Lab 4: Bandwidth Estimation

Harmanprit Tatla – tatlahar
Shunta Chimura – chimuras

Content

60

presentation:

20

code:

20

Exercise 1.5 Evaluation

Provide the plots and discussions of the graphs generated in Exercise 1.5

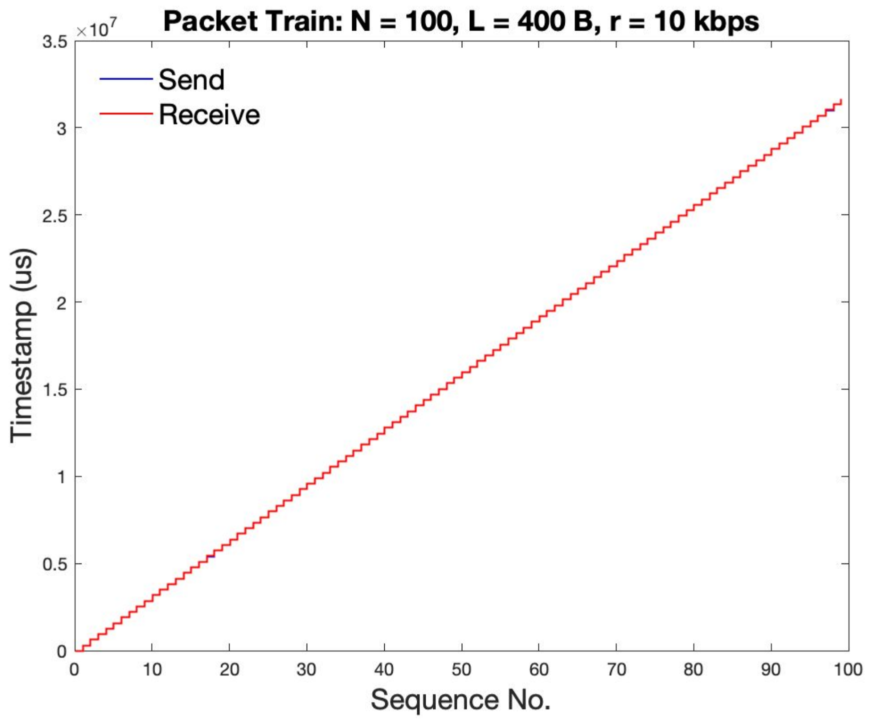


Figure 1: Send and Receive Timestamps of packet train with N = 100, L = 400 B and r = 10 Kbps

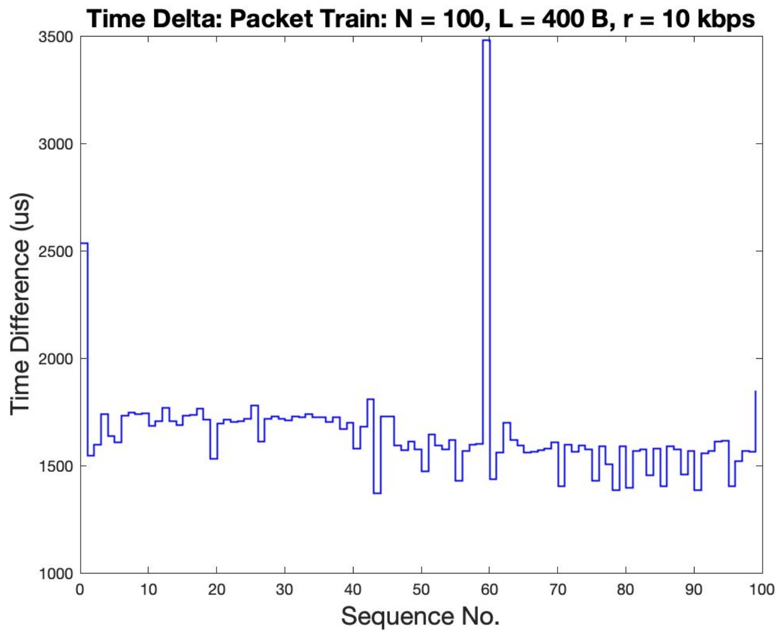


Figure 1a: Difference of Send and Receive Timestamps of packet train with N = 100, L = 400 B and r = 10 Kbps

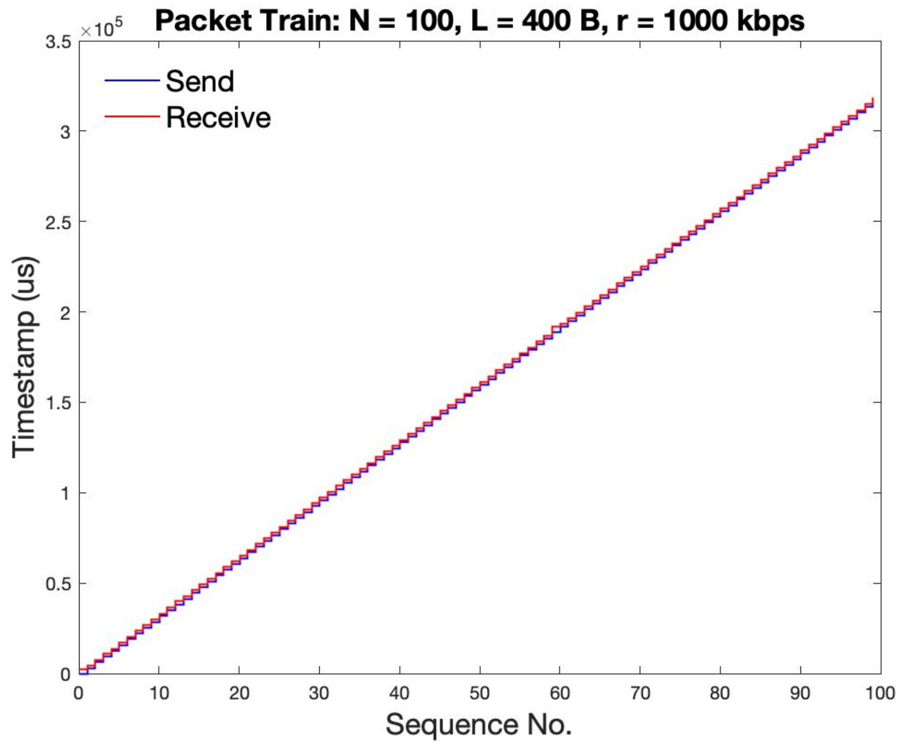


Figure 2: Send and Receive Timestamps of packet train with N = 100, L = 400 B and r = 1000 Kbps

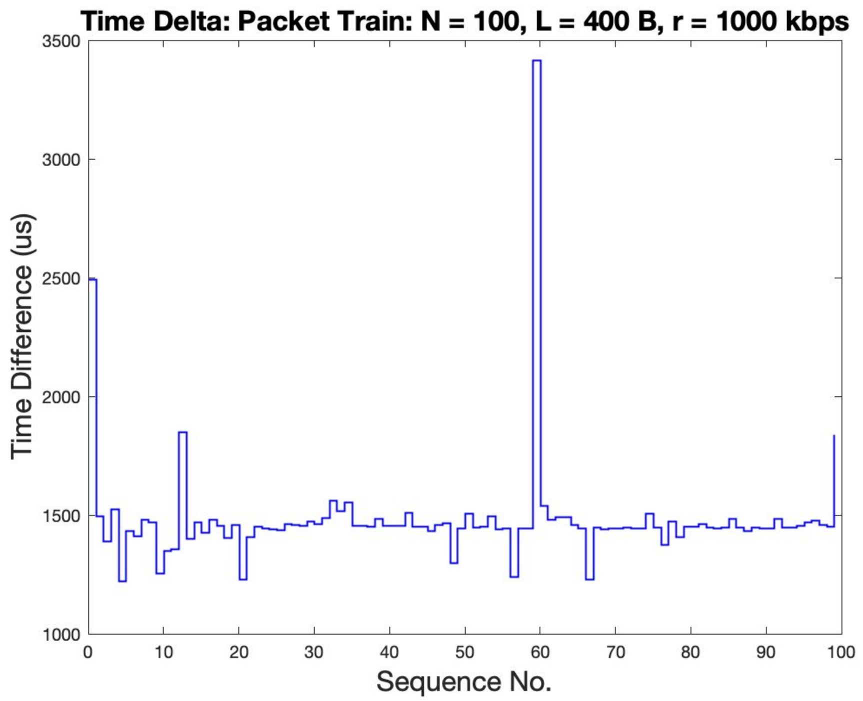


Figure 2a: Difference of Send and Receive Timestamps of packet train with N = 100, L = 400 B and r = 1000 Kbps

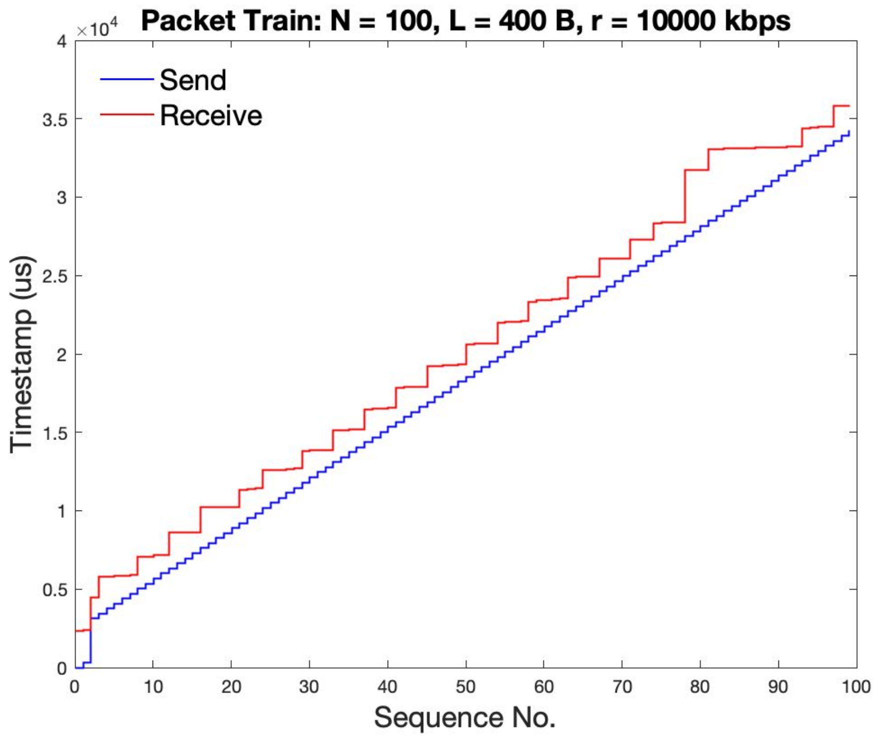


Figure 3: Send and Receive Timestamps of packet train with N = 100, L = 400 B and r = 10000 Kbps

