EE4371 Data Structures and Algorithms Midterm 2019

Rishhanth Maanav V
EE16B036

1. Introduction

In this problem, we simulate a network router receiving and transmitting 2 types of packets- data and video. This simulation has been done using different types of queues. The receiving channel has high bandwidth support and hence can support any input rate of packets of both types. However, the transmission channel has limited bandwidth and can only transmit data upto a specific rate. Both the data and video packets have to be transmitted considering different delay sensitivities and input rates of both the types of packets.
Video packets are 512 bytes in size and arrive at a rate of 48 kbytes per second. Data packets are 1024 bytes in size and arrive at in the middle of a second as a cluster of size 6 kbytes. In addition to this, there may be a burst of data packets of size 512 kbytes initially. Video packets are more delay sensitive and hence expire within 1 second as compared to data packets which are less sensitive to delay and expire after 30 seconds. The transmission rate is limited to 64 kbytes per second. In this problem, we try 2 types of queues (FIFO queues and Priority Queues). We have to keep the dropped video packets to less than 5%.

2. Question 1

In this question, we have been asked to keep the initial data burst as 0. We have been also asked to use FIFO Queue to simulate the router.

2.1. Observation

We observe that none of the packets, video or data, are dropped. This is due the fact that the total size of packets received per second is lesser than the maximum output transmission rate.

$$InSizePerSecond = 96 * VideoSize + 6 * HTTPSize$$

$$InSizePerSecond = 96 * 0.5 + 6 * 1 = 54kbytes$$

$$InSizePerSecond = 54kbytes < OutRate = 64kbytes$$

Hence,
Data Drop% = 0
Video Drop% = 0

Also, the observed delays times, which are constant throughout, are:

Avg. Data Waiting time: 0.0416658 sec
Data Transmission time: 0.015625 sec
Avg. Data Delay: 0.057289 sec

Avg. Video Waiting time: 0.0206702 sec
Video Transmission time: 0.0078125 sec
Avg. Video Delay: 0.0284827 sec

In this case of 0 data burst, the FIFO queue satisfies of a maximum video drop of 5%.

2.2. Pseudocode for FIFO Queue

```
Implementation of FIFO Queue
using struct PACKET in a linked list

    Initialize PACKET front = NULL, rear = NULL
    Initialize int len = 0

    func isempty()
    # Check if the queue is empty
        if front is NULL then
            return true
        else
            return false

    func push (PACKET x)
    #Add an element to the queue at rear
        if isEmpty() then
            # first element
            front = x
            rear = front
        else
            # link to next packet
            assign x to rear.next
            rear = rear.next
        len = len+1
```

```
func pop()
#Remove element from front of queue
    if not isempty() then
        if front == rear then
            front = rear = NULL
        else
            front = front.next
        len = len-1

func front_val()
# return front val of queue
    return front

func qlen()
#length of queue
    return len
```

3. Question 2

Now, with the same FIFO Queue, we now additionally have 512 kbytes of initial data burst. There is no change in any other parameter from the previous question.

3.1. Observations

Even in this case, there are no data packet drops. However, between t=8s and t=9s, we observe a drop of 745 video packets. The reason attributed to the no drop of data packets is due to their high tolerance to delay. They can take delays of upto 30 seconds whereas video packets can only take delays upto 1 second. After t = 9s, there are no drops of either types of packets.
For t=120s:

Total Video Packets Dropped = 745
Video Drop Percentage = 100*745/(96*120) = 6.46%

Data Drop Percentage = 0%

Since there are no data packets dropped, retransmission of data packets wouldn't change this answer.

We observe that after t=15s, the router behaves similar to the way it behaved in question 1. This can we attributed to transmitting all excess data packets from the burst and reaching a 'steady state'. This behaviour can also be seen from the delay times of transmitted data and video packets. We plot the delay times to appreciate this better in Figure 1 and 2.

