# ECE466 LAB4

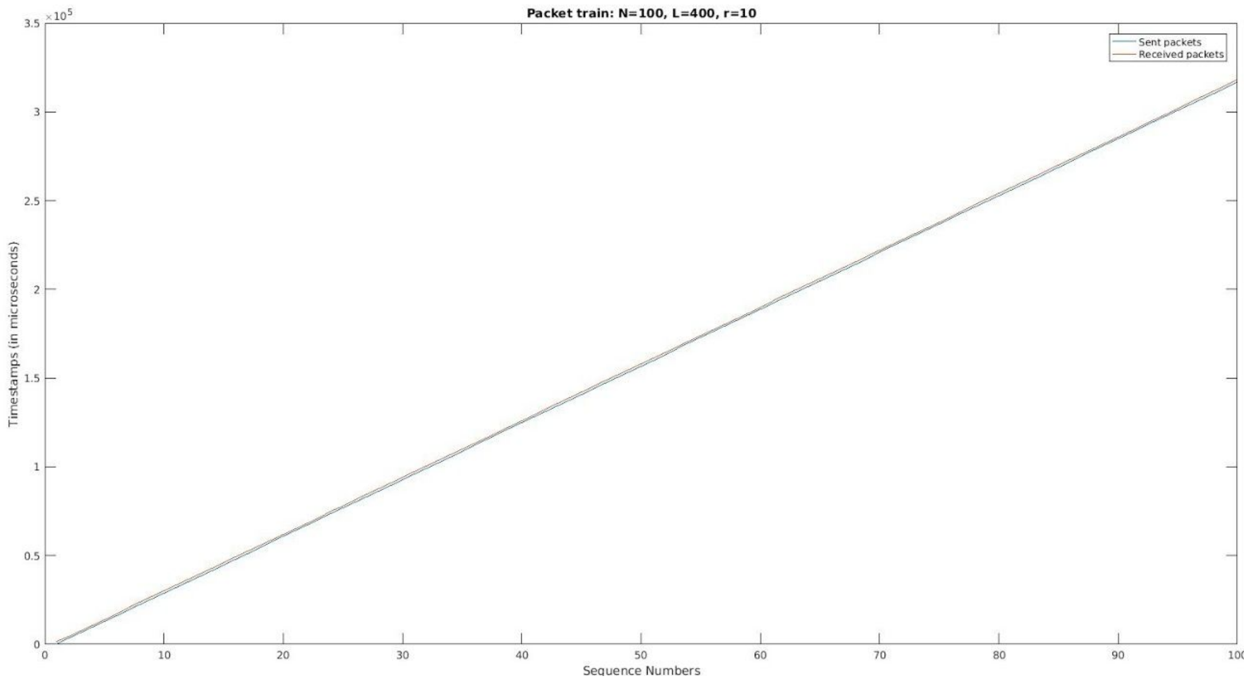
**Chengchen Yang 1000285880**
**Yu Li 1000173554**

# Part 1 Generating and Time-stamping Packet Trains

### Exercise 1.5 Evaluation

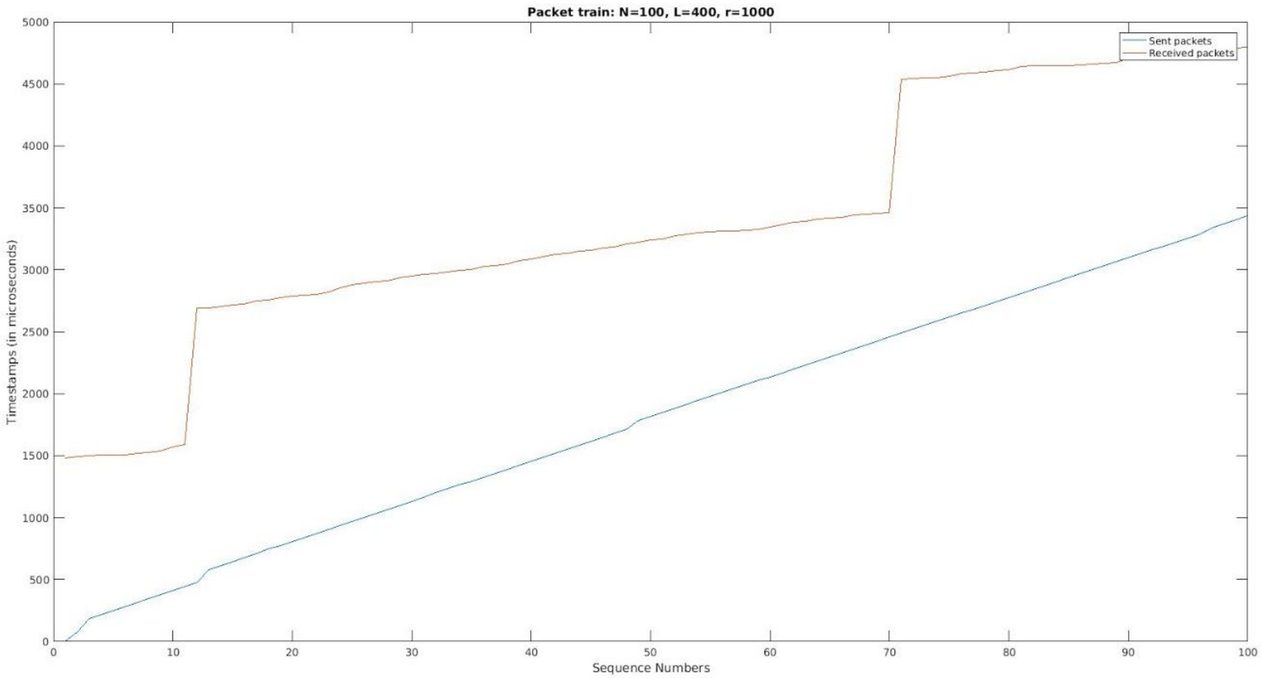
Provide the plots and a discussions of the graphs generated in Exercise 1.5. Include the source code for the Estimator, that was used for Exercise 1.5.