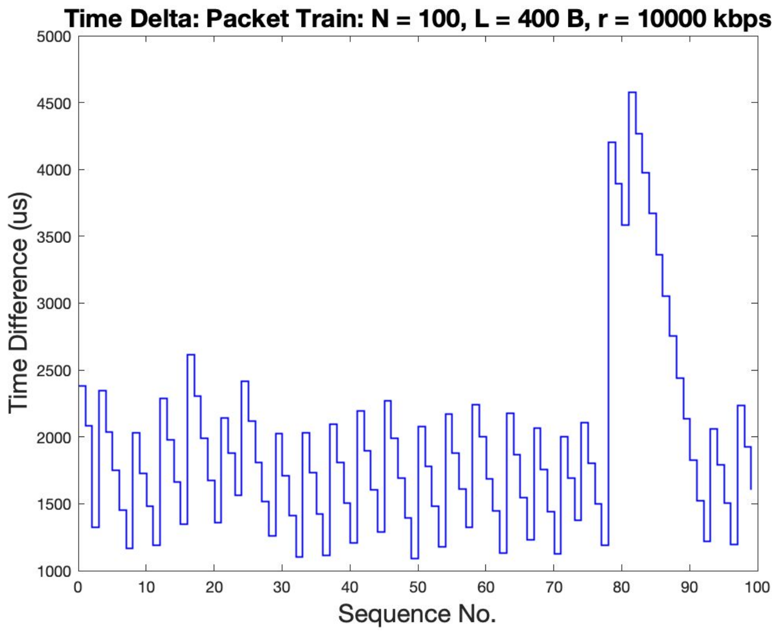
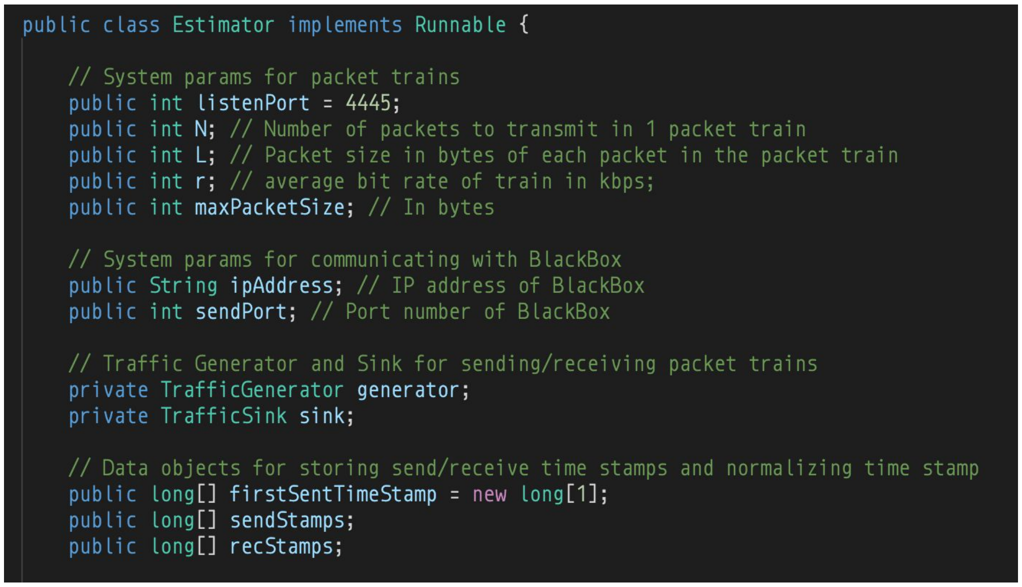


Figure 3a: Difference of Send and Receive Timestamps of packet train with N = 100, L = 400 B and r = 10,000 Kbps

Discussion of Plots

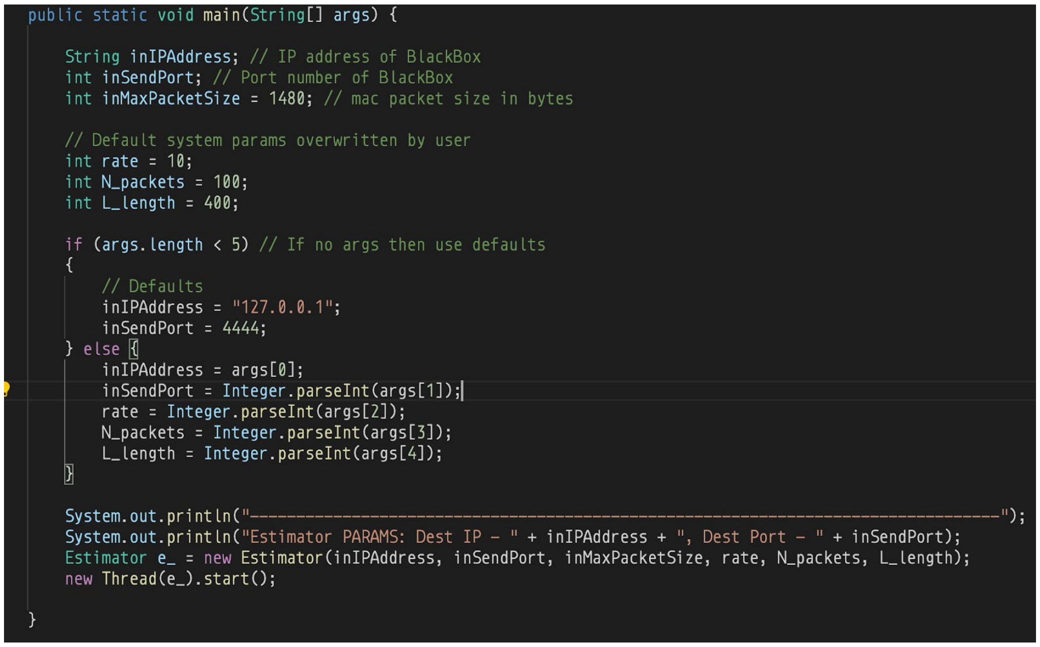
From figures 1 through 3 we observe that as we increase the packet train arrival rate r, the delay between the send and receive timestamps oscillates more frequently between two extremes, i.e. 1000 μs + overhead#1 and 1000 μs + overhead#2, where the overhead values are system specific delays incurred by the BlackBox, generator and receiver that vary depending on arrival rate.

Include the source code for the Estimator, that was used for Exercise 1.5



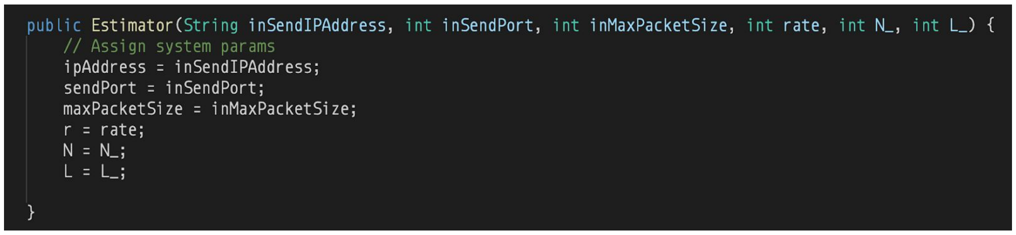
```java
public class Estimator implements Runnable {

    // System params for packet trains
    public int listenPort = 4445;
    public int N; // Number of packets to transmit in 1 packet train
    public int L; // Packet size in bytes of each packet in the packet train
    public int r; // average bit rate of train in kbps;
    public int maxPacketSize; // In bytes

    // System params for communicating with BlackBox
    public String ipAddress; // IP address of BlackBox
    public int sendPort; // Port number of BlackBox

    // Traffic Generator and Sink for sending/receiving packet trains
    private TrafficGenerator generator;
    private TrafficSink sink;

    // Data objects for storing send/receive time stamps and normalizing time stamp
    public long[] firstSentTimeStamp = new long[1];
    public long[] sendStamps;
    public long[] recStamps;
```

