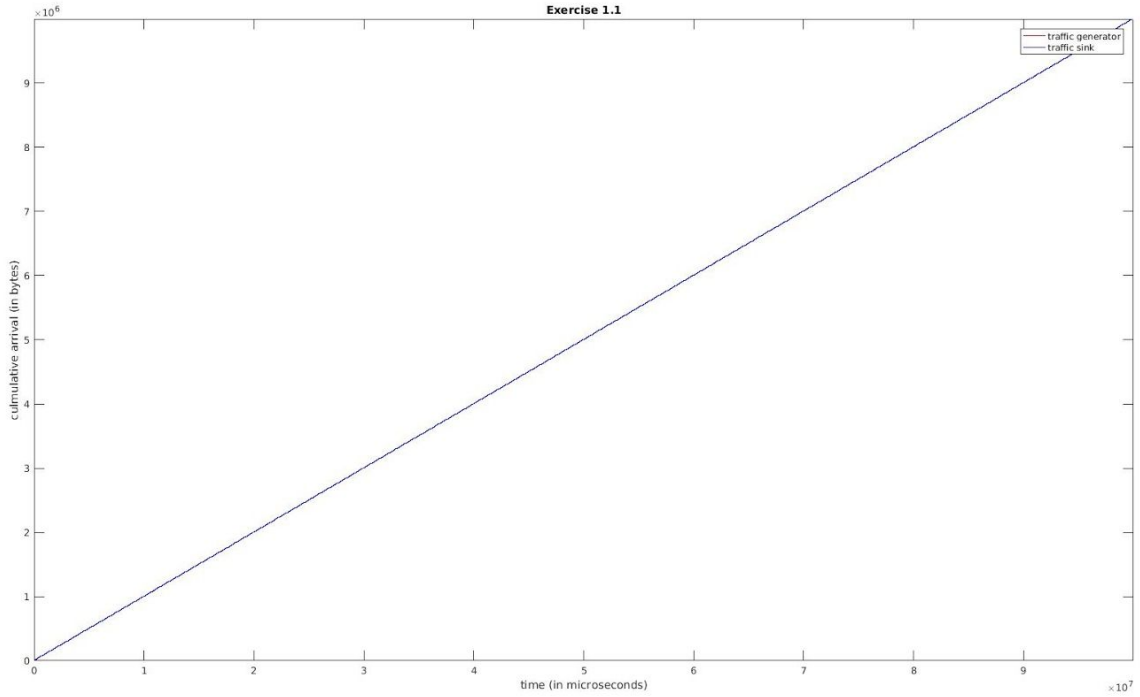
ECE466 LAB2b

Chengchen Yang 1000285880
Yu Li 1000173554

Part1 A Reference Traffic Generator

Exercise 1.1 Create and Test Reference Traffic Generator

- Prepare a plot that shows the difference of trace file and the output file. For example, you may create two functions that show the cumulative arrivals of the trace file and the output file, respectively, and plot them as a function of time.
- The figure below is the traffic generator versus the traffic sink.



In the Traffic Generator, we set the values of T = 100ms, N = 100 packets, and L = 100 bytes. We plotted an sample of 100,000 points in the graph above, by using the m1/plot11.m file. Compare the difference between the generator and sink, we could see that they are only few very small to no delay.

Part2 Token Bucket Implementation

Describe the design of your implementation for the Token Bucket. Include your source code in the lab report. (Only include the source code of methods that you have modified).

Exercise 2.1 Complete the implementation of the Token Bucket

- Completed three functions getNoTokens, removeTokens, getWaitingTime in Bucket.java

- Below is completed implementation of token bucket implementation, basically, when updating tokens, we will use the time difference since last update then multiply by the token increasing rate. One thing we need to make sure is no of token cannot exceed the maximum token bucket size. In addition, remove tokens is implemented as following:

```
private void updateNoTokens()
{
    // Currently this code segment is empty.
    //
    // In Lab 2B, you  add the code that performs the following tasks:
    //     1. Compute the elapsed time since the last time  the token bucket was updated
    //        and update the variable "lastTime"
    //     2. Update the variable noTokens, which stores the updated content of the token bucket
    //

    //get curr_time
    long curr_time = System.nanoTime();
    //get time diff since last update
    long time_diff = curr_time - lastTime;
    //update noTokens accordingly
    noTokens += time_diff/tokenInterval;
    if (noTokens > size) {
        noTokens = size;
    }
    //
    lastTime=curr_time-(time_diff%tokenInterval);
}
```

- Remove tokens

```
    updateNoTokens();

    // Currently this code segment is empty.
    //
    // In Lab 2B, you  add the code that removes the required number of tokens
    // and returns false if there are not enough tokens.
    //
    if (noToRemove < noTokens) {
        noTokens = noTokens - noToRemove;
    }
    else {
        return false
    }
    return true;
}
```

Exercise 2.2 Validate your implementation

- We have added all the data information in the table below

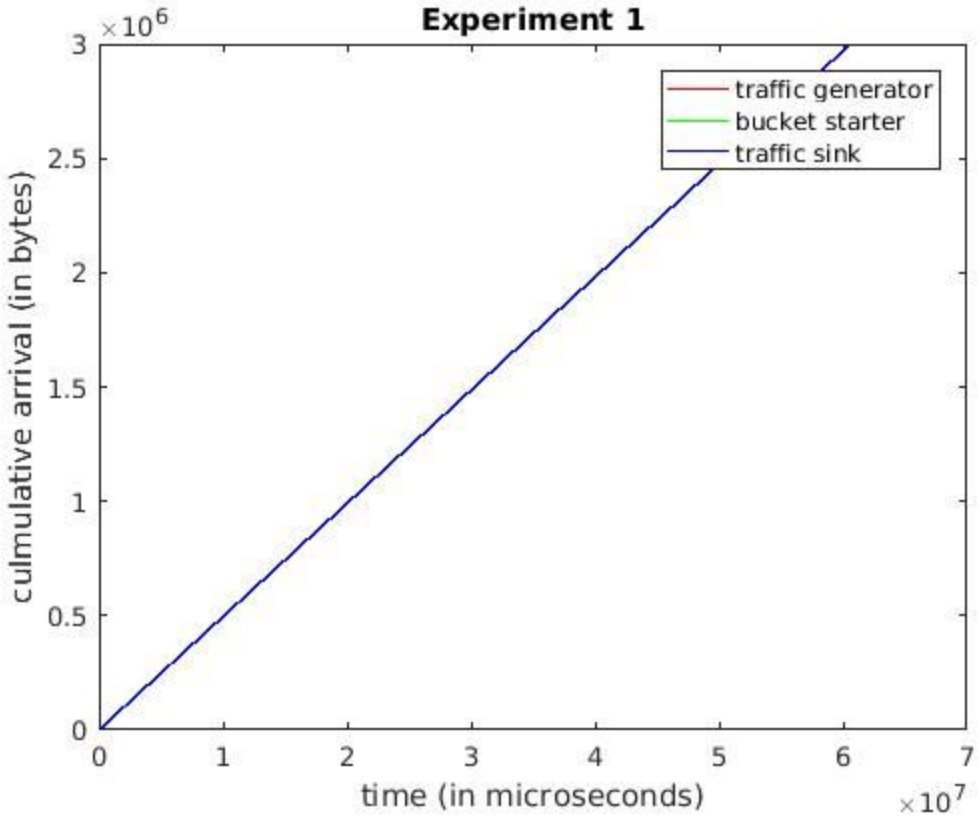
Experiment	N (packets)	L (bytes)	T (ms)	TB size (bytes)	Source Rate (Mbps)	TB Rate (Mbps)
1	1	100	2	100	0.4	2
2	10	100	10	500	0.8	1

