Computer Engineering 4DK4

Lab 2: Packet Switched Network & Integrated Voice Performance

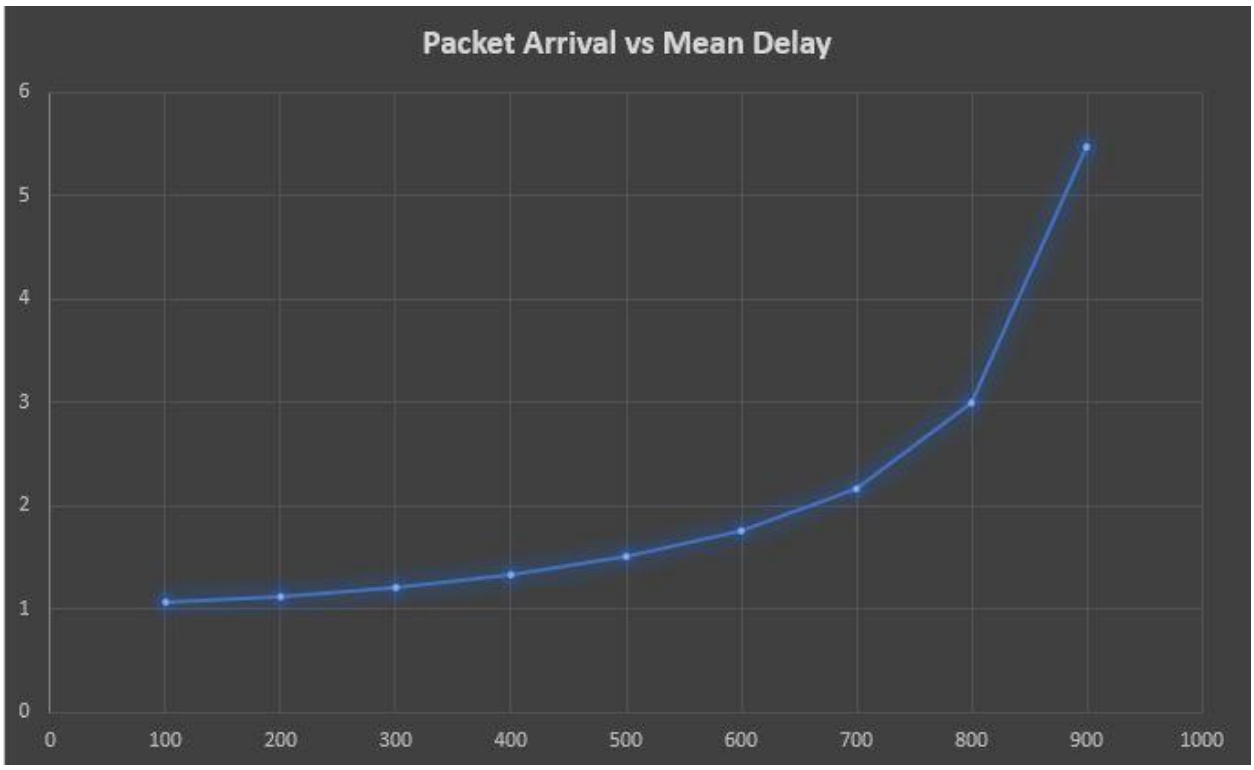
Instructor: Prof. T. Todd

Moshiur Howlader - 1316948

Ankur Bargotra – 1306226

2)

Packet Arrival Rate/Random Seed	7385384	5848481	6154815	2156181	8781819	4218916	1306226	1316948	Mean
100	1.06	1.06	1.06	1.06	1.06	1.06	1.06	1.06	1.06
200	1.12	1.12	1.12	1.13	1.12	1.13	1.13	1.12	1.12375
300	1.21	1.21	1.21	1.21	1.21	1.21	1.21	1.21	1.21
400	1.33	1.33	1.33	1.33	1.33	1.33	1.33	1.33	1.33
500	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50	1.50
600	1.75	1.75	1.75	1.75	1.75	1.75	1.75	1.75	1.75
700	2.16	2.17	2.17	2.17	2.16	2.17	2.17	2.17	2.1675
800	2.99	3	3	3	2.99	3	3	3	2.9975
900	5.46	5.47	5.47	5.52	5.44	5.5	5.48	5.48	5.4775
1000	423.85	519.34	637.17	1667.92	967.21	1109.58	1617.1	993.86	992



