CSE 322 : Computer Networks Sessional
NS3 Project Report

Project Supervisor:
Md. Toufikuzzaman
Lecturer, Department of CSE
BUET

Std Name : Nuraiyad Nafiz Islam
Std Id: 1705114
L/T : 3-2
Department of CSE
BUET

Network Topologies :

For both the tasks (A and B) I used dumbbell topology. In task A, I was assigned to work on wireless network, so I used a wireless dumbbell topology. For task B, the paper I was working on had a wired setup. So I followed their network topology.

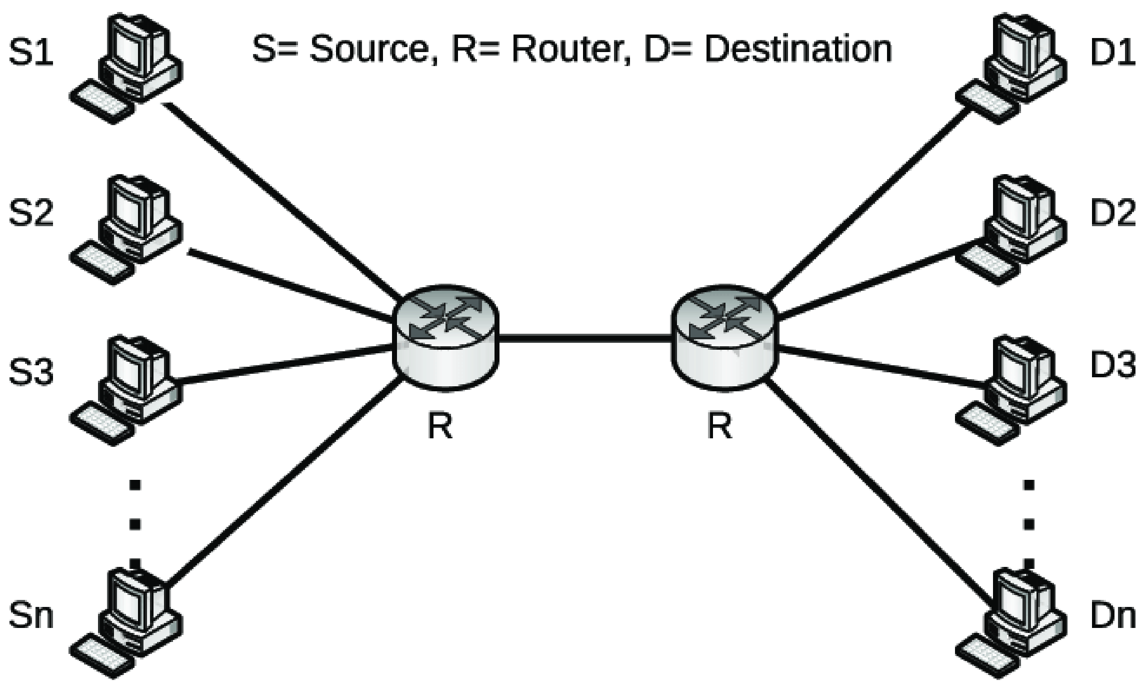


Fig 1 : Dumbbell Topology

Details of topology :

In a dumbbell topology there are three networks. For both wireless(task A) and wired(task B) Left side nodes are in network 10.1.2.0, 255.255.255.0 And right side nodes are in 10.1.3.0, 255.255.255.0. And the routers are connected in 10.1.1.0 network.

Variation of parameters :

Task A :

- Number of Nodes : 20,40,60,80,100
- Number of Flows : 10,20,30,40,50
- Packets per second : 100,200,300,400,500
- Coverage : Tx_range = 2
2xTx_range
3xTx_range
4xTx_range
5xTx_range

Task B :

- Number of Flows : 5,15,25,35,45,55,65,75,85,95

Overview of proposed algorithm :

RED(Random Early Detection) is a classic algorithm to ease congestion and then to improve the quality of service of the network.When average queue size is near to the minimum and maximum threshold, the loss rate is unreasonable. To improve the performance of linear approach of RED, a nonlinear approach is proposed in the paper.