3	1	100	1	100	0.8	0.8

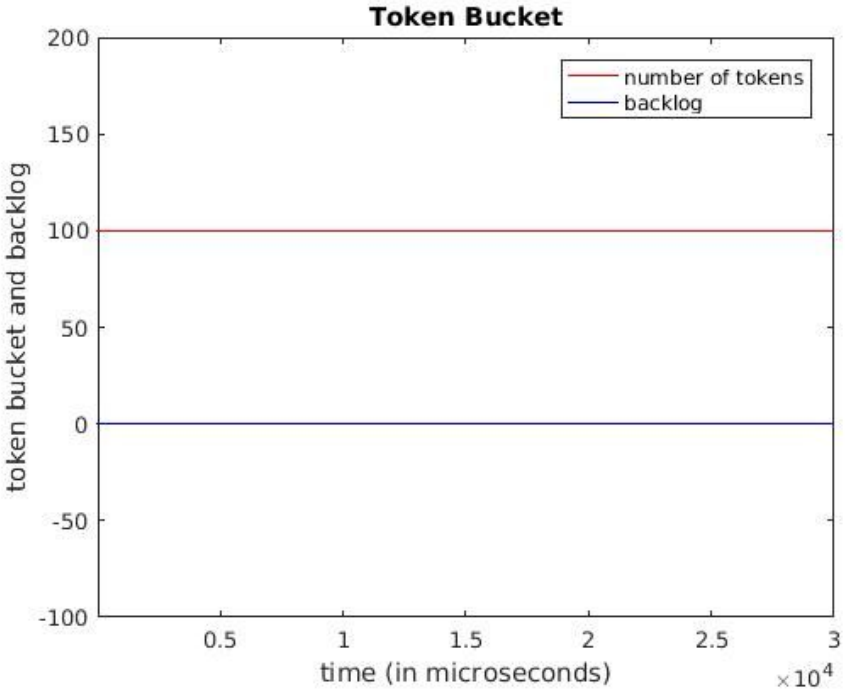
The Experiment 1 has source Rate < TB rate and source rate is much smaller than TB rate.
The Experiment 2 has source rate < TB rate and source rate is closer to the TB rate.
The Experiment 3 has source rate equal to the TB rate.

Exercise 2.3 Generate plots for the experiments

- This exercise is used to show the plot from the previous exercise
- **Experiment 1 Source Rate < TB rate, N = 1, TB size = 100 bytes**

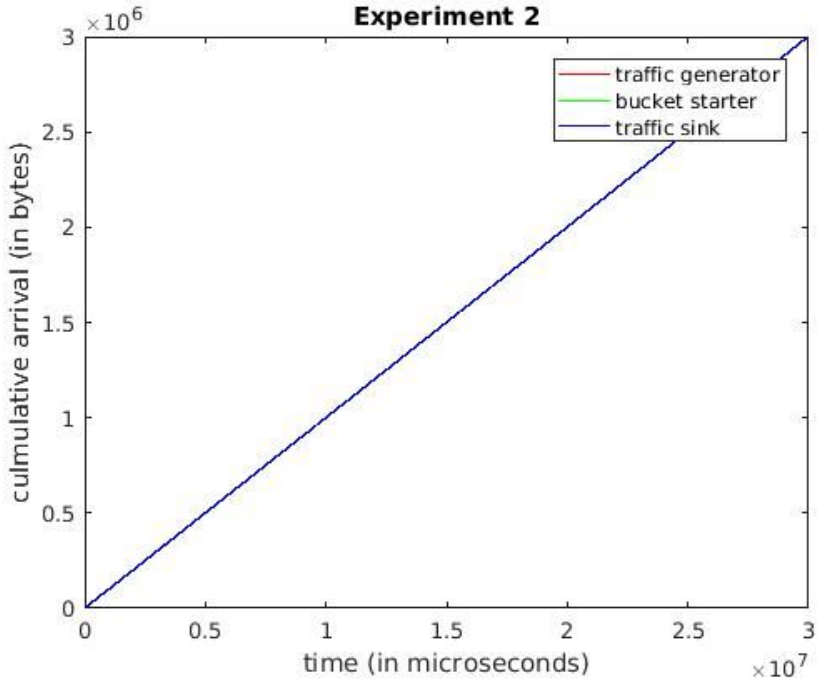


-
- The three output curves are close to each other and have no delays, because that the token bucket could handle all the incoming arrival packets

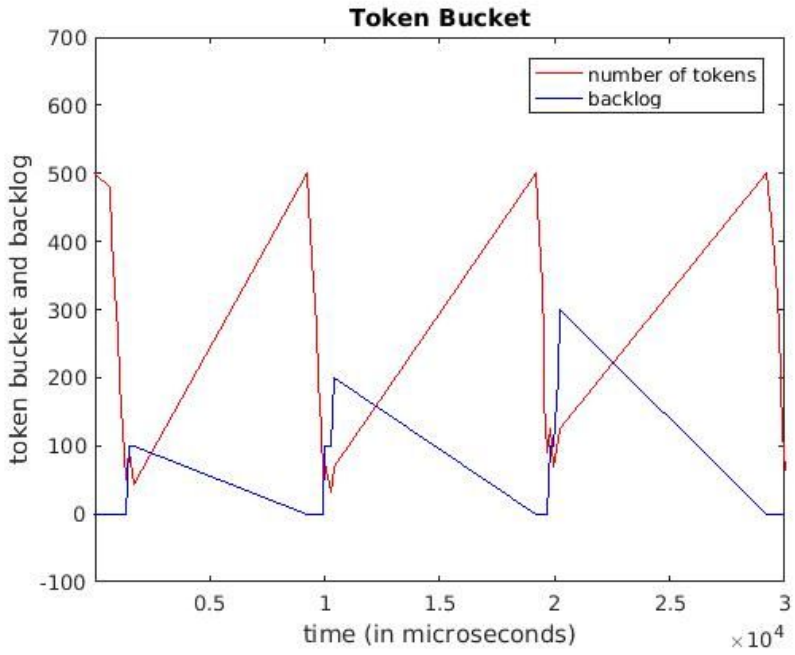
- 

- There is no backlog, because the rate of token bucket is bigger than the arrival (source) rate, so the token bucket is always full.