Figure 4: System Parameters for the estimator



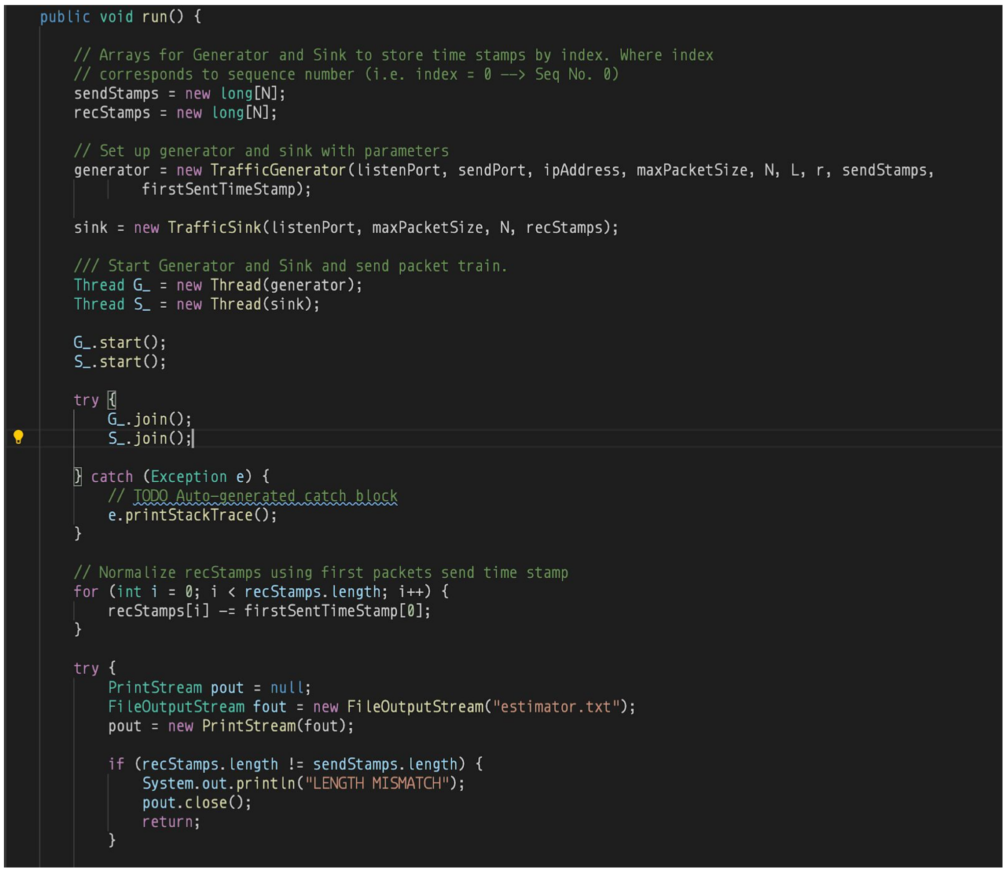
```java
public static void main(String[] args) {

    String inIPAddress; // IP address of BlackBox
    int inSendPort; // Port number of BlackBox
    int inMaxPacketSize = 1480; // mac packet size in bytes

    // Default system params overwritten by user
    int rate = 10;
    int N_packets = 100;
    int L_length = 400;

    if (args.length < 5) // If no args then use defaults
    {
        // Defaults
        inIPAddress = "127.0.0.1";
        inSendPort = 4444;
    } else {
        inIPAddress = args[0];
        inSendPort = Integer.parseInt(args[1]);
        rate = Integer.parseInt(args[2]);
        N_packets = Integer.parseInt(args[3]);
        L_length = Integer.parseInt(args[4]);
    }

    System.out.println("--------------------------------------------------------------------------------");
    System.out.println("Estimator PARAMS: Dest IP - " + inIPAddress + ", Dest Port - " + inSendPort);
    Estimator e_ = new Estimator(inIPAddress, inSendPort, inMaxPacketSize, rate, N_packets, L_length);
    new Thread(e_).start();

}
```

Figure 5: Estimator Main function to acquire user packet train parameters

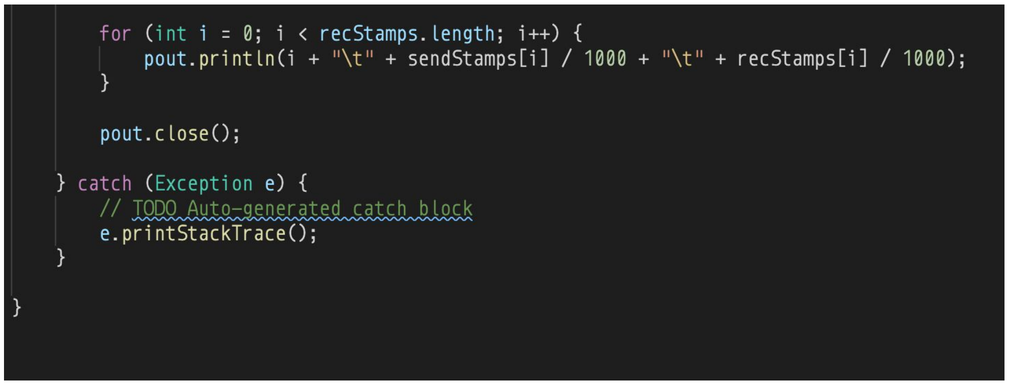


```java
public Estimator(String inSendIPAddress, int inSendPort, int inMaxPacketSize, int rate, int N_, int L_) {
    // Assign system params
    ipAddress = inSendIPAddress;
    sendPort = inSendPort;
    maxPacketSize = inMaxPacketSize;
    r = rate;
    N = N_;
    L = L_;

}
```

Figure 6: Estimator Constructor to assign packet train parameters



```java
    public void run() {

        // Arrays for Generator and Sink to store time stamps by index. Where index
        // corresponds to sequence number (i.e. index = 0 --> Seq No. 0)
        sendStamps = new long[N];
        recStamps = new long[N];

        // Set up generator and sink with parameters
        generator = new TrafficGenerator(listenPort, sendPort, ipAddress, maxPacketSize, N, L, r, sendStamps,
                firstSentTimeStamp);

        sink = new TrafficSink(listenPort, maxPacketSize, N, recStamps);

        /// Start Generator and Sink and send packet train.
        Thread G_ = new Thread(generator);
        Thread S_ = new Thread(sink);

        G_.start();
        S_.start();

        try {
            G_.join();
            S_.join();

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        // Normalize recStamps using first packets send time stamp
        for (int i = 0; i < recStamps.length; i++) {
            recStamps[i] -= firstSentTimeStamp[0];
        }

        try {
            PrintStream pout = null;
            FileOutputStream fout = new FileOutputStream("estimator.txt");
            pout = new PrintStream(fout);

            if (recStamps.length != sendStamps.length) {
                System.out.println("LENGTH MISMATCH");
                pout.close();
                return;
            }
```



```java
            for (int i = 0; i < recStamps.length; i++) {
                pout.println(i + "\t" + sendStamps[i] / 1000 + "\t" + recStamps[i] / 1000);
            }

            pout.close();

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

}
```

Figure 7: Estimator Run function to send and receive from/to BlackBox

Exercise 2.1 Implement and test the probing methodology

Please note that on the UG machines we were unable to use the stated rates of R = 100 Mbps and R = 1000 Mbps for this section of the lab due to the following frequently occurring error - "*Exception in thread "Thread-3" java.lang.IllegalArgumentException: nanosecond timeout value out of range*". To work around this issue, we chose to use **R = 1 Mbps** in place of R = 100 Mbps and **R = 5 Mbps** in place of R = 1000 Mbps. Furthermore, all service curve estimates titled $S\#^{Double}$ correspond to packet trains where N = 200 and L = 800 Bytes (i.e. double the initial values stated in the lab) whereas estimates of the form $S\#$ used N = 100 and L = 400 Byes.

Provide a description of how you implemented the probing methodology as an addition to the Estimator

Implementation of the probing methodology involved the following where each bullet point summarizes our implementation of the iterative algorithm in an order similar to that provided in the lab handout:

- For section 2.1 we selected initial packet train rates r close to the value R of the BlackBox, where in each iteration of the algorithm we increased r by some amount (i.e. dependent on the BlackBox in question). Each value of r is added to an array list.

- We transmitted a packet train with rate r using the implementation from part 1 of the lab.

- To compute A(t), we took the send timestamp array (of size N) and computed its cumulative sum. As for D(t), we repeated this step but instead for the receive timestamps array. The index into the A(t) or D(t) arrays corresponds to the cumulative number of bytes (i.e. for A[i] the corresponding cumulative bytes for this time instance is i * L). See figure 8 below.

- In order to compute $B_{max}(r)$, we found for each cumulative departure time the nearest cumulative arrival time (i.e. for D[i] we found nearest A[j]). From here, we took the difference of the indexes into each array and multiplied by the number of bits per packet (i.e. L * 8) to get the backlog. The $B_{max}(r)$ corresponding to each r is stored in an array list. See figure 9 below.

- Upon finding $B_{max}(r)$, we then compute the estimate for S(t). This is done by computing rt - $B_{max}(r)$ for each r and its corresponding $B_{max}(r)$ at t = 100 ms, the combination that produces the largest S(t) estimate is chosen as the best 'r' and '$B_{max}(r)$' found so far. See figure 10 below.

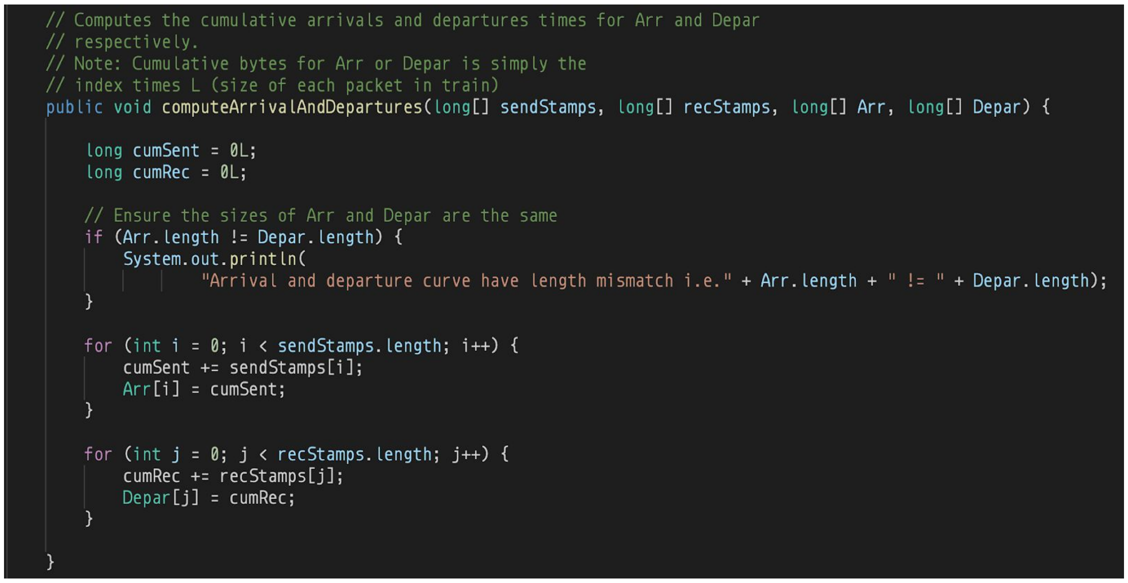
- If the S(t) estimate is larger than the last one, the algorithm continues to iterate and increase rate r, otherwise, if the estimate is the same as or smaller than the former estimate, the algorithm halts and returns the r and $B_{max}(r)$ combination leading to the best S(t) estimate found so far.

