Bangladesh University of Engineering and Technology
Department of Computer Science and Engineering

CSE322 - Computer Networks Sessional

Report on

CURVILINEAR RED : AN IMPROVED RED ALGORITHM FOR INTERNET ROUTERS

Submitted by:
1705065 - Taseen Mubassira

Paper by:
Ayodeji Oluwatope - Obafemi Awolowo University
Samuel Hassan - Olabisi Onabanjo University
Conference and Date:
Conference : iasted
Date : September,2014

February 23, 2022

Contents

1 Introduction

This project is about modifying the existing RED algorithm for active queue management in Network Simulator 3. Random Early Detection (RED), an active queue management (AQM) scheme has been known to address the problem of network congestion in Internet routers. However,RED suffers from large delay which can be partly traced to the single linear packet drop function it deploys.The existing RED implementations can be upgraded/replaced with the proposed CLRED scheme so as to help RED overcome its large delay drawback. The paper that I've followed is this.

2 Network Topologies Under Simulation

The topology I followed for this simulation is a **Dumbbell** topology (as per paper) and As per assignment I've to simulate this topology for a *wired connection* and for *wireless low rate mobile connection* in my network.

2.1 Wired Connection

I've simulated Task A ad Task B in this network(As per my paper CLRED works for wired connections). Properties of my topology are:

- A dumbbell topology
- PointToPoint connection for the nodes and router
- Two routers form bottleneck link
- N nodes on the left side(N can be varied)
- 1 node on the right side

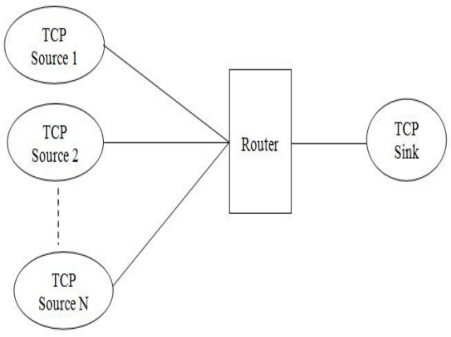

Figure 1: Wired Connection proposed in paper

Figure 2: Wired Connection in my topology

2.2 Wireless Low Rate Connection Mobile

I've simulated Task A in this connection. Properties of my topology are:

- N wifi Nodes (N can be varied)
- 2 Csma Nodes
- UDP packets were sent
- Default red-queue-disc simulation was done

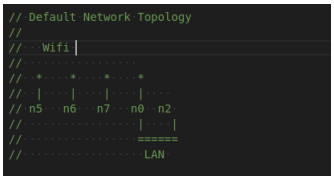

Figure 3: Wireless Connection in my topology

3 Variation of Parameters

The number of nodes needed to be varied is from 20 to 100. The other four parameters needed to be varied are flow count,number of packets per second,speed of nodes(for mobility) and coverage area(for static nodes).I've varied my parameters according to my network.

3.1 For Wired

- Number of Nodes (from 20 to 100)
- Number of flows (from 10 to 50)
- Number of packets per second (from 100 to 500)

3.2 For Wireless Low Rate Mobile

- Number of Nodes (from 20 to 100)
- Number of flows (from 10 to 50)
- Number of packets per second (from 100 to 500)
- Speed of Nodes (from 5m/s to 25m/s)

4 Overview of Proposed Algorithm

Curvilinear Random Early Detection (CLRED) is a modified scheme of existing RED which modified the single linear dropping function of RED with *a two-segment (that is, a quadratic and a linear) dropping functions* to address the large delay which can be partly traced to the *single linear* packet drop function that RED deploys.Therefore, the existing RED implementations can be upgraded/replaced with the proposed CLRED scheme so as to help RED overcome its large delay drawback.

4.1 Random Early Detection : RED

The operation of the RED scheme is quite simple but profound. For each new packet that arrives the gateway, RED computes the average queue size (avg) (meaning, the average number of packets in the buffer of the router) and compares the result with two preset thresholds: the minimum threshold (Min_{th}) and the maximum threshold (Max_{th}).