- **Experiment 2 Source Rate < TB rate, N = 10, TB size = 500 bytes**

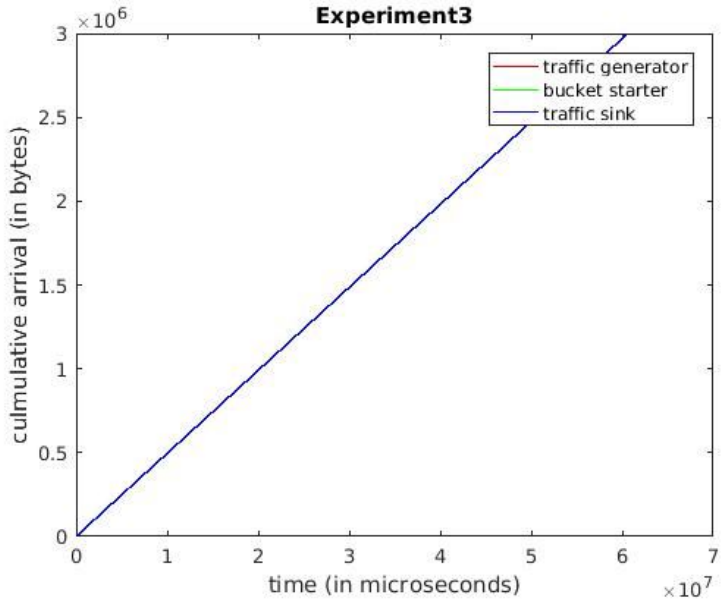
- 

- Since the TB rate is larger than the source rate, so that the token bucket and the traffic sink curves are able to keep track of the traffic generator's shape.

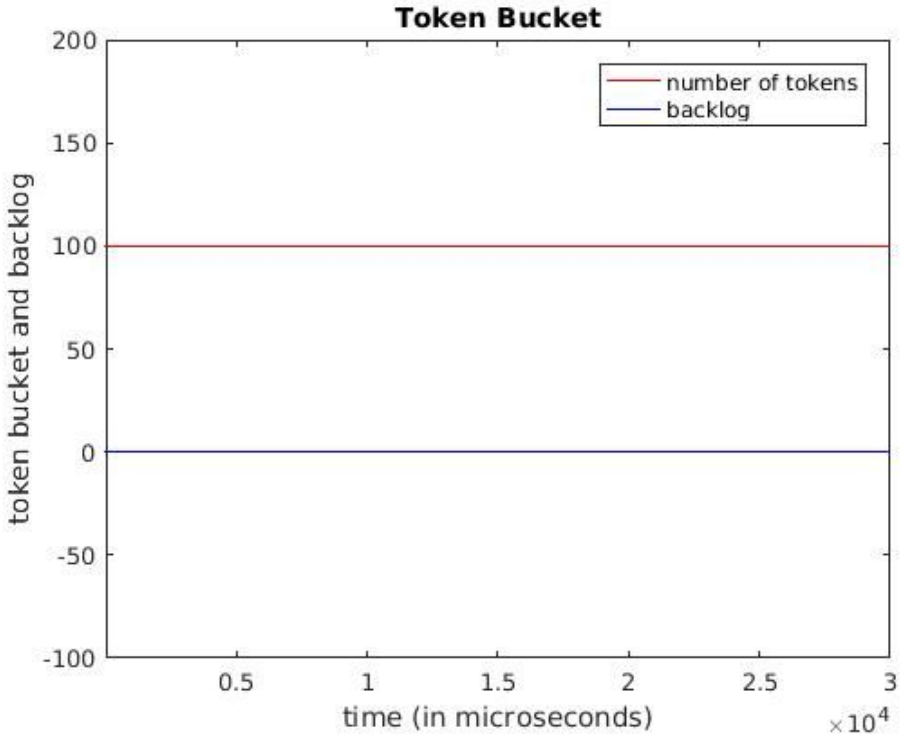


- The token bucket starts at a full size 500 bytes and reduce to 0, which means it releases 5 packets quickly and then stores tokens once all packets have been sent out. The backlog reaches maximum as the number of tokens in token bucket reduce to 0.

- **Experiment 3 Source Rate ~= TB rate, N = 1, TB size = 100 bytes**



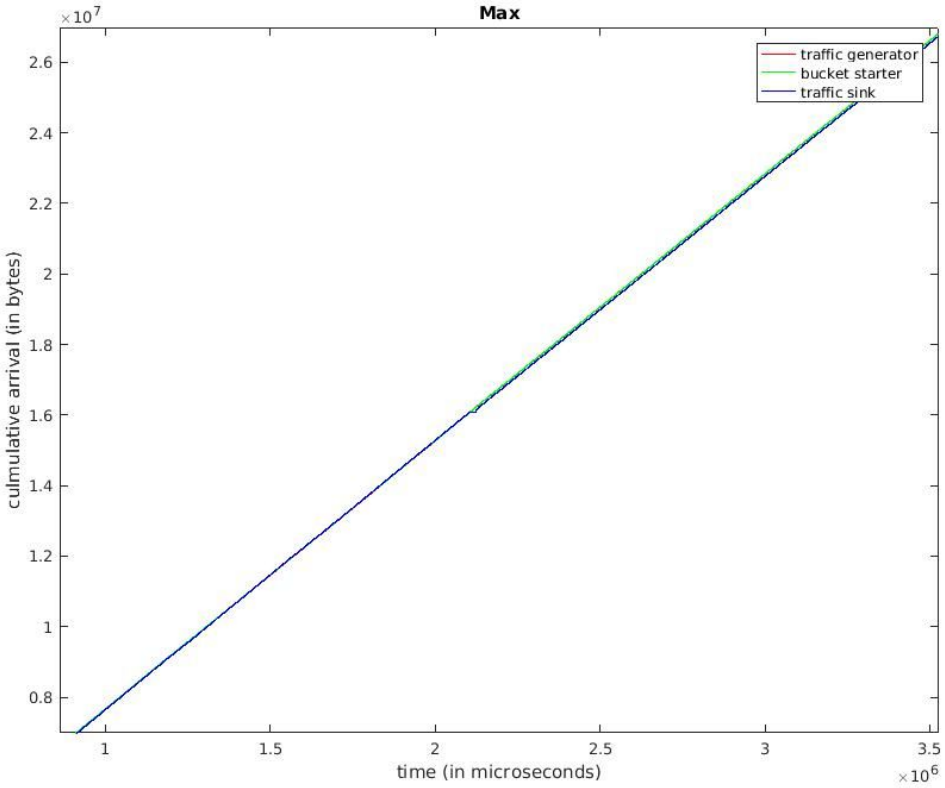
- The three curves are close to each other, as the rate of the TB rate is equal to the source rate.



-
- The graph shows that our outcome is the expected behaviour, there is no backlog in the buffer as the token bucket rate is equal to the source rate and starts with a full bucket.

Exercise 2.4 Maximum rate of Token Bucket

- Determine the maximum arrival rate of traffic that can be supported by your Token Bucket. Support your findings with an experiment and (at least) one data plot.
- We found that the maximum arrival rate that can be supported by our token bucket with 10000 bytes size and rate of 64 Mbps. N = 8 packets, L = 955 bytes, T = 1ms, we initially set up L = 1000 bytes and found out it is too big, and we found that the 900 is too small, then we use binary search to find the L = 955 bytes. So that maximum rate of arrival is 61.12Mbps.