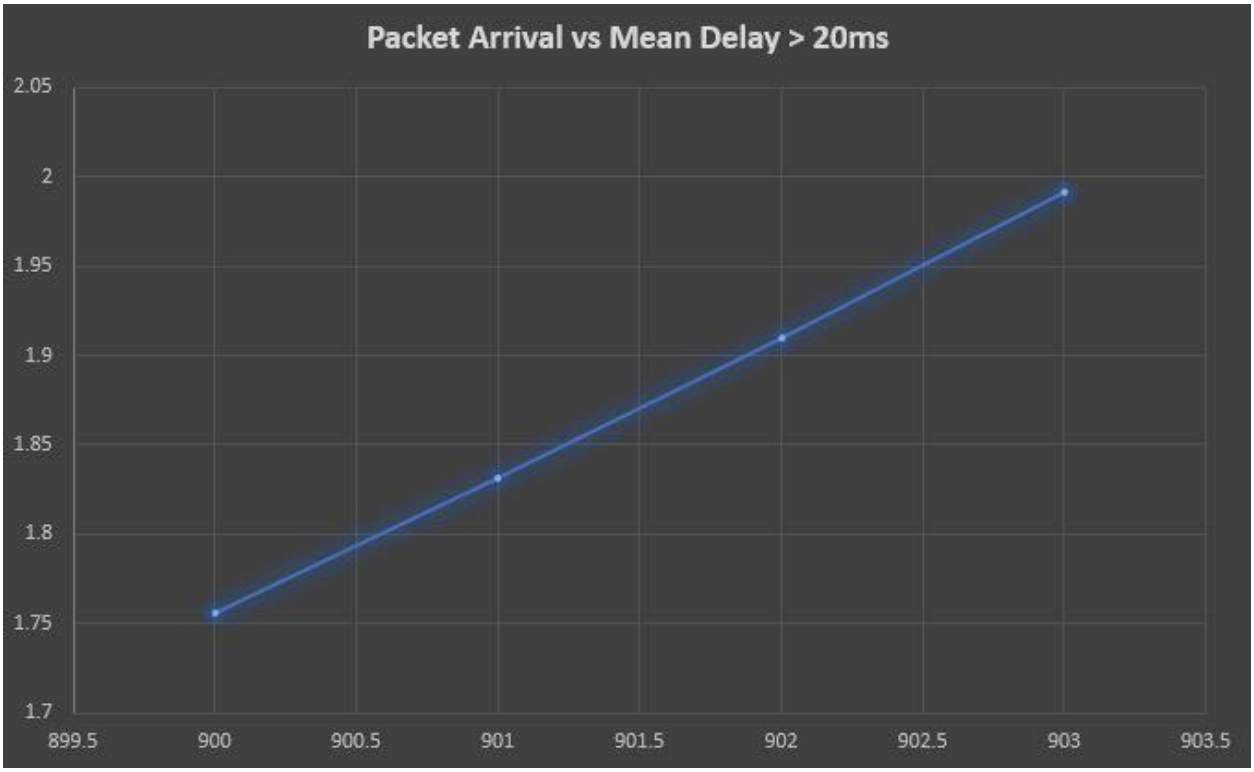
As packet arrival rate increases, so does the mean delay – indicating a correlation between the two variables. The packet arrival rate vs mean delay behaves asymptotically as the packet arrival rate reached 1000 packets/sec at which point the mean delay starts to become 'infinite'. This is because the service fraction become greater than 1, in other words we need to process the packets at a higher bit rate. The bit

rate for this case was 1Mbps or 1,000,000bps. With a packet arrival rate of 1000 packets/s and a packet length of 1000 bits, the bit rate is maxed out, therefore for a stable system:

Packet Arrival Rate * Packet Length << Link Bit Rate

3)

Packet Arrival Rate/Random Seed	7385384	5848481	6154815	2156181	8781819	4218916	1306226	1316948	Mean
900	1.7294	1.7503	1.7434	1.7750	1.7532	1.7648	1.7654	1.7682	1.7562
901	1.8024	1.8252	1.8181	1.8514	1.8293	1.8409	1.8413	1.8444	1.8316
902	1.8788	1.9025	1.8949	1.9303	1.9078	1.9195	1.9198	1.9222	1.9095
903	1.9588	1.9840	1.9763	2.0129	1.9898	2.0015	2.0019	2.0043	1.9912




```
Random Seed = 5848481
Packet arrival count = 10000005
Transmitted packet count  = 10000000 (Service Fraction =
1.00000)
Arrival rate = 903.000 packets/second
Mean Delay (msec) = 5.63
Number of Packets with Mean Delay > 20ms = 396805.000000000000
Number of Packets Transmitted = 20000000.000000000000
Percent of Packets with Mean Delay > 20ms = 1.984025000000
```

