Lab 2b: Traffic Shaping II

Harmanprit Tatla – tatlahar
Shunta Chimura – chimuras

Exercise 1.1 Create and Test Reference Traffic Generator

Prepare a plot that shows the difference of trace file and the output file:

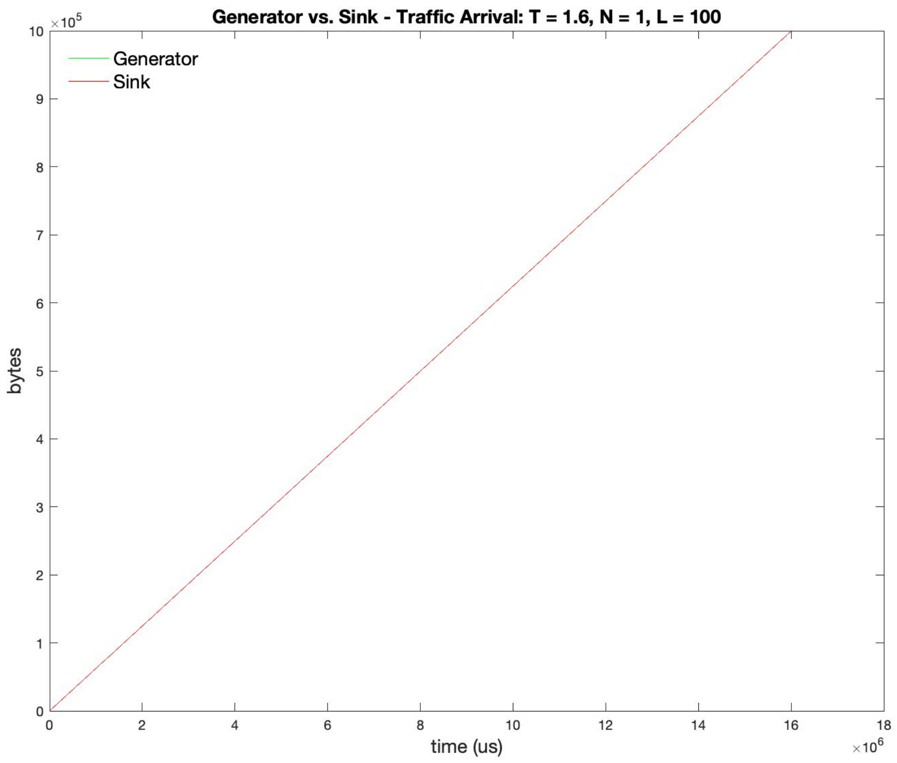


Figure 1: Cumulative Arrivals for Generator and Sink with T = 1.6 ms, N = 1, L = 100 B, $R_{source} = 0.5\ Mbps$

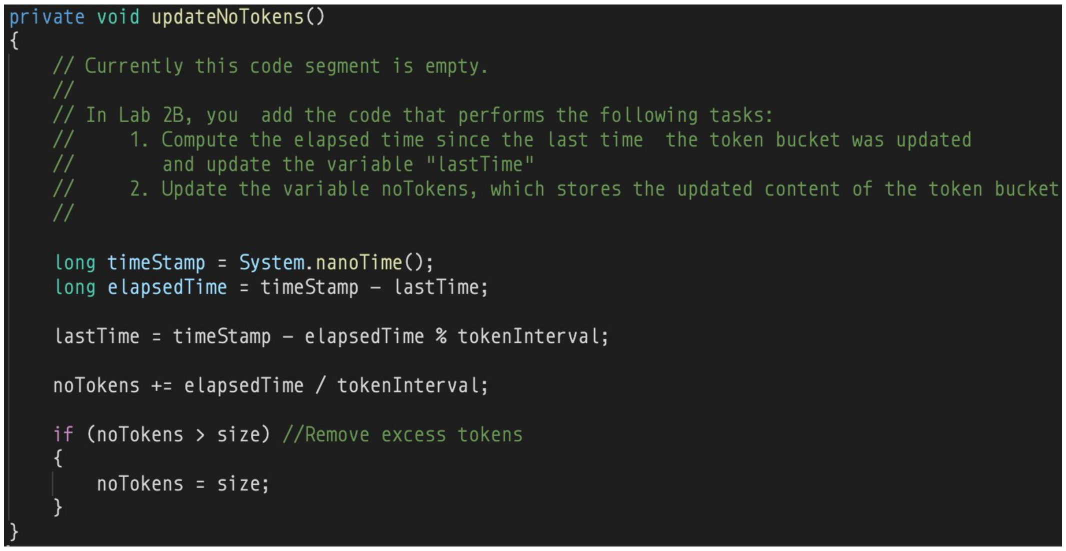
For the above plot, we enforced the generator to transmit 1 packet of 100 B in size after a delay of 1.6 ms, which corresponds to a long-term average traffic rate of 0.5 Mbps. As seen in the above plot, the cumulative arrivals for the traffic generator and sink are visibly identical. That is to say, there is no noticeable delay nor packet loss between the generator and sink and that both arrival curves achieve identical long-term average traffic rates (i.e. 0.5 Mbps).

Exercise 2.1 Complete the implementation of the Token Bucket

Note: We assume that each token = 1 byte in our implementation.

Describe the design of your implementation for the Token Buffer. Include your source code in the lab report.

updateNoTokens()

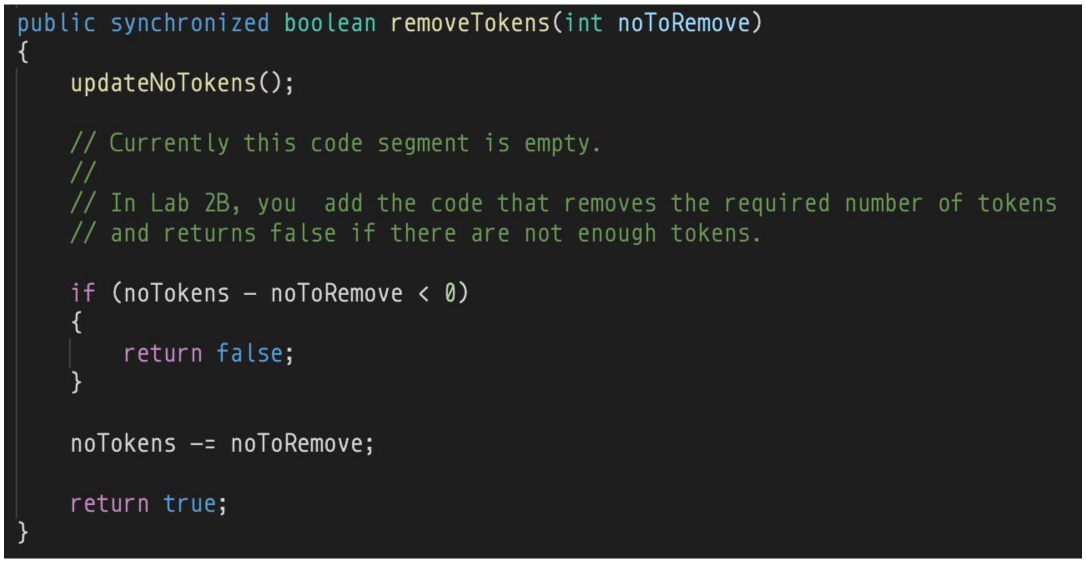


```java
private void updateNoTokens()
{
    // Currently this code segment is empty.
    //
    // In Lab 2B, you  add the code that performs the following tasks:
    //     1. Compute the elapsed time since the last time  the token bucket was updated
    //        and update the variable "lastTime"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

    long timeStamp = System.nanoTime();
    long elapsedTime = timeStamp - lastTime;

    lastTime = timeStamp - elapsedTime % tokenInterval;

    noTokens += elapsedTime / tokenInterval;

    if (noTokens > size) //Remove excess tokens
    {
        noTokens = size;
    }
}
```

Figure 2: updateNoTokens() source code

The updateNoTokens() function is implemented by first taking a timestamp of when this function is called and then determining the elapsed time, which is the difference of said timestamp and the last time this function was called. The 'lastTime' variable is then updated to equal the aforementioned timestamp minus the fragment of the token interval that the elapsed time had taken (i.e. 'lastTime' is always set to the start of a token interval). Finally, we update the total number of tokens by adding to it the amount generated during the elapsed time, if this value exceeds the maximum token bucket size we reset the number of tokens to the max bucket size.

removeTokens()

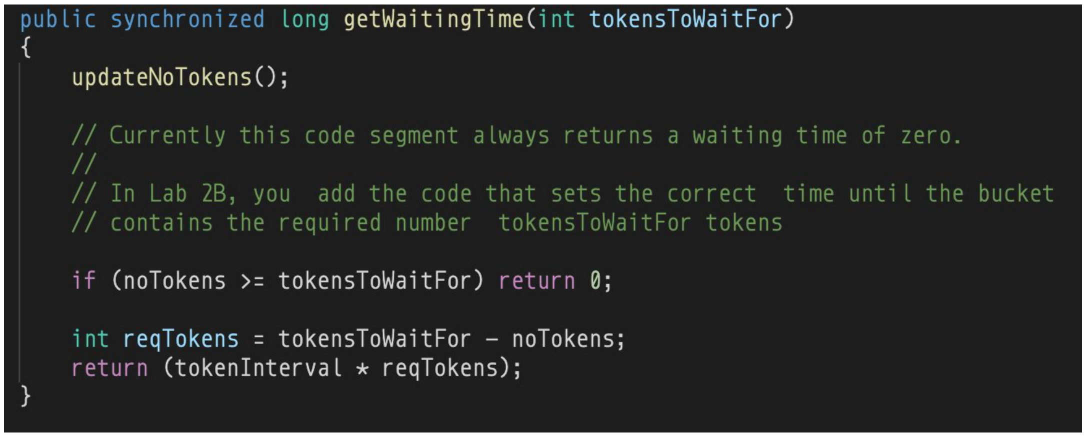


```java
public synchronized boolean removeTokens(int noToRemove)
{
    updateNoTokens();

    // 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 (noTokens - noToRemove < 0)
    {
        return false;
    }

    noTokens -= noToRemove;

    return true;
}
```