Note: The value t = 100 ms was chosen from trial and error to best determine which r and $B_{max}(r)$ combination led to the best S(t) estimate.

Discuss your algorithm for selecting the rates

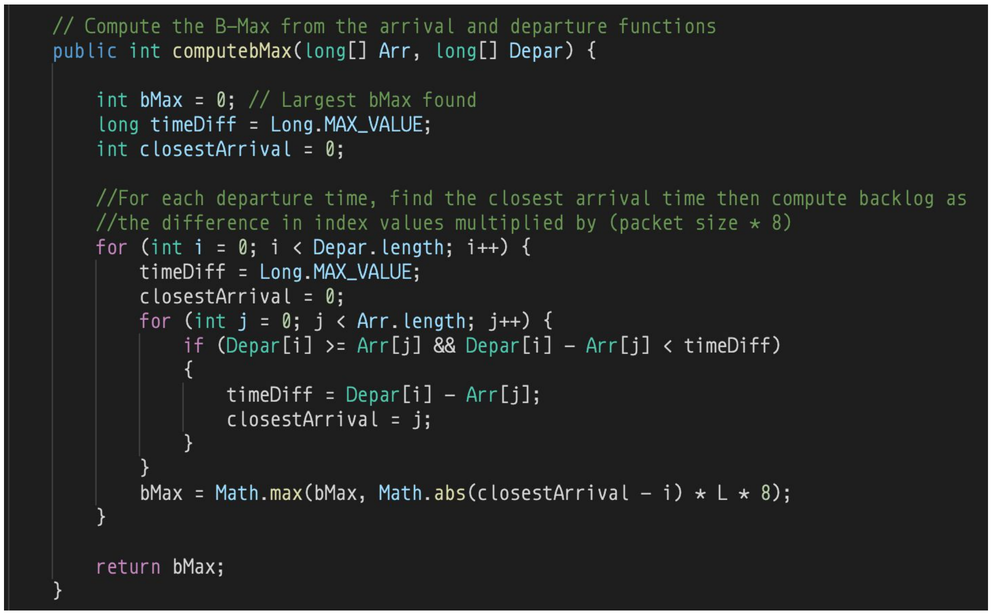
As parameters of each BlackBox were known, in particular, their long term rates 'R', we decided to choose arrival rates r that were close to R (i.e. 50 to 100 Kbps less than R). The objective was to see if the service curves obtained by selecting packet train arrival rates at or around the long term rate of the BlackBox led to service curve estimates close to the ideal. As for iteratively increasing the rate r after each iteration, we tried increases of 5 Kbps to 25 Kbps, and found that the latter produced the best results.

Include the source code for the Estimator, that was used for Exercises 2.1 and 2.2



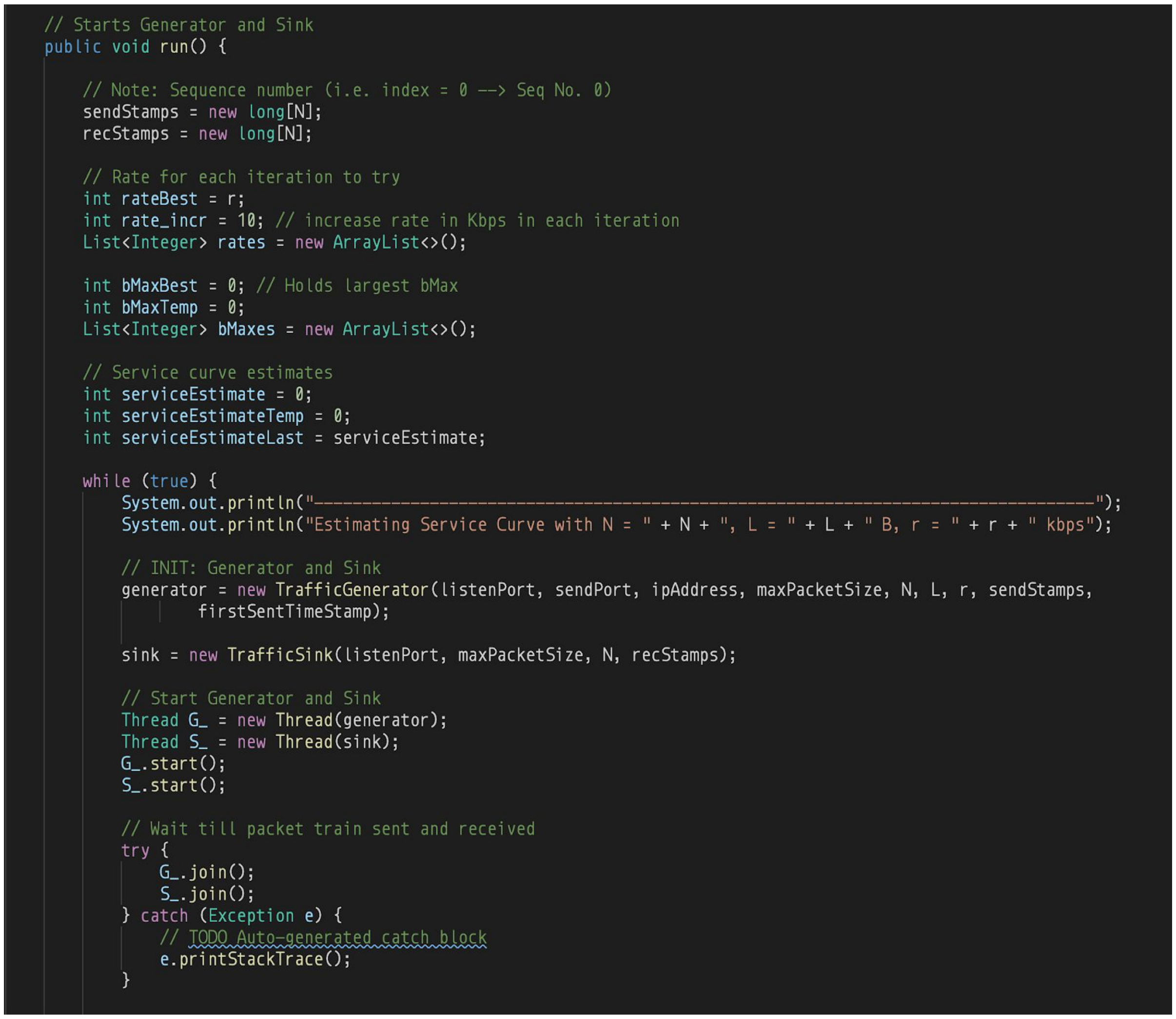
```
// Computes the cumulative arrivals and departures times for Arr and Depar
// respectively.
// Note: Cumulative bytes for Arr or Depar is simply the
// index times L (size of each packet in train)
public void computeArrivalAndDepartures(long[] sendStamps, long[] recStamps, long[] Arr, long[] Depar) {

    long cumSent = 0L;
    long cumRec = 0L;

    // Ensure the sizes of Arr and Depar are the same
    if (Arr.length != Depar.length) {
        System.out.println(
                "Arrival and departure curve have length mismatch i.e." + Arr.length + " != " + Depar.length);
    }

    for (int i = 0; i < sendStamps.length; i++) {
        cumSent += sendStamps[i];
        Arr[i] = cumSent;
    }

    for (int j = 0; j < recStamps.length; j++) {
        cumRec += recStamps[j];
        Depar[j] = cumRec;
    }

}
```

Figure 8: Estimator code to compute A(t) and D(t)