1. Packet train: N=100, L=400, r=10;



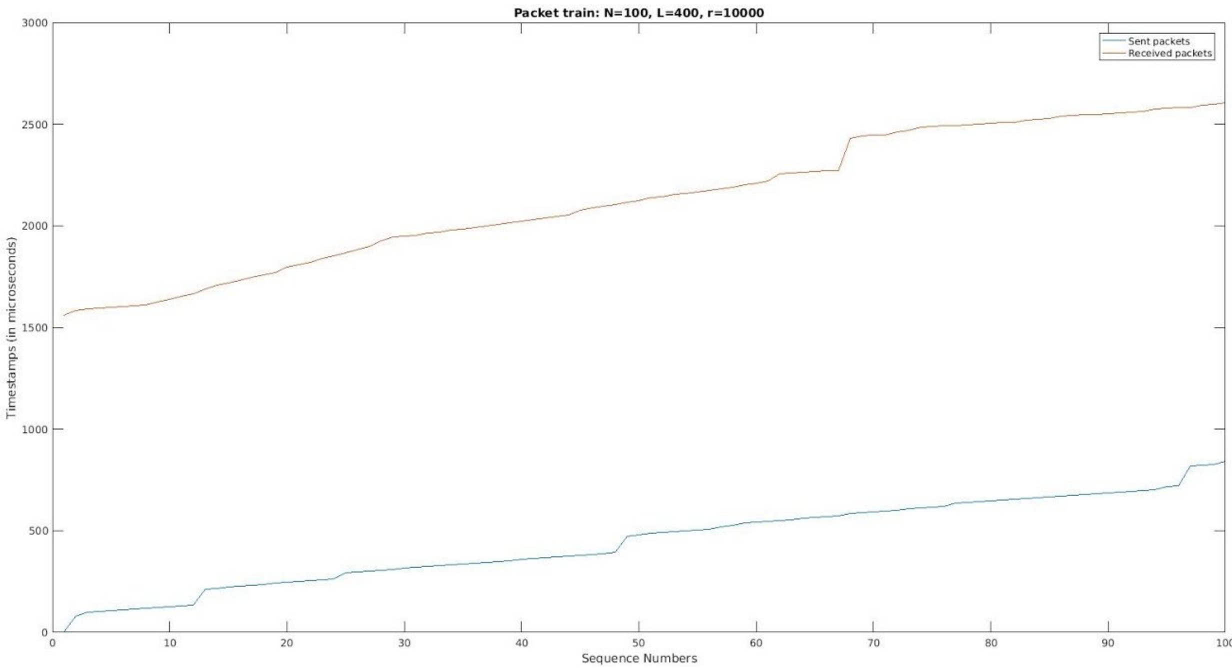

From the graph above, the two curves represent the sent packets and received packets. As can be seen, the two curves are almost the same and very close to each other, which means that the BlackBox is able to receive and transmit the packets efficiently with TrafficGenerator and TrafficSink. There is also a small delay showing up in the figure as well.

2. Packet train: N=100, L=400, r=1000;




In the graph above, we can see that the receive timestamp is above the send timestamp. It is because that the increasing of the packet train rate. It causes the backlog in the BlackBox. However, since the rate at BlackBox is pretty fast, so when it passes the delay, it sends out the packet at a faster rate than the receive rate, which leads to the jump in the curve.

3. Packet train: N=100, L=400, r=10000;




With the rate r increased to 10 times of the previous one, we still can see a similar pattern as the previous in the graph above. But as the rate increases, the speed of send packet has increased and we could see that the send time is pretty low. Therefore, the packets can only be sent after a certain delay and backlogged at the BlackBox for a while.

Source code for the Estimator: basically we created a concurrentHashMap in order to store the timestamps for the packets. And created two public functions to access the map. Meanwhile, in the main function of the Estimator, we first read the user inputs from arguments and then created two thread for TrafficGenerator and TrafficSink in order to send and receive packets to and from BlackBox.

```java
public class TrafficEstimator {

    private static ConcurrentHashMap<Integer,Timestamp> map = new ConcurrentHashMap<Integer, Timestamp>(

    public static void mapPut(Integer key, Timestamp value){
        map.put(key, value);
    }

    public static Timestamp mapGet(Integer key){
        return map.get(key);
    }

    public static void main(String[] args) throws IOException {
        // BlackBox is started at localhost with default port 4444
        // also initialize the sink port for the TrafficSink to receive packets later
        String hostname = args[0];
        InetAddress addr = InetAddress.getByName(hostname);
        int port = Integer.parseInt(args[1]);
        int sinkPort = 4445;

        // read the user input for given parameters N L r
        int N = Integer.parseInt(args[2]);
        int L = Integer.parseInt(args[3]);
        int r = Integer.parseInt(args[4]);

        // start the traffic generator to send packets to blackbox
        new TrafficGenerator(addr, port, N, L, r, sinkPort).start();

        // start the traffic sink to receive packets from blackbox and compute timestamps
        new TrafficSink(sinkPort,N, L, r).start();
    }
}
```

# Part 2 Bandwidth Estimation of Black Boxes

### Exercise 2.1 Implement and test the probing methodology

We started with the L = 400 Bytes, N = 100. But we found that because the L is too small, so that all packets are transmitted before BlackBox to send the packet, so we increase the N and L to N = 1000 and L = 100,000 Bytes.
And we found that if we set the R = 100 Mbps and set the r = 100,000 Kbps, it could cause some internal exception in the nanosecond timeout value out of range. Therefore, we changed the R = 1 Mbps instead of 100 Mbps in order to use the probing probing methodology to find the service curve. And we could set the r to a value that is slightly higher than the R’s value.
In order to select the rate, I use the probing methodology to determine the service curve. Basically, I start with a rate that close to the rate R and start to compare the estimated service curve to the ideal service curve and adjust the rate r correspondingly.
We use the below code inside the TrafficSink.java to calculate the maximum backlog.
After all the packets are received from BlackBox, we calculate the backlog based on the sequence numbers between the arrival and departure functions.

```
long Bmax = 0;
for (int i = 1; i <= N; i++) {
    long temp = 0;
    long receiveTime = TrafficEstimator.mapGet(i).getReceive();
    for (int j = 1; j <= N; j++){
        long sendTime = TrafficEstimator.mapGet(j).getSend();
        if (sendTime > receiveTime){
            temp = (j - i)*(L/N)*8;
            break;
        }
    }

    if (Bmax < temp)
        Bmax = temp;
}
System.out.println("The max backlog is " + Bmax + " bits");
```

