CS4150: Computer Networks Lab

Lab5

111901030
Mayank Singla

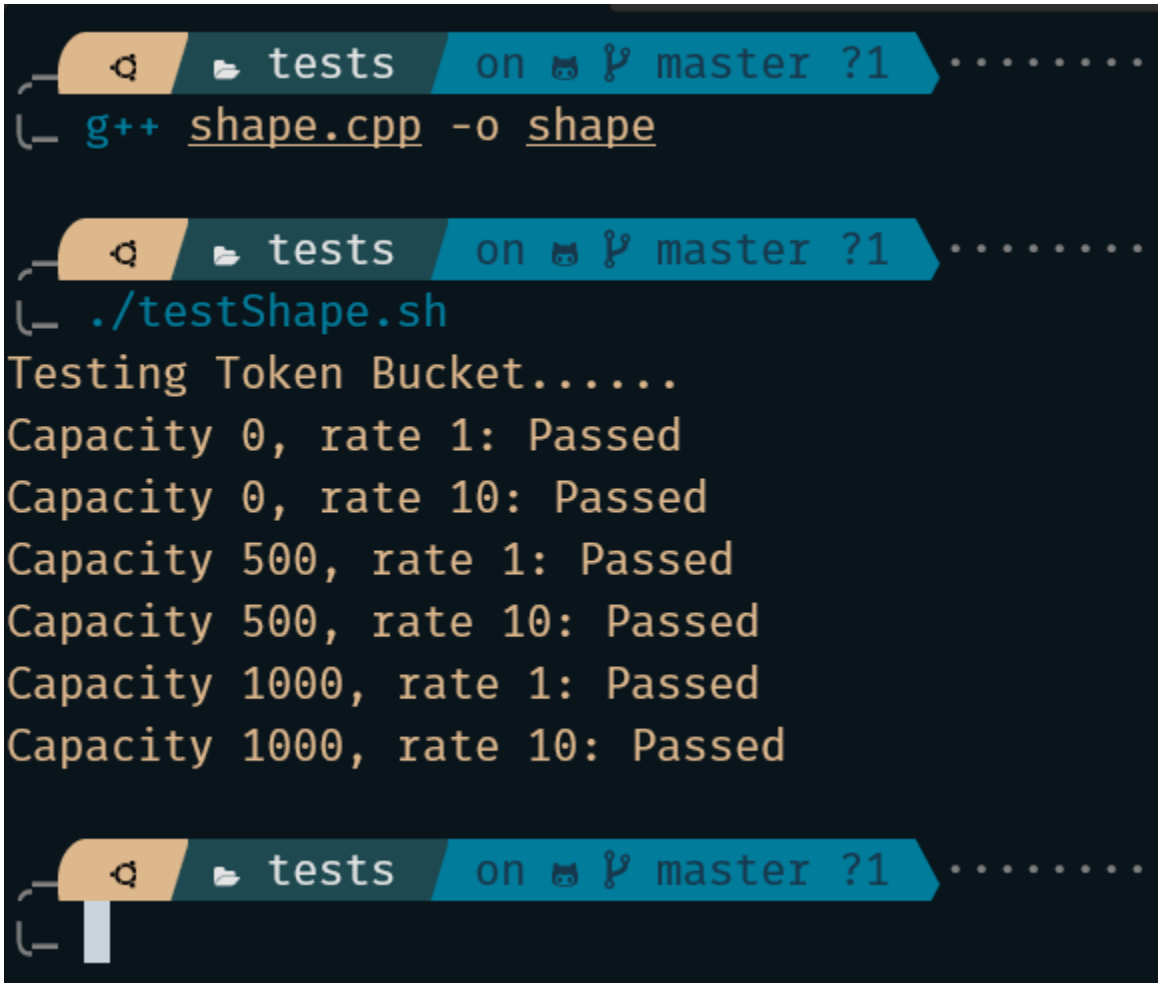
Q1. Write a C program called "shape" that implements token bucket-based traffic shaping. This function should have two input arguments. The first argument is an integer and denotes the size of the token bucket, whereas the second argument is the rate at which tokens fall into the token bucket. The file "arrivals.txt" contains the list of packet arrivals. A line denotes each packet arrival in this file as "X Y Z", where X is arrival time, Y is packet ID, and Z is packet length. You should be able to invoke your program as follows:

```
./shape 250 4.0 < arrivals.txt
```

The above invocation should shape the traffic in the file "arrivals.txt" using a token bucket of capacity 250 and rate 4.0. The output of the above invocation should be, for each packet, "U V W" (without quotes), where U is the transmission time of the packet ("%.2f" format), V is the packet ID ("%d" format), and W is the length of the packet ("%d" format). Your program will be evaluated for 6 test cases by running the script "./testShape", and 5 points will be awarded for each test case passed.