Figure 3: removeTokens() source code

The removeTokens() function is implemented by first making a call to updateNoTokens() to get the most recent number of available tokens, thereafter, if the number of tokens to remove exceeds the amount available, we return false, otherwise, we deduct the required number of tokens and return true.

getWaitingTime()



```java
public synchronized long getWaitingTime(int tokensToWaitFor)
{
    updateNoTokens();

    // Currently this code segment always returns a waiting time of zero.
    //
    // In Lab 2B, you  add the code that sets the correct  time until the bucket
    // contains the required number  tokensToWaitFor tokens

    if (noTokens >= tokensToWaitFor) return 0;

    int reqTokens = tokensToWaitFor - noTokens;
    return (tokenInterval * reqTokens);
}
```

Figure 4: getWaitingTime() source code

For the implementation of getWaitingTime(), we first call updateNoTokens() to get the most recent number of available tokens. Thereafter, we check if the number of available tokens is equal to or exceeds the amount desired number of tokens; if this is the case we return a wait of 0 nanoseconds. If the required number of tokens is less than what is available we determine how many more tokens we need and multiply the result with the token interval (i.e. the time it takes to generate 1 token), the result which is the required time to wait (in nanoseconds) is then returned.

Exercise 2.3 Generate plots for the experiments

Note: We assume that each token = 1 byte in our implementation. This is taken into account when setting up the token bucket for use between the generator and sink. For example, if $rate_{TB}$ = 5 Mbps then the equivalent rate in megabytes per second is 0.625 Mbps and so the required bucket rate is set to 625,000 tokens/second.

Experiment #1: $Rate_{source} < rate_{TB}$, N = 1, $size_{TB}$ = 100 B

For this experiment, $Rate_{source}$ = 0.5 Mbps, $rate_{TB}$ = 5 Mbps, T = 1.6 msec and bucket rate was set to 625,000 tokens/second (i.e. each token = 1 byte).

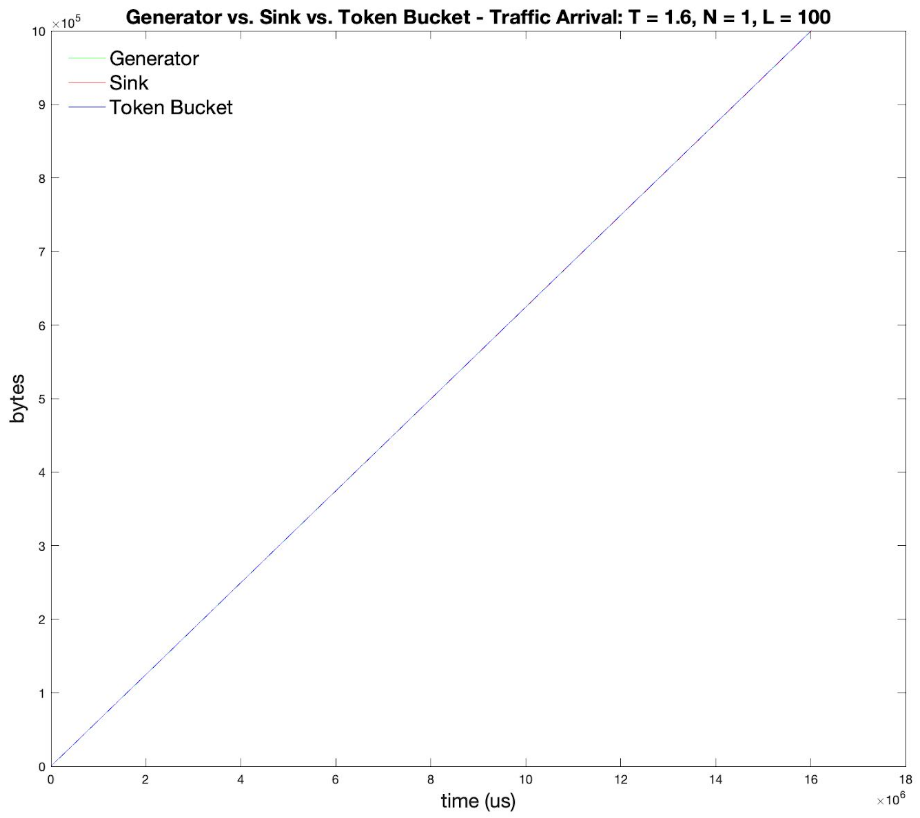


Figure 5: Experiment 1 - Generator vs. Sink vs. Token Bucket Cumulative Arrivals

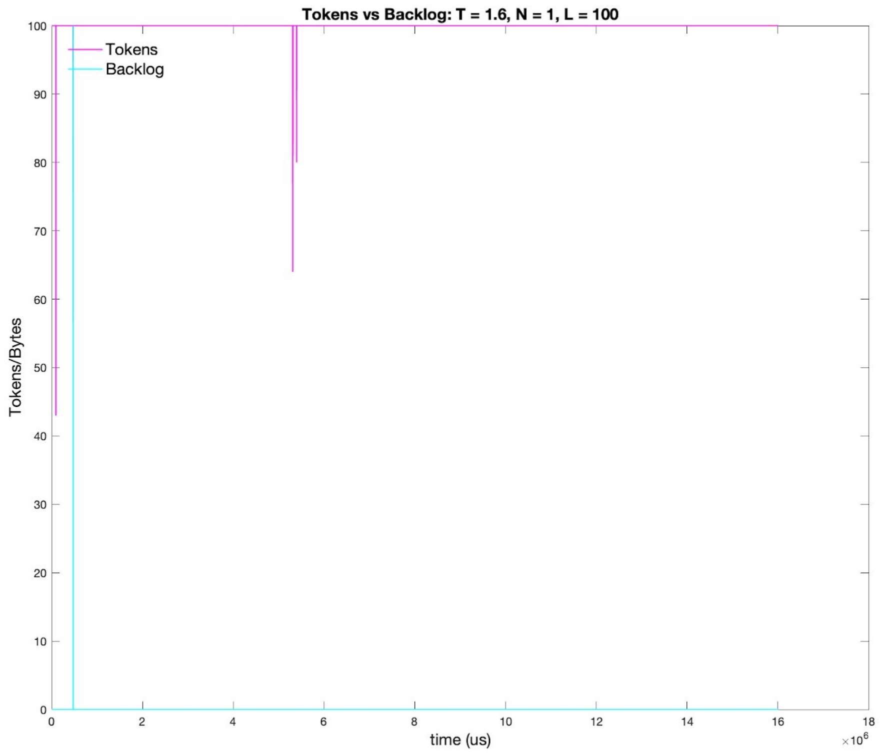


Figure 6: Experiment 1 - Content of Token Bucket vs. Backlog

As seen in figure 5, the cumulative arrival curves of the traffic generator, token bucket, and sink are visibly identical, thus suggesting, all arrival curves converge to the same number of cumulative bytes (i.e. no packet losses) and long term average traffic rate. In figure 6, aside from a few anomalies, we observe that the backlog remains constant at 0 and the number of tokens also remains constant at 100 tokens. These results match what was expected as there are always enough tokens to transmit single equally-spaced 100 B packets immediately.

***Note**: We know that the backlog should remain constant at 0 and that the number of tokens should also remain constant at 100 tokens. The above anomalies we believe are possibly due to the device we tested on having too high of a CPU usage at the time.*

Experiment #2: $Rate_{source} < rate_{TB}$, N = 10, $size_{TB}$ = 500 B

For this experiment, $Rate_{source}$ = 0.5 Mbps, $rate_{TB}$ = 5 Mbps, T = 16 msec and bucket rate was set to 625,000 tokens/second (i.e. each token = 1 byte).