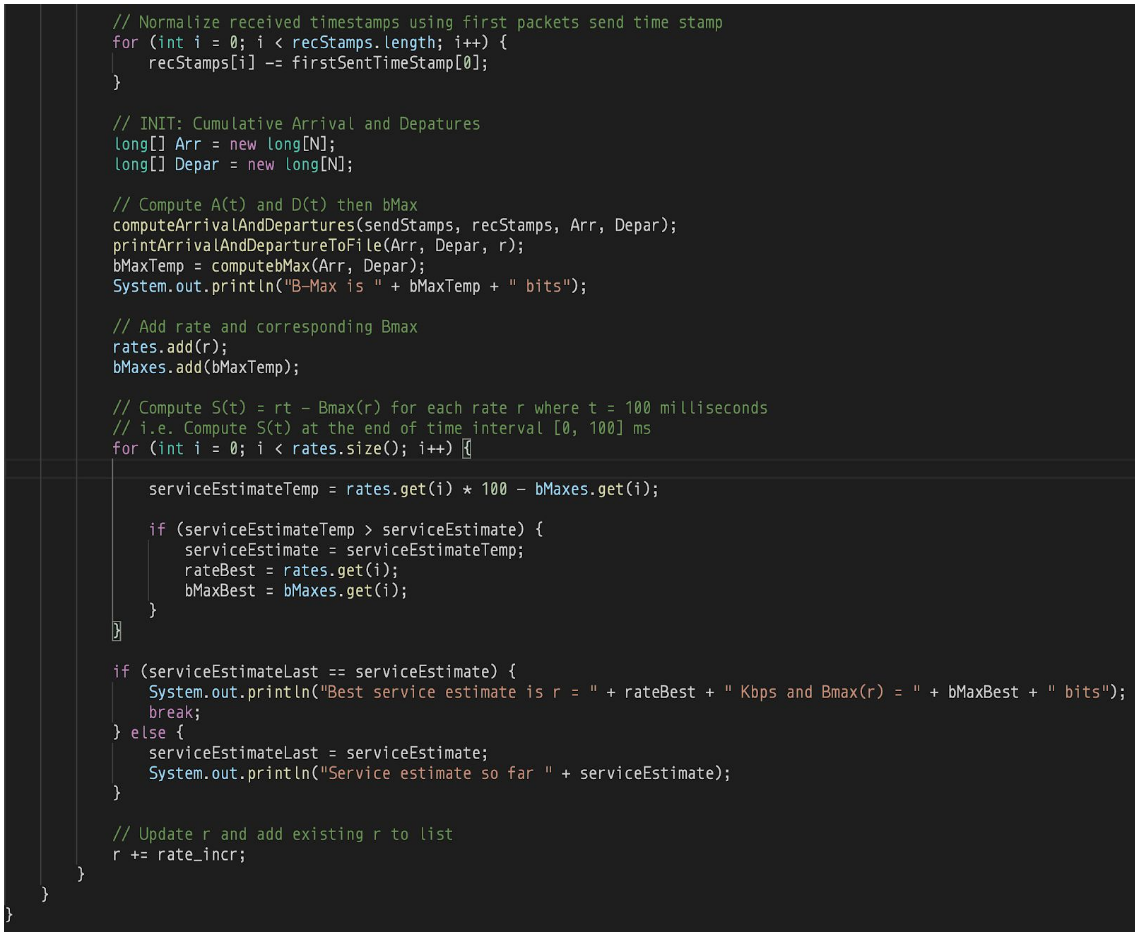


```java
// Compute the B-Max from the arrival and departure functions
public int computebMax(long[] Arr, long[] Depar) {

    int bMax = 0; // Largest bMax found
    long timeDiff = Long.MAX_VALUE;
    int closestArrival = 0;

    //For each departure time, find the closest arrival time then compute backlog as
    //the difference in index values multiplied by (packet size * 8)
    for (int i = 0; i < Depar.length; i++) {
        timeDiff = Long.MAX_VALUE;
        closestArrival = 0;
        for (int j = 0; j < Arr.length; j++) {
            if (Depar[i] >= Arr[j] && Depar[i] - Arr[j] < timeDiff)
            {
                timeDiff = Depar[i] - Arr[j];
                closestArrival = j;
            }
        }
        bMax = Math.max(bMax, Math.abs(closestArrival - i) * L * 8);
    }

    return bMax;
}
```

Figure 9: Estimator code to compute $B_{max}(r)$



```java
// Starts Generator and Sink
public void run() {

    // Note: Sequence number (i.e. index = 0 --> Seq No. 0)
    sendStamps = new long[N];
    recStamps = new long[N];

    // Rate for each iteration to try
    int rateBest = r;
    int rate_incr = 10; // increase rate in Kbps in each iteration
    List<Integer> rates = new ArrayList<>();

    int bMaxBest = 0; // Holds largest bMax
    int bMaxTemp = 0;
    List<Integer> bMaxes = new ArrayList<>();

    // Service curve estimates
    int serviceEstimate = 0;
    int serviceEstimateTemp = 0;
    int serviceEstimateLast = serviceEstimate;

    while (true) {
        System.out.println("---------------------------------------------------------------------------------");
        System.out.println("Estimating Service Curve with N = " + N + ", L = " + L + " B, r = " + r + " kbps");

        // INIT: Generator and Sink
        generator = new TrafficGenerator(listenPort, sendPort, ipAddress, maxPacketSize, N, L, r, sendStamps,
                firstSentTimeStamp);

        sink = new TrafficSink(listenPort, maxPacketSize, N, recStamps);

        // Start Generator and Sink
        Thread G_ = new Thread(generator);
        Thread S_ = new Thread(sink);
        G_.start();
        S_.start();

        // Wait till packet train sent and received
        try {
            G_.join();
            S_.join();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```



```
            // Normalize received timestamps using first packets send time stamp
            for (int i = 0; i < recStamps.length; i++) {
                recStamps[i] -= firstSentTimeStamp[0];
            }

            // INIT: Cumulative Arrival and Depatures
            long[] Arr = new long[N];
            long[] Depar = new long[N];

            // Compute A(t) and D(t) then bMax
            computeArrivalAndDepartures(sendStamps, recStamps, Arr, Depar);
            printArrivalAndDepartureToFile(Arr, Depar, r);
            bMaxTemp = computebMax(Arr, Depar);
            System.out.println("B-Max is " + bMaxTemp + " bits");

            // Add rate and corresponding Bmax
            rates.add(r);
            bMaxes.add(bMaxTemp);

            // Compute S(t) = rt - Bmax(r) for each rate r where t = 100 milliseconds
            // i.e. Compute S(t) at the end of time interval [0, 100] ms
            for (int i = 0; i < rates.size(); i++) {

                serviceEstimateTemp = rates.get(i) * 100 - bMaxes.get(i);

                if (serviceEstimateTemp > serviceEstimate) {
                    serviceEstimate = serviceEstimateTemp;
                    rateBest = rates.get(i);
                    bMaxBest = bMaxes.get(i);
                }
            }

            if (serviceEstimateLast == serviceEstimate) {
                System.out.println("Best service estimate is r = " + rateBest + " Kbps and Bmax(r) = " + bMaxBest + " bits");
                break;
            } else {
                serviceEstimateLast = serviceEstimate;
                System.out.println("Service estimate so far " + serviceEstimate);
            }