If the calculated average queue size is found to be less than the Min_{th}, then the packet is dropped with prob- ability 0 (that is, allowed into the gateways buffer). If the calculated average queue size is found to exceed the Max_{th}, then the packet is dropped with probability 1. However, if the average queue size is found to be a value between the Min_{th} and Max_{th} , then the packet is dropped linearly from 0 to Max_p with probability P_a. The dropping probability function of RED is given below -

$$P_d(avg) = \begin{cases} 0 & avg < Min_{th} \\ Max_p(\frac{avg - Min_{th}}{Max_{th} - Min_{th}}) & Min_{th} \leq avg < Max_{th} \\ 1 & Max_{th} \leq avg \end{cases} \quad (4)$$

Figure 4: Dropping Function of RED

4.2 Curvilinear Random Early Detection : CLRED

For each packet arrival, CLRED computes the average queue size (avg) just like RED but the diffference is it uses two minimum threshold points,Min_{th1} and Min_{th2}. The dropping Function for CLRED is:

$$P_d(avg) = \begin{cases} 0, & avg < Min1_{th}, \quad P_b = 4(1 \\ 4(1 - Max_p)(\frac{avg - Min1_{th}}{Max_{th} - Min1_{th}})^2, & Min1_{th} \leq avg < Min2_{th}, \\ (1 - Max_p) + 2Max_p(\frac{avg - Min2_{th}}{Max_{th} - Min1_{th}}), & Min2_{th} \leq avg < Max_{th}, \\ 1, & Max_{th} \leq avg. \end{cases}$$

Figure 5: Dropping Function of CLRED

Where, Min_{th1} is the first minimum threshold (same as Min_{th1} for RED); Max_{th} is the maximum threshold (same as Max_{th} of RED); and Min_{th2} is the mid-point threshold between Min_{th1} and Max_{th}.

Pseudocode for CLRED Algorithm:

Initialization:
$avg \leftarrow 0$
$count \leftarrow -1$
For each packet arrival,
Calculate the average queue size avg
if the buffer of the router is non-empty **then**
    $avg \leftarrow ((1 - W_q) \times avg') + (W_q \times q)$
else $m \leftarrow f(time - q\_idle\_time)$
    $avg \leftarrow ((1 - W_q)^m \times avg')$
end if
if $avg < Min1_{th}$ **then**
    No packet drop
    Set $count \leftarrow -1$
else if $Min1_{th} \leq avg < Min2_{th}$ **then**
    Set $count \leftarrow count + 1$
    Calculate the packet drop probability P_a
    $P_b \leftarrow 4(1 - Max_p) \times ((avg - Min1_{th})/(Max_{th} - Min1_{th}))^2$
    $P_a \leftarrow P_b/(1 - count.P_b)$
    mark the arriving packet with probability P_a:
    $count \leftarrow 0$
    Drop the packet

Figure 6: Pseudocode for CLRED (contd)

else if $Min2_{th} \leq avg < Max_{th}$ **then**
Set $count \leftarrow count + 1$
Calculate the packet drop probability P_a
$P_b \leftarrow (1 - Max_p) + 2Max_p((avg - Min2_{th})/(Max_{th} - Min1_{th}))$
$P_a \leftarrow P_b/(1 - count.P_b)$
mark the arriving packet with probability P_a:
Set $count \leftarrow 0$
Drop the packet
else if $Max_{th} \leq avg$ **then**
Drop the arriving packet
Set $count \leftarrow 0$
else $count \leftarrow -1$
when the buffer of the router becomes empty
Set $q\_idle\_time \leftarrow time$
end if

Figure 7: Pseudocode for CLRED

So,what basically the CLRED Algorithm does is :

- if avg is found to be value between 0 and Min_{th1} , then the packet will not be dropped.
- if avg is found to be a value between Min_{th1} and Min_{th2} , then the packet is dropped with probability P_a.

That is, $P_a = P_b/(1 - count.P_b)$

$$P_b = 4(1 - Max_p)(\frac{avg - Min1_{th}}{Max_{th} - Min1_{th}})^2$$

- similarly, if avg is found to be a value between Min_{th2} and Max_{th} , then the packet is dropped with probability P_a (as in ii.). However Now the calculation for P_b will be changed.

$$P_b = (1 - Max_p) + 2Max_p(\frac{avg - Min2_{th}}{Max_{th} - Min1_{th}})$$

- Lastly, if avg is found to exceed Max_{th},then the packet will be dropped.