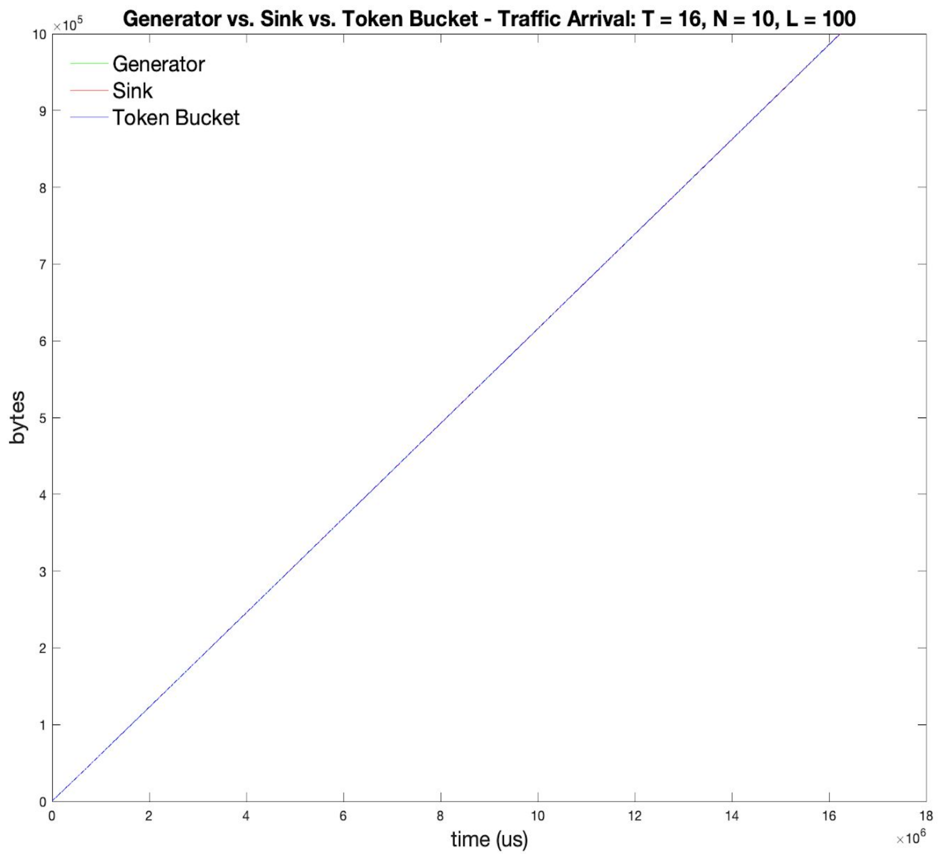


Figure 7: Experiment 2 - Generator vs. Sink vs. Token Bucket Cumulative Arrivals

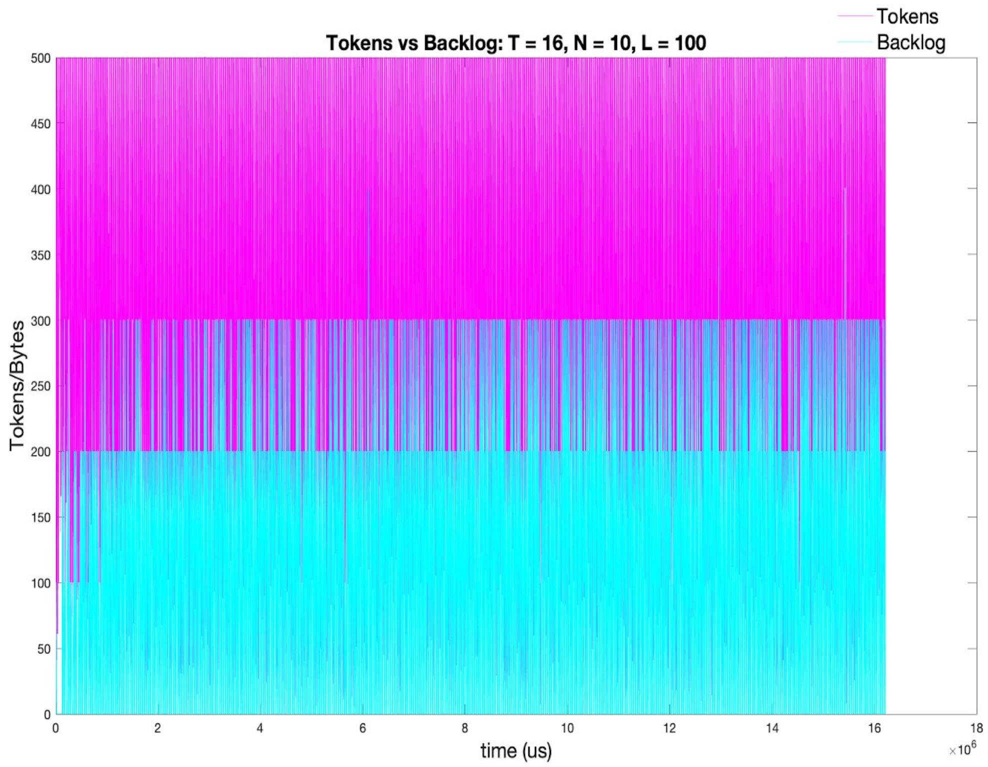


Figure 8: Experiment 2 - Content of Token Bucket vs. Backlog

Observing figure 7, we see that the cumulative arrival functions of the traffic generator, token bucket and sink are all identical as in experiment 1. Based on this result, it is evident that the long term average traffic rate is the same for all arrivals and that there are no packet losses. Now looking at figure 8, we see that the backlog oscillates frequently between ~ 0 and 400 bytes and in a similar manner, the number of available tokens oscillates between ~ 25 and 500 tokens. These results suggest that whenever there is a burst of 10 packets, approximately 4 to 5 packets are immediately sent (as made evident by the number of available tokens periodically reaching nearly 0). To clarify the case for 5 packets being sent, there may be times when the backlog has 4 packets (as seen in figure 8) and a new packet has just arrived, and at this time instance, there may be enough tokens to transmit all packets. Overall these experimental results are consistent with the expected results.

Experiment #3: $Rate_{source} \approx rate_{TB}$, N = 1, $size_{TB}$ = 100 B

For this experiment, $Rate_{source}$ = 0.5 Mbps, $rate_{TB}$ = 0.7 Mbps, T = 1.6 msec, and bucket rate was set to 87,500 tokens/second (i.e. each token = 1 byte).

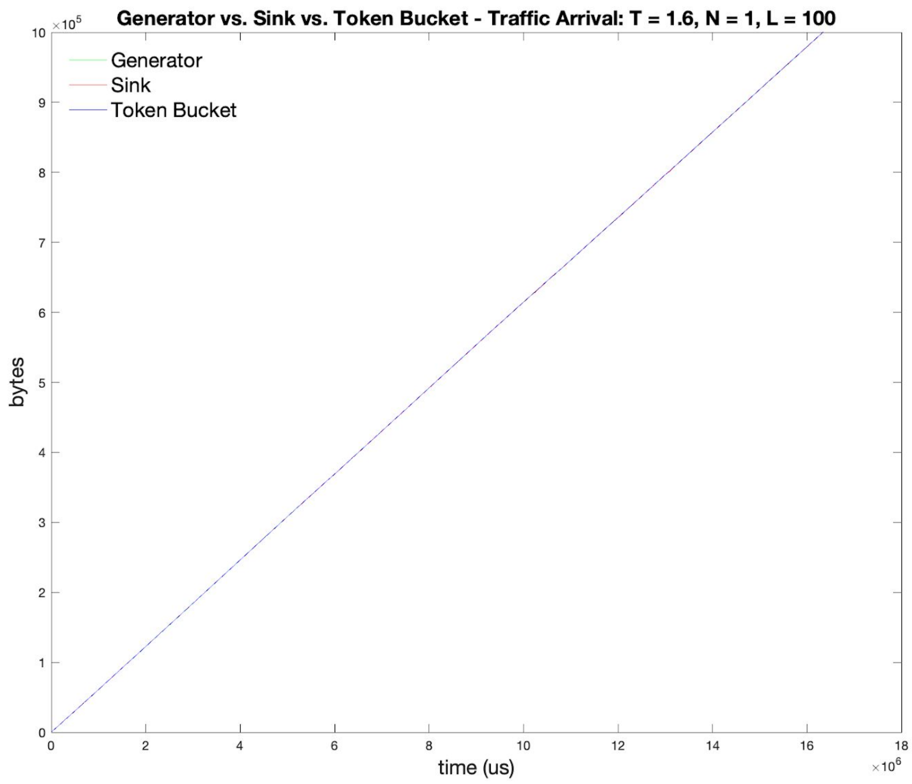


Figure 9: Experiment 3 - Generator vs. Sink vs. Token Bucket Cumulative Arrivals

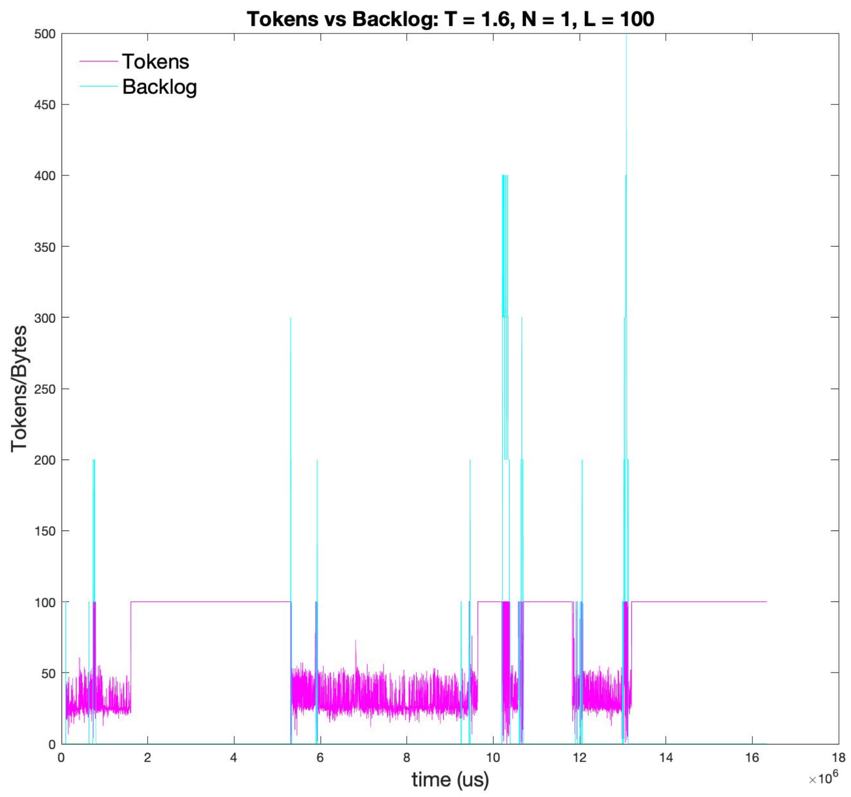


Figure 10: Experiment 3 - Content of Token Bucket vs. Backlog

From figure 9, we make an identical observation as in experiments 1 and 2, we see that the cumulative arrivals for the traffic generator, token bucket and sink are all nearly identical, which suggests nearly identical long term average traffic rates and no packet losses. Observing figure 10, we see that the backlog and the contents of the token bucket oscillate frequently but with the backlog never overflowing (i.e. there are enough tokens to eventually service the entire backlog). These results make sense given that the rates of the generator and long term token bucket rate are very close (i.e. $Rate_{source} \approx rate_{TB}$). Specifically, it is expected that most of the time there are just enough tokens to service some or all of the incoming packets and hence we may see oscillations in the backlog and the available number of tokens (but of course without buffer overflow).