-

- As can be seen in the graph, we can see that the sink has little offset of the generator. Curves are close and the backlog is will not increase to infinity, thus we have reached a maximum rate of arrival that the token bucket can support.

Part3 Engineering Token Bucket Parameters

Discuss your method for selecting the token bucket parameters

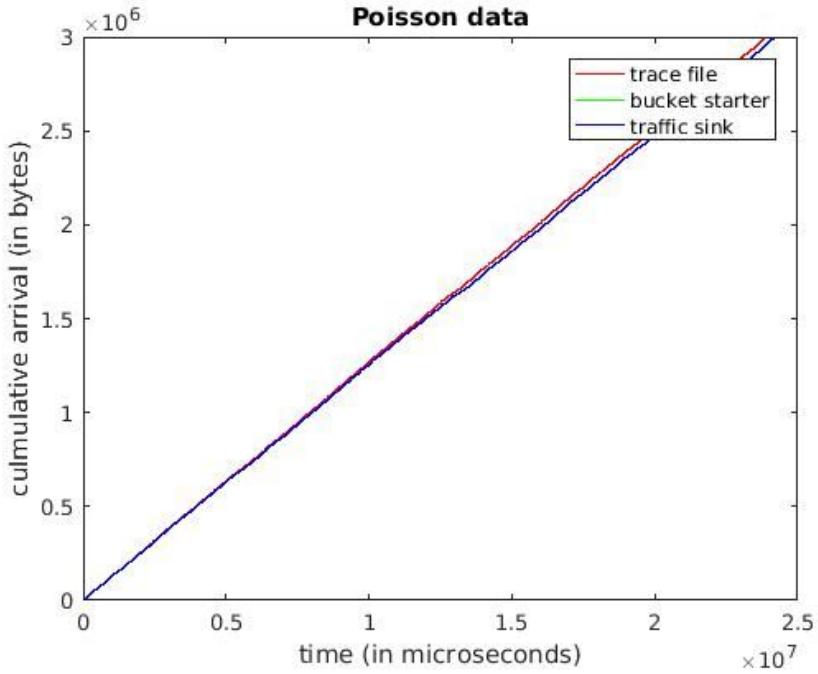
We estimated the initial token bucket parameters by analysing the trace file, specifically, set the max_burst(size_TB) equal to the largest packet size in the trace file, and set rate_tb as the average rate of trace file which will be less than the peak rate. After we set the initial parameter, we then use binary search to adjust the parameters.

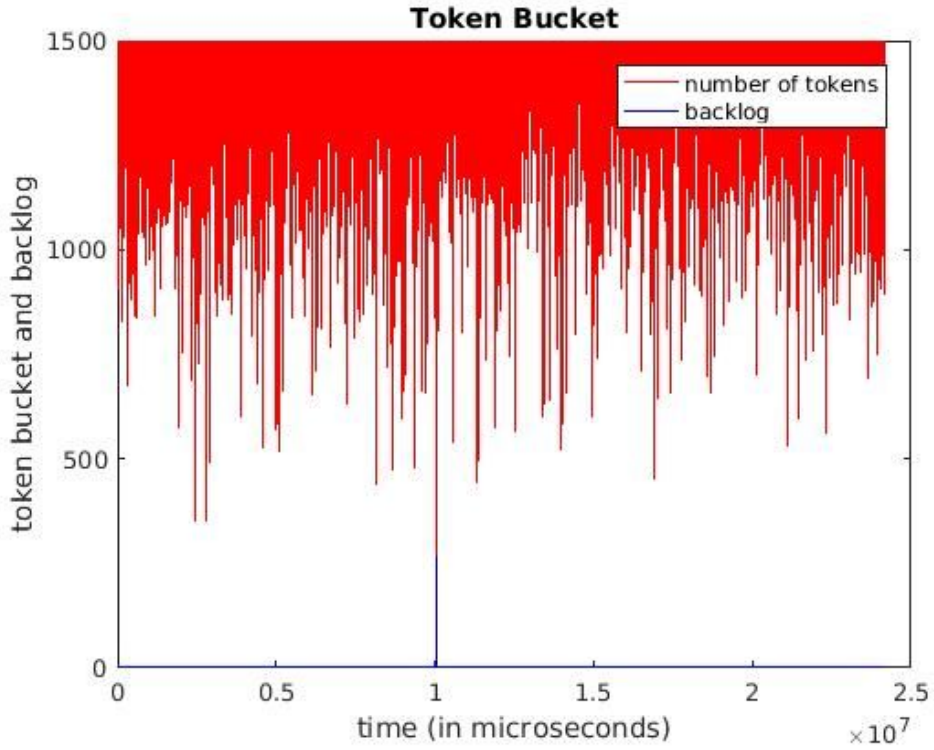
Exercise 3.1 (Long-term) Bandwidth is cheap

1.Poisson Traffic: size_TB = 1500 Bytes rate_TB = 1.0006Mbps -> 2.4 Mbps

- The maximum size of the packet from poisson data is around 1500 bytes, so that we choose 1500 bytes as the size of token bucket. We initially chose 1.0006 Mbps as the token bucket rate, because this is the rate that we got from lab1.

However, we found out the token bucket rate is not good enough for the trace file, so we increased the speed to 2.4 Mbps and got the following results in the figures below.

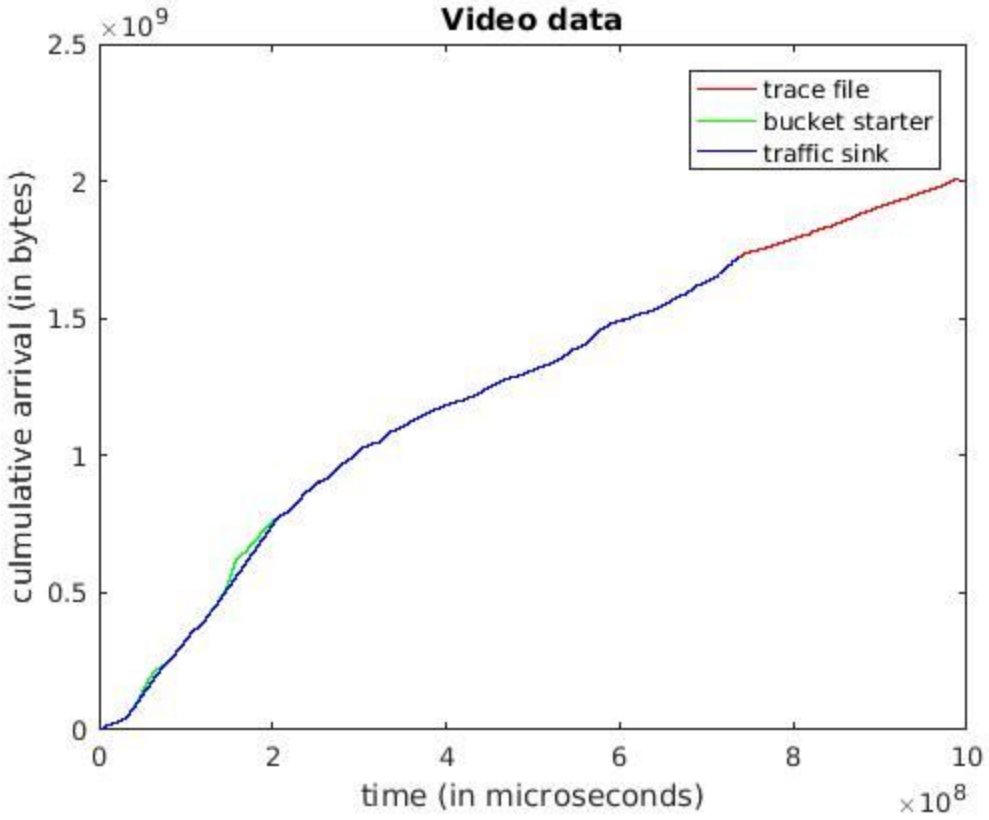
- 

- The three curves are close to each other and behave as expected.
- 

- The backlog and number of tokens from the output file behave as expected.