4.3 Comaprison Between RED and CLRED

Dropping Function for RED :

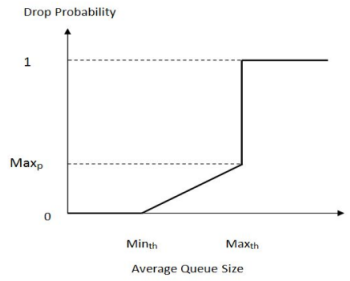

Figure 8: RED Dropping Function

Dropping Function for CLRED :

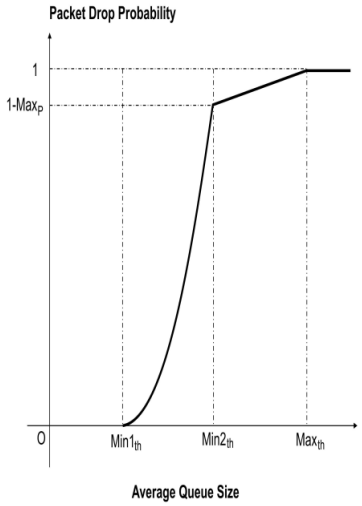

Figure 9: CLRED Dropping Function

5 Modifications in Simulator to implement Algorithm

Three new Variables are declared in red-queue-disc.h header file.

```
bool m_isCLRED;           //!< True to enable curvilinear RED
double m_minTh2;          //!< Minimum threshold2 for m_qAvg (bytes or packets)
int n_CLRED;              //!<parameters for calculating pb
```

Figure 10: New Variables in red-queue-disc.h

Here, `m_isCLRED` is a boolean to check if CLRED is being run, `m_minTh2` is the second threshold value and `n_CLRED` is a integer used as a parameter to calculate the value of P_b for different scenarios.

Again,3 new functions were declared inside red-queue-disc.h header file.

```
/**
 * \brief Set the minTh2 value to CLRED.
 *
 *
 */
void SetCLREDMinTh2 (double minTh2);

/**
 * \brief Get the minTh2 value to CLRED.
 *
 *
 */
double GetCLREDMinTh2 (void);
```

Figure 11: New Functions in red-queue-disc.h

These two functions are used like setter and getter of the new variable `m_minTh2`

```
/**
 * \brief Returns a probability  for CLRED using these function parameters for the DropEarly function
 * \returns Prob. of packet drop before "count"
 */
double CalculatePbCLRED (int n,double m_qAvg);
```

Figure 12: New Function in red-queue-disc.h

This Function is used for calculating the value of P_b for different scenarios with different calculations that is when avg is found to be a value between Min_{th1} and Min_{th2} and when avg is found to be a value between Min_{th2} and Max_{th}.

Now,in red-queue-disc.cc I've added two attributes CLRED and MinTh2 as previously mentioned, these were also added in the header file.

```
.AddAttribute ("CLRED",
               "True to enable CLRED",
               BooleanValue (false),
               MakeBooleanAccessor (&RedQueueDisc::m_isCLRED),
               MakeBooleanChecker ())
```

Figure 13: New attribute in red-queue-disc.cc

```
.AddAttribute ("MinTh2",
               "Minimum Threshhold 2 for CLRED",
               DoubleValue (10),
               MakeDoubleAccessor (&RedQueueDisc::m_minTh2),
               MakeDoubleChecker<double> ())
```

Figure 14: New attribute in red-queue-disc.cc

As previously mentioned,I've declared the setter and getter for MinTh2 in the header file,now I've defined them in the cc file.

```
void
RedQueueDisc::SetCLREDMinTh2 (double minTh2)
{
  NS_LOG_FUNCTION (this << minTh2);
  m_minTh2 = minTh2;
}

double
RedQueueDisc::GetCLREDMinTh2 (void)
{
  NS_LOG_FUNCTION (this);
  return m_minTh2;
}
```

Figure 15: Setter and Getter for MinTh2 in red-queue-disc.cc

In the function InitializeParams I've initialised my added variables and set count for -1 as my paper suggests.

```
//added
if (m_isCLRED)
  {
    // Set m_minTh, m_maxTh and m_qW to zero for automatic setting
    m_minTh = 0;
    m_minTh2 = 0;
    m_maxTh = 0;
    m_qW = 0;
    n_CLRED = 0;
  }
```