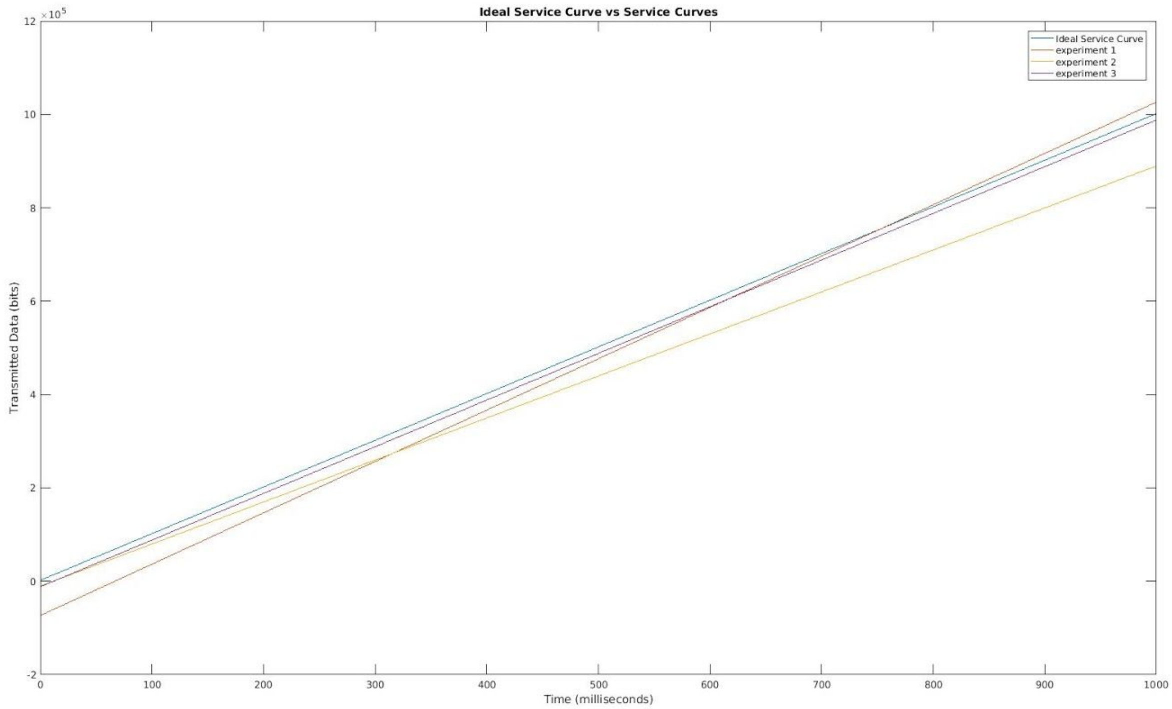
Bmax(r) = sup{A(t) - D(t)}, we could use matlab to retrieve it from sink output
S(t) = max(rt - Bmax(r))