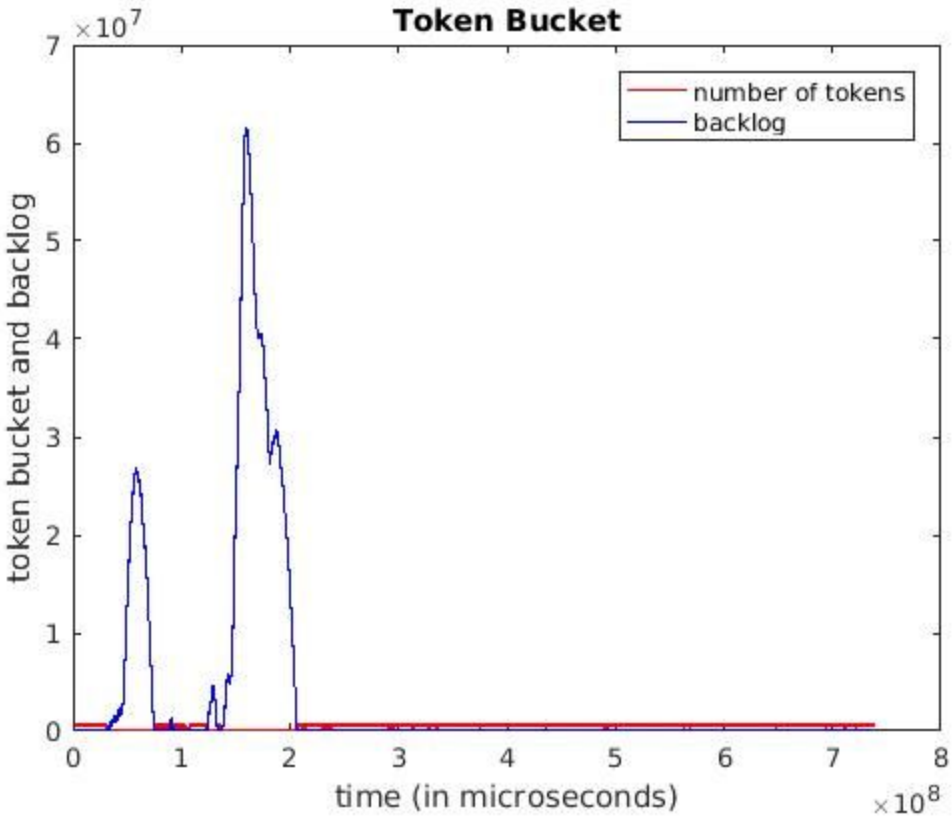
2.Video Traffic: Size_TB = 660000 Bytes rate_TB = 14.4 Mbps -> 35.856 Mbps

- The maximum size of the packet from video data is around 660000 bytes, so that we choose 660000 bytes as the size of token bucket. We initially chose 14.4 Mbps as the token bucket rate, because this is the rate that we got from lab1. However, we found out the token bucket rate is not good enough for the trace file, so we increased the speed to 35.856 Mbps and got the following results in the figures below.



-

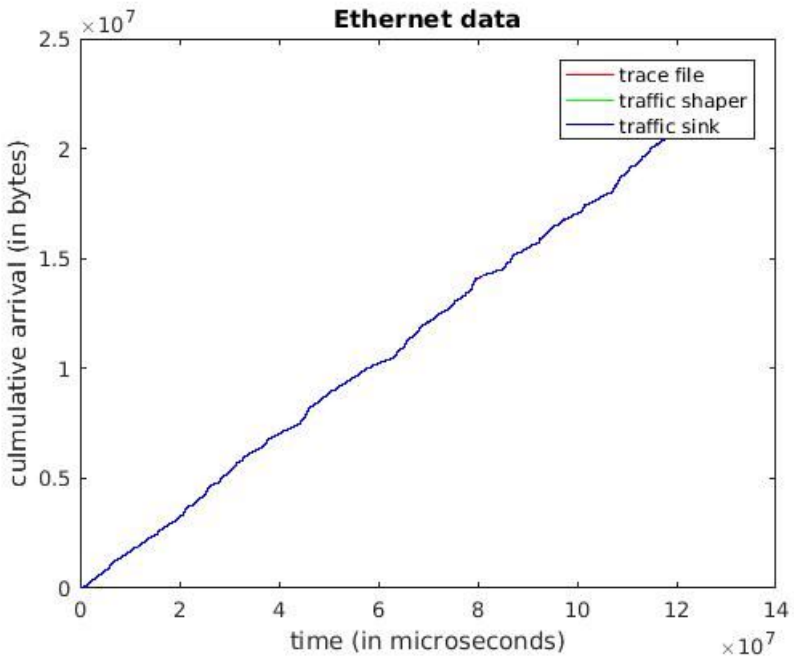
- We can see that there is some delay in the graph above, but the overall curves trend are close to each other. In the place where delay happened, where the backlog was higher than 0.