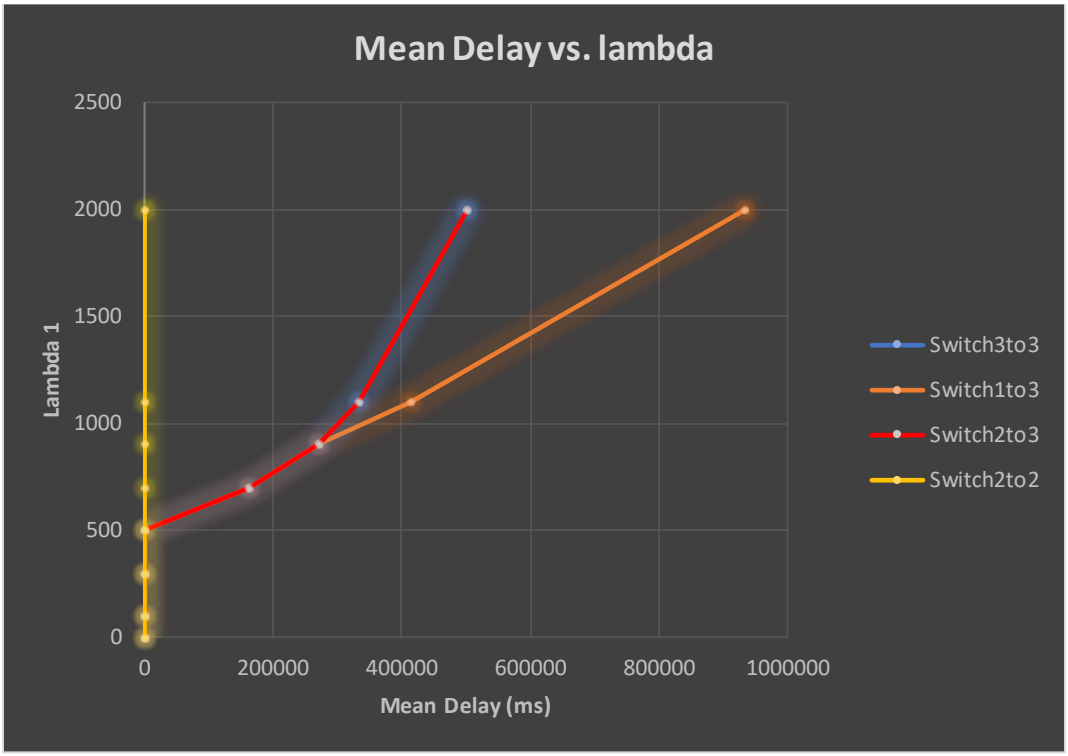
λ = 903 packets/sec

903 packet/sec is the maximum arrival rate to ensure the percentage of packets with a mean delay less than 20ms is less than 2%. This was obtained by averaging various random seeds.

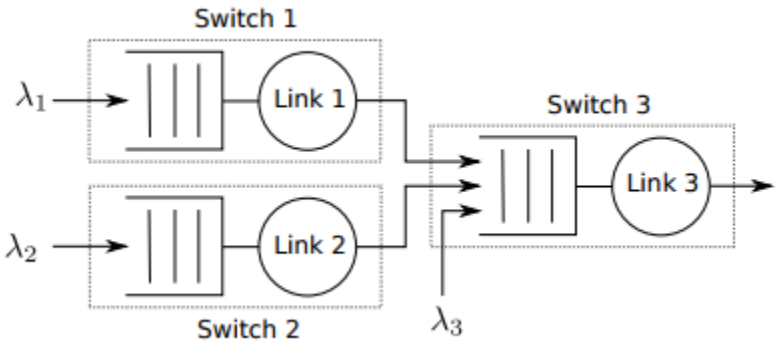
NOTE: At 902 packets/sec the value NEVER exceeded 2% for any random seed, so this too can be used as a final answer, however in aggregate, 903 packets/sec results in an average less than 2% so it is more efficient to use rather than 902.

4)

lambda1	Time spent in each switch	385384	5848481	6154815	2156181	8781819	4218916	1306226	1316948	Mean
1	Switch1	0	0	0	0	0	0	0	0	0
	Switch2	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18
	Switch3	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
100	Switch1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	Switch2	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
	Switch3	1.07	1.06	1.06	1.06	1.07	1.07	1.06	1.07	1.065
300	Switch1	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24
	Switch2	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11	0.11
	Switch3	1.61	1.6	1.6	1.61	1.61	1.61	1.61	1.61	1.6075
500	Switch1	0.39	0.39	0.39	0.39	0.39	0.39	0.4	0.4	0.3925
	Switch2	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09
	Switch3	5.94	5.91	5.92	5.91	5.87	5.91	5.92	6.01	5.92375
700	Switch1	0.71	0.71	0.71	0.71	0.71	0.71	0.71	0.72	0.71125
	Switch2	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08
	Switch3	162903.6	163310.7	163036.7	162533.5	162006.4	162790.8	162314.9	163127.7	162753
900	Switch1	2.16	2.14	2.14	2.14	2.15	2.14	2.17	2.17	2.15125
	Switch2	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
	Switch3	269978.8	270031.7	270019.5	269796.6	269423.2	269769.4	269648	270007.5	269834.3
1100	Switch1	79441.1	78708.31	78822.14	79241	78523.25	79151.51	79397.73	79357.95	79080.37
	Switch2	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
	Switch3	333942.2	334252.6	334294.4	334173.2	333777.9	333914.4	333876.5	334177.4	334051.1
2000	Switch1	432816.8	432816.8	432655.3	432820.4	432655.3	432655.3	432816.8	432655.3	432736.5
	Switch2	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
	Switch3	502338.8	502327.8	502332.5	502338.8	502327.8	502332.5	502338.8	502327.8	502333.1



For question 4, the diagram for the system is as follows:



Switch2 and Switch 3's Packet Arrival rate are fixed. However, as the rate of Packet Arrival increases for Switch1, there is significant bottleneck of packets coming in from beginning to the end of switch 1. Since the arrival rate of switch 2 is fixed and low amounts of packets are coming in, the curve of just switch 2 is a straight line along the y axis, but since we are plotting from switch 2 to 3, the plot of the switch 2 to 3 and switch 3 to 3 are nearly identical. There is significant bottleneck of packets coming in from beginning of Switch 1 to end of switch 3. The results make sense based on what we are simulating.