Exercise 2.4 Maximum rate of Token Bucket

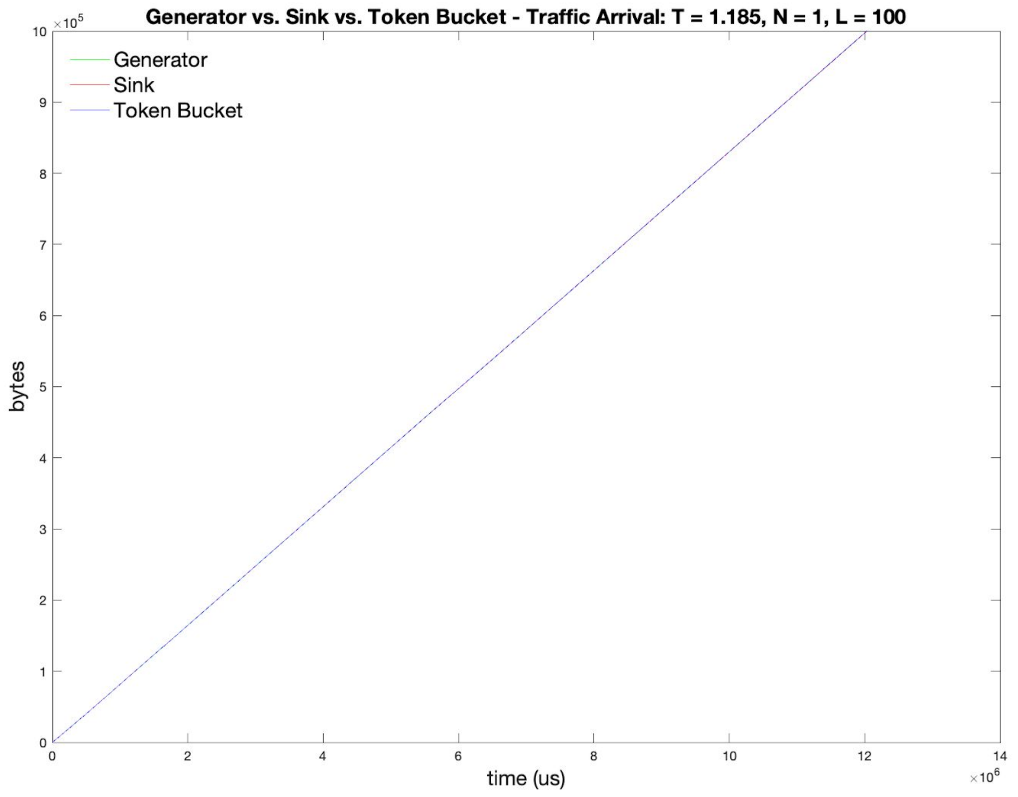


Figure 12: Generator vs. Sink vs. Token Bucket Cumulative Arrivals for max rate

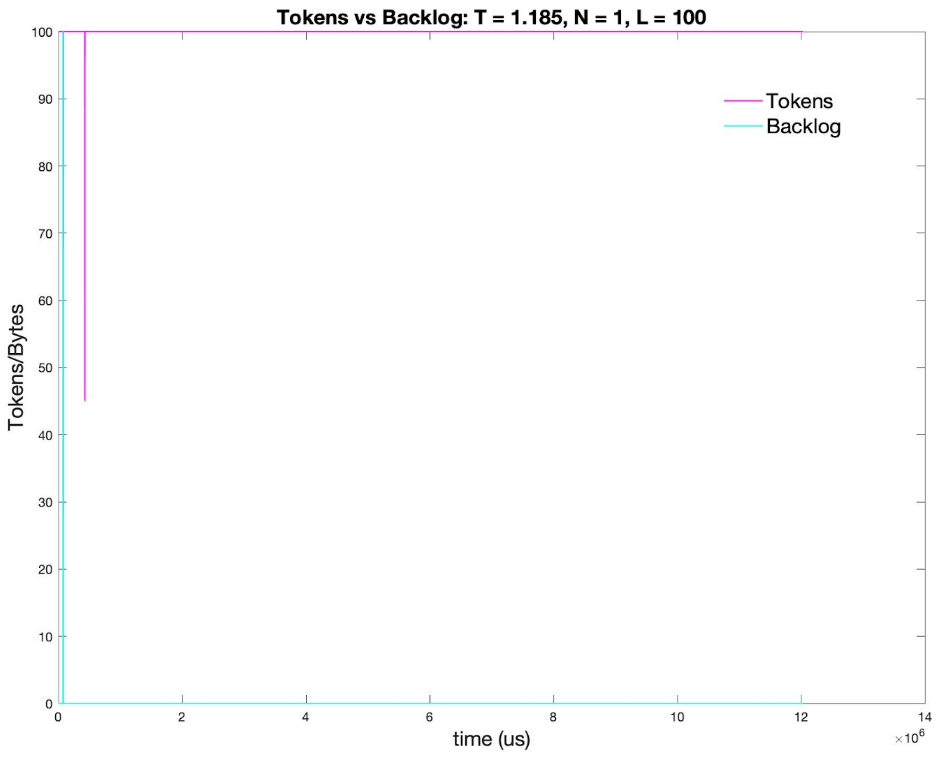


Figure 13: Content of Token Bucket vs. Backlog for max rate

To determine the maximum supported arrival rate we began by assigning $rate_{TB}$ = 5 Mbps, $size_{TB}$ = 100 B, bucket rate = 625,000 tokens/second, N = 1 and L = 100 B. From this point onwards, we tested values for $Rate_{source}$ beginning from 0.5 Mbps and onwards. We found via trial and error that the maximum traffic arrival rate supported by the token bucket for the stated parameters is 0.8 Mbps. In general, we found that going any higher than 0.8 Mbps led to the backlog overflowing and any less eventually led to an increase in the number of available tokens such that backlog remained 0 for any time interval. Figure 12 shows the cumulative arrivals for the generator, token bucket and sink from which we can see that they are all nearly identical and converge to the same cumulative bytes (i.e. no packet loss). As for figure 13, we observe that the backlog remains approximately at 0 during the entire trace and that there are always enough tokens available to service it.

Exercise 3.1 (Long-term) Bandwidth is cheap

<u>Poisson 3 Trace Parameters</u>

$size_{TB}$ = 1500 bytes	Buffer Capacity = 4000 bytes	$rate_{TB}$ = 1.20 Mbps or 150,000 bytes/sec
Max Delay = 13.413 msec		

For the Poisson 3 trace, we observed in lab 1 that the largest packet length is 1469 bytes, the mean bit rate is 1.006 Mbps and the measured peak rate is 7.39 Mbps. Based on these results we set $size_{TB}$ to 1500 bytes to accommodate the largest packet in the trace along with additional variable-length packets that might arrive in quick succession. The buffer capacity was set to 4000 bytes after observing that the maximum backlog often grew close to or above ~3500 bytes when testing with the stated $size_{TB}$ and various $rate_{TB}$ values. As for $rate_{TB}$, we tried setting it to the mean bit rate of 1.006 Mbps, however, this led to severe packet losses. As bandwidth is cheap, we incrementally increased $rate_{TB}$ and after some trial and error, observed that a value of 1.20 Mbps combined with the above values for $size_{TB}$ and buffer capacity allowed us to achieve the smoothest arrival curves for the sink and token bucket along with minimal delay (as seen in figure 14).

As for the content of the token bucket and backlog, figure 15 illustrates the number of tokens and backlog constantly oscillating with the backlog never overflowing and there eventually being enough tokens to service the backlog.