**BlackBox 1: b = Lmax, R = 1 Mbps, T = 10,000 us, Ideal Service Curve: S(t) = 11840 + 1000(t - 10)**

Experiment 1: r = 1100 Kbps; Bmax = 73600 bits; S(t) = 1100*t - 73600

Experiment 2: r = 900 Kbps; Bmax = 10400 bits; S(t) = 900*t - 10400

Experiment 3: r = 1000 Kbps; Bmax = 12000 bits; S(t) = 1000*t - 12000



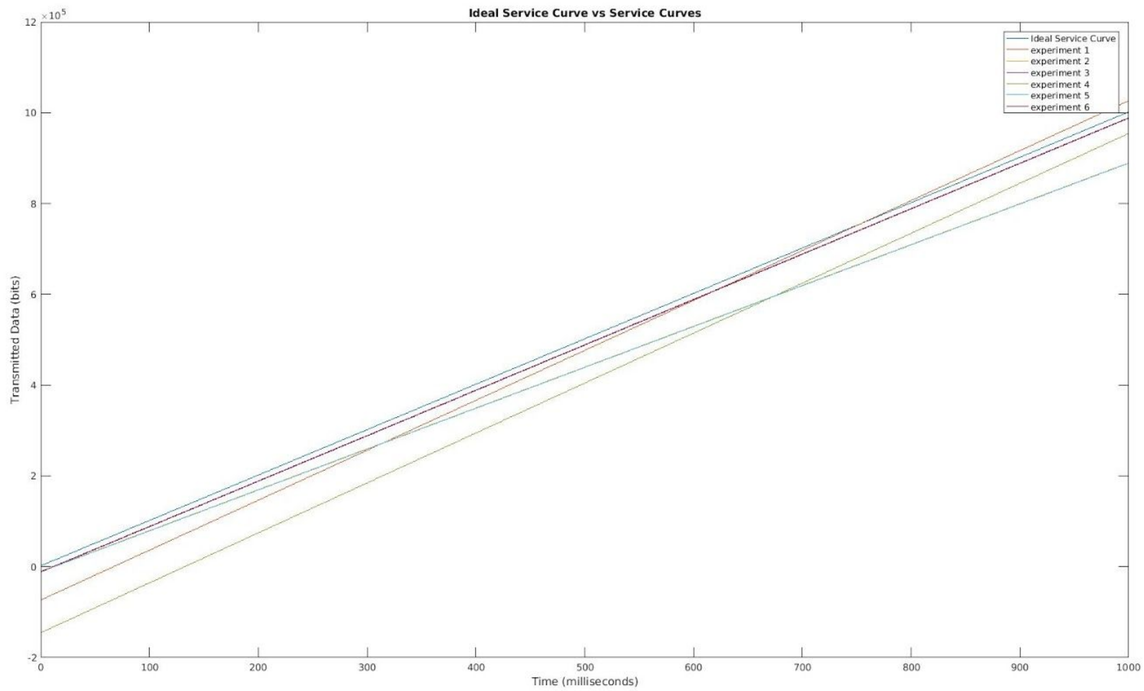

As shown in the image above, we could see that the experiment service curves are close and overlapping to the ideal service curve. The BlackBox is able to transmit the packets from TrafficGenerator to TrafficSink. Furthermore, from the closer comparison, we could see that the combination of the service curve (S(t) = {max(rt-Bmax(r)) | r belongs to R}) is closest to the ideal service curve.

Repeat the experiment with N = 2N = 2000 and L = 2L = 200,000

Experiment 4: r = 1100 Kbps; Bmax = 145600 bits; S(t) = 1100*t - 145600

Experiment 5: r = 900 Kbps; Bmax = 11200 bits; S(t) = 900*t - 11200

Experiment 6: r = 1000 Kbps; Bmax = 11200 bits; S(t) = 1000*t - 11200



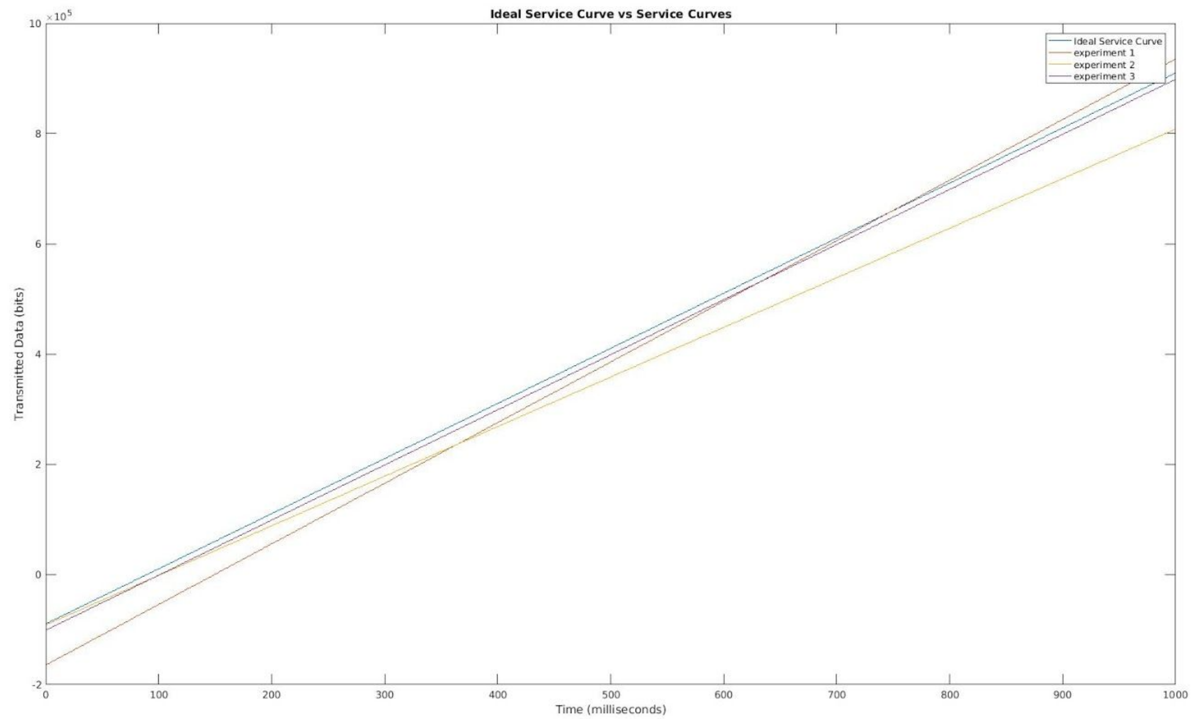

By repeating the experiment above, we still get the similar results. The image shown above displays all six experiments with the ideal service curve. With the length of packet train increased, the maximum backlog increases as well. When r > R and as the number of the transmitted bit increases, the backlog increases because of the rates difference between r and R.