            // Update r and add existing r to list
            r += rate_incr;
        }
    }
}
```

Figure 10: Estimator run function that implements the probing algorithm

Please note that for each BlackBox instance below we discuss the following: (1) Comparison of estimates to ideal service curve (2) Effects of repeating experiments twice and (3) Effects of doubling packet trains and packet size (4) Any Conclusions/Explanations.

BlackBox 1: b= 11,840 bits , R=1 Mbps, T = 10,000 μs

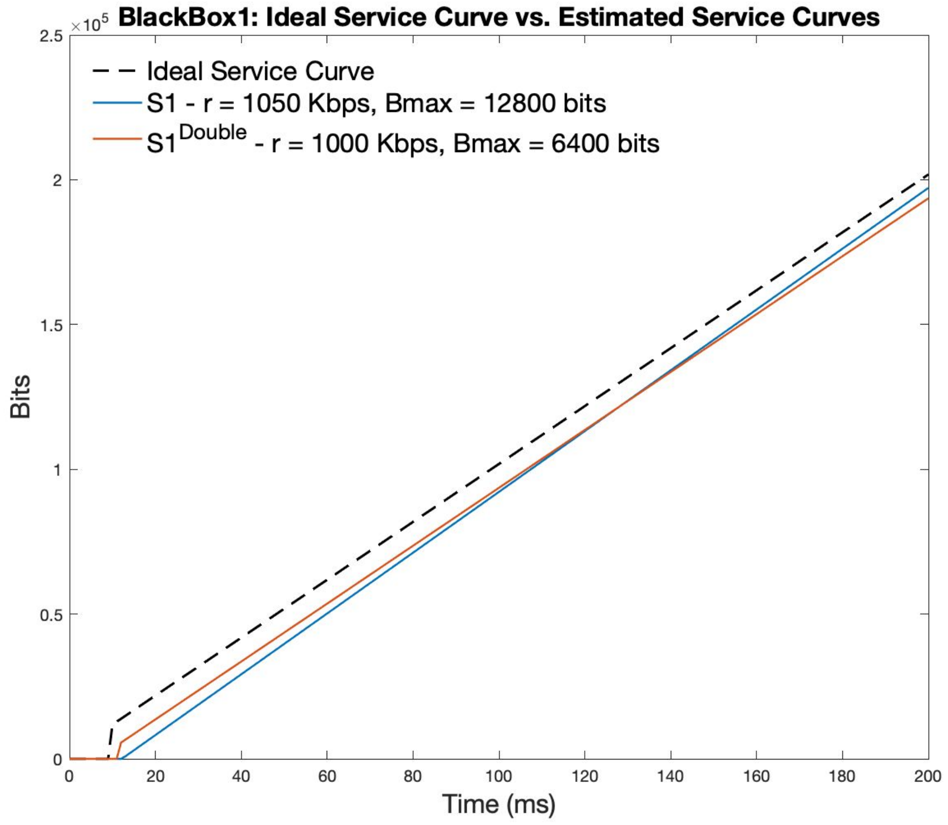


Figure 11: Ideal Service Curve Vs. Estimated Service Curves for BlackBox 1

Observations

For each of the two service curve estimates above we chose an initial packet arrival rate of r = 900 Kbps and rate increase of 50 Kbps . From figure 11, we see that the S1 estimate with N = 100 and L = 400 B, we achieve a service curve that when compared to the ideal service curve has a slightly more conservative delay, similar long term rate (i.e. 1050 Kbps) but with a much smaller burst size (i.e. $b_{estimate} < b = 11,840\ bits$). When repeating the experiments several more times using the same parameters, we often achieved the same final rate of 1050 Kbps (sometimes we achieved 1000 Kbps exactly) though with identical or slightly larger $B_{max}(r)$ values.

When we had doubled the packet train such that N = 200 and L = 800 Bytes we achieved the service curve estimate $S1^{Double}$ seen in figure 11 above. Observing the $S1^{Double}$ service curve estimate, we see that it has a delay and burst size closer to the ideal service curve than the S1 estimate has. In addition, the $S1^{Double}$ service estimate also has a long term rate equal to the ideal service curve (i.e. 1000 Kbps).

BlackBox 2: b=10,000 bits, R = 1 Mbps, T = 100,000 μs

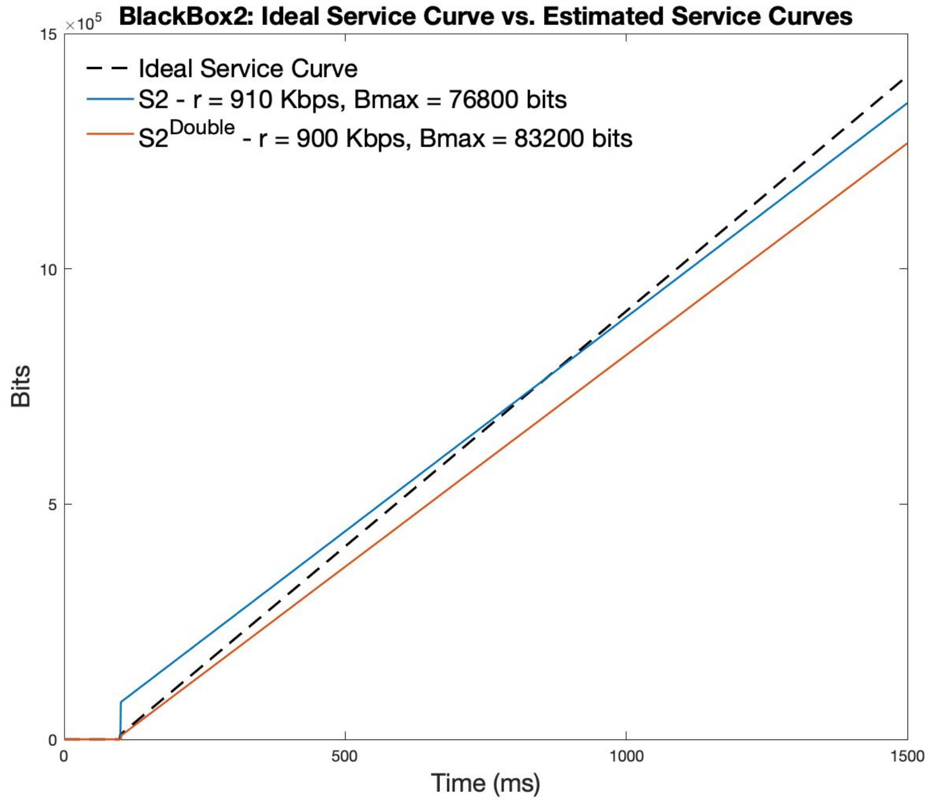


Figure 12: Ideal Service Curve Vs. Estimated Service Curves for BlackBox 2

Observations

For each of the two service curve estimates above we chose an initial packet arrival rate of r = 900 Kbps and rate increase of 10 Kbps. From figure 12, we observe that the service curve estimates with and without doubling the packet train both have long term rates that are less than the ideal (i.e. 1000 Kbps). Furthermore, for the service curve estimate S2, it appears to have a delay and burst nearly identical to that of the ideal service curve. As for the service curve estimate $S2^{Double}$, it has a much larger burst but comparatively similar delay to the ideal service curve.

Repeating the experiments also led to similar results where the algorithm often halts at the same rate but with varying $B_{max}(r)$.

We observed that the enormous delay incurred by the BlackBox resulted in large $B_{max}(r)$ values for each rate increase and thus led to service curve estimates that stopped improving (i.e. rt - $B_{max}(r)$ evaluated at t = 100 ms became smaller as $B_{max}(r)$ grew large). The effect of this was that our algorithm halted before reaching rates r that were much closer to the ideal rate.

BlackBox 3: b= 11,840 bits, R = 5 Mbps, T = 20,000 μs

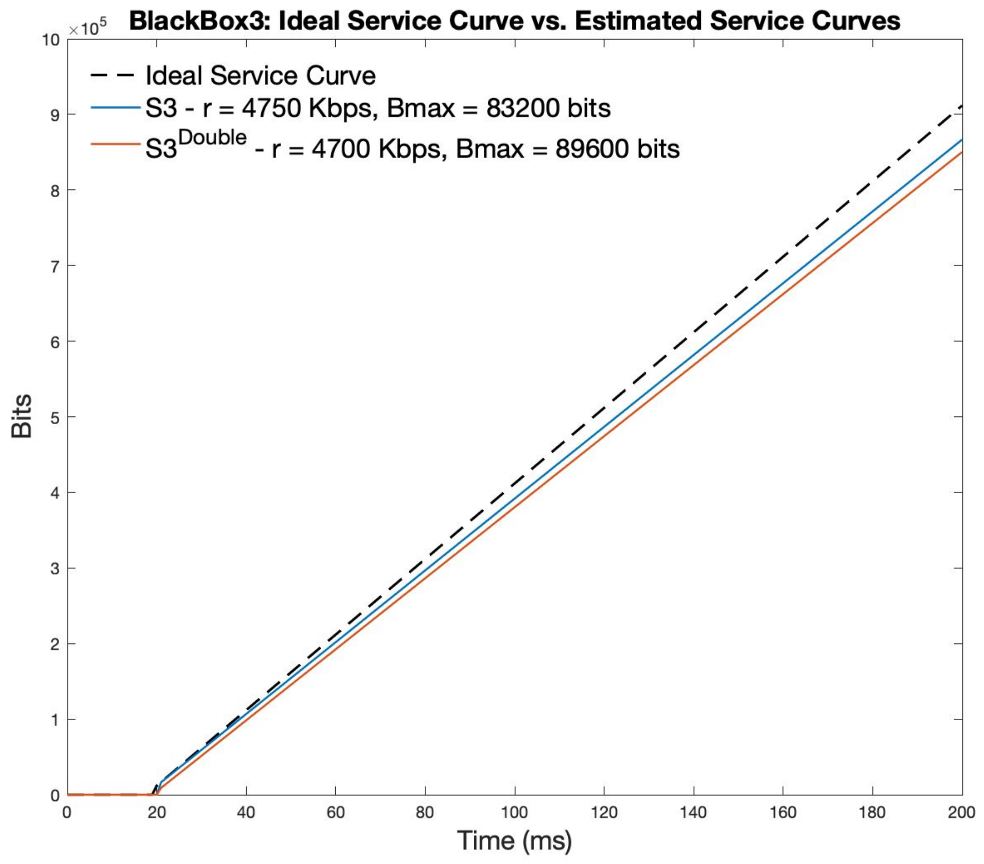


Figure 13: Ideal Service Curve Vs. Estimated Service Curves for BlackBox 3

Observations