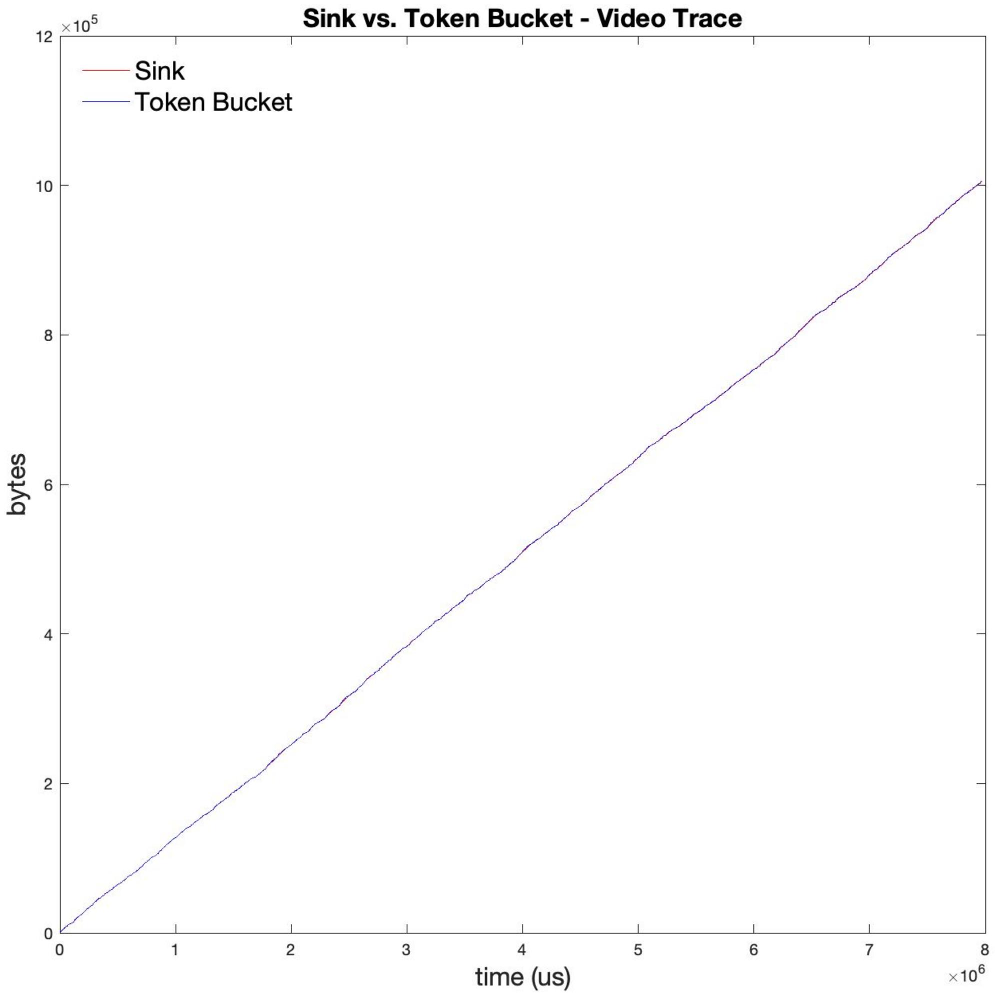


Figure 14: Poisson 3 Trace - Sink vs. Token Bucket Cumulative Arrivals

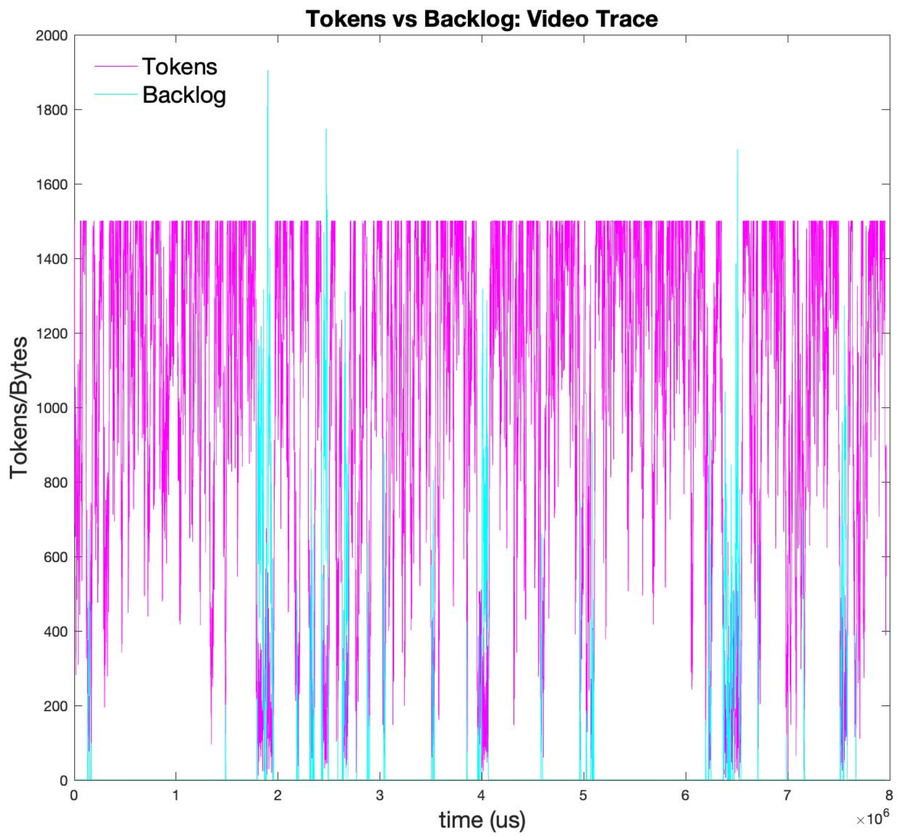


Figure 15: Poisson 3 Trace - Content of Token Bucket vs. Backlog

Video Trace Parameters

Note: For video trace, we only consider 1000 frames due to computational limitations

$size_{TB}$ = 60,000 bytes	Buffer Capacity = 660,000 bytes	$rate_{TB}$ = 40 Mbps or 5,000,000 bytes/second
Max Delay = 52.373 msec		

For the video trace, we know from lab 1 that the mean bit rate is 14.268 Mbps, the peak rate is 157.88 Mbps and the largest frame size is 657,824 bytes. Accordingly, we selected $size_{TB}$ through trial and error to be 60,000 bytes, where this value accommodates the burst of packets associated with large frame sizes (note frame sizes exceeding 1024 bytes will arrive as a burst of packets of max length 1024 bytes each). We also set the buffer capacity to be 660,000 bytes to handle the case of the largest frame arriving. As for $rate_{TB}$, we initially tried using the mean bit rate but this led to many packets being dropped, eventually, we found that a bit rate of 40 Mbps (combined with the above values for $size_{TB}$ and buffer capacity) yielded the smoothest arrival curves for the sink and token bucket with minimal delay (see figure 16).

As for the content of the token bucket and backlog for the above parameters we see that in figure 17 the number of available tokens and the backlog constantly oscillate, but with the property that the backlog never overflows as there is always sufficient tokens to service it.

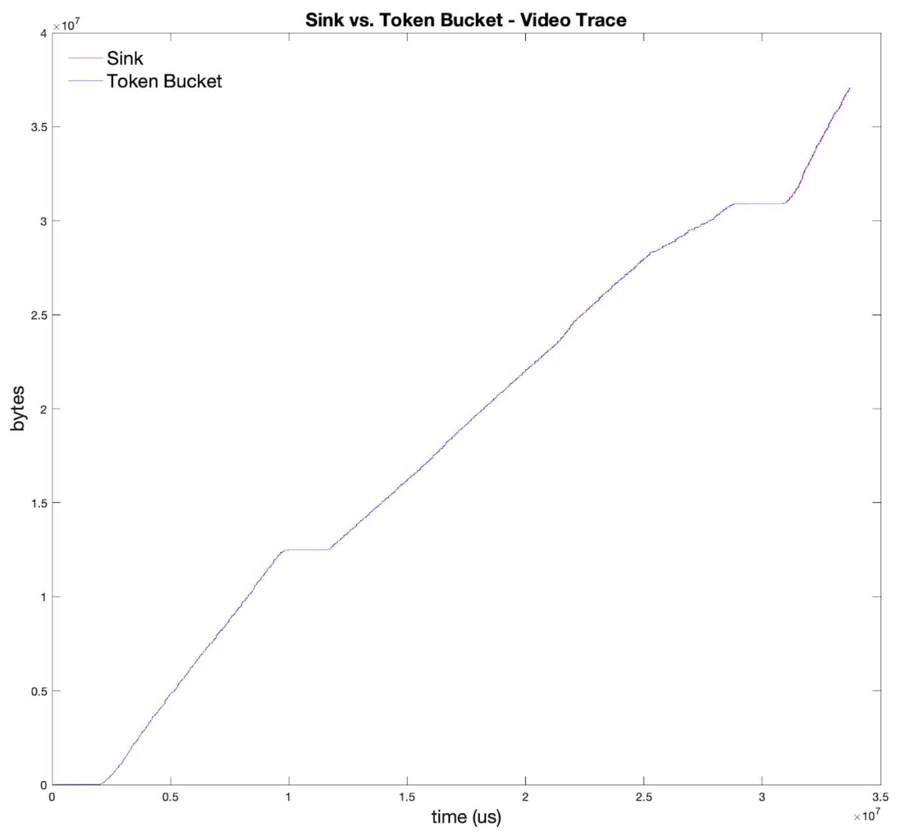


Figure 16: Video Trace - Sink vs. Token Bucket Cumulative Arrivals

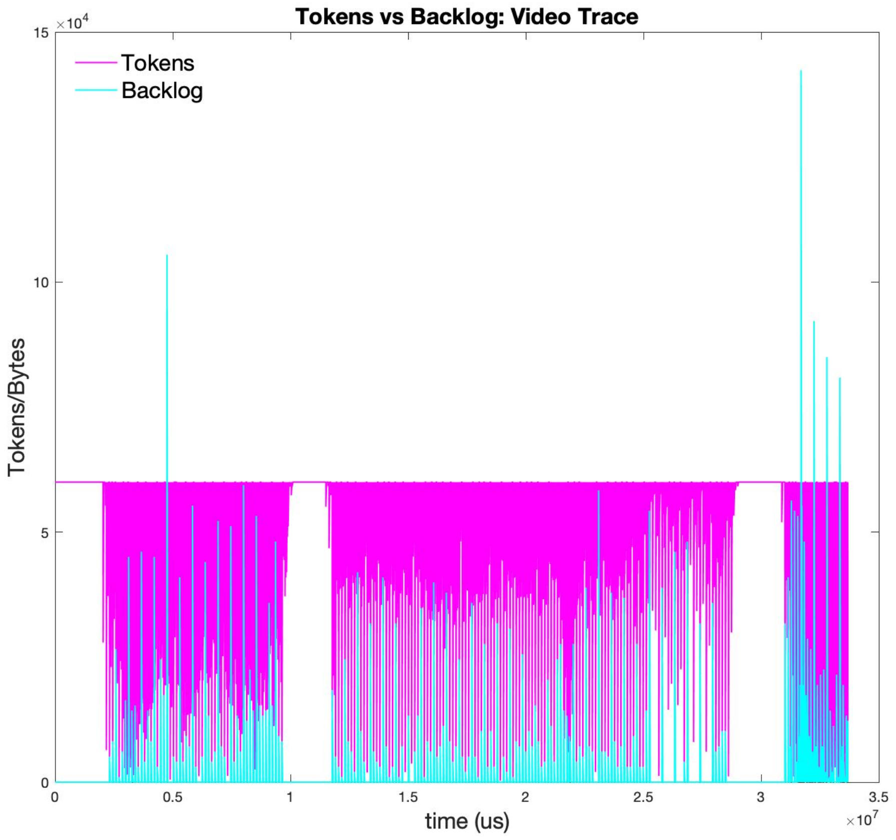


Figure 17: Video Trace - Content of Token Bucket vs. Backlog

Ethernet Trace Parameters

$size_{TB}$ = 1520 bytes	Buffer Capacity = 45,000 bytes	$rate_{TB}$ = 3.5 Mbps or 437,500 bytes/sec
Max Delay = 66.429 msec		

For the Ethernet trace (i.e. 'BC-pAug89-small.TL'), we found that the mean bit rate was 1.47 Mbps, the peak rate was 16.5 Mbps and the largest frame size was 1518 bytes. Based on these results we set $size_{TB}$ to 1520 bytes to accommodate the largest packet in the trace. For the buffer capacity, we found that through trial and error a value of 45,000 bytes is sufficient for this trace (note that buffer capacity was estimated by setting $size_{TB}$ to its stated value and trying various rates for $rate_{TB}$). As for $rate_{TB}$, we tried setting it to the mean bit rate of 1.47 Mbps, however, this led to severe packet loss. Since bandwidth is cheap, we incrementally increased $rate_{TB}$ and found that a value of 3.5 Mbps combined with the stated $size_{TB}$ and buffer capacity led to the smoothest arrival curves for the sink and token bucket along with minimal delay (as seen in figure 18).