In the proposed Flexible RED (FXRED), the router's queue with finite capacity is divided into four segments (A, B, C and D) via threshold values minth, Δ and maxth where Δ = 0.5 (minth + maxth)

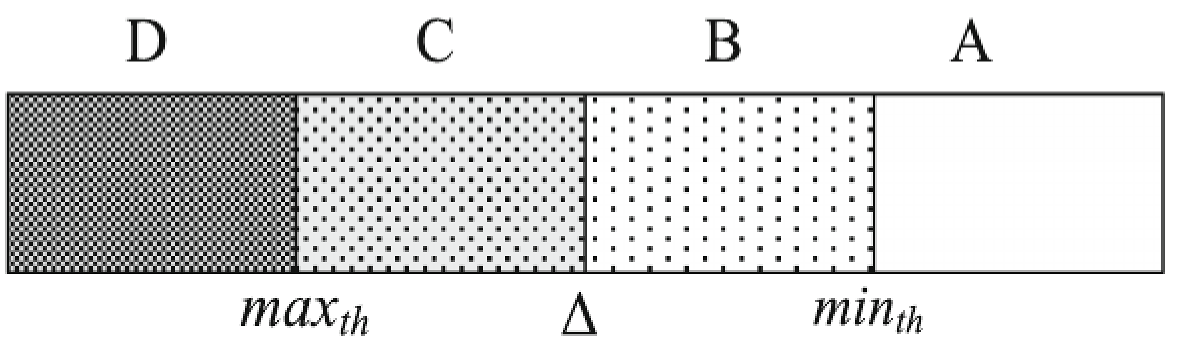


Fig 2 : FXRED Queue

Just like in traditional RED and its other variants, in FXRED, average queue length (avg) is computed,
avg = (1 − w)avg' + wq(t)
λ(t) is total data arrival rate, μ the bandwidth of the bottleneck link and the traffic load at time
t is denoted by ρ(t) and expressed by
ρ(t) = λ(t)/μ
Based on traffic load three states are

State-1: $\rho(t) < 1,$
State-2: $\rho(t) \approx 1,$
State-3: $\rho(t) > 1.$

Probability function :

$$
p_d = \begin{cases} 0 & avg < min_{th}, \\ 2^{\lfloor k \rfloor} \left(\frac{avg - min_{th}}{max_{th} - min_{th}} \right)^{\lfloor k \rfloor} \cdot (1 - \epsilon) & min_{th} \le avg < \Delta, \\ 2\epsilon \left(\frac{avg - \Delta}{max_{th} - min_{th}} \right) + (1 - \epsilon) & \Delta \le avg < max_{th}, \\ 1 & avg \ge max_{th}. \end{cases}
$$

$$
k = c^{\frac{1}{\gamma}}, c \ge 2. \qquad \begin{cases} \gamma < 1 & \text{for } \textbf{State-1}, \\ \gamma \approx 1 & \text{for } \textbf{State-2}, \\ \gamma \ge c & \text{for } \textbf{State-3}. \end{cases}
$$

$$
\epsilon = c^{-\gamma}.
$$

Modification made in the simulator :

Two source files of traffic control models are modified.
One is red-queue-disc.cc and other one is red-queue-disc.h

In red-queue-disc.cc the modification is done the RedQueueDisc::calculatePnew() function.

Added attributes(red-queue-disc.cc) :

```
.AddAttribute ("Miu",
               "Miu bottleneck",
               DoubleValue (5),
               MakeDoubleAccessor (&RedQueueDisc::d_miu),
               MakeDoubleChecker<double> ())
.AddAttribute ("C",
               "Constant C",
               DoubleValue (2),
               MakeDoubleAccessor (&RedQueueDisc::c),
               MakeDoubleChecker<double> ())
.AddAttribute ("Gamma",
               "Constant Gamma",
               DoubleValue (1),
               MakeDoubleAccessor (&RedQueueDisc::gamma),
               MakeDoubleChecker<double> ())
```

```
    .AddAttribute ("FXRED",
                   "True to enable Nonlinear RED",
                   BooleanValue (false),
                   MakeBooleanAccessor (&RedQueueDisc::m_isNonlinear),
                   MakeBooleanChecker ())
```

Algorithm implementation in red-queue-disc.cc :

```
void
RedQueueDisc::InitializeParams (void)
{
  NS_LOG_FUNCTION (this);
  NS_LOG_INFO ("Initializing RED params.");

  m_wallClock.Start();
```

```
  m_wallClock.End();
   //std::cout<<"bytes" + m_countBytes*1000<<std::endl;
  //std::cout<<"Megabytes" + m_countBytes/1000000<<std::endl;
  //std::cout<<(m_countBytes/m_wallClock.GetElapsedUser())*0.001<<std::endl;

  double lambda = (m_countBytes/m_wallClock.GetElapsedUser())*0.001;
  d_rho = lambda/d_miu;
  //std::cout<<"lambda" << lambda<<std::endl;
  //std::cout<<"d_rho-->" << d_rho<<std::endl;
  //m_wallClock.Start();
  if(d_rho<0.75)
  {
    gamma = 0.25;
  }
  else if(d_rho>=0.75)
  {
    gamma = 1;
  }
  else
  {
    gamma = c;
  }
  //std::cout<<"gamma-->" << (1/gamma)<<std::endl;
  k = pow(c,(1/gamma));
  //std::cout<<"k-->" << k<<std::endl;
  epsilon = pow(c,-1*gamma);
  //std::cout<<"epsilon-->" << epsilon<<std::endl;