For each of the two service curve estimates above we chose an initial packet arrival rate of r = 4500 Kbps and rate increases of 50 Kbps. From figure 13, we see that the service curve estimates for the regular packet train and doubled packet train provide very similar results where both have a burst size and delay nearly identical to that of the ideal service curve. We also see from figure 13 that the regular packet train led to a service curve estimate that has a rate closer to that of the ideal service curve (i.e. 4750 Kbps vs. 5000 Kbps) than that provided by the doubled packet train (i.e. 4700 Kbps vs. 5000 Kbps).

As with prior exercises, when repeating the experiments the rates often ended up being the same rate though with some variance in the final $B_{max}(r)$ values.

As to why the service curve estimates for the regular and doubled packet train produced nearly identical results, we believe this is because of the long term rate of the BlackBox which is much higher in this instance than the other BlackBox instances and also because of the small imposed delay. Ultimately, doubling the packet train would not necessarily lead to a large increase in $B_{max}(r)$ for each rate r as N and L are very small to begin with.

For Experiment 2.1, discuss when your methodology was able to find the correct parameters and when it had difficulties finding the correct parameters. Also, describe the impact of increasing the length of the packet trains and the size of the probe packets on the outcome of the measurements.

From the results obtained above we see that when the delay is very large the probing algorithm performed relatively poorly since the burst size and rate estimates were noticeably off from the ideal values (i.e. BlackBox 3 - see figure 13). Furthermore, when the delay is small and the long term rate is very large, we found that although the estimated burst sizes and delays were relatively accurate when compared to the ideal values, however, the estimated long term rates were somewhat inaccurate (i.e. BlackBox 2 - see figure 12). When the rate is modest and the delay is not too large, we found that the approximations given by our service curve estimates were fairly accurate (i.e. BlackBox 1 - see figure 11).

As for doubling the packet trains, in some instances it improved estimates as is the case for BlackBox 1 in figure 11, however, in other experiments it produced results ranging from being slightly to significantly inaccurate as was evident in BlackBox 2 and 3 presented in figures 12 and 13 respectively. Accordingly, we believe that when the BlackBox delay is not too large, doubling packet trains could help improve estimates for burst size and delay.

Exercise 2.2 Evaluation of BlackBox with unknown parameters

<u>For Experiment 2.2, provide a discussion of your estimates of the values for b, R, and T.</u>

<u>**Unknown BlackBox 1**</u>

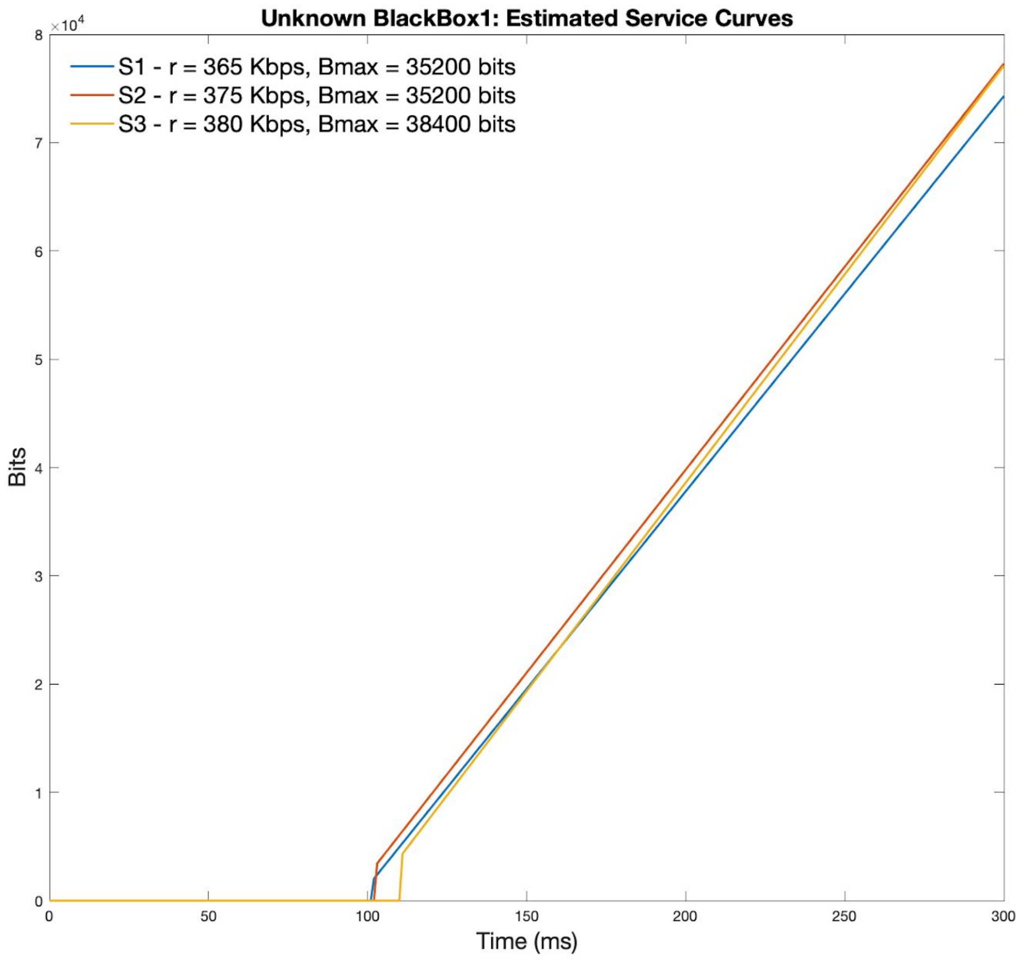


Figure 14: Service Curve Estimates of Unknown BlackBox 1

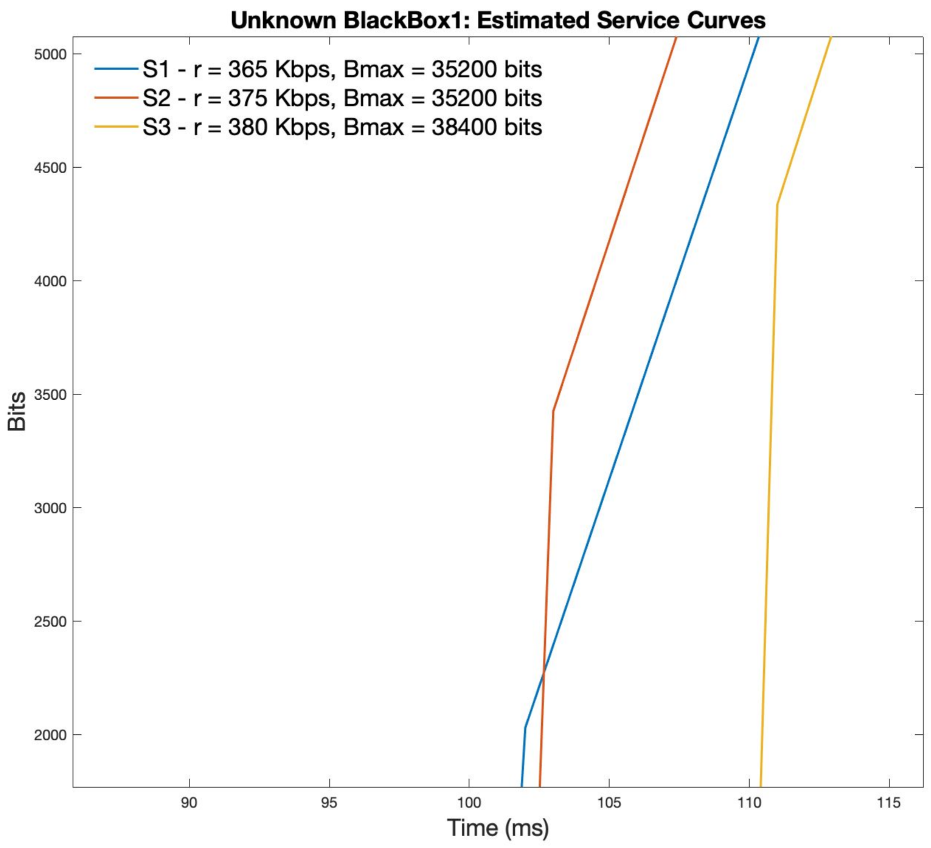


Figure 15: Service Curve Estimates of Unknown BlackBox 1 - Zoomed in

Discussion

