Mid-Semester Assignment

Sanjana S Prabhu

October 6, 2019

1 Introduction

In this assignment,a mix of video on demand and data packets arrive at a router along a high bandwidth link. The router has to schedule them to a home client through a low bandwidth link (512 kbps or 64 kbytes per second). The data packets are somewhat sensitive to delay (not more than 30 seconds delay permitted) while the video on demand is very delay sensitive (max of 1 second delay is acceptable) and arrive at a rate of 48 kbytes per second. The data packets are 1024 bytes in size,while the video packets are 512 bytes long. Keeping this in mind, we have to design a queue such that the drop percentage of videos is less than 5 percent every second. Also, we have to account for the retransmitted packets.

2 Part 1 and Part 2

In the first question, we are asked to use a FIFO queue for queueing the received packets and the initial burst is not given. Whereas in the second question, the burst is given, therefore the structure of the queue is the same for both the questions, but there are extra packets initially for the second question. The pseudo code for the queue and the packet simulation is given in the next section(for first question, the variable packets=0).

2.1 Pseudo Code

```
function addone ( i: integer ):
    begin
    return (( i mod maxlength ) + 1)
end; { addone }

procedure MAKENULL ( var Q: QUEUE );
    begin
    Q.front := 1;
    Q.rear := maxlength
```

```
end; { MAKENULL }

function EMPTY ( var Q: QUEUE ):
    begin
    if addone(Q.rear) = Q.front then
        return (true)
    else
        return (false)
end; { EMPTY }

function FRONT ( var Q: QUEUE ): elementtype;
    begin
    if EMPTY(Q) then
        error('queue is empty')
    else
        return (Q.elements[Q.front])
end; { FRONT }

procedure ENQUEUE ( x: elementtype; var Q: QUEUE );
begin
    if addone (addone(Q.rear)) = Q.front then
        error('queue is full')
    else begin
        Q.rear := addone(Q.rear);
        Q.elements[Q.rear] := x
        end
end; { ENQUEUE )

procedure DEQUEUE ( var Q: QUEUE );
begin
    if EMPTY(Q) then
        error('queue is empty')
    else
        Q.front := addone(Q.front)
end; { DEQUEUE }
```

Structure of an element of the queue:

```
struct Packet
{
char type_of_packet;
double time_waiting;
int time_received_sec;
};
```

For the packet simulation:

```
packet_simulation(Queue q, int time_waiting, int time_transmission, int packets):
struct Packet * p;//pointer to the packet
    for(i=0:i<96*t_max:i++)
        Enqueue(q)//add video packets
        if((i-48) mod maxlength==0)
            for(j=0;j<6;j++)//add 6 data packets
                Enqueue(q)
        if(i>=96 and i mod 96==0)
            for(j=0;j<dropped[i/96 -1];j++)//Re-transmission of packets of each
            type dropped the previous second
                Enqueue(q)
        p=Front(q)
        p->time_waiting-=1/96;//Every iteration is of 1/96
        seconds
        for(m=0;!(Empty(q));m++)//when any packet arrives in the front of the
        queue it is checked for the following
            if(i/96 -p->time_received >=max_delay)//checked if it can be dropped
                drop(p);
            else if(p->time_waiting<=0)//checked if it can be transmitted
                transmit(p);
                (p+1)->time_waiting+=transmission_time
            else//if none of these are possible the loop is broken
                break;
```

2.2 The Code

So, for Q1 and Q2, the pseudocode for the implementation of the queue and the packet simulation link remains the same. However the number of packets for the first question is 0. The code for the questions are uploaded on Moodle as ee17b072.c.

2.3 Conclusions and Results

- We observe that the for the first question, the overall performance is clearly better, that is the drop percentage of the video packets is lesser and in the second question, it is higher, sometimes 100 percent for some particular times of transmission and waiting. However, even when the burst is not present, the efficiency what we want(less than 5 percent video drop) is not achieved.

- In the second question, most of the initial video packets are completely dropped, that is the percentage drop in the video packets is

100 percent, every second, due to the large burst of data packets in the beginning. Towards the end, that is after around 100s , the video drop percentage is same as that of the drop percentage without the burst. However during the first 10-20 s, the drop is almost 100 percent.

- This is due to the presence of the data packets, as well as the time of transmission. Since these have a maximum delay of 30 seconds, they will generally be transmitted before they are dropped. However they are a hurdle to the video packets, as their maximum delay limit might be reached if there are some data packets infront of video packets.
- Also, the transmission time for the data packets is greater than the video packets(by a factor of 2), therefore again causing some kind of delay to the video packets behind them in the queue.
- So we have clearly noticed that the presence of the data packets infront of video packets causes the high drop percentage of the video packets every second, so **the obvious solution is to ensure that the data packets are always behind the video packets** and hence do not lead to a delay greater than that of the maximum delay of the video packets.
- This is what is asked in the last question, Q3.

3 Part 3-Designing a different type of queue

As concluded in the above section, we need a different type of queue, with some priority condition, such that video packets are never behind the data packets, as this causes drop in video packets. The priority is given for video packets, that is video packets are transmitted first. We assume that the video packets are generally queued before the data packets and the data packets are always added in the end. However, to ensure that some of the data packets are also transmitted, I have given the following condition for video packets to be dropped: **video packets are dropped when the packets transmitted in a second is greater than 95 percent of the packets received in that second.** As we can observe from the output, this will lead to less than 5 percent video drop every second. As for the data drop, it depends on the burst given at t=0, if the burst is of zero data packets, then most of the initial data packets will be transmitted.

3.1 Pseudocode

Below is the pseudocode for the special queue or priority queue:

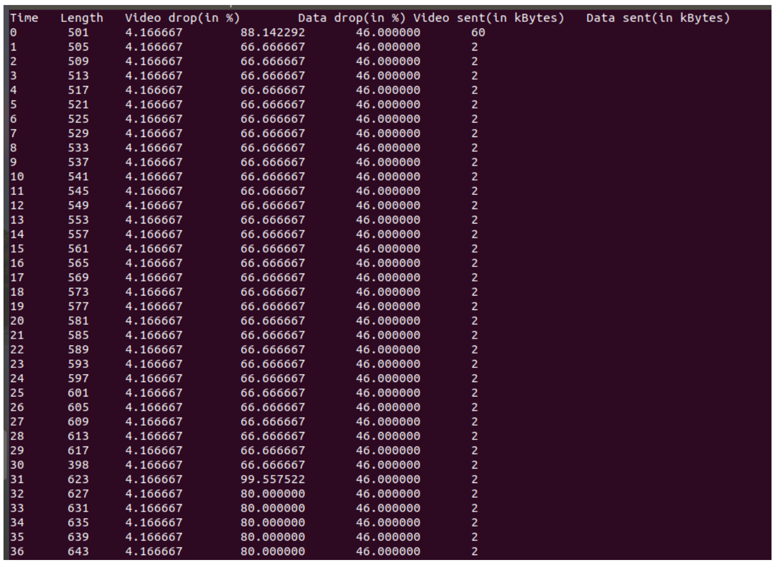
```
function makenull:
    begin
```

```
head=ptr;
data_rear=head;
video_rear=head;
head->next=NULL;
length=0;
end
function insert(x):
    begin
ptr=head;
for(i=0;ptr->next!=NULL;i++){
if((ptr->next)->type_of_packet=='d')
{
video_rear=ptr;
break;
}
else
ptr=ptr->next;
}// To find the last video packet
if(priority){//priority=1 for video and 0 for data
temp=video_rear->next;
video_rear->next=ptr;
video_rear=ptr;
ptr.element=x; }
else{
data_rear->next=ptr;
data_rear=ptr;
ptr->next=NULL;
ptr.element=x; }
length++;
end
function drop_packet(ptr)
    begin
ptr->next=(ptr->next)->next;
length--;
    end
function dequeue()
begin
    ptr = head;
    head = head->next;
    free(ptr);
    length--;
end
```

Pseudocode for the simulation function: Same as the one for Q1 and Q2, except the condition for dropping video packets is different.

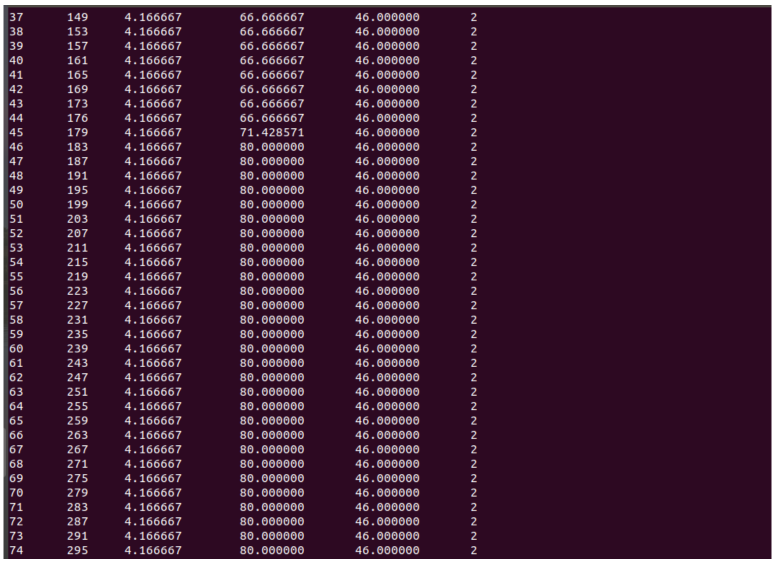
3.2 Results

The following table is obtained by using transmit time=0.015625 s(64 kbps):



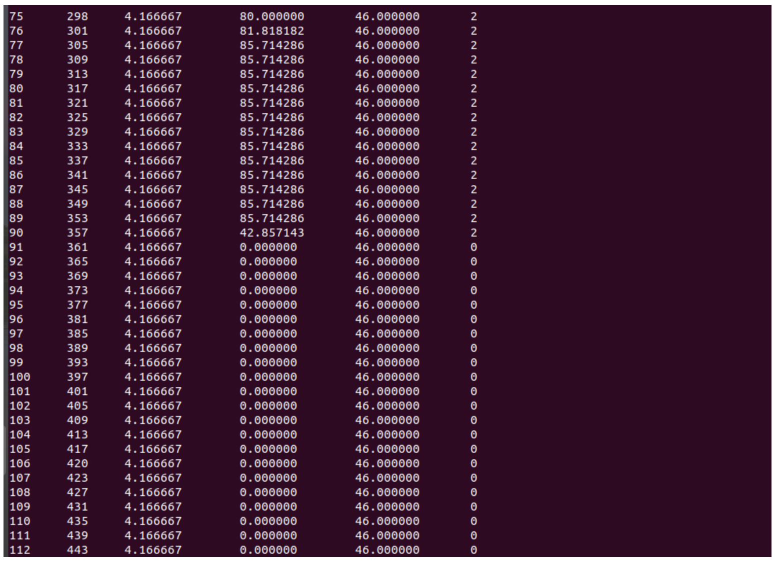
```
Time    Length   Video drop(in %)        Data drop(in %) Video sent(in kBytes)   Data sent(in kBytes)
0        501     4.166667        88.142292       46.000000       60
1        505     4.166667        66.666667       46.000000       2
2        509     4.166667        66.666667       46.000000       2
3        513     4.166667        66.666667       46.000000       2
4        517     4.166667        66.666667       46.000000       2
5        521     4.166667        66.666667       46.000000       2
6        525     4.166667        66.666667       46.000000       2
7        529     4.166667        66.666667       46.000000       2
8        533     4.166667        66.666667       46.000000       2
9        537     4.166667        66.666667       46.000000       2
10       541     4.166667        66.666667       46.000000       2
11       545     4.166667        66.666667       46.000000       2
12       549     4.166667        66.666667       46.000000       2
13       553     4.166667        66.666667       46.000000       2
14       557     4.166667        66.666667       46.000000       2
15       561     4.166667        66.666667       46.000000       2
16       565     4.166667        66.666667       46.000000       2
17       569     4.166667        66.666667       46.000000       2
18       573     4.166667        66.666667       46.000000       2
19       577     4.166667        66.666667       46.000000       2
20       581     4.166667        66.666667       46.000000       2
21       585     4.166667        66.666667       46.000000       2
22       589     4.166667        66.666667       46.000000       2
23       593     4.166667        66.666667       46.000000       2
24       597     4.166667        66.666667       46.000000       2
25       601     4.166667        66.666667       46.000000       2
26       605     4.166667        66.666667       46.000000       2
27       609     4.166667        66.666667       46.000000       2
28       613     4.166667        66.666667       46.000000       2
29       617     4.166667        66.666667       46.000000       2
30       398     4.166667        66.666667       46.000000       2
31       623     4.166667        99.557522       46.000000       2
32       627     4.166667        80.000000       46.000000       2
33       631     4.166667        80.000000       46.000000       2
34       635     4.166667        80.000000       46.000000       2
35       639     4.166667        80.000000       46.000000       2
36       643     4.166667        80.000000       46.000000       2
```

Figure 1:



```
37       149     4.166667        66.666667       46.000000       2
38       153     4.166667        66.666667       46.000000       2
39       157     4.166667        66.666667       46.000000       2
40       161     4.166667        66.666667       46.000000       2
41       165     4.166667        66.666667       46.000000       2
42       169     4.166667        66.666667       46.000000       2
43       173     4.166667        66.666667       46.000000       2
44       176     4.166667        66.666667       46.000000       2
45       179     4.166667        71.428571       46.000000       2
46       183     4.166667        80.000000       46.000000       2
47       187     4.166667        80.000000       46.000000       2
48       191     4.166667        80.000000       46.000000       2
49       195     4.166667        80.000000       46.000000       2
50       199     4.166667        80.000000       46.000000       2
51       203     4.166667        80.000000       46.000000       2
52       207     4.166667        80.000000       46.000000       2
53       211     4.166667        80.000000       46.000000       2
54       215     4.166667        80.000000       46.000000       2
55       219     4.166667        80.000000       46.000000       2
56       223     4.166667        80.000000       46.000000       2
57       227     4.166667        80.000000       46.000000       2
58       231     4.166667        80.000000       46.000000       2
59       235     4.166667        80.000000       46.000000       2
60       239     4.166667        80.000000       46.000000       2
61       243     4.166667        80.000000       46.000000       2
62       247     4.166667        80.000000       46.000000       2
63       251     4.166667        80.000000       46.000000       2
64       255     4.166667        80.000000       46.000000       2
65       259     4.166667        80.000000       46.000000       2
66       263     4.166667        80.000000       46.000000       2
67       267     4.166667        80.000000       46.000000       2
68       271     4.166667        80.000000       46.000000       2
69       275     4.166667        80.000000       46.000000       2
70       279     4.166667        80.000000       46.000000       2
71       283     4.166667        80.000000       46.000000       2
72       287     4.166667        80.000000       46.000000       2
73       291     4.166667        80.000000       46.000000       2
74       295     4.166667        80.000000       46.000000       2
```

Figure 2:



```
75      298     4.166667        80.000000       46.000000       2
76      301     4.166667        81.818182       46.000000       2
77      305     4.166667        85.714286       46.000000       2
78      309     4.166667        85.714286       46.000000       2
79      313     4.166667        85.714286       46.000000       2
80      317     4.166667        85.714286       46.000000       2
81      321     4.166667        85.714286       46.000000       2
82      325     4.166667        85.714286       46.000000       2
83      329     4.166667        85.714286       46.000000       2
84      333     4.166667        85.714286       46.000000       2
85      337     4.166667        85.714286       46.000000       2
86      341     4.166667        85.714286       46.000000       2
87      345     4.166667        85.714286       46.000000       2
88      349     4.166667        85.714286       46.000000       2
89      353     4.166667        85.714286       46.000000       2
90      357     4.166667        42.857143       46.000000       2
91      361     4.166667        0.000000        46.000000       0
92      365     4.166667        0.000000        46.000000       0
93      369     4.166667        0.000000        46.000000       0
94      373     4.166667        0.000000        46.000000       0
95      377     4.166667        0.000000        46.000000       0
96      381     4.166667        0.000000        46.000000       0
97      385     4.166667        0.000000        46.000000       0
98      389     4.166667        0.000000        46.000000       0
99      393     4.166667        0.000000        46.000000       0
100     397     4.166667        0.000000        46.000000       0
101     401     4.166667        0.000000        46.000000       0
102     405     4.166667        0.000000        46.000000       0
103     409     4.166667        0.000000        46.000000       0
104     413     4.166667        0.000000        46.000000       0
105     417     4.166667        0.000000        46.000000       0
106     420     4.166667        0.000000        46.000000       0
107     423     4.166667        0.000000        46.000000       0
108     427     4.166667        0.000000        46.000000       0
109     431     4.166667        0.000000        46.000000       0
110     435     4.166667        0.000000        46.000000       0
111     439     4.166667        0.000000        46.000000       0
112     443     4.166667        0.000000        46.000000       0
```

Figure 3:

4 Answers

- 1.This type of a FIFO queue does not meet requirements due to the data packets infront of it.
- 2.Due to the burst the drop of video packets in the first few seconds is even worse than the first case, this is due to the burst of data packets. The retransmission of the dropped data packets only makes the drop of video packets even worse.
- 3.A priority queue which gives higher priority to videos is the best, as videos are always queued at the front due to this. However,a condition is imposed on the video packets such that even the data packets can be transmitted.As seen in the results, the drop in video packets is always less than 5 percentage. Also all the video packets have delay less than 1 second because they have a delay equal to the transmit time of the previous packet, which is less than 1 second.