**BlackBox 2: b =  10000bits, R = 1 Mbps, T = 100,000 us, Ideal Service Curve: S(t) = 10000 + 1000(t - 100)**

Experiment 1: r = 1100 Kbps; Bmax = 164800 bits; S(t) = 1100*t - 164800

Experiment 2: r = 900 Kbps; Bmax = 92000 bits; S(t) = 900*t - 92000

Experiment 3: r = 1000 Kbps; Bmax = 101600 bits; S(t) = 1000*t - 101600



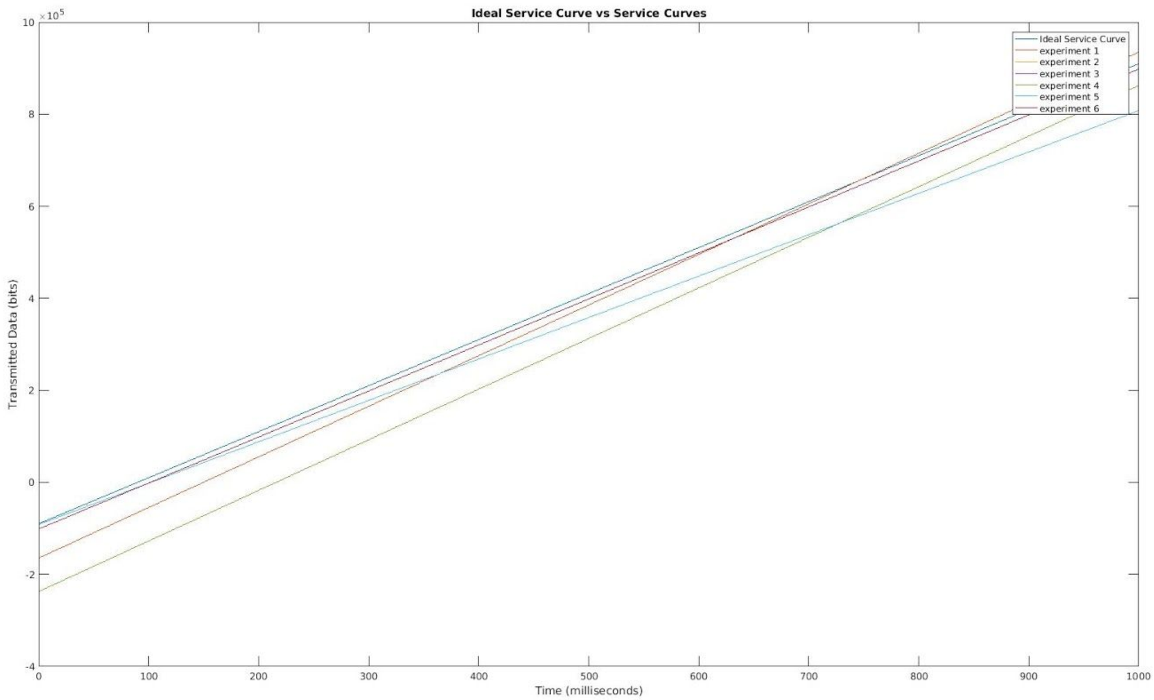

The graph above appears to be similar to the previous BlackBox that we tried. It has the similar results as the results from probing methodology is close to the ideal service curve. The only big difference between this and BlackBox1 is that this has a larger Bmax.

Repeat the experiment with N = 2N = 2000 and L = 2L = 200,000

Experiment 4: r = 1100 Kbps; Bmax = 237600 bits; S(t) = 1100*t - 237600

Experiment 5: r = 900 Kbps; Bmax = 92000 bits; S(t) = 900*t - 92000

Experiment 6: r = 1000 Kbps; Bmax = 101600 bits; S(t) = 1000*t - 101600



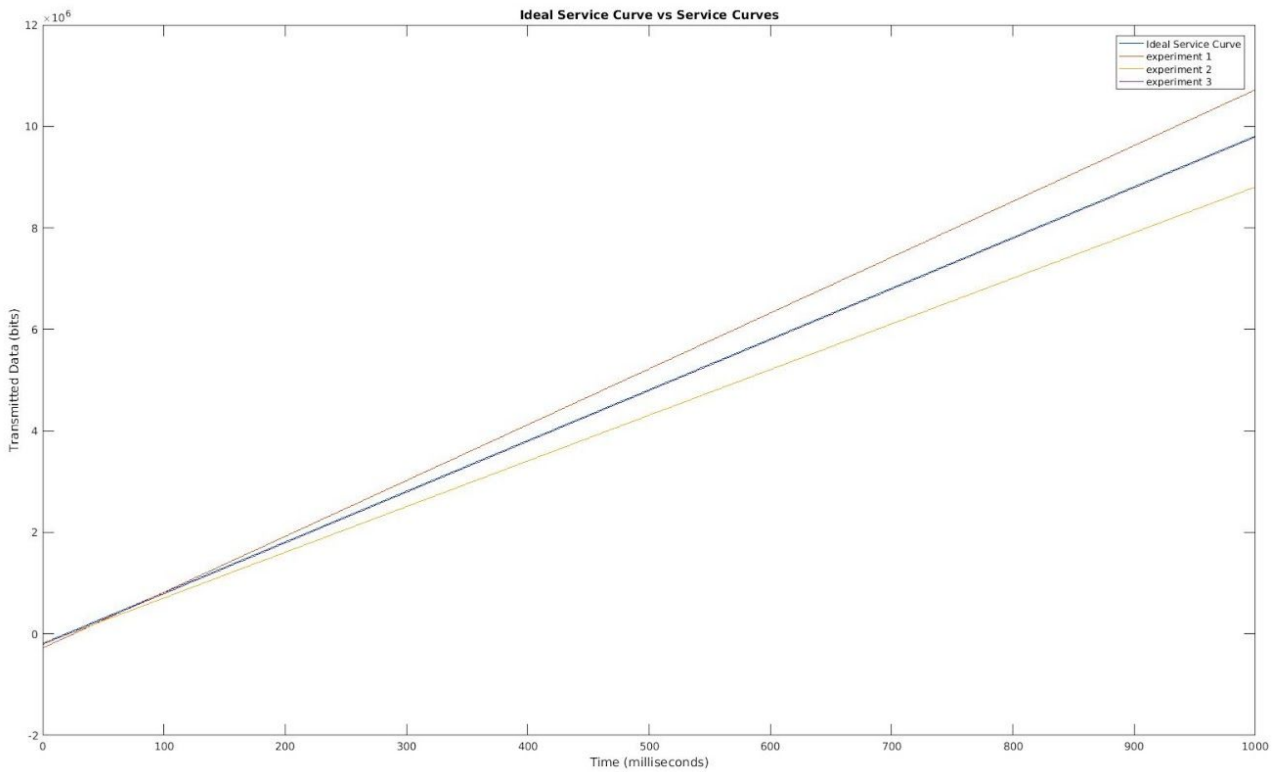

The graph has the similar results as the one from BlackBox1, as the N and L increases, the maximum backlog increases as well.