5)

DATA PACKETS:

Packet Arrival Rate/Random Seed	7385384	5848481	6154815	2156181	8781819	4218916	1306226	1316948	Mean
3	47.17	47.01	47.16	47	47.26	47.21	47.2	47.2	47.1513
6	56.81	56.67	57.02	56.79	56.95	56.91	56.74	56.96	56.8563
9	71.68	71.51	71.86	71.51	71.4	71.49	71.66	71.55	71.5825
12	96.54	96.88	96.97	96.29	96.56	96.39	96.75	96.18	96.57
15	149.25	148.86	150.78	147.87	148.49	148.3	148.76	147.56	148.738
18	323.11	323.89	328.5	317.67	324.88	322.75	323.63	316.5	322.616
21	745014	756935	827657	695679	733514	765850	734343	747899	750861

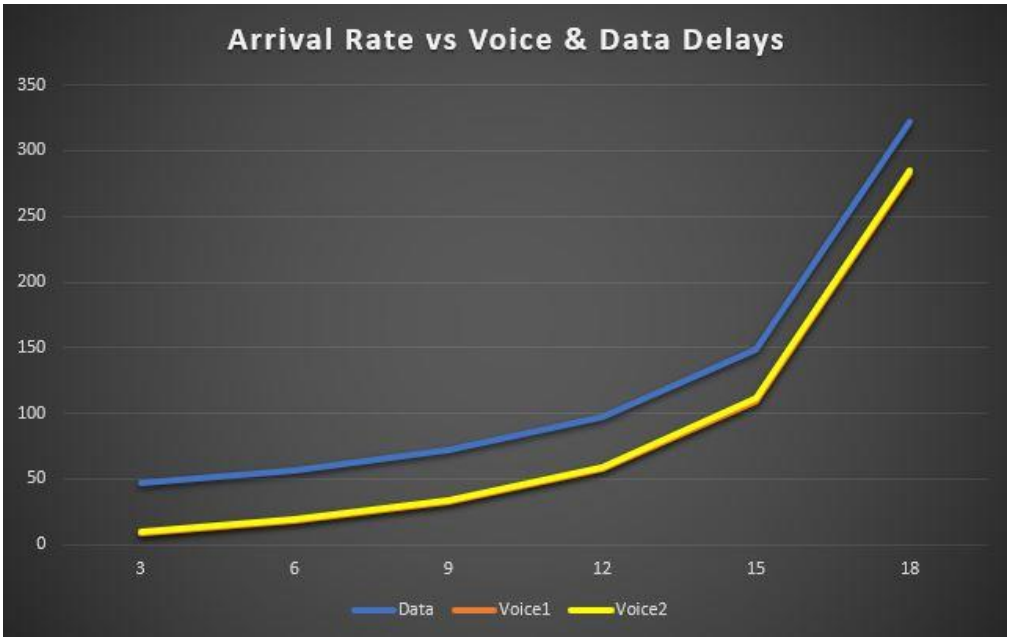
VOICE1 PACKETS:

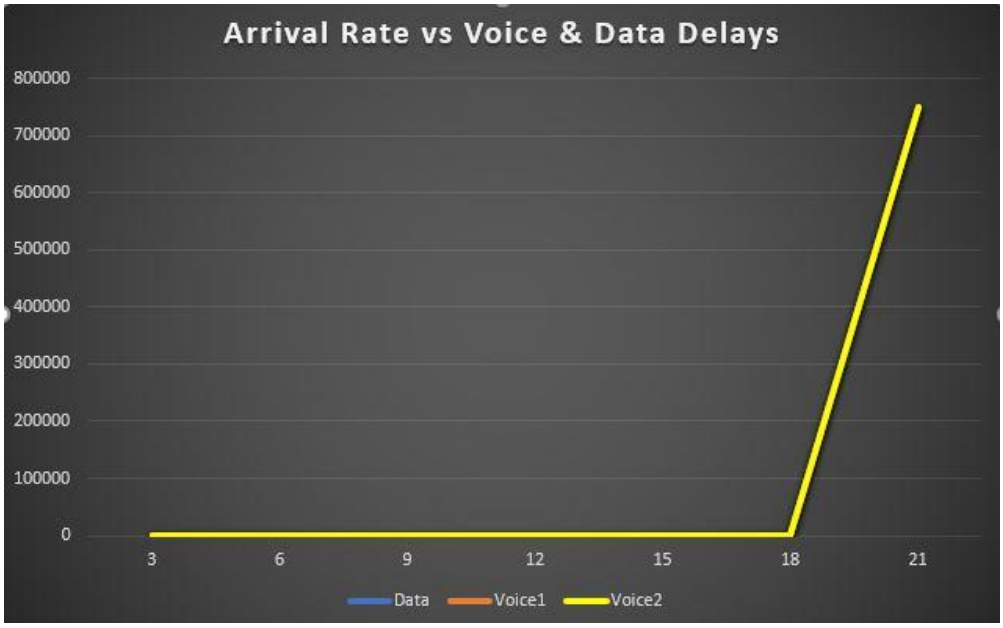
Packet Arrival Rate/Random Seed	7385384	5848481	6154815	2156181	8781819	4218916	1306226	1316948	Mean
3	8.40	8.39	8.41	8.38	8.42	8.42	8.41	8.41	8.405
6	17.84	17.82	17.99	17.87	17.87	17.91	17.83	17.93	17.8825
9	32.44	32.37	32.6	32.36	32.26	32.34	32.42	32.41	32.4
12	57.16	57.49	57.56	57	57.24	57.08	57.3	56.92	57.2188
15	109.41	109.34	111.01	108.44	109.02	108.83	109.16	108.05	109.158
18	283.16	284.58	288.71	278.24	284.59	282.65	283.98	277.06	282.871
21	745063	757272	827252	695461	733097	765794	734044	747669	750706

VOICE2 PACKETS:

Packet Arrival Rate/Random Seed	7385384	5848481	6154815	2156181	8781819	4218916	1306226	1316948	Mean
3	10.18	10.17	10.19	10.15	10.20	10.19	10.19	10.19	10.1825
6	19.62	19.59	19.77	19.65	19.64	19.69	19.60	19.71	19.6588
9	34.21	34.14	34.38	34.13	34.04	34.11	34.2	34.19	34.175

12	58.93	59.26	59.34	58.77	59.01	58.86	59.08	58.69	58.9925
15	111.18	111.12	112.78	110.22	110.79	110.61	110.94	109.83	110.934
18	284.93	286.35	290.49	280.01	286.36	284.42	285.76	278.84	284.645
21	745065	757274	827254	695463	733099	765796	734046	747670	750708





All packets, both voice and data are placed in the same buffer and served in FCFS order. This is why both voice and data packets have such a high mean delay, since they are all in the same queue.