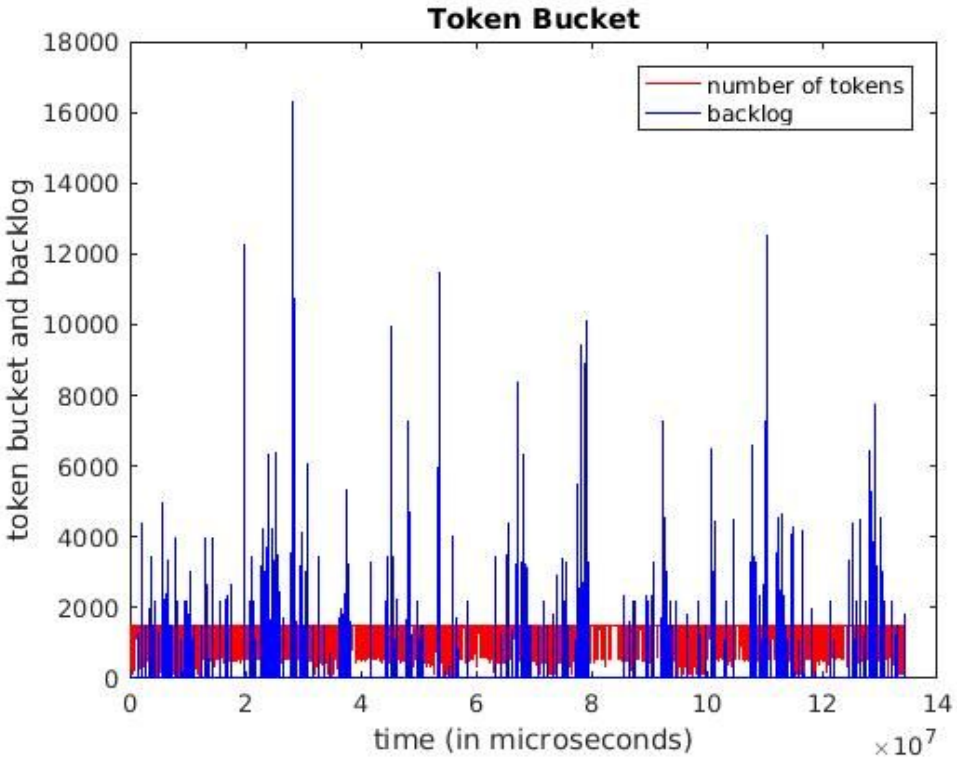
-

- As mentioned above, there were delays in the transfering trace file. In the places where backlog is not 0, it means that it receives large amount of data and then as the token bucket keeps filling tokens, it brings the backlog back to 0.

3. Ethernet Traffic: Size_TB = 1518 Bytes rate_TB = 1.1 Mbps -> 8 Mbps
    - The maximum size of the packet from video data is around 1518 bytes, so that we choose 1518 bytes as the size of token bucket. We initially chose 1.1 Mbps as the token bucket rate, because this is the rate that we got from lab1. However, we found out the token bucket rate is not good enough for the trace file, so we increased the speed to 8 Mbps and got the following results in the figures below.



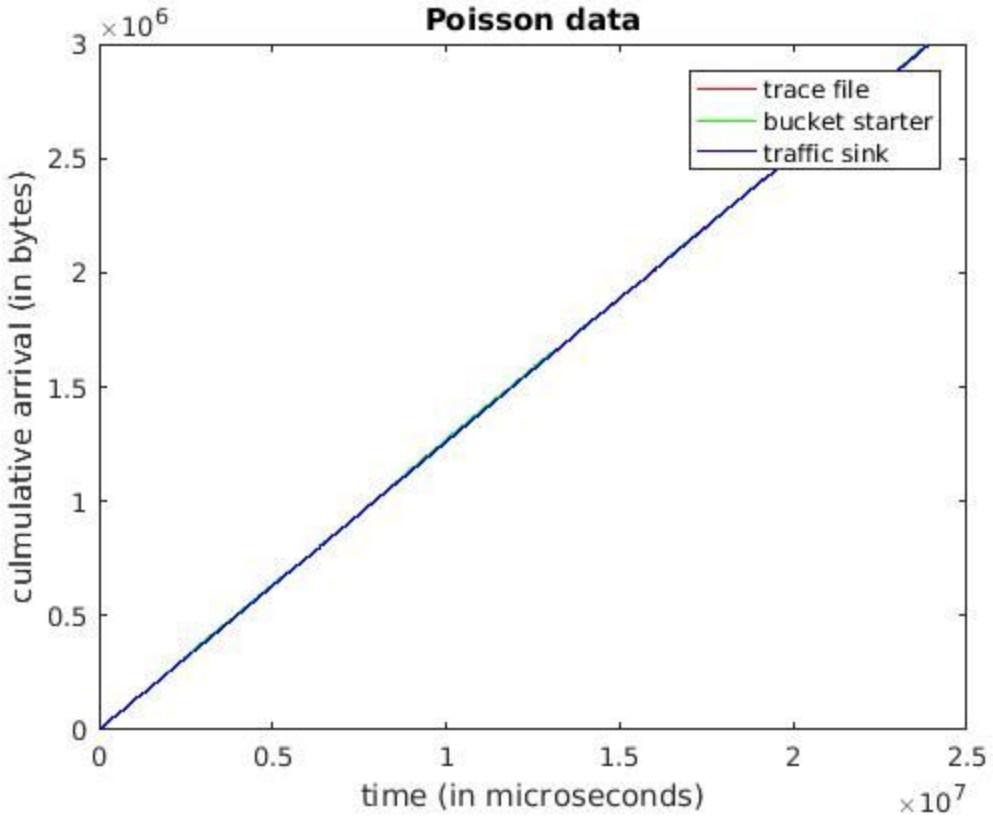
- The token bucket rate allows the sink and the generator have the similar trend in the graph shown above.



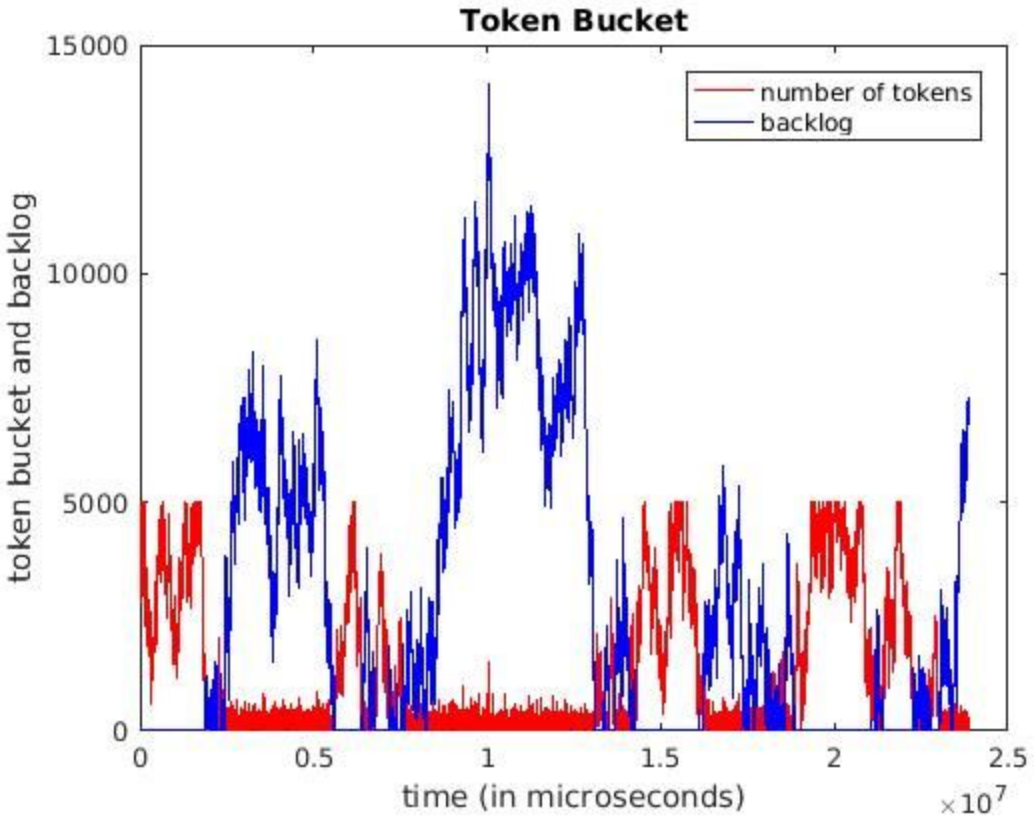
- From the image above, there are some places where the backlog is not 0, but the backlog is then be brought down by the token bucket rate.