Figure 16: Inside InitializeParams red-queue-disc.cc

```
//added
if(m_isCLRED){
  m_count = -1;
}
```

Figure 17: Setting count in InitializeParams in red-queue-disc.cc

Now,In the DoEnqueue Function I've done the following changes according to my CLRED Algorithm.

```
if(m_isCLRED){
  NS_LOG_DEBUG ("In CLRED");

    if(m_qAvg < m_minTh){
      //no packet drop
      m_count = -1;
    }
    else if(m_minTh <= m_qAvg && m_qAvg < m_minTh2){
        n_CLRED = 1;
        if(DropEarly(item,nQueued)){
            NS_LOG_LOGIC ("DropEarly returns 1");
            dropType = DTYPE_UNFORCED;
        }
    }
    else if(m_minTh2 <= m_qAvg && m_qAvg < m_maxTh){
        n_CLRED = 2;
        if(DropEarly(item,nQueued)){
            NS_LOG_LOGIC ("DropEarly returns 1");
            dropType = DTYPE_UNFORCED;
        }
    }
    else{
        NS_LOG_DEBUG ("adding DROP FORCED MARK");
        dropType = DTYPE_FORCED;
    }
}
```

Figure 18: Modifications in DoEnqueue function in red-queue-disc.cc

I've added a new function CalculatePbCLRED in red-queue-disc.cc and this Function is used for calculating the value of P_b for different scenarios with different calculations that is when avg is found to be a value between Min_{th1} and Min_{th2} and when avg is found to be a value between Min_{th2} and Max_{th}.

```
//added
// Returns a probability for CLRED using these function parameters for the DropEarly function
double
RedQueueDisc::CalculatePbCLRED (int n,double m_qAvg)
{
  NS_LOG_FUNCTION (this);
  double pb;

  //between min 1 and 2
  if( n == 1){
    pb = 4 * (1 - m_curMaxP) * ((m_qAvg - m_minTh) / (m_maxTh - m_minTh)) * ((m_qAvg - m_minTh) / (m_maxTh - m_minTh));
  }

  //between min2 and max
  else{
    pb = (1 - m_curMaxP) + (2 * m_curMaxP * ((m_qAvg-m_minTh2) / (m_maxTh-m_minTh2)));
  }

  return pb;
}
```

Figure 19: CalculatePbCLRED function in red-queue-disc.cc

Now, In the drop Early function this CalculatePbCLRED is called to calculate P_b when CLRED is set.

```
if(m_isCLRED){
  prob1 = CalculatePbCLRED(n_CLRED,m_qAvg);
}
```

Figure 20: Modifications in DropEarly function in red-queue-disc.cc

6 Results and Explanation of Task A

With no modifications in the RED algorithm simulations were run for wired and wireless connection

6.1 Wired connection

For wired connection, the graphs are plotted with CLRED in task B for better visualization of Modification.

6.2 Wireless Low Rate Mobile Connection

For varying Nodes in Wifi:

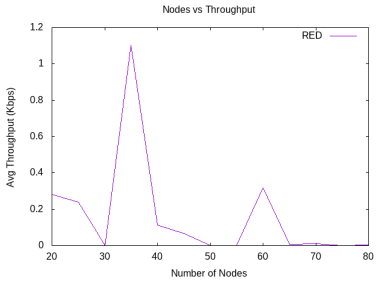

Figure 21: Wireless Nodes vs Throughput

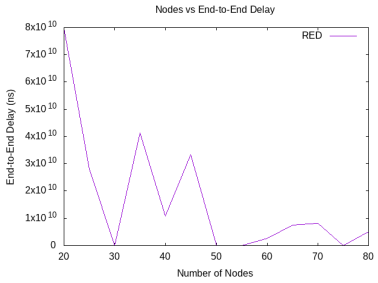

Figure 22: Wireless Nodes vs End-to-End Delay

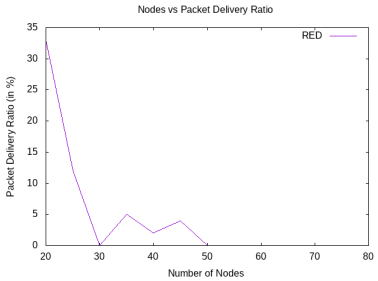

Figure 23: Wireless Nodes vs Packet Delivery Ratio

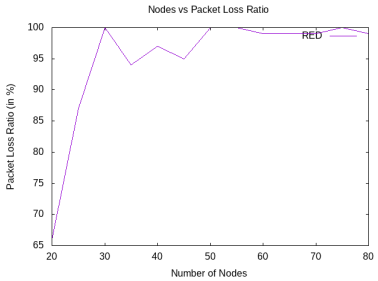

Figure 24: Wireless Nodes vs Packet Loss Ratio

For varying Speed of Nodes:

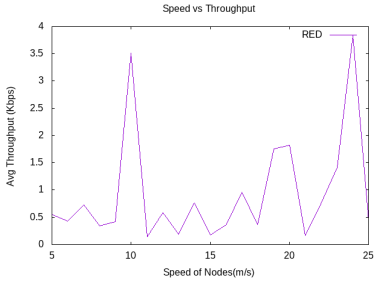

Figure 25: Speed vs Throughput

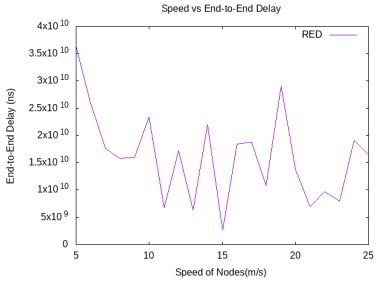

Figure 26: Speed vs End-to-End Delay

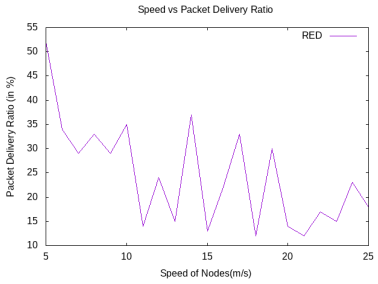

Figure 27: Wireless Speed vs Packet Delivery Ratio

Figure 28: Wireless Speed vs Packet Loss Ratio

For varying interval of packets:

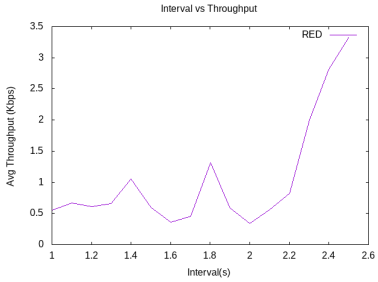

Figure 29: Interval vs Throughput

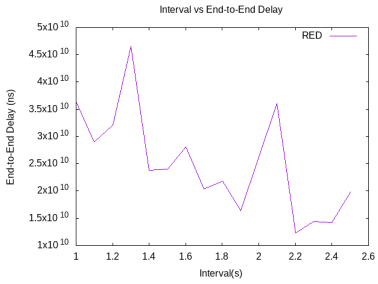

Figure 30: Interval vs End-to-End Delay

Figure 31: Wireless interval vs Packet Delivery Ratio

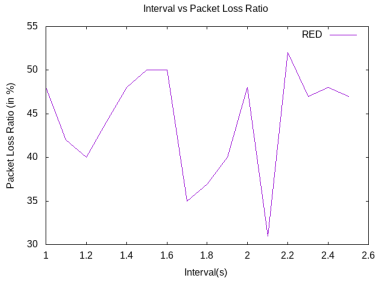

Figure 32: Wireless interval vs Packet Loss Ratio

Explanation:
In this simulation we can see,the packet loss is huge because the lrw-pan in ns3 is under developed and as my network was in low rate and also mobile so packet delivery was inconsistent and thus monotonous graphs were not observed.

7 Results and Explanation of Task B

In this task,I've added two graphs together for better comparison in the metrics. I've varied number of nodes,flows and packets per second and this simulation is done in wired connection.