For the content of the token bucket and backlog, figure 19 depicts the number of tokens and backlog constantly oscillating with the backlog never overflowing and there always being enough tokens to service the backlog.

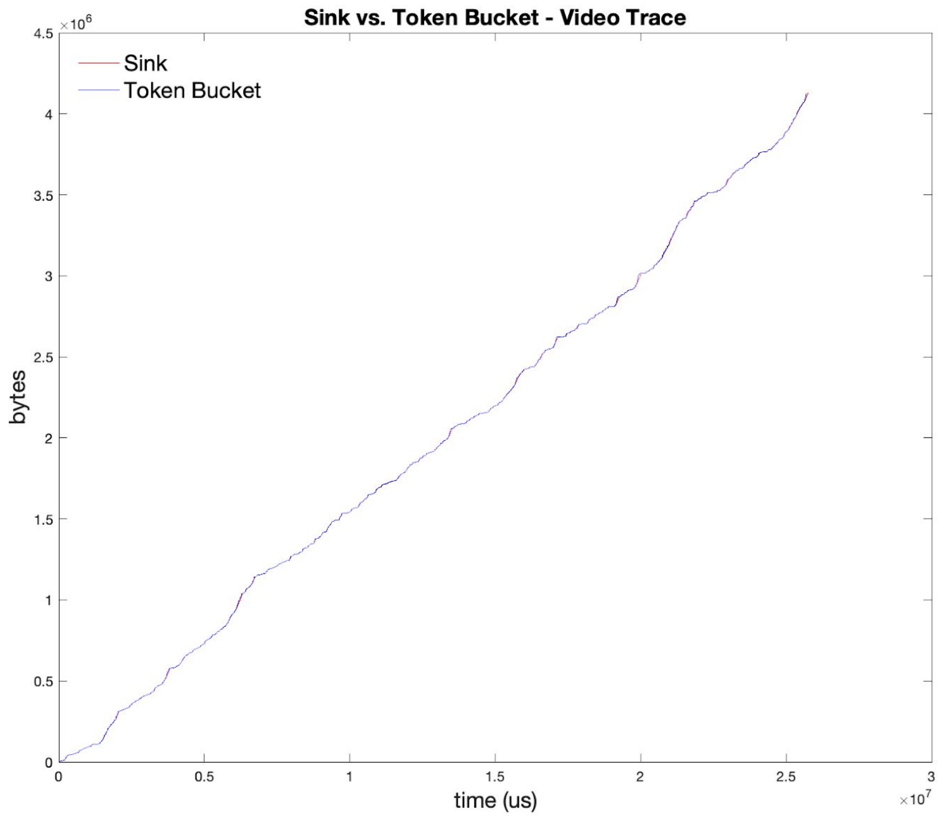


Figure 18: Ethernet Trace - Sink vs. Token Bucket Cumulative Arrivals

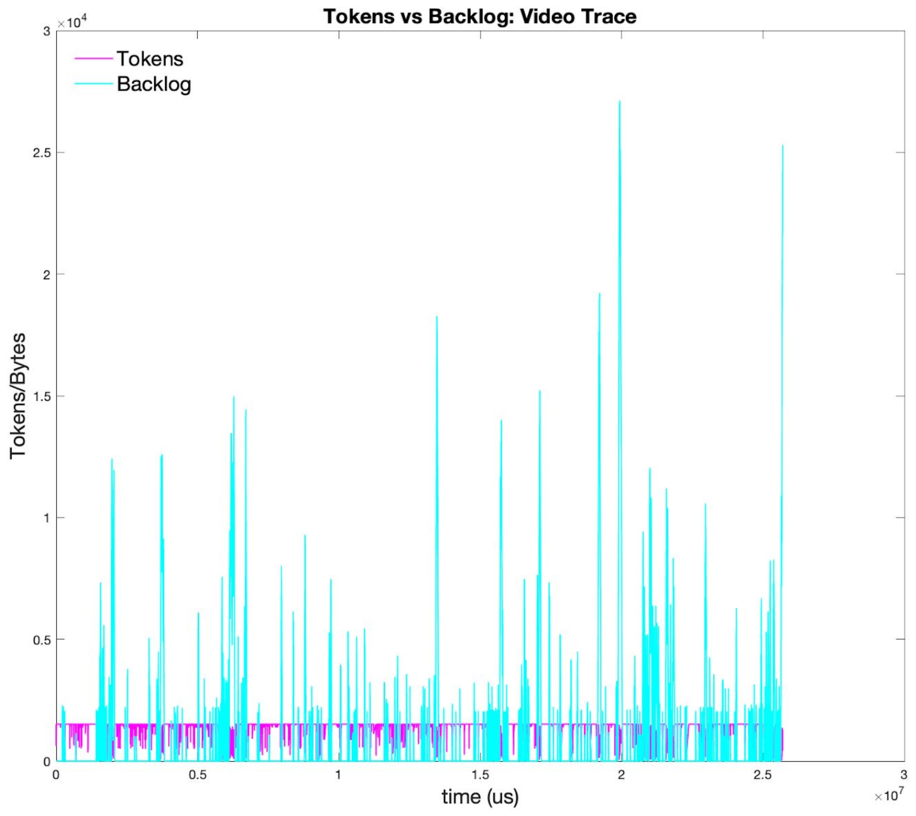


Figure 19: Ethernet Trace - Content of Token Bucket vs. Backlog

Exercise 3.2 (Long-term) Bandwidth is expensive

Poisson 3 Trace Parameters

$size_{TB}$ = 7000 bytes	Buffer Capacity = 15,000 bytes	$rate_{TB}$ = 1.006 Mbps
Max Delay = 12.251 ms		

For the Poisson 3 trace, we set $rate_{TB}$ to the mean bit rate of 1.006 Mbps that we computed in lab 1. Moving forward, we found through trial and error a Buffer Capacity of 15,000 bytes and $size_{TB}$ = 7000 bytes achieved the smoothest cumulative arrival curves for the sink and token bucket along with minimal delay (as seen in figure 20).

As for the content of the token bucket and backlog, we see from figure 21 there are always enough tokens to cover the contents of the backlog and as such the backlog never overflows.