6)

DATA PACKETS:

Packet Arrival Rate/Random Seed	7385384	5848481	6154815	2156181	8781819	4218916	1306226	1316948	Mean
3	48.6	48.43	48.58	48.41	48.71	48.65	48.64	48.64	48.5825
6	60.27	60.12	60.51	60.26	60.44	60.4	60.19	60.44	60.3288
9	78.3	78.11	78.52	78.11	77.98	78.08	78.28	78.14	78.19
12	108.49	108.9	109	108.2	108.52	108.31	108.74	108.05	108.526
15	172.53	172.06	174.38	170.87	171.61	171.38	171.94	170.47	171.905
18	383.9	384.85	390.43	377.29	385.07	383.47	384.51	375.84	383.17
21	890831	903737	986768	832073	873973	914676	876581	893307	896493

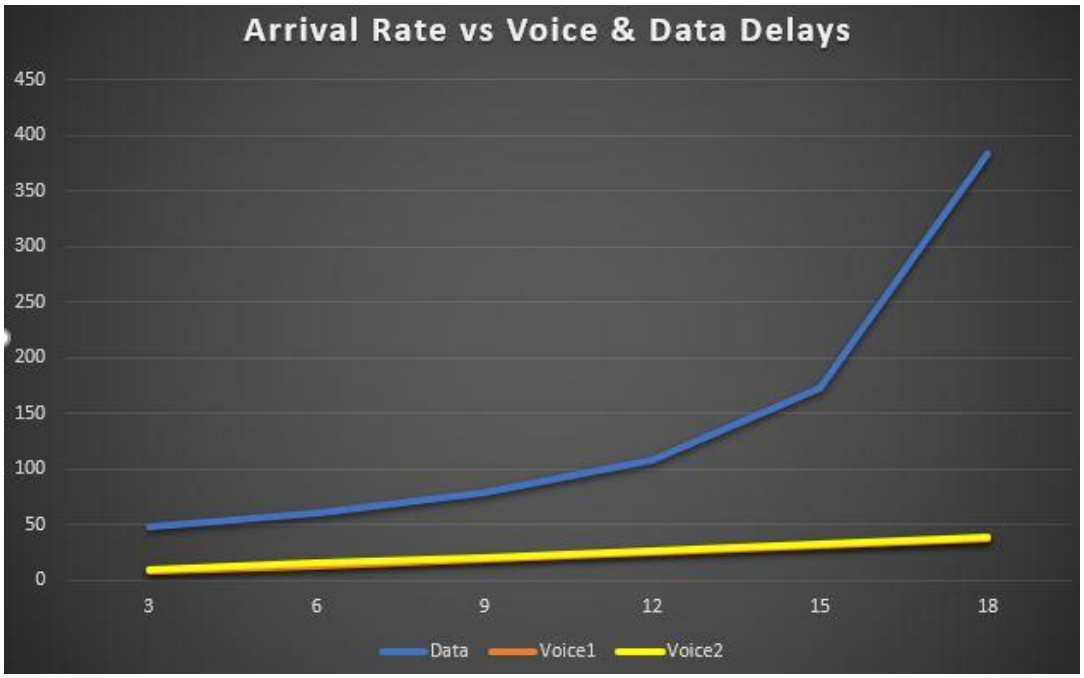
VOICE1 PACKETS [Priority]:

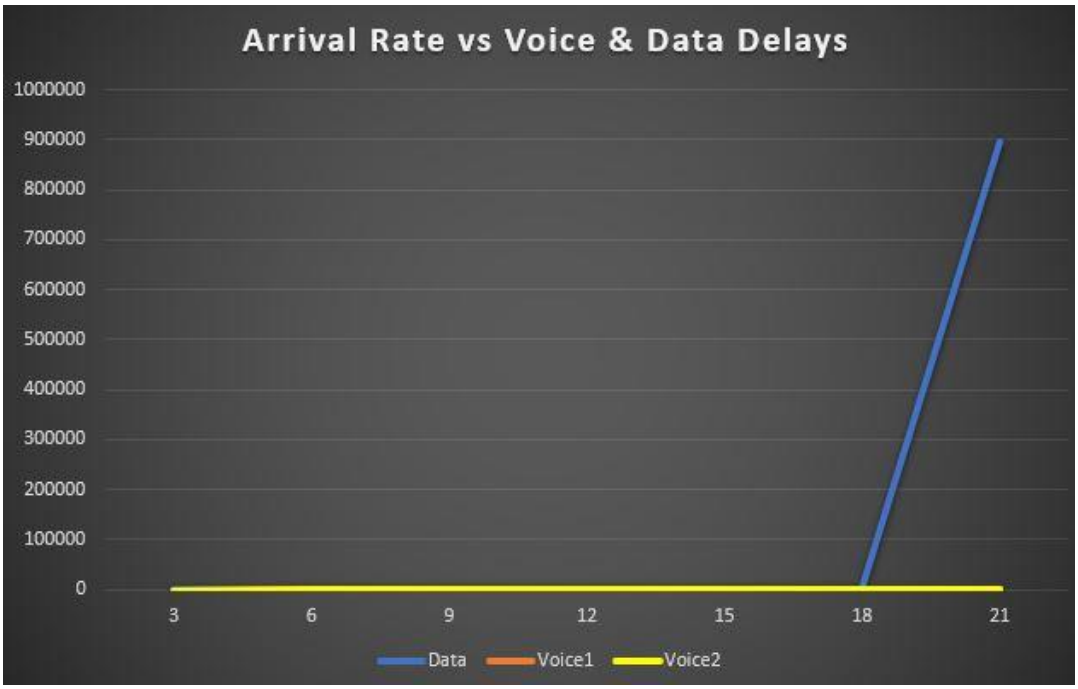
Packet Arrival Rate/Random Seed	7385384	5848481	6154815	2156181	8781819	4218916	1306226	1316948	Mean
3	7.44	7.41	7.44	7.43	7.44	7.45	7.45	7.44	7.4375
6	13.17	13.14	13.24	13.19	13.15	13.19	13.15	13.22	13.1813
9	18.99	18.95	19.06	18.97	18.92	18.97	18.97	19.03	18.9825
12	24.81	24.85	24.9	24.79	24.84	24.82	24.84	24.86	24.839
15	30.7	30.76	30.83	30.67	30.74	30.73	30.77	30.75	30.7438
18	36.64	36.73	36.83	36.59	36.7	36.69	36.75	36.74	36.7088
21	41.78	41.82	41.82	41.76	41.83	41.77	41.83	41.83	41.805

VOICE2 PACKETS [Priority]:

Packet Arrival Rate/Random Seed	7385384	5848481	6154815	2156181	8781819	4218916	1306226	1316948	Mean
3	9.22	9.19	9.21	9.2	9.21	9.23	9.22	9.21	9.2113
6	14.95	14.92	15.02	14.97	14.92	14.97	14.93	14.99	14.9588
9	20.77	20.73	20.83	20.75	20.7	20.75	20.74	20.8	20.7588
12	26.58	26.63	26.67	26.57	26.62	26.6	26.62	26.64	26.616

15	32.48	32.54	32.61	32.44	32.51	32.51	32.54	32.53	32.52
18	38.41	38.5	38.61	38.37	38.47	38.47	38.53	38.52	38.485
21	43.56	43.6	43.6	43.54	43.61	43.55	43.61	43.61	43.858





In this case, we create 2 separate buffers, one for voice and one for data, with voice being given priority, in other words voice packets will be served first and then data. This is seen by the mean delay for the voice packets decreasingly significantly as compared to previously, however data mean delay grows exponentially because it is not prioritized.

Appendix

2)

Multiple seeds used for simulation

```
#define PACKET_ARRIVAL_RATE 1000 /* packets per second */
#define PACKET_LENGTH 1e3 /* bits */
#define LINK_BIT_RATE 1e8 /* bits per second */
#define RUNLENGTH 10e6 /* packets */

/* Comma separated list of random seeds to run. */
#define RANDOM_SEED_LIST 7385384,5848481,6154815,2156181,8781819,4218916,1306226,1316948

#define PACKET_XMT_TIME ((double) PACKET_LENGTH/LINK_BIT_RATE)
#define BLIPRATE (RUNLENGTH/1000)
```

3)

Added additional variables to data struct in main.h to track total packets transmitted and packets with delay

```
typedef struct _simulation_run_data_
{
  Fifoqueue_Ptr buffer;
  Server_Ptr link;
  long int blip_counter;
  long int arrival_count;
  long int number_of_packets_processed;
  double packet_count;
  double delay_count;
  double accumulated_delay;
  unsigned random_seed;
} Simulation_Run_Data, * Simulation_Run_Data_Ptr;
```

Added code in packet_transmission.c to increment variables for each packet transmission and when the delay for packet transmission exceeds 20ms

```
data->packet_count = data->packet_count+1;
if(1000*(simulation_run_get_time(simulation_run) - this_packet->arrive_time) > 20){
    data->delay_count = data->delay_count+1;
}
```

Added code in output.c to display results

```
printf("Number of Packets with Mean Delay > 20ms = %.12f \n", data->delay_count);
printf("Number of Packets Transmitted = %.12f \n", data->packet_count);
printf("Percent of Packets with Mean Delay > 20ms = %.12f \n", 100*(data->delay_count/data->packet_count));
```

```
#define PACKET_ARRIVAL_RATE 903 /* packets per second */
#define PACKET_LENGTH 1e3 /* bits */
#define LINK_BIT_RATE  1e6 /* bits per second */ //was 1e8
#define RUNLENGTH 10e6 /* packets */

/* Comma separated list of random seeds to run. */
#define RANDOM_SEED_LIST 7385384, 5848481, 6154815, 2156181, 8781819, 4218916, 1306226, 1316948
```