Exercise 3.2 (Long-term) Bandwidth is expensive

1.Poisson Traffic: size_TB = 1500 Bytes -> 5000 bytes rate_TB = 1.0006Mbps

- The maximum size of the packet from poisson data is around 1500 bytes, and in this case, we have to increase the size of the token bucket to adapt the traffic coming in. So that we choose 5000 bytes as the size of token bucket. And we chose 1.0006 Mbps as the token bucket rate, because this is the rate that we got from lab1.
- 

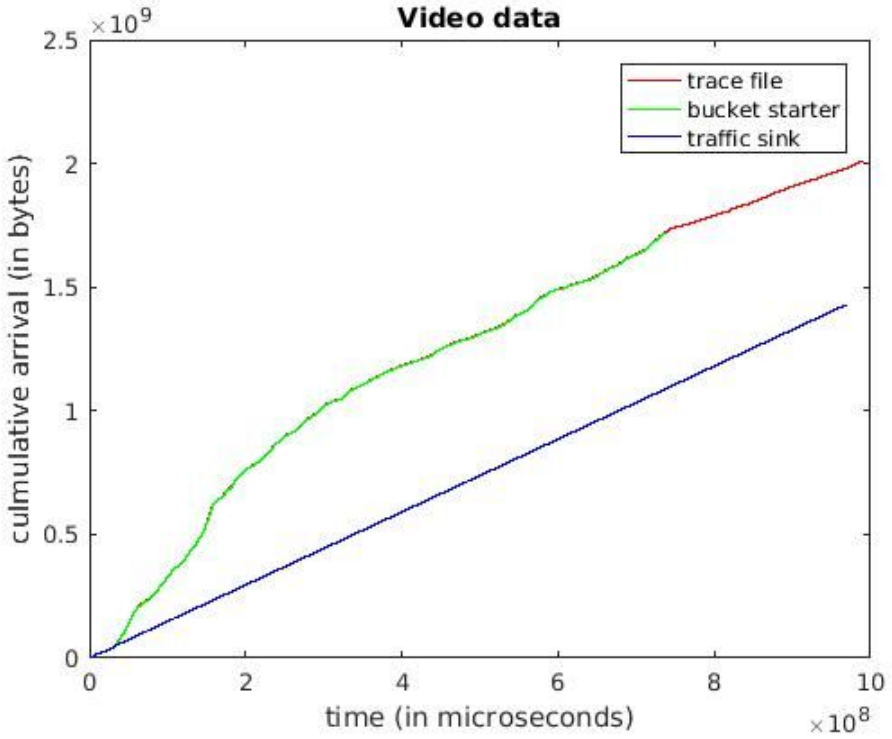
- From the figure above, we can see that the three files outputs are close to each other and as expected



-
- We set a high burst for the token bucket, so the backlog is finally brought down back to the 0.

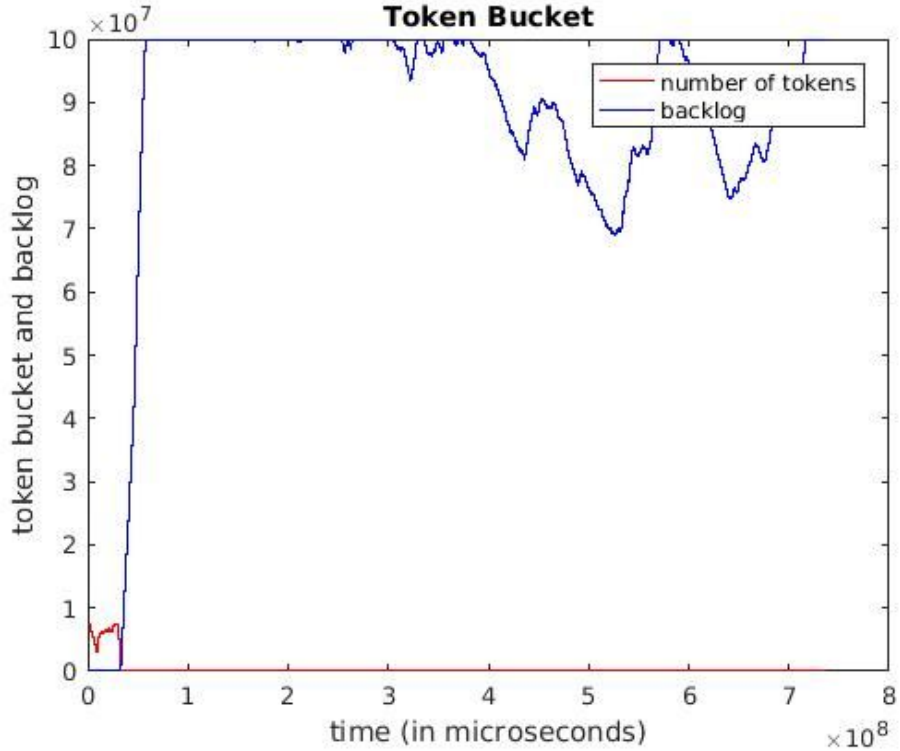
2.Video Traffic: Size_TB = 660000 Bytes -> 7543400 rate_TB = 14.4 Mbps

- The maximum size of the packet from poisson data is around 66000 bytes, and in this case, we have to increase the size of the token bucket to adapt the traffic coming in. So that we choose 7543400 bytes as the size of token bucket. And we chose 14.4 Mbps as the token bucket rate, because this is the rate that we got from lab1.
-



-

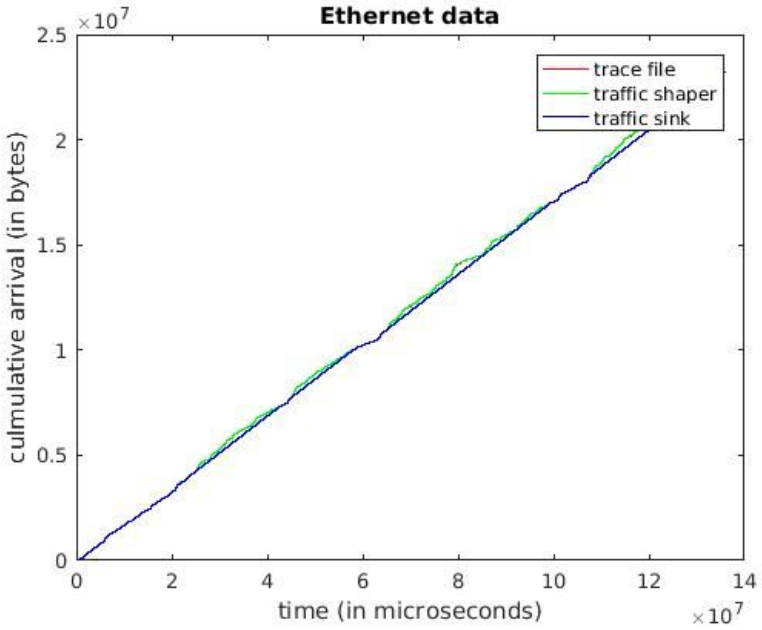
- The output from the traffic sink fails to catch up with the traffic trace file.