Steady State Data Delay: 0.057289 s
Steady State Video Delay: 0.0284821 s
Avg. Data Delay: 1.87962 s
Avg. Video Delay: 0.0540962 s

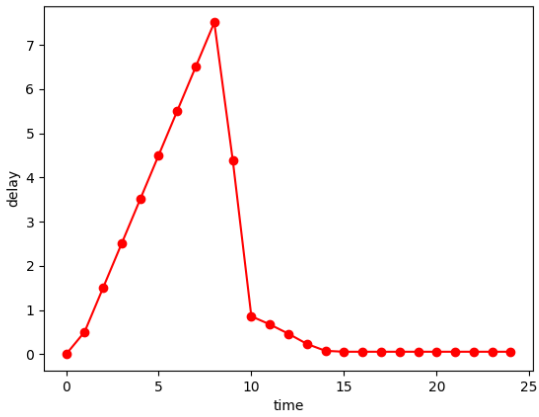

Figure 1. Plot of Delay Time for Data Packets

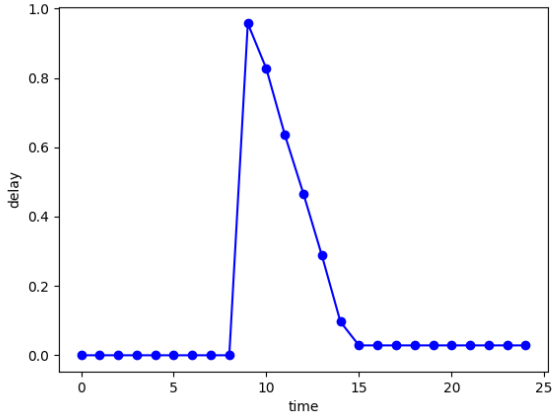

Figure 2. Plot of Delay Time for Video Packets

4. Question 3

In this question, we have been asked to design a data structure such that video packets are pushed at the front and data packets at the back. This is essentially a Priority Queue where videos are given more priority than data packets. We use a linked list implementation of the priority queue where the elements given more priority are assigned a lower priority number than those which are given less priority. We have an initial data burst of size 512 kbytes in this question.

4.1. Pseudocode for Priority Queue

```
Implementing Priority Queue
using struct PACKET in a Linked List

Priority_Queue

    initialize PACKET ptr front = NULL
    # front pointer to serve as head
```

```
initialize int len=0

func isempty()
    return front == NULL

func pop()
    if not isempty()
    initialize temp = front
    # move head to next element
    head = head->next
    delete temp
    len = len-1

func push(PACKET p)
    initialize a = front
    # ptr a points to front
    initialize PACKET ptr temp
    assign value of p to temp
    if (front->priority > p->priority)
    then
        #low priority val than front
        #insert at front
        temp->next = front
        front = temp
    else
        while (start->next != NULL and
        start->next->priority < p) do
            # Traverse through the q
            # until we find the index
            # to insert new packet
            a = a -> next
        temp->next = a->next
        a->next = temp
    len = len+1

func frontval()
# return front value of queue
    return *head

func qlen()
    return len
```

4.2. Methods and Observations

The priority value for video packets should be lesser than those of data packets arriving at the same time. We use 4 types of priority functions in this implementation.

4.2.1 Priority Function 1

One trivial assignment of priority is,
Priority = 0 for video packets
Priority = 1 for data packets

By this assignment, we make sure that if there is any video packet in the queue then it is more towards the front than the first data packet. Thus, we expect to transmit all video packets in the queue before the first data packet is transmitted. This leads to 0 drop percentage for video packets. However, due to the initial burst of data, all initial data packets may not be transmitted and some of them are dropped after 30 seconds, between t=30s and t=31s, due to the max delay of 30 seconds for data. After they are dropped, they are retransmitted the next second. But the output bandwidth is high enough so that all the data packets are transmitted before the next 30 sec interval ending at t=60s. No packets are dropped after t=31s. The 'steady state' is reached within t=60s. The following is the observation for an initial data burst of size 512 kbytes,

For t=120s
%Dropped Video Packets = 0%
Number of Data Packets Dropped = 32
%Dropped Data Packets = 32/(120*6+512) = 2.597%

The 32 dropped are from the initial burst data only

We now make the initial data burst size 5120 kbytes. Now, between t=30s and t=31s, 4640 data packets are dropped. There are no video packet drops. Dropping a high number of data packets is unavoidable due to the constraints on the dropping of video packets. However, as we see in the next types of priority functions, we can still reduce this number of dropped data packets.

