near one, the system has very low mean delay because of the same packet transfer speed between the main data channel and the S-ALOHA channel.

Below are the code changes to implement this scenario:

Main.c

```
for(i=0; i<NUMBER_OF_STATIONS; i++) {
  (data.stations+i)->id = i;
  (data.stations+i)->buffer = fifoqueue_new();
  (data.stations+i)->packet_count = 0;
  (data.stations+i)->accumulated_delay = 0.0;
  (data.stations+i)->mean_delay = 0;
}

/* Create and initialize the channel. */
data.channel = channel_new();
data.S_ALOHA_channel = channel_new();

/* Schedule initial packet arrival. */
schedule_packet_arrival_event(simulation_run,
                simulation_run_get_time(simulation_run) +
                exponential_generator((double) 1/PACKET_ARRIVAL_RATE));
```

```
typedef struct _simulation_run_data_
{
  Station_Ptr stations;
  Channel_Ptr channel;
  Channel_Ptr S_ALOHA_channel;
  long int blip_counter;
  long int arrival_count;
  long int packets_processed;
  long int number_of_packets_processed;
  long int number_of_collisions;
  double accumulated_delay;
  unsigned random_seed;
} Simulation_Run_Data, * Simulation_Run_Data_Ptr;
```

```c
#ifndef _SIMPARAMETERS_H_
#define _SIMPARAMETERS_H_

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

#define NUMBER_OF_STATIONS 10
#define MEAN_PACKET_DURATION 1       /* normalized packet Tx time*/
#define PACKET_ARRIVAL_RATE 0.0345  /* packets per Tx time */
#define MEAN_BACKOFF_DURATION 5     /* in units of packet transmit time, Tx */
#define RUNLENGTH 70000000
#define BLIPRATE 100000
#define GUARD_TIME 0.01
#define Xr 3

/* Comma separated list of random seeds to run. */
#define RANDOM_SEED_LIST 1316948, 222222, 44444

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

#endif /* simparameters.h */
```

Packet_arrival.c

```c
packet_arrival_event(Simulation_Run_Ptr simulation_run, void* dummy_ptr)
{
  int random_station_id;
  Station_Ptr station;
  Packet_Ptr new_packet;
  Buffer_Ptr stn_buffer;
  Time now;
  Simulation_Run_Data_Ptr data;

  now = simulation_run_get_time(simulation_run);

  data = (Simulation_Run_Data_Ptr) simulation_run_data(simulation_run);
  data->arrival_count++;

  /* Randomly pick the station that this packet is arriving to. Note
     that randomly splitting a Poisson process creates multiple
     independent Poisson processes.*/
  random_station_id = (int) floor(uniform_generator()*NUMBER_OF_STATIONS);
  station = data->stations + random_station_id;

  new_packet = (Packet_Ptr) xmalloc(sizeof(Packet));
  new_packet->arrive_time = now;
  new_packet->service_time = get_packet_duration();
  new_packet->status = WAITING;
  new_packet->collision_count = 0;
  new_packet->station_id = random_station_id;

  /* Put the packet in the buffer at that station. */
  stn_buffer = station->buffer;
  fifoqueue_put(stn_buffer, (void *) new_packet);
```

```
station = data->stations + random_station_id;

new_packet = (Packet_Ptr) xmalloc(sizeof(Packet));
new_packet->arrive_time = now;
new_packet->service_time = get_packet_duration();
new_packet->status = WAITING;
new_packet->collision_count = 0;
new_packet->station_id = random_station_id;

/* Put the packet in the buffer at that station. */
stn_buffer = station->buffer;
fifoqueue_put(stn_buffer, (void *) new_packet);

/* If this is the only packet at the station, transmit it (i.e., the
   ALOHA protocol). It stays in the queue either way. */
if(fifoqueue_size(stn_buffer) == 1) {
  /* Transmit the packet. */
    //however for Q6 we send to separate s-ALOHA
    schedule_transmission_start_event(simulation_run, now+GUARD_TIME, (void *) new_packet);

  //schedule_transmission_start_event(simulation_run, now+GUARD_TIME, (void *) new_packet);
}

/* Schedule the next packet arrival. */
schedule_packet_arrival_event(simulation_run,
              now + exponential_generator((double) 1/PACKET_ARRIVAL_RATE));
```

Packet_transmission.c

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

  event.description = "Retransmission of this packet";
  event.function = schedule_transmission_start_event_S_ALOHA;
  event.attachment = this_packet;

  return simulation_run_schedule_event(simulation_run, event+Xr+GUARD_TIME, event_time);
}
```

```
transmission_end_event(Simulation_Run_Ptr simulation_run, void * packet)
{
  Packet_Ptr this_packet, next_packet;
  Buffer_Ptr buffer;
  Time backoff_duration, now;
  Simulation_Run_Data_Ptr data;
  Channel_Ptr channel;
  Channel_Ptr S_aloha_channel;

  data = (Simulation_Run_Data_Ptr) simulation_run_data(simulation_run);
  channel = data->channel;
  S_aloha_channel = data->S_ALOHA_channel;

  now = simulation_run_get_time(simulation_run);

  this_packet = (Packet_Ptr) packet;
  buffer = (data->stations+this_packet->station_id)->buffer;

  /* This station has stopped transmitting. */
  decrement_transmitting_stn_count(S_aloha_channel);


  /* Check if the packet was successful. */
  if(get_channel_state(S_aloha_channel) == SUCCESS) {
     schedule_transmission_start_event(simulation_run,
                                       now+Xr,
                                       (void*) this_packet);
     //this packet must now be sent to FIFO channel
    /* Transmission was a success. The channel is now IDLE. */
    set_channel_state(S_aloha_channel, IDLE);
```

```
TRACE(printf("Success on S-Aloha channel, will transmit to data channel.\n"));

/* Collect statistics. */
data->number_of_packets_processed++;

(data->stations+this_packet->station_id)->packet_count++;
(data->stations+this_packet->station_id)->accumulated_delay +=
  now - this_packet->arrive_time;

data->number_of_collisions += this_packet->collision_count;
data->accumulated_delay += now - this_packet->arrive_time;

output_blip_to_screen(simulation_run);

/* See if there is another packet at this station. If so, enable
   it for transmission. We will transmit immediately. */
if(get_channel_state(S_aloha_channel) == SUCCESS){
        /* This packet is done. */
    free((void*) fifoqueue_get(buffer));
    decrement_transmitting_stn_count(channel);
if(fifoqueue_size(buffer) > 0) {
  next_packet = fifoqueue_see_front(buffer);

  schedule_transmission_start_event(simulation_run,
                                    now,
                                    (void*) next_packet);
}
```