**BlackBox 3: b = Lmax, R = 10 Mbps, T = 20,000 us, Ideal Service Curve: S(t) = 11840 + 1000(t - 20)**

Experiment 1: r = 11000 Kbps; Bmax = 280000 bits; S(t) = 11000*t - 280000

Experiment 2: r = 9000 Kbps; Bmax = 193600 bits; S(t) = 9000*t - 193600

Experiment 3: r = 10000 Kbps; Bmax = 210400 bits; S(t) = 10000*t - 210400



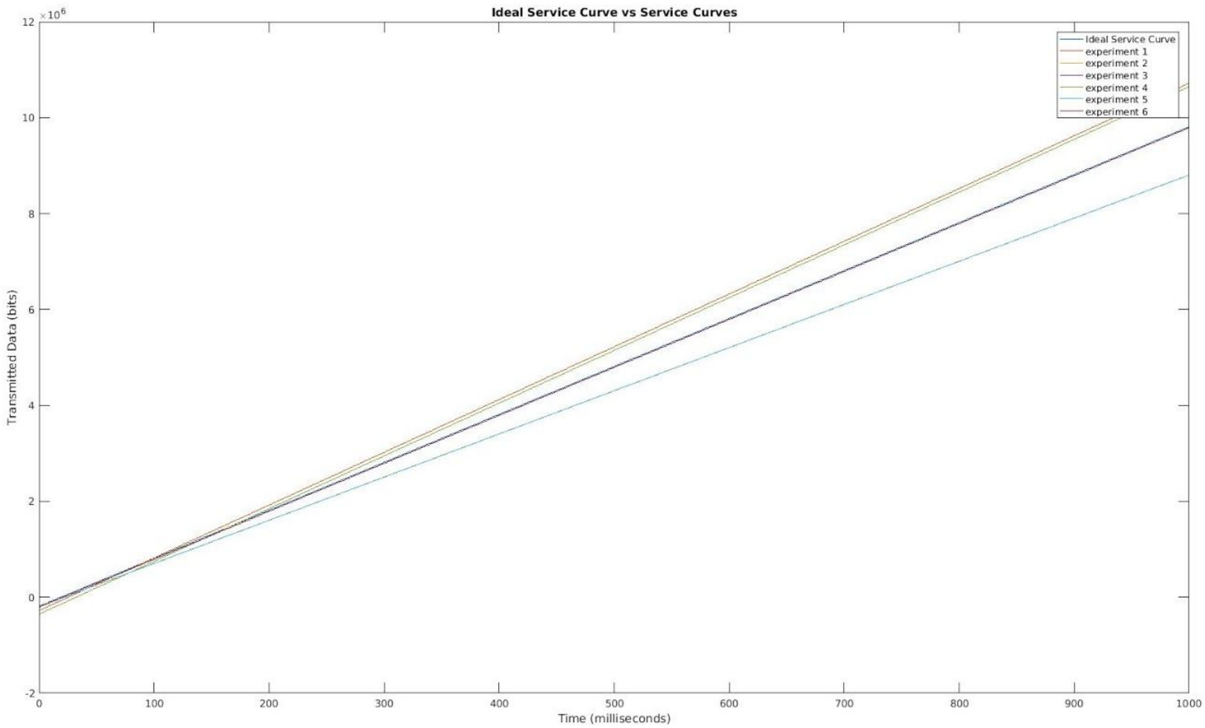

In this BlackBox, the rate R has increased to the 10 times of the two previous. We found that the combination of the estimated service curves are kind of far from ideal service curve. Only if when the selected rate r is close to the rate in the BlackBox.

Repeat the experiment with N = 2N = 2000 and L = 2L = 200,000

Experiment 4: r = 11000 Kbps; Bmax = 354400 bits; S(t) = 11000*t - 354400

Experiment 5: r = 9000 Kbps; Bmax = 197600 bits; S(t) = 9000*t - 197600

Experiment 6: r = 10000 Kbps; Bmax = 210400 bits; S(t) = 10000*t - 210400




The repeated experiment with double N and L shows the similar results in the previous image in the same BlackBox.

The estimates seem to be pretty close to the ideal service curves, so we are pretty confident about the estimations. When the estimated service curve could not have a higher line it means that it has found the final results. But there is need to use the binary search method to find the correct parameters, which requires pretty much effort on it.

**Exercise 2.2 Evaluation of BlackBox with unknown parameters**

**Estimate the unknown parameters with N = 1000, L = 100,000**

**BlackBox1.jar**
First, we tested the burst size by changing N to different number and check what is the maximum packet size it could receive, which is the burst for the BlackBox.
We tried in the following procedure by setting N 1000 -> 100 -> 200 -> 150 -> 180 -> 170 -> 165. And we found that 170 could pass and 165 could not, so the burst equals to 100000/170 = 588 bytes = 4704 bits