For the packet trains sent to this BlackBox we used the same values for N and L used in section 2.1, that is, N = 100 and L = 400 Bytes (this was chosen arbitrarily). We used an initial rate of 300 Kbps and rate increments of 5 or 15 Kbps via trial and error. After running the probing algorithm several times we plotted the three best service curve estimates obtained in figures 14 and 15.

From figures 14 and 15, we estimate that the delay of the BlackBox is **T ≅ 100 ms**, since the delay of two of three service curve estimates hover around this value while the other service curve's delay is not too far off. We estimate that the burst size is **b ≅ 4400 bits**, where this value was chosen from the largest burst size seen amongst the three service curves. Finally, as two of the service curve estimates have nearly identical rates (i.e. 375 Kbps vs. 380 Kbps), we estimate that the long term rate of the BlackBox is **R ≅ 375 Kbps** (i.e. the lower of the two similar estimated rates).

Unknown BlackBox 2

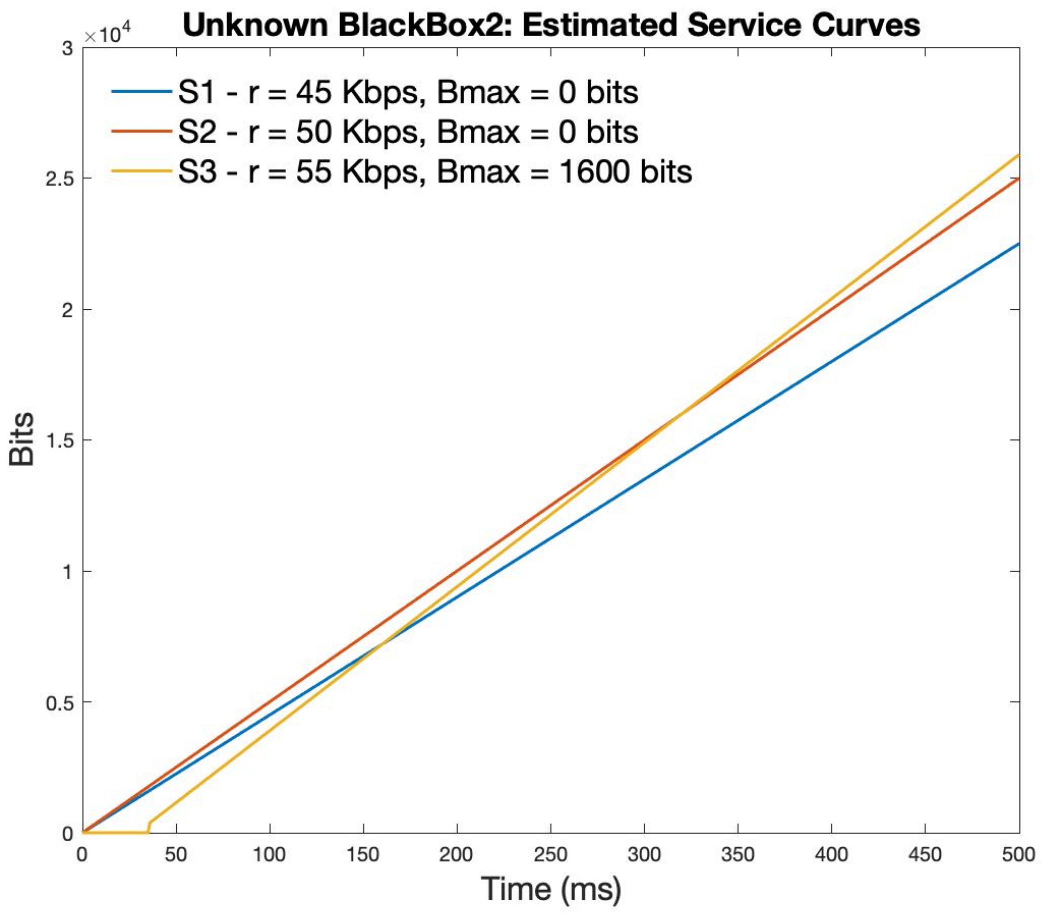


Figure 16: Service Curve Estimates of Unknown BlackBox 2

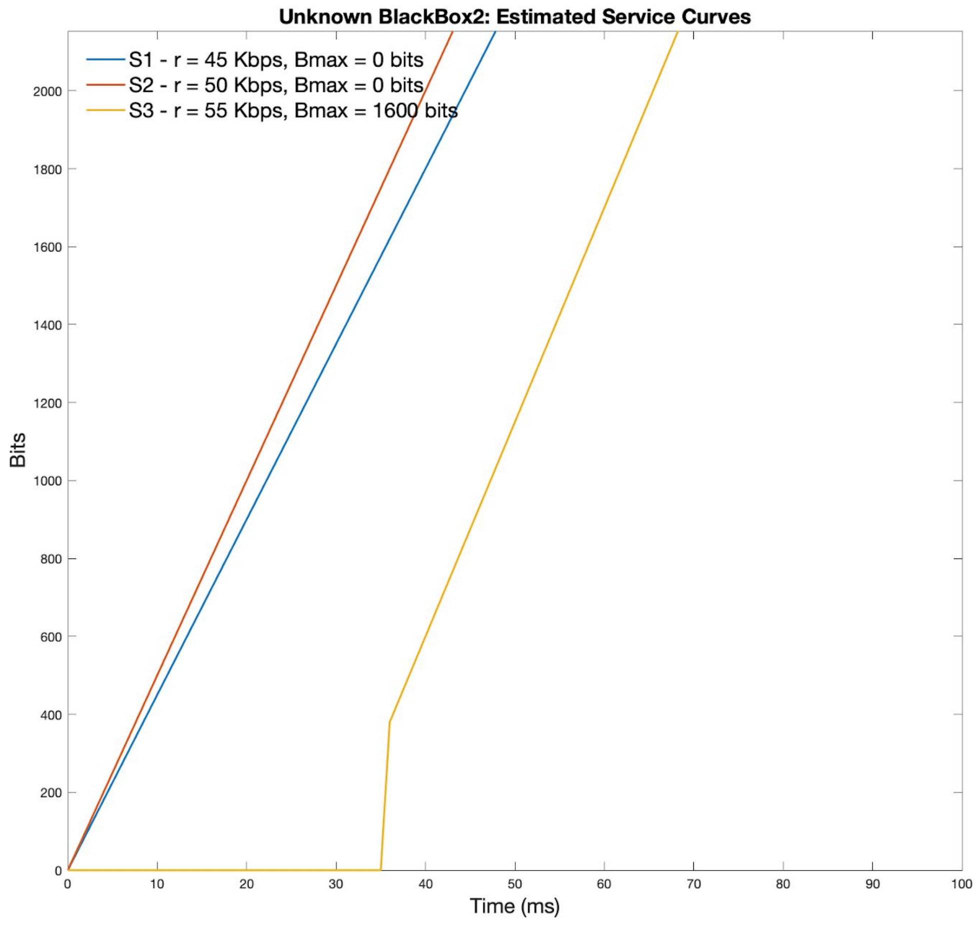


Figure 17: Service Curve Estimates of Unknown BlackBox 2 - Zoomed in

Discussion

For the packet trains sent to this BlackBox we used N = 100 (as before) and L = 100 B via trial and error. We used an initial rate of 10 Kbps and rate increments of 5 Kbps via trial and error. After running the probing algorithm several times we plotted the three best service curve estimates obtained in figures 16 and 17.

Observing figures 16 and 17, we estimate that the delay of the BlackBox is **T ≅ 0 ms**, since the delay of two of three service curve estimates have almost exactly this delay. As for the burst size, we estimate that it is **b ≅ 400 bits**, where this value was chosen from the largest burst size seen amongst the three service curves. Finally, as two of the service curve estimates are very close to one another and have similar rates (i.e. 50 Kbps vs. 55 Kbps), we estimate that the long term rate of the BlackBox is **R ≅ 50 Kbps** (i.e. the lower of the two similar estimated rates).

Unknown BlackBox 3

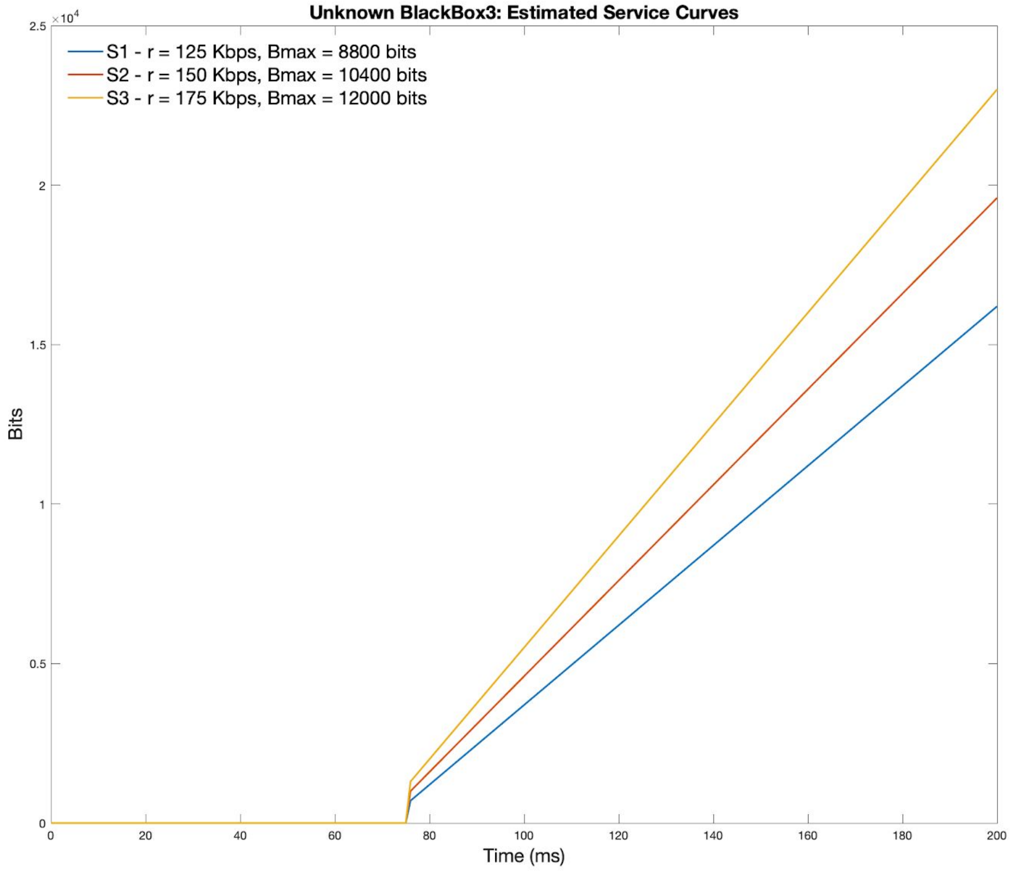


Figure 18: Service Curve Estimates of Unknown BlackBox 3

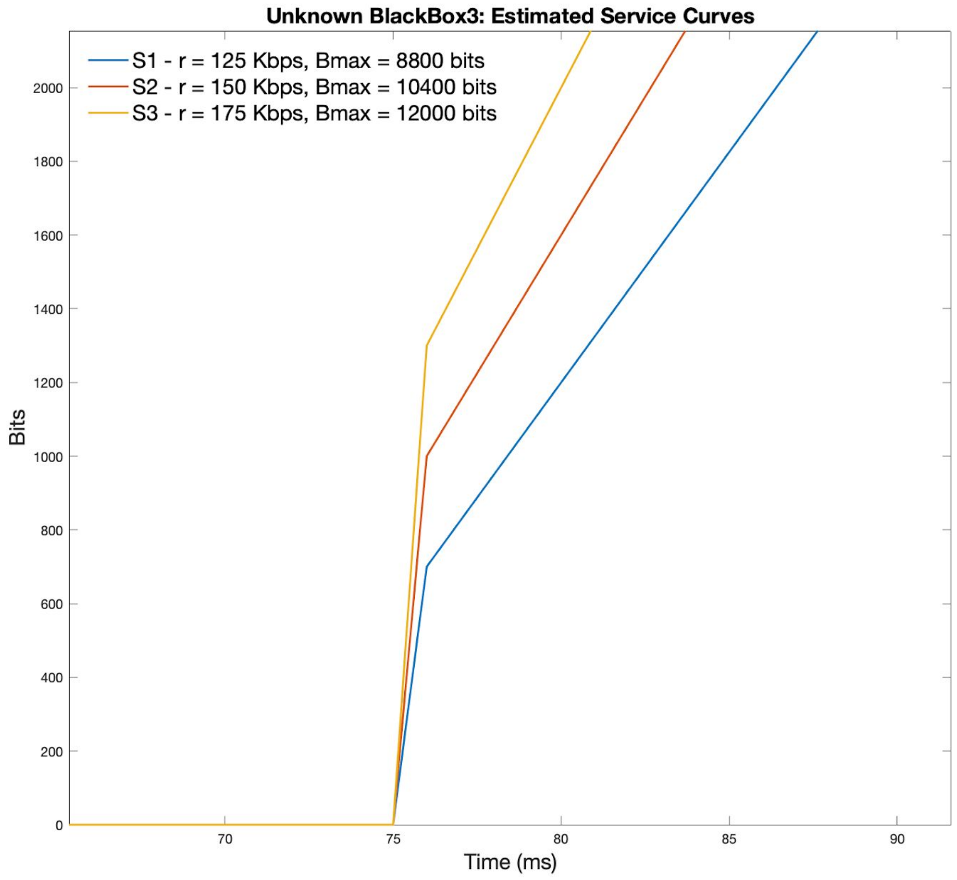


Figure 19: Service Curve Estimates of Unknown BlackBox 3 - Zoomed in

Discussion

For the packet trains sent to this BlackBox we used N = 100 and L = 100 B via trial and error. We used an initial rate of 100 Kbps and rate increments of 25 Kbps via trial and error. After running the probing algorithm several times we plotted the three best service curve estimates obtained in figures 18 and 19.

Observing figures 16 and 17, we estimate that the delay of the BlackBox is **T ≅ 75 ms**, since the delay of all three service curve estimates is almost exactly this delay. As for the burst size, we estimate that the burst size is **b ≅ 1300 bits**, where this value was chosen from the largest burst size seen amongst the three service curves. Finally, as two of the service curve estimates are fairly close to one another and have similar rates (i.e. 150 Kbps vs. 175 Kbps), we estimate that the long term rate of the BlackBox is **R ≅ 150 Kbps** (i.e. the lower of the two similar estimated rates).