```

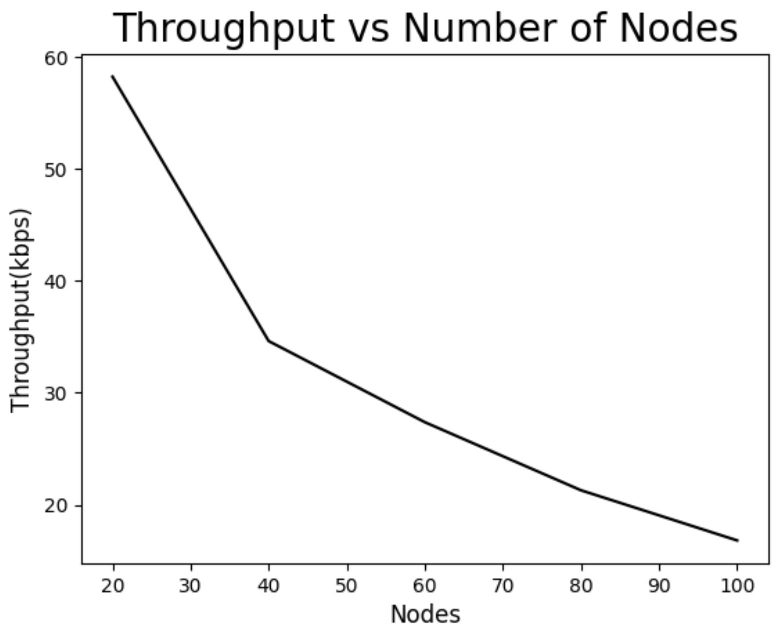
```
else
  {
    if(m_minTh <= m_qAvg && m_qAvg < m_delta)
    {
      p = m_vA * m_qAvg + m_vB;
      p = pow(2,k) * pow(p,k) *(1-epsilon);
    }

    else if(m_delta <= m_qAvg && m_qAvg < m_maxTh)
    {
      p = m_vA * m_qAvg - m_delta*m_vA;
      p = (2 * epsilon * p) + 1 - epsilon;
    }

    else if(m_qAvg >= m_maxTh)
    {
      p = 1.0;
    }
  }

```

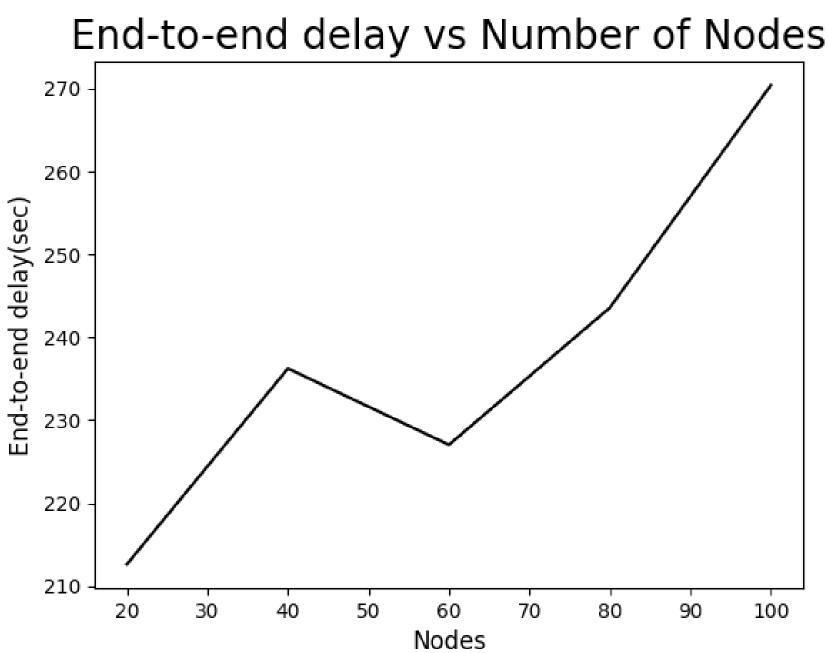
A program is written in scratch folder to simulate the modified algorithm. Name of the file is red-vs-fxred.cc.