Experiment 1: r = 100 Kbps; Bmax = 10400 bits; S(t) = 100*t - 10400

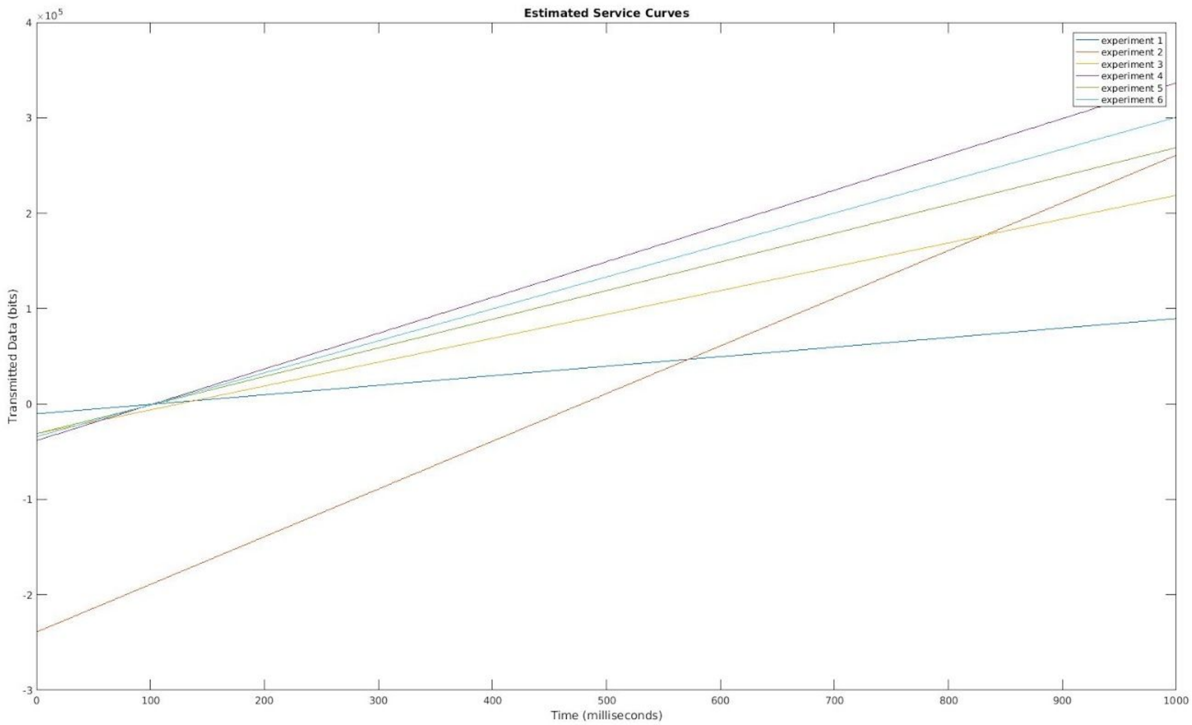
Experiment 2: r = 500 Kbps; Bmax = 239200 bits; S(t) = 500*t - 239200

Experiment 3: r = 250 Kbps; Bmax = 31200 bits; S(t) = 250*t - 31200

Experiment 4:  r = 375 Kbps; Bmax = 38400 bits; S(t) = 375*t - 38400

Experiment 5: r = 300 Kbps; Bmax = 31200 bits; S(t) = 300*t - 31200

Experiment 6: r = 335 Kbps; Bmax = 34400 bits; S(t) = 335*t - 34400




From the image shown above, it is the drawings of all S(t) experiments we have done. So from the image, we could see that the curve with rate r = 375 Kbps appears to be rate for BlackBox since there is no other curves higher than it.
From the sinkOutputBB1.txt, we could know that the T delay is about 100 ms from the difference between the send and receive time. We got T = 100 ms.
So b = 4704 bits, T = 100 ms, r = 375 Kbps

**BlackBox2.jar**

Similarly, we used the same method to find b as the previous BlackBox1.jar. We tried in the following procedure by setting N 1000 -> 500 -> 200 -> 350 -> 275 -> 237 -> 250 -> 270 -> 265 .And we found that 270 could pass and 265 could not, so the burst equals to 100000/270 = 370 bytes = 2960 bits

Experiment 1: r = 100 Kbps; Bmax = 399600 bits; S(t) = 100*t - 399600

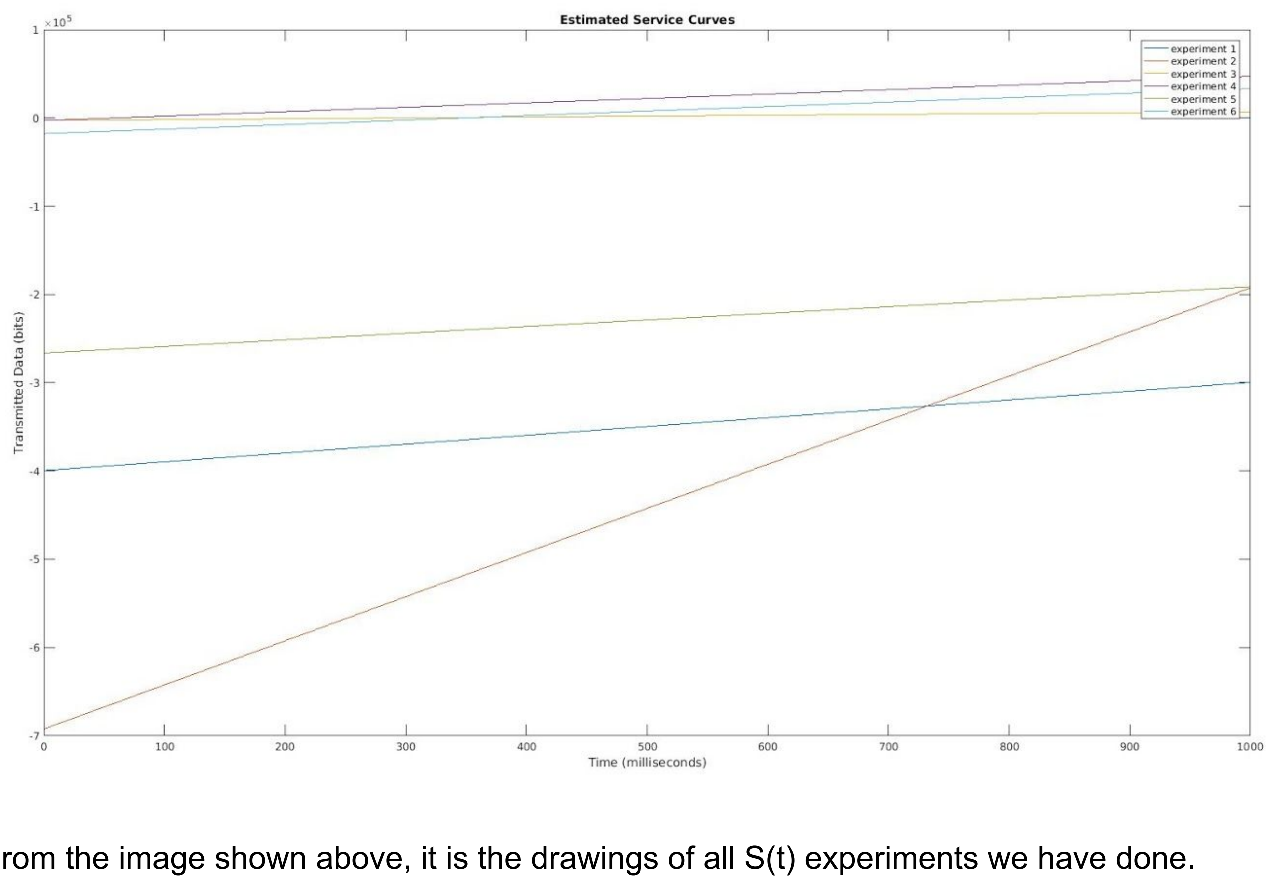
Experiment 2: r = 500 Kbps; Bmax = 692640 bits; S(t) = 500*t - 692640