For varying Nodes from 20 to 100:

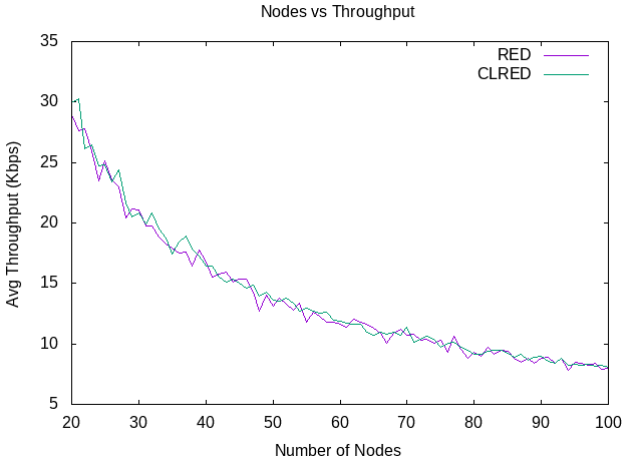

Figure 33: Nodes vs Throughput

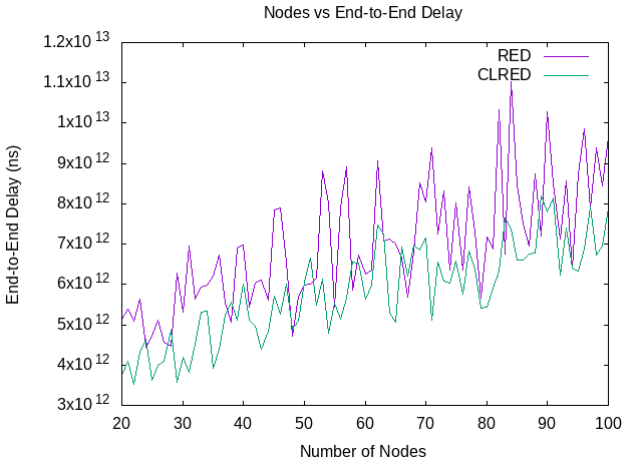

Figure 34: Nodes vs End-to-End Delay

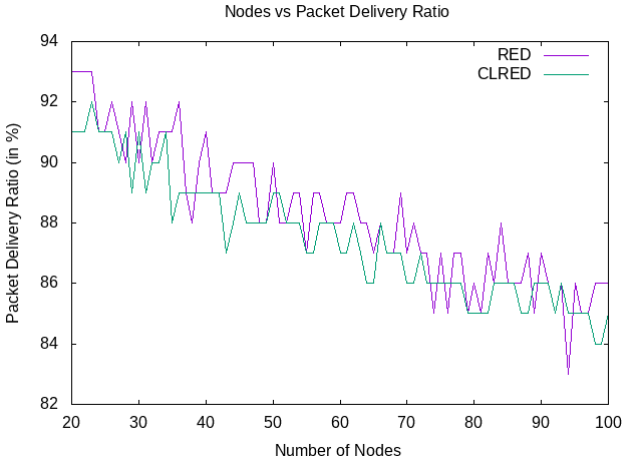

Figure 35: Nodes vs Packet Delivery Ratio

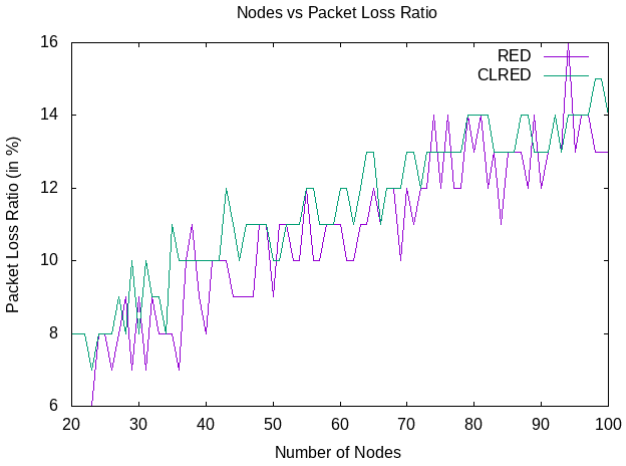

Figure 36: Nodes vs Packet Loss Ratio

For varying Flows from 10 to 50:

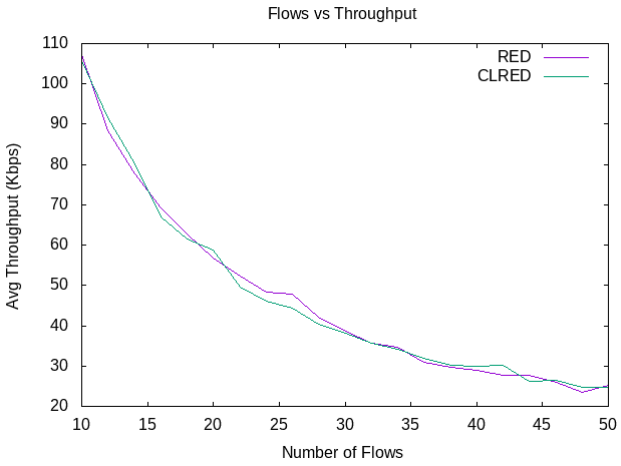

Figure 37: Flows vs Throughput

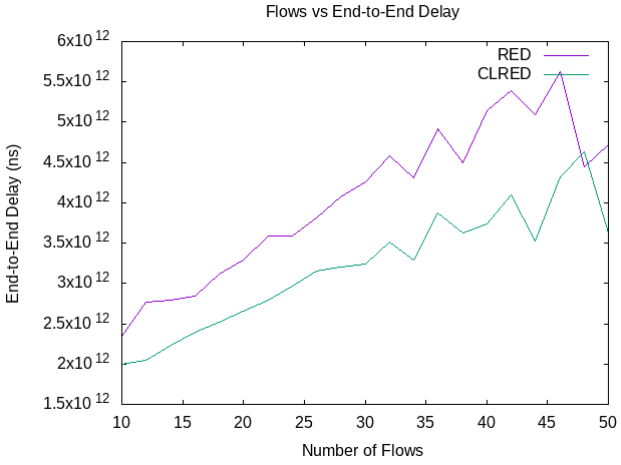

Figure 38: Flows vs End-to-End Delay

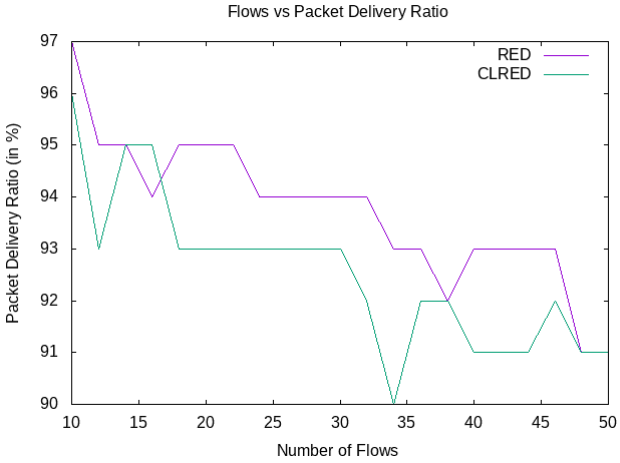

Figure 39: Flows vs Packet Delivery Ratio

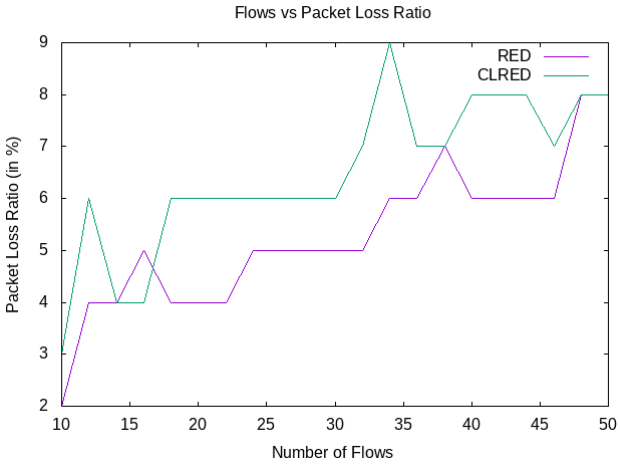

Figure 40: Flows vs Packet Loss Ratio

For varying Packets per second from 100m/s to 500m/s:

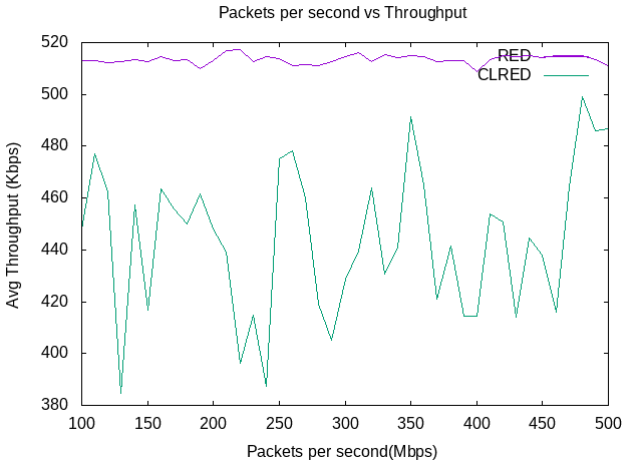

Figure 41: Packets per second vs Throughput

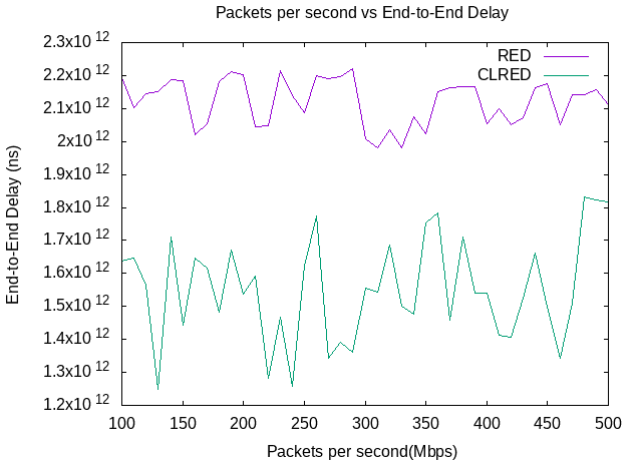

Figure 42: Packets per second vs End-to-End Delay

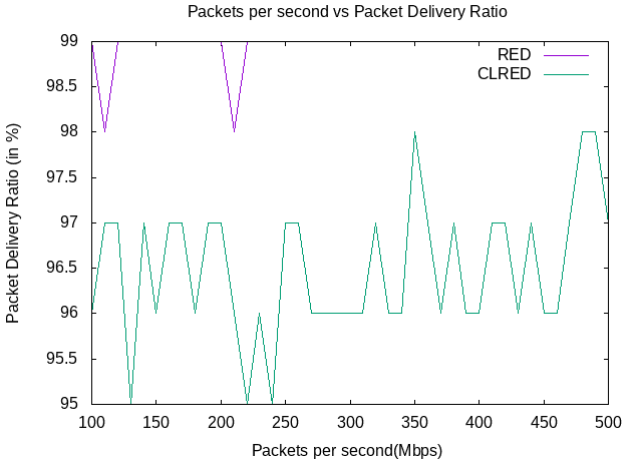

Figure 43: Packets per second vs Packet Delivery Ratio

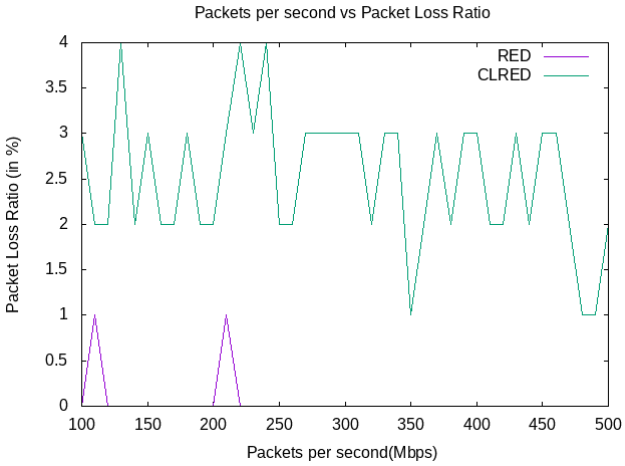

Figure 44: Packets per second vs Packet Loss Ratio

Explanation:
For all those varied parameters we can see the End-to-End Delay in CLRED is always less than actual RED which my paper proposed.So, the modifications are successful.