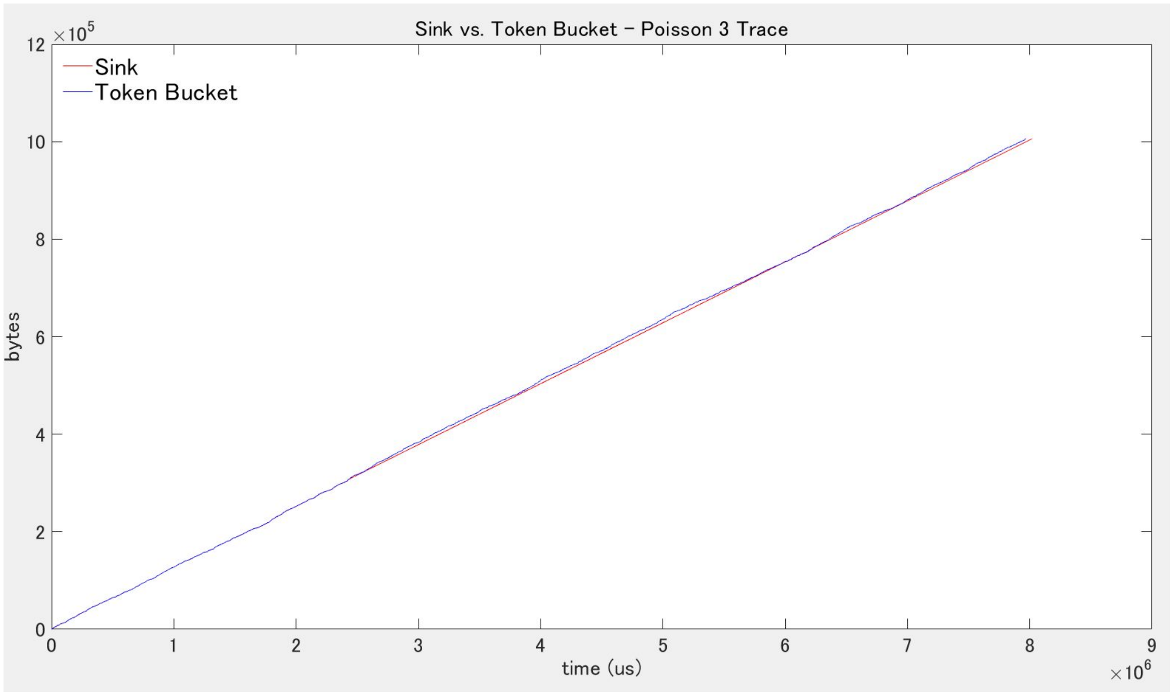


Figure 20: Poisson 3 Trace - Sink vs. Token Bucket Cumulative Arrivals

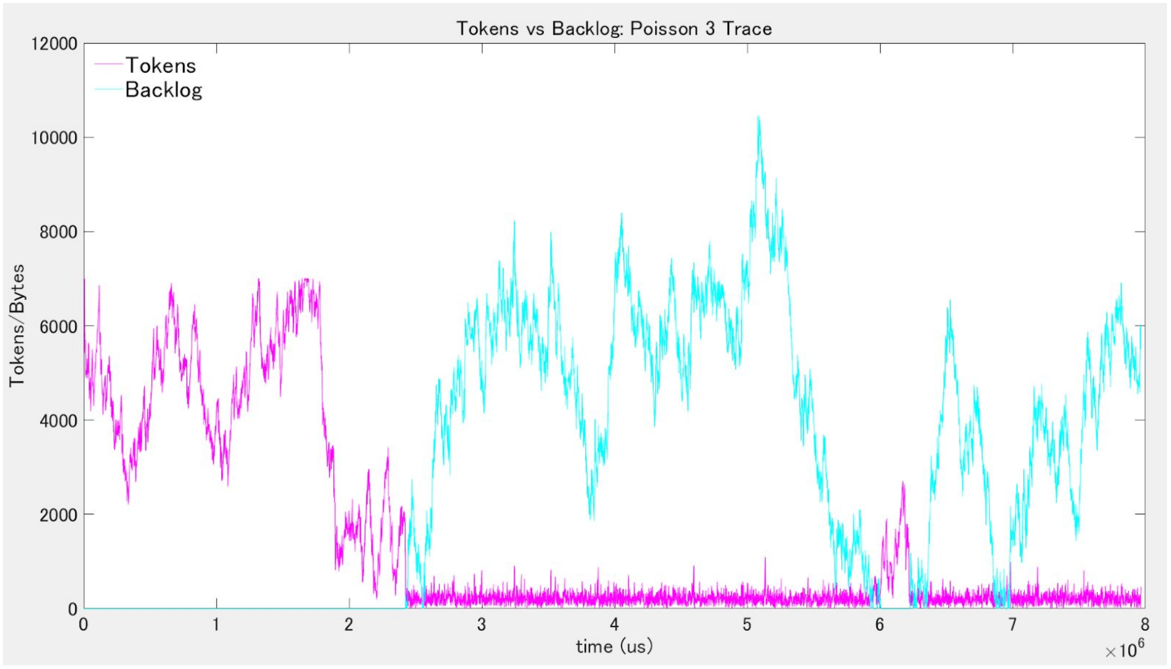


Figure 21: Poisson 3 Trace - Content of Token Bucket vs. Backlog

Video Trace Parameters

Note: For video trace, we only consider 1000 frames due to computational limitations

$size_{TB}$ = 1,500,000 bytes	Buffer Capacity = 660,000 bytes	$rate_{TB}$ = 14.3 Mbps or 1,787,500 bytes/second
Max Delay = 0.889467 msec		

For this experiment, we decided to set the $rate_{TB}$ to be the mean bit rate measured in lab 1 of 14.3 Mbps and we kept the same buffer capacity from section 3.1 for the same reasons. From this point onward, we experimented with various values for $size_{TB}$ and found that a value of 1,500,000 bytes led to the smoothest cumulative arrival curves for the sink and token bucket along with minimal delay (as seen in figure 22).

Analyzing figure 23, we see that the backlog and number of available tokens continuously vary, though the backlog never overflows as there are always enough tokens to service it.

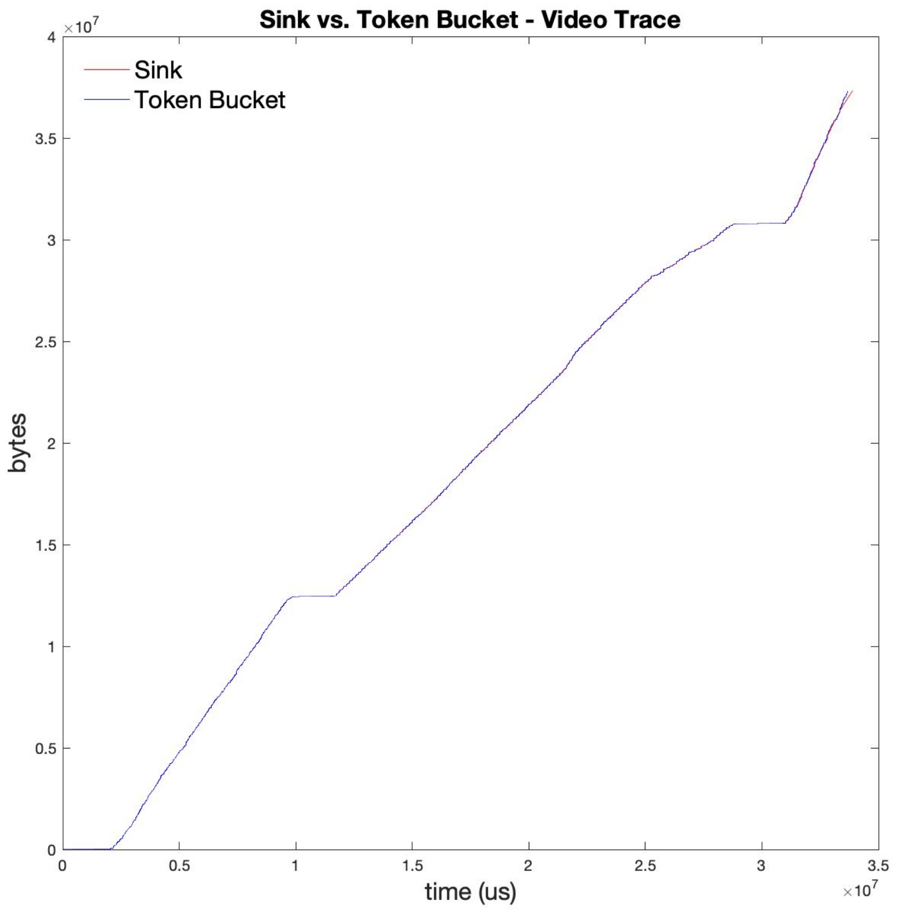


Figure 22: Video Trace - Sink vs. Token Bucket Cumulative Arrivals

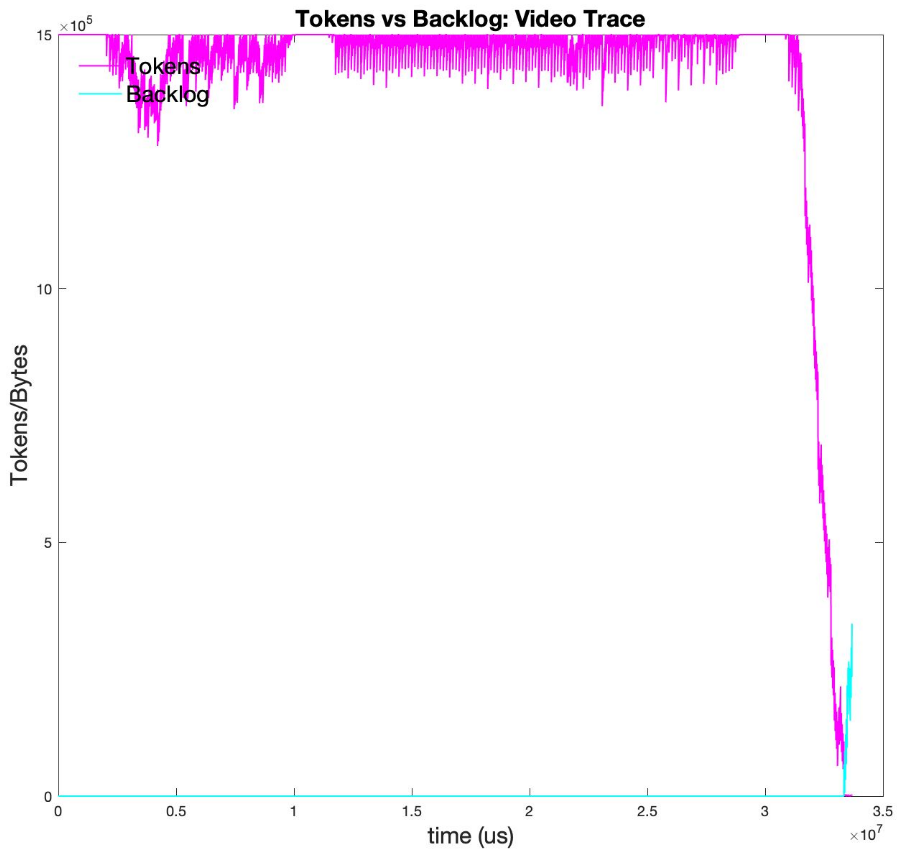


Figure 23: Video Trace - Content of Token Bucket vs. Backlog

<u>Ethernet Trace Parameters</u>

$size_{TB}$ = 250,000 bytes	Buffer Capacity = 45,000 bytes	$rate_{TB}$ = 1.47 Mbps or 183,750 bytes/second
Max Delay = 0.780753 msec		

For this experiment, we decided to set the $rate_{TB}$ to be the mean bit rate measured for the trace of 1.47 Mbps and we kept the same buffer capacity from section 3.1 for the same reasons. Thereafter, we experimented with various values for $size_{TB}$ and found that a value of 250,000 bytes led to the smoothest cumulative arrival curves for the sink and token bucket along with minimal delay (as seen in figure 24).