Experiment 3: r = 10 Kbps; Bmax = 2960 bits; S(t) = 10*t - 2960

Experiment 4:  r = 50 Kbps; Bmax = 2960 bits; S(t) = 50*t - 2960

Experiment 5: r = 75 Kbps; Bmax = 26400 bits; S(t) = 75*t - 266400

Experiment 6: r = 51 Kbps; Bmax = 2960 bits; S(t) = 51*t - 17760




From the image shown above, it is the drawings of all S(t) experiments we have done. So from the image, we could see that the curve with rate r = 50 Kbps appears to be rate for BlackBox since when r = 51 Kbps, the change in Bmax is significant.

From the sinkOutputBB2.txt, we could know that the T delay is about 0 ms from the difference between the send and receive time. We got T = 0 ms.
So b = 2960 bits, T = 0 ms, r = 50 Kbps

**BlackBox3.jar**
Similarly, we used the same method to find b as the previous BlackBox1.jar. We tried in the following procedure by setting N 1000 -> 500 -> 300 -> 400 -> 350 -> 375 -> 365 -> 370. And we found that 370 could pass and 365 could not, so the burst equals to 100000/370 = 270 bytes = 2160 bits

Experiment 1: r = 100 Kbps; Bmax = 8640 bits; S(t) = 100*t - 8640

Experiment 2: r = 500 Kbps; Bmax = 546480 bits; S(t) = 500*t - 546480

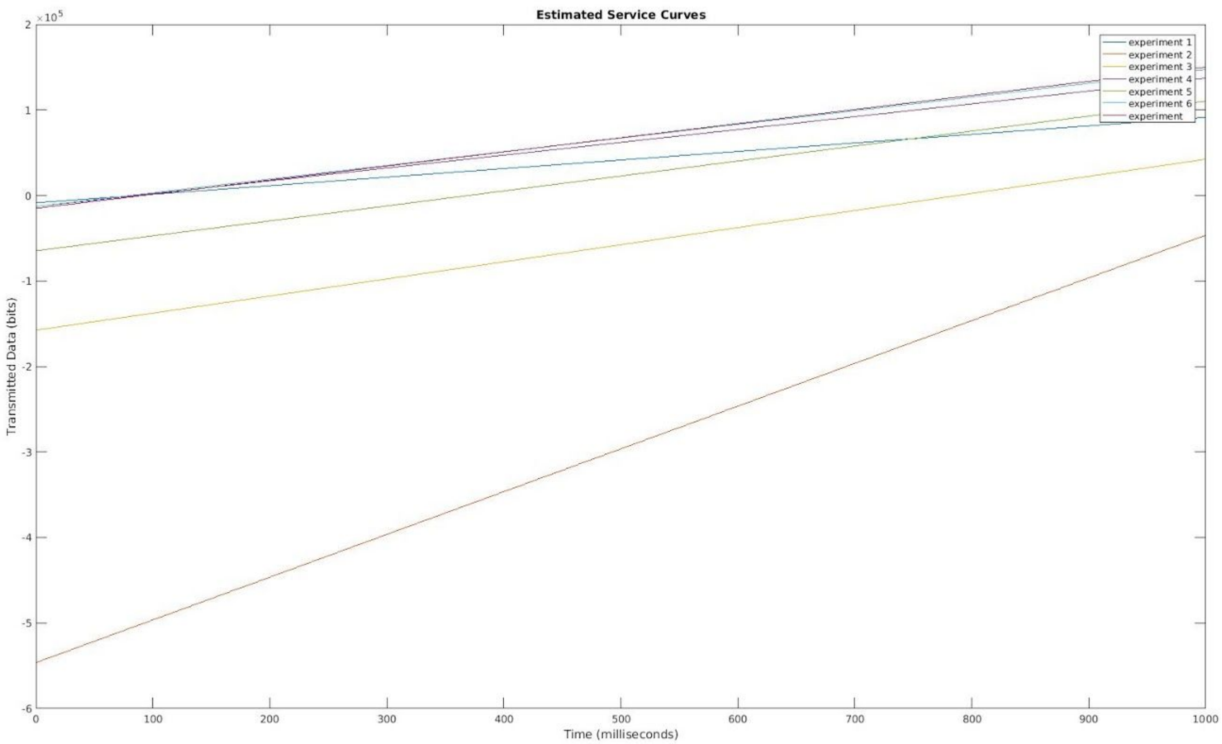
Experiment 3: r = 200 Kbps; Bmax = 157680 bits; S(t) = 200*t - 157680

Experiment 4: r = 150 Kbps; Bmax = 12960 bits; S(t) = 150*t - 12960

Experiment 5: r = 175 Kbps; Bmax = 64800 bits; S(t) = 175*t - 64800

Experiment 6: r = 160 Kbps; Bmax = 12960 bits; S(t) = 160*t - 12960

Experiment 7: r = 165 Kbps; Bmax = 15120 bits; S(t) = 165*t - 15120




From the image shown above, it is the drawings of all S(t) experiments we have done. So from the image, we could see that the curve with rate r = 165 Kbps appears to be rate for BlackBox since it is the higher one in the figure.
From the sinkOutputBB3.txt, we could know that the T delay is about 75 ms from the difference between the send and receive time. We got T = 75 ms.
So b = 2160 bits, T = 75 ms, r = 165 Kbps