4) In main.h, added 3 delay variables for the three switches. In main.c we initialize these variables accordingly.

```
typedef struct _simulation_run_data_
{
  Fifoqueue_Ptr buffer1;
  Server_Ptr link1;
  Fifoqueue_Ptr buffer2;
  Server_Ptr link2;
  Fifoqueue_Ptr buffer3;
  Server_Ptr link3;
  long int blip_counter;
  long int arrival_count;
  long int number_of_packets_processed;
  double accumulated_delay1;
  double accumulated_delay2;
  double accumulated_delay3;
  unsigned random_seed;
} Simulation_Run_Data, * Simulation_Run_Data_Ptr;
```

In packet_transmission.c, various functions for Switch 1,2, and 3 were added.

```
void
start_transmission_on_link1(Simulation_Run_Ptr, Packet_Ptr, Server_Ptr);

void
start_transmission_on_link2(Simulation_Run_Ptr, Packet_Ptr, Server_Ptr);

void
start_transmission_on_link3(Simulation_Run_Ptr, Packet_Ptr, Server_Ptr);

void
end_packet_transmission_s1_event(Simulation_Run_Ptr, void*);

void
end_packet_transmission_s2_event(Simulation_Run_Ptr, void*);

void
end_packet_transmission_s3_event(Simulation_Run_Ptr, void*);

double
get_packet_transmission_time_13(void);

double
get_packet_transmission_time_2(void);
```

Similarly for packet_arrival.h

```
void
packet_arrival_s1_event(Simulation_Run_Ptr, void*);

void
packet_arrival_s2_event(Simulation_Run_Ptr, void*);

void
packet_arrival_s3_event(Simulation_Run_Ptr, void*);

long
schedule_packet_arrival_s1_event(Simulation_Run_Ptr, double);

long
schedule_packet_arrival_s2_event(Simulation_Run_Ptr, double);

long
schedule_packet_arrival_s3_event(Simulation_Run_Ptr, double);
```

Major change was in packet_transmission.c, where the simulation for packet transmission from switch 1 and 2 were passed to switch 3. Sample for end_packet_transmission_s1_event() :

```
  /* Collect statistics. */
  data->number_of_packets_processed++;
  data->accumulated_delay1 += simulation_run_get_time(simulation_run) - this_packet->arrive_time;

  /* Output activity blip every so often. */
  output_progress_msg_to_screen(simulation_run);

    if(server_state(data->link3) == BUSY) {
        //put in buffer S3
        fifoqueue_put(data->buffer3, (void*) this_packet);
    } else {
        //else pass packet to S3 link
        start_transmission_on_link3(simulation_run, this_packet, data->link3);
    }

  /*
   * See if there is are packets waiting in the buffer. If so, take the next one
   * out and transmit it immediately.
   */

  if(fifoqueue_size(data->buffer1) > 0) {
    next_packet = (Packet_Ptr) fifoqueue_get(data->buffer1);
    start_transmission_on_link1(simulation_run, next_packet, link);
  }
}
```

We check if Switch 3's link is busy, if so pass the packet from switch 1 to switch 3 buffer. Otherwise, directly pass the packet to link of the switch 3. Done similarly for switch 2. Switch 3 is where freeing of the memory for the data is called after packet is done transmitting

5)

In simparameters.h, the length of the voice packets were specified, as well as their xmt time.

```
#define PACKET_ARRIVAL_RATE 21 /* packets per second */
#define PACKET_LENGTH 1e3 /* bits */
#define VOICE1_LENGTH 1776 /* bits [64kbps*20ms [Codec*Voice Time] + 62*8 [Header]] */
#define VOICE2_LENGTH 1776 /* bits [64kbps*20ms [Codec*Voice Time] + 62*8 [Header]]*/
#define LINK_BIT_RATE 1e6 /* bits per second */
#define RUNLENGTH 10e6 /* packets */

#define MEAN_SERVICE_TIME 40e-3 /* seconds */
#define INTERPACKET_ARRIVAL_TIME 20e-3 /* seconds */

/* Comma separated list of random seeds to run. */
#define RANDOM_SEED_LIST 1306226

#define PACKET_XMT_TIME ((double) PACKET_LENGTH/LINK_BIT_RATE)
#define VOICE1_XMT_TIME ((double) VOICE1_LENGTH/LINK_BIT_RATE)
#define VOICE2_XMT_TIME ((double) VOICE2_LENGTH/LINK_BIT_RATE)
#define BLIPRATE (RUNLENGTH/1000)
```

In main.h, variables to get the # of processed voice and data packets as well as their delay were created in the data struct. Also a packet type was created in the packet struct with the possible values of DATA, VOICE1 or VOICE2

```
typedef struct _simulation_run_data_
{
  Fifoqueue_Ptr buffer;
  Server_Ptr link;
  long int blip_counter;
  long int arrival_count;
  long int number_of_packets_processed;
  long int data_packets_processed;
  long int voice1_packets_processed;
  long int voice2_packets_processed;
  double accumulated_delay;
  double voice1_delay;
  double voice2_delay;
  double data_delay;
  unsigned random_seed;
} Simulation_Run_Data, * Simulation_Run_Data_Ptr;
```