Analyzing figure 25, we notice that the backlog remains nearly constant at 0, but towards the end, there is a large spike for which there are just enough tokens to service it. Furthermore, we see that the number of available tokens throughout the plot in figure

25 continuously oscillates, which is expected from the Bellcore trace consisting of variable size packets with variably spaced arrival times.

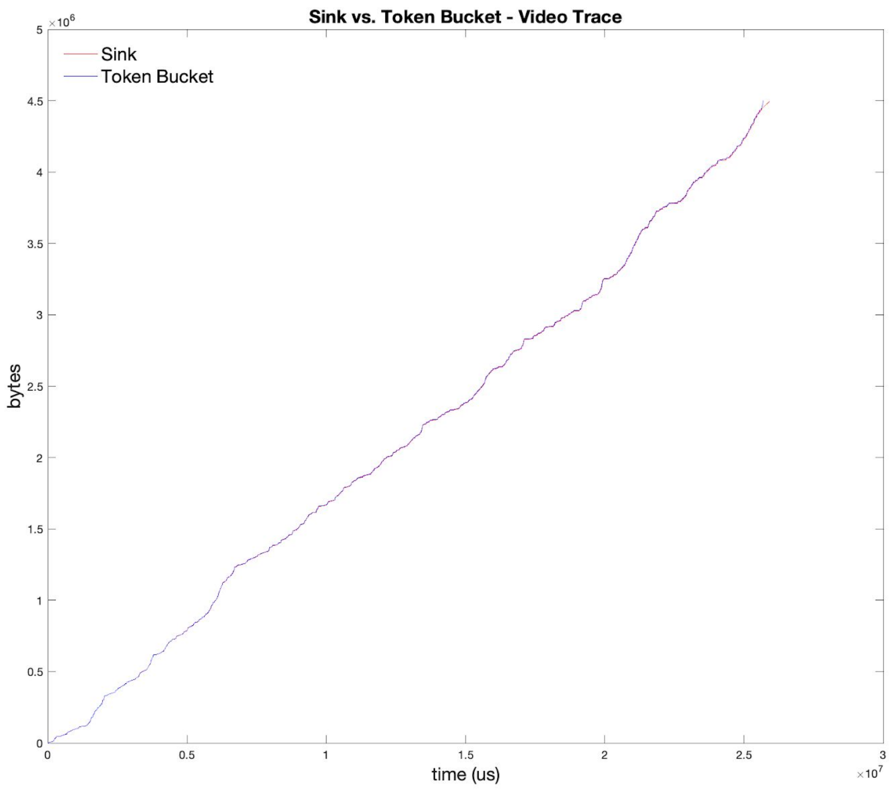


Figure 24: Ethernet Trace - Sink vs. Token Bucket Cumulative Arrivals

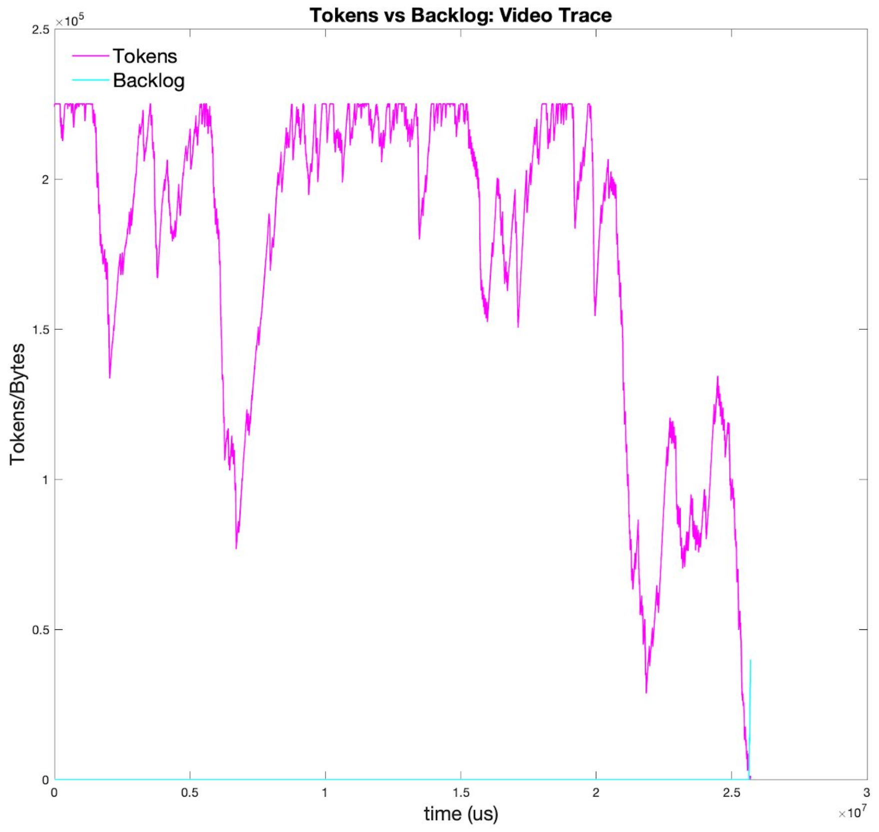


Figure 25: Ethernet Trace - Content of Token Bucket vs. Backlog

Compare the results for the cheap bandwidth and expensive bandwidth scenarios

Poisson 3 Trace

Cheap bandwidth: $size_{TB}$ = 1500 bytes, Buffer Capacity = 4000 bytes, $rate_{TB}$ = 1.20 Mbps, Max Delay = 13.413 msec

Expensive Bandwidth: $size_{TB}$ = 7000 bytes, Buffer Capacity = 15,000 bytes, $rate_{TB}$ = 1.006 Mbps, Max Delay = 12.251 msec

In both the cheap and expensive bandwidth cases for the Poisson trace we were able to achieve similar max delays, however, in the cheap bandwidth case, the traffic appears smoother (see figure 14) than in the expensive bandwidth case (see figure 20).

Video Trace

Cheap bandwidth: $size_{TB}$ = 60,000 bytes, Buffer Capacity = 660,000 bytes, $rate_{TB}$ = 40 Mbps, Max Delay = 66.429 msec

Expensive Bandwidth: $size_{TB}$ = 1,500,000 bytes, Buffer Capacity = 660,000 bytes, $rate_{TB}$ = 14.3 Mbps, Max Delay = 0.889467 msec

For the video trace, we obtained a markedly lower max delay in the expensive bandwidth case than in the cheap bandwidth scenario. Additionally, we see that the arrival curves for the cheap bandwidth case appear much smoother (see figure 16) than in the expensive bandwidth case (see figure 22).

Ethernet Trace

Cheap bandwidth: $size_{TB}$ = 1520 bytes, Buffer Capacity = 45,000 bytes, $rate_{TB}$ = 3.5 Mbps, Max Delay = 66.429 msec

Expensive Bandwidth: $size_{TB}$ = 250,000 bytes, Buffer Capacity = 45,000 bytes, $rate_{TB}$ = 1.47 Mbps, Max Delay = 0.780753 msec

As for the ethernet trace, we observed (similar to the video trace) that in the expensive bandwidth scenario we achieved a much lower max delay than in the cheap bandwidth case, however, both in the cheap and expensive bandwidth cases the arrival curves appear equally smooth (see figure 18 and figure 24).

Compare the results for the three traffic types (i.e. which type of traffic is most demanding in terms of bandwidth or memory requirements? Which type of traffic benefits the most from traffic shaping?)

From the three traces, we analyzed we see that the video trace is the most demanding in both bandwidth and memory requirements (i.e. it requires the highest $rate_{TB}$ and $size_{TB}$ in either cheap or bandwidth scenarios compared to other traces – see above comparisons for quantitative results). This is due to the fact that in comparison, the frame sizes in the video trace can be several magnitudes larger than any packet size in the Poisson 3 or Ethernet trace (i.e. Video – Largest Frame: 657,824 bytes vs. Poisson – Largest Packet: 1469 bytes vs. Ethernet – Largest Packet: 1518 bytes). Furthermore, as large frame sizes are fragmented into several smaller sized packets (i.e. each packet will be of length at most 1024 bytes), this suggests that the video trace also produces more aggressive bursty behavior than any of the other traces. To support this claim, when comparing the Token Bucket vs Backlog plots of the three traces, we observe that on average there are much larger backlogs that develop in the video trace than in any other trace (for Poisson see figures 15 or 21; Video see figures 17 or 23; Ethernet see figures 19 or 25).

Finally, when evaluating which trace benefits the most from traffic shaping we expect that it is the ethernet trace. Although the video trace has the most variance in terms of frame size (see table 1), however, frame arrivals follow an overarching pattern as defined by video's GOP, and when performing traffic smoothing (see sections 3.1 and 3.2), the effects mostly improved the delay between the sink and token bucket than any noticeable smoothing of their arrival curves. As for the Poisson 3 trace, although packets may arrive at highly variable times there is much less variance for packet sizes when compared with the other traces (see table 1), and traffic smoothing did not appear to have much of an effect other than reducing the delay between the source and sink arrival curves (see figure 14 and 20). As for the ethernet trace, it has more variability with respect to its packet size than the Poisson trace (see table 1), combining this with the fact that its packet arrival times (although less variable than the Poisson trace) is still highly variable nonetheless, explains why its arrival curves for the sink and token bucket are more bursty than that of the Poisson trace (see figure 14 for Poisson and figure 18 for ethernet trace). Accordingly, the ethernet trace with its more bursty arrival curves when compared to any other trace stands most to benefit from traffic shaping.

Table 1: Variance of packet size and arrival times of various traces

	Poisson 3	Video	Ethernet
Variance of Packet Size	9.9628e+03	4.5336e+09	2.2363e+05
Variance of Arrival Time	8.2456e+14	Cannot use Frame Display Time	5.3998e+03