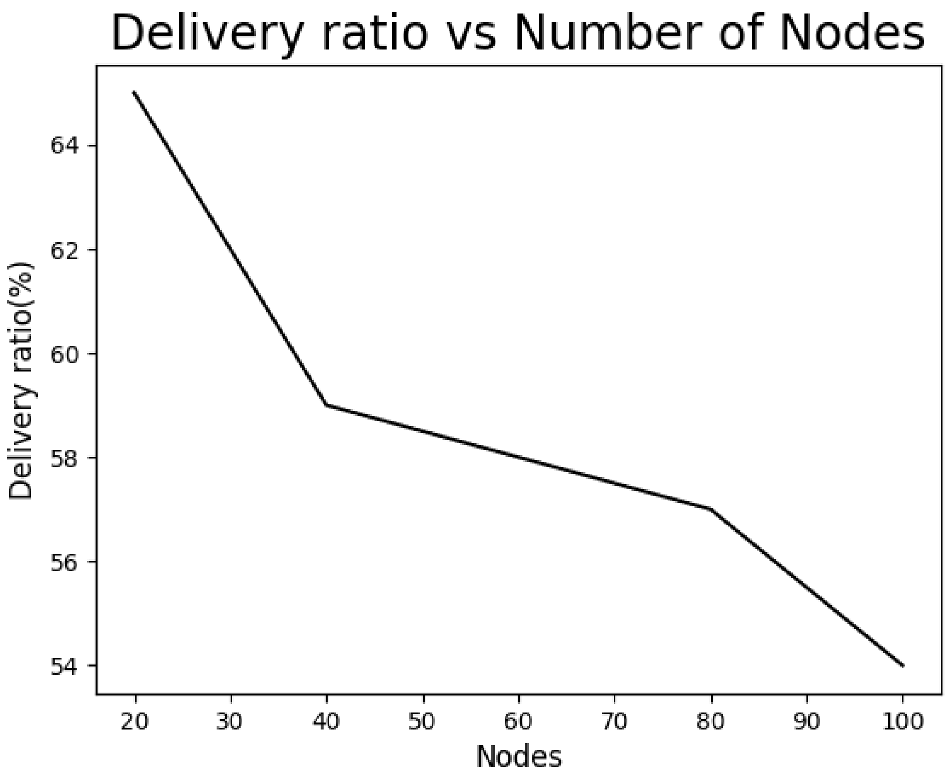
Results :

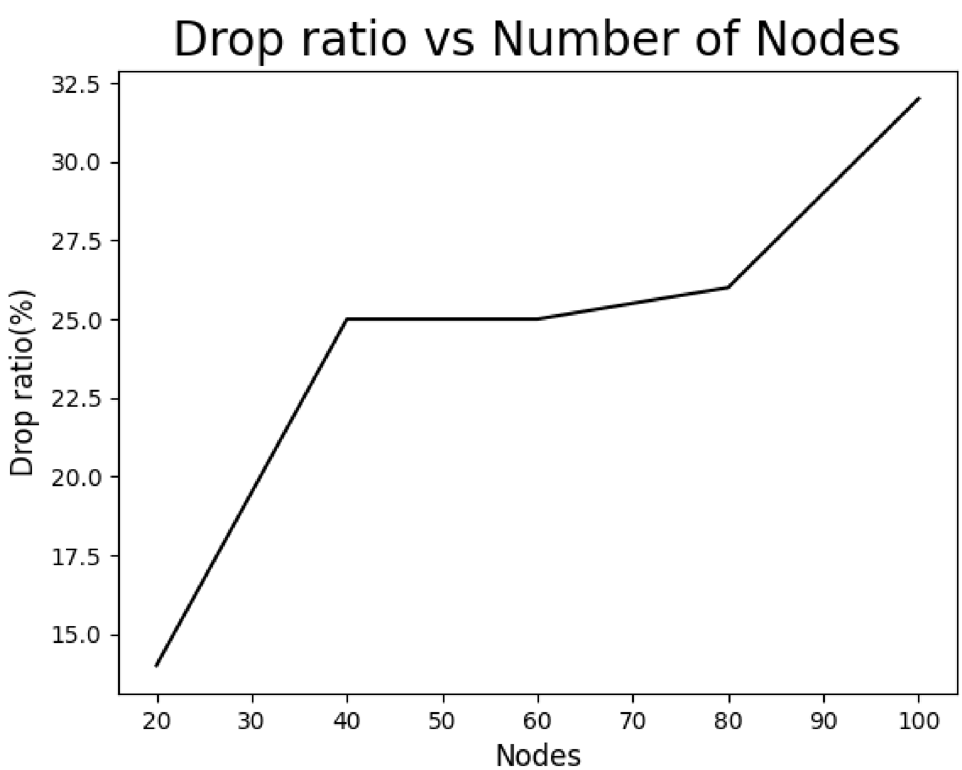
Task A :