I keep track of the number of tokens present at any time in the bucket and the transmission time of the last packet that was transmitted. Then as I keep reading the input packet, I checked first if the arrival time of the incoming packet is greater than the transmission time of the last packet. If it is not, then the transmission time for the current packet will start from the transmission time of the last packet. Else if it is, then in between this time, there will be tokens generated within this time interval up to the bucket size. I added them to the number of tokens generated and updated the transmission time for this packet as the arrival time for this packet. Then I checked if the packet length is less than the number of tokens I have in the bucket and in that case, I simply reduced the number of tokens by the packet length. If the packet length is greater than the number of tokens, then my transmission time will increase by the time for generating the extra packets needed for the extra packet length. Finally, I printed the transmission time for each packet as expected.

Here is the result of all the test cases.



Q2. Write a C program called "fifo" that implements a finite capacity FIFO queue. This function should have two input arguments. The first argument is an integer and denotes the buffer size (including the packet being served / HOL packet), whereas the second argument is the constant rate at which data is pushed to the output line of the queue. The HOL packet is held in a buffer until it is completely transmitted. The file "arrivals.txt" contains the list of packet arrivals. A line denotes each packet arrival in this file as "X Y Z", where X is arrival time, Y is packet ID, and Z is packet length. You should be able to invoke your program as follows:

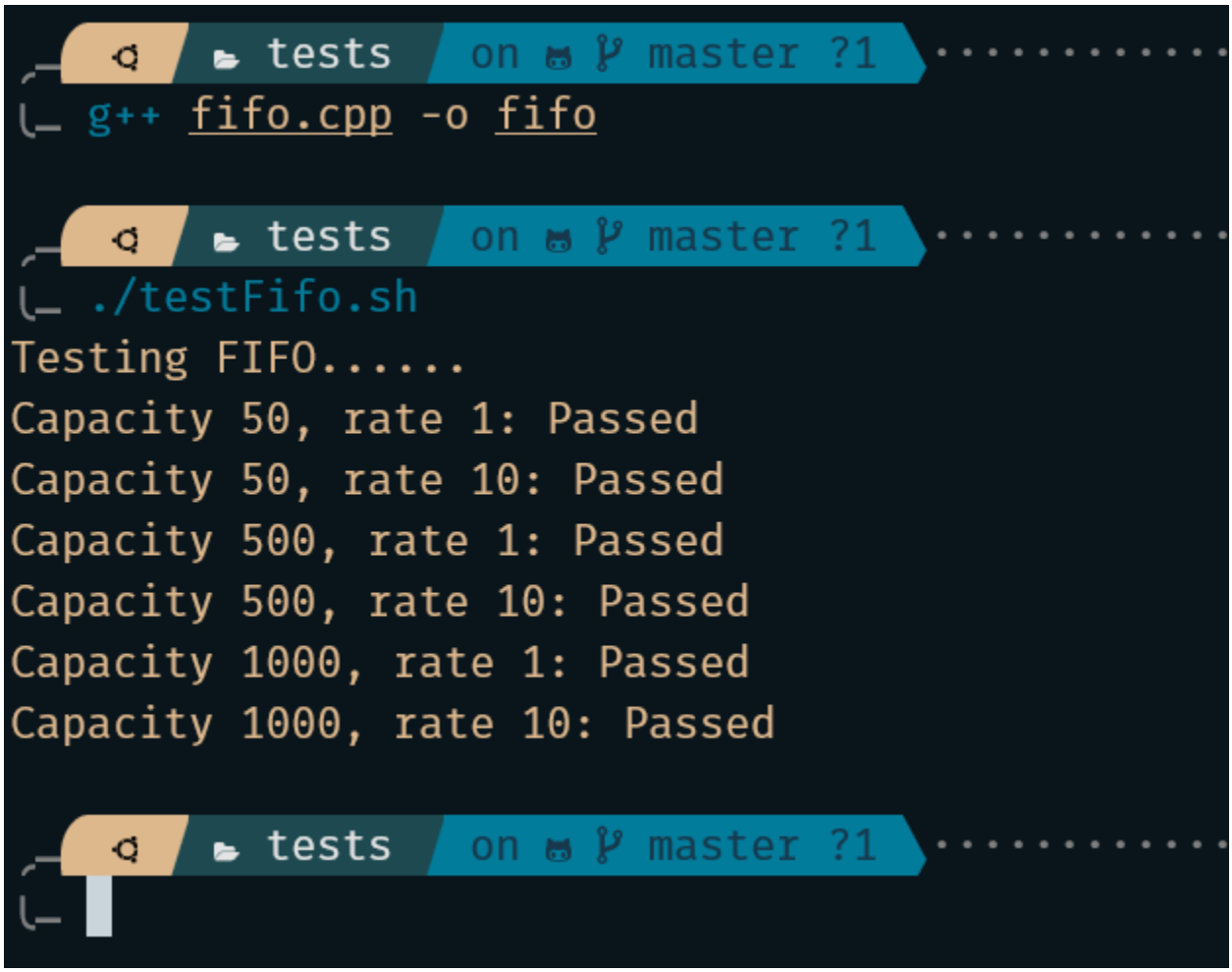
```
./fifo 250 4.0 < arrivals.txt
```

The above invocation should simulate the traffic in the file "arrivals.txt" passing through a FIFO queue of capacity 250 and output rate 4.0. The output of the above invocation should be, for each packet, "U V W" (without quotes), where U is the transmission completion time of the packet ("%.2f" format), V is the packet ID ("%d" format), and W is the length of the packet ("%d" format). Your program will be evaluated for 6 test cases by running the script "./testFifo", and 5 points will be awarded for each test case passed.

I am using a **deque** to simulate an iterative FIFO queue which will carry the information of the packets currently in the queue. For each input packet arrival, I am removing the packets from the

FIFO queue which are finished transmitting till the arrival of the current packet. Then I am computing the remaining capacity left of the FIFO queue. Here also, I am keeping track of the transmission time of the last packet in the queue. If the arrival time of the packet in the head of the queue is greater than that transmission time, I update the transmission time to the arrival time of the packet at the head since it also denotes the time at which the transmission for the next packet will start. Then, I am computing the length of the packet that could be sent till the current arrival time and if this length is less than the length of the packet at the head of the queue, then no further transmission will be possible till now and I simply subtract the lengths of the packets from the capacity. If it was greater, then the packet could be transmitted and update the transmission time by the time it takes the packet to transmit based on the output data rate and printed the transmission time for that packet. After calculating the capacity left in the queue, if the new packet length is less than that, then I push the packet into the queue else there will be a packet loss.

Here is the result of all the test cases.



Q3. The traffic in file "arrivals.txt" is first passed through a token bucket of capacity 500 and rate x, and then passed through a FIFO queue of capacity 1000 and rate 10.0. Write a C code/bash script to find the largest x (up to 6 decimal places) which ensures that no packets are dropped by the FIFO queue.

We can do this question by iterating for all the values of x starting from x = 0.000001 as it can have a maximum of 6 decimal places to x = 10, as 10 is the out data rate of the FIFO queue and token rate greater than that will always lead to some packet loss. But, if we observe, we can see that if we can achieve no packet loss for some value of x, then we will be able to achieve no packet loss for values smaller than that value also, so we need to check for the values of x greater than that only. Also, if for some value of x there is a packet loss, then for all the value of x greater than that, there will be again packet loss. So, based on this we can do a binary search for the values of x from 0.000001 to 10 and optimally find the largest value of x.
So, in my bash script, I simply wrote a basic iterative algorithm for the binary search on the values of x. To check if we have a packet loss for some particular value of x, I created a function **checkAtX** and I pass that value to the **shape** program, then pipe the output of that to the **fifo** program. If there is no packet loss, then the original number of lines in the input file (**arrivals.txt**) and the final output of the **fifo** program should be the same. Based on this logic, I am checking if there is any packet loss or not for a certain value of x.
The largest value of x computed is **1.956414**

```
└─ bash q3.sh
The largest value of x is: 1.956414
```