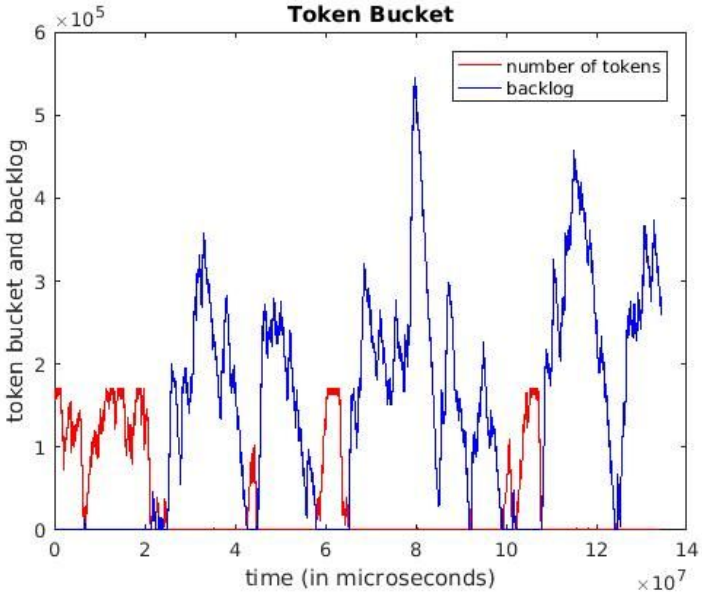
-

- Even though we have increase the size of token bucket, we still could not handle the large incoming packets.

3. Ethernet Traffic: Size_TB = 1518 bytes -> 170000 bytes rate_TB = 1.1 Mbps
   - The maximum size of the packet from poisson data is around 1518 bytes, and in this case, we have to increase the size of the token bucket to adapt the traffic coming in. So that we choose 170000 bytes as the size of token bucket. And we chose 1.1 Mbps as the token bucket rate, because this is the rate that we got from lab1.
   -



   - As the size of token bucket increases, we could see that the curves are close to each other, and low delays happen.
   -



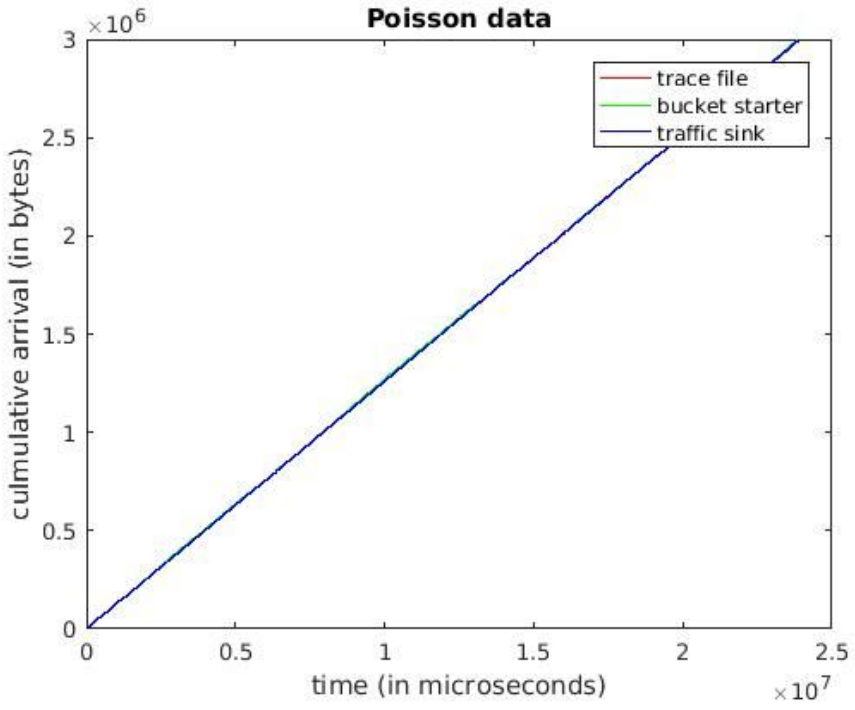
- As mentioned above, we still have some delays. However, these delays are bounded and finally brought down by token bucket rate.

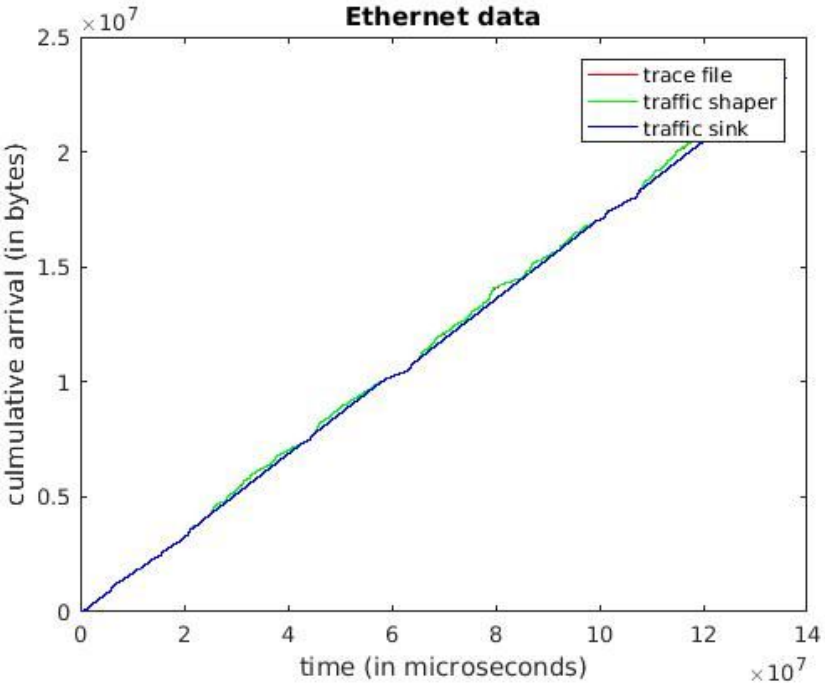
Compare the results for the three traffic types. For example, which type of traffic is most demanding in terms of bandwidth or memory requirements?

Traffic	Bandwidth Demanding Rank	Memory Requirements Rank
Poisson	3	3
Video	1	1
Ethernet	2	2

Which type of traffic benefits the most from traffic shaping? Support your findings with data (plots).

Since we can build a traffic shaper with a fix token bucket rate for both Poisson and ethernet Traffic, we believe that these two traffic benefits the most from traffic shaping.

- 

- 
Ethernet data
×10^7
2.5
2
1.5
1
0.5
0
culmulative arrival (in bytes)
0
2
4
6
8
10
12
14
time (in microseconds)
×10^7
trace file
traffic shaper
traffic sink