In main.c, variables were initialized to 0, and initial arrival events for all packets were called for the first time

```
schedule_packet_arrival_event(simulation_run, simulation_run_get_time(simulation_run));
schedule_voice1_arrival_event(simulation_run, simulation_run_get_time(simulation_run));
schedule_voice2_arrival_event(simulation_run, simulation_run_get_time(simulation_run));
```

```
typedef enum {XMTTING, WAITING} Packet_Status;
typedef enum {DATA, VOICE1, VOICE2} Packet_Type;

typedef struct _packet_
{
  double arrive_time;
  double service_time;
  int source_id;
  int destination_id;
  Packet_Type packet_type;
  Packet_Status status;
} Packet, * Packet_Ptr;
```

voice_packet_arrival.h was created to store function prototype definitions for voice packet events

```
void
voice2_arrival_event(Simulation_Run_Ptr, void*);

void
voice1_arrival_event(Simulation_Run_Ptr, void*);

long
schedule_voice1_arrival_event(Simulation_Run_Ptr, double);

long
schedule_voice2_arrival_event(Simulation_Run_Ptr, double);
```

voice_packet_arrival.c was created to process the arrival if voice packets, either voice 1 or 2

```
long int
schedule_voice1_arrival_event(Simulation_Run_Ptr simulation_run,
                  double event_time)
{
  Event event;

  event.description = "Voice 1 Arrival";
  event.function = voice1_arrival_event;
  event.attachment = (void *) NULL;

  return simulation_run_schedule_event(simulation_run, event, event_time);
}

long int
schedule_voice2_arrival_event(Simulation_Run_Ptr simulation_run,
                              double event_time)
{
    Event event;

    event.description = "Voice 2 Arrival";
    event.function = voice2_arrival_event;
    event.attachment = (void *) NULL;

    return simulation_run_schedule_event(simulation_run, event, event_time);
}
```

```
void
voice1_arrival_event(Simulation_Run_Ptr simulation_run, void * ptr)
{
  Simulation_Run_Data_Ptr data;
  Packet_Ptr new_packet;

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

  new_packet = (Packet_Ptr) xmalloc(sizeof(Packet));
  new_packet->arrive_time = simulation_run_get_time(simulation_run);
  new_packet->service_time = get_voice1_transmission_time();
  new_packet->status = WAITING;
  new_packet->packet_type = VOICE1;

  /*
   * Start transmission if the data link is free. Otherwise put the packet into
   * the buffer.
   */

  if(server_state(data->link) == BUSY) {
    fifoqueue_put(data->buffer, (void*) new_packet);
  } else {
    start_transmission_on_link(simulation_run, new_packet, data->link);
  }

  /*
   * Schedule the next packet arrival. Independent, exponentially distributed
   * interarrival times gives us Poisson process arrivals.
   */

  schedule_voice1_arrival_event(simulation_run,
            simulation_run_get_time(simulation_run) +
            INTERPACKET_ARRIVAL_TIME);
}
```

In packet_transmission.c, an if-else statement is created to track which packets are being transmitted based on type and calculate their delay

```
if(this_packet->packet_type == DATA){
    data->data_packets_processed++;
    data->data_delay += simulation_run_get_time(simulation_run) - this_packet->arrive_time;
}
else if(this_packet->packet_type == VOICE1){
    data->voice1_packets_processed++;
    data->voice1_delay += simulation_run_get_time(simulation_run) - this_packet->arrive_time;
}
else if(this_packet->packet_type == VOICE2){
    data->voice2_packets_processed++;
    data->voice2_delay += simulation_run_get_time(simulation_run) - this_packet->arrive_time;
}
```

Also, statements were added to calculate voice packet transmission times

```
double
get_packet_transmission_time(void)
{
    return exponential_generator((double) MEAN_SERVICE_TIME);
}

double
get_voice1_transmission_time(void)
{
    return ((double)VOICE1_XMT_TIME);
}

double
get_voice2_transmission_time(void)
{
    return ((double)VOICE2_XMT_TIME);
}
```

In output.c, we simply print the results

```
printf("Transmitted Data Packet Count  = %ld \n", data->data_packets_processed);
printf("Transmitted Voice1 Packet Count  = %ld \n", data->voice1_packets_processed);
printf("Transmitted Voice2 Packet Count  = %ld \n", data->voice2_packets_processed);

printf("Data Packet Mean Delay (msec) = %.2f \n", 1e3*data->data_delay/data->data_packets_processed);
printf("Voice1 Packet Mean Delay (msec) = %.2f \n", 1e3*data->voice1_delay/data->voice1_packets_processed);
printf("Voice2 Packet Mean Delay (msec) = %.2f \n", 1e3*data->voice2_delay/data->voice2_packets_processed);
```

6)

In main.h a second buffer for voice packets was created, in main,c it was initialized.

```
typedef struct _simulation_run_data_
{
  Fifoqueue_Ptr buffer;
  Fifoqueue_Ptr voice_buffer;
  Server_Ptr link;


data.buffer = fifoqueue_new();
data.voice_buffer = fifoqueue_new();
data.link = server_new();
```

In voice_packet_arrival.c, we switch the code from Q5 putting voice data into the voice buffer we created

```
if (server_state(data->link) == BUSY) {
    fifoqueue_put(data->voice_buffer, (void*)new_packet);
}
else {
    start_transmission_on_link(simulation_run, new_packet, data->link);
}
```

In packet_transmission.c we and an additional if-else statement, this is the primary statement that gives voice packets priority. Only processing data packets if the voice buffer is empty

```
if (fifoqueue_size(data->voice_buffer) > 0) {
    next_packet = (Packet_Ptr)fifoqueue_get(data->voice_buffer);
    start_transmission_on_link(simulation_run, next_packet, link);
}
else if (fifoqueue_size(data->buffer) > 0) {
    next_packet = (Packet_Ptr)fifoqueue_get(data->buffer);
    start_transmission_on_link(simulation_run, next_packet, link);
}
```