4.2.2 Priority Function 2

We now use a different priority function. We use the time when a packet would be expired as the the priority value. When a data packet arrives, we assign priority as, time of arrival + 30. When a video packet arrives, we assign priority as, time of arrival + 1. Hence, when a data packet and video packet arrive simultaneously, video, having lower priority value, is inserted at the front and data goes to the rear. This assignment makes the drop percentage for both data and video as 0. This priority function, by enqueing packets according to the expiry time prevents any drop from happening. The following is the observation for an initial data burst of size 512 kbytes,

For t=120s,
%Drop Video Packets = 0%
%Drop Data Packets = 0%

We now make the initial data burst size 5120 kbytes. Now, between t=30s and t=31s, 4593 data packets are dropped. There are no video packet drops. We see that the number of dropped data packets has reduced from the

previous priority function.

4.2.3 Priority Function 3

In this case, we slightly modify priority function 2. The priority is the same as function 2 for data packets. But, for video packets, we can exploit the 5% allowance over the drop percentage. We do this by changing the video priority to, time of arrival + (1+0.05)*1 .
Thus, we have slightly decreased the priority of video by decreasing the priority value of video. This is to reduce the drop of data packets when the burst size is higher. This would drop some video packets, within the tolerance of 5%, to allow for transmission of some data packets. For an initial data burst of size 512 kbytes, the dropped video and data packets still remain 0 with this priority function.

We now increase the initial data burst size to 5120 kbytes. Now, between t=30s and t=31s, 4591 data packets are dropped. There are no video packet drops. Here, the number of dropped data packets is lesser than in the case of the previous 2 priority functions.

4.2.4 Priority Function 4

In the last case, we assign,
Priority = 0 for video packets
Priority = 1 for data packets
In addition to this we assign some video packets, less than 5 % of incoming packets, a priority of 2. In effect, we push them to the end of the priority queue. By this, we intentionally drop some video packets, within the drop tolerance, to allow for more data packets to be transmitted.

In our case of a data burst of 512 kbytes, the first 168 video packets are dropped between t=42s ans t=43s. These have arrived between t=0s and t=42s and have been pushed to the back of the priority queue. There are no video or data drops elsewhere.

For t=42s,
%Drop Data Packets = 0%
%Drop Video Packets = 168/(42*96) * 100 = 4.16667%
%Drop Video Packets $<$ 5%

For t=120s,
%Drop Data Packets = 0%
%Drop Video Packets = 168/(120*96) * 100 = 1.45833%

We now make the initial data burst size 5120 kbytes. Now, between t=30s and t=31s, 4580 data packets are dropped. There are no video packet drops. Here, the number of dropped data packets is the least among all of our priority functions.

4.3. Comparison of Priority Functions

Priority function 1, makes sure that we don't drop any video packets in any case. However, for a higher size of initial data burst, the number of data packets dropped becomes too high. Even when the initial data burst is small in size, some data packets are dropped.

Priority function 2 too maintains a zero video drop. Additionally, it also maintains zero data drop for a smaller initial data burst. It performs better by dropping a significantly smaller number of data packets even in the case of a large initial data burst.

Priority function 3 performs similar to priority function 2 in the case of a small initial data burst. It performs marginally better with a larger data burst. Similar to priority function 2, video packets are never dropped.

Priority function 4 performs worse than all the other priority functions in terms of video packet drops, by having a constant video drop rate. However, this video drop rate is beneficial in the case of a large initial data burst when this function has the least number of dropped data packets. This video drop rate is however unnecessary in the case of a small data burst.

5. Code Input/Output

There are 2 codes submitted. Code 1, ee16b036_q2.cpp, implements question 1 and 2. Code 2, ee16b036_q3.cpp, implements question 3.

Code 1 takes 4 parameters as inputs:
transmission rate in kbytes per sec
video input rate in kbytes per sec
initial data burst in kbytes
data inter-second arrival size in kbytes

Code 2 takes 5 parameters as inputs:
transmission rate in kbytes per sec
video input rate in kbytes per sec
initial data burst in kbytes
data inter-second arrival size in kbytes
priority function to be used (1,2,3 or 4)

Both the codes output the parameters as required in the question every second. The condition parameter displays "EXPIRED" if packets have expired and have been dropped. Also, at the end, they output the average delay times in the last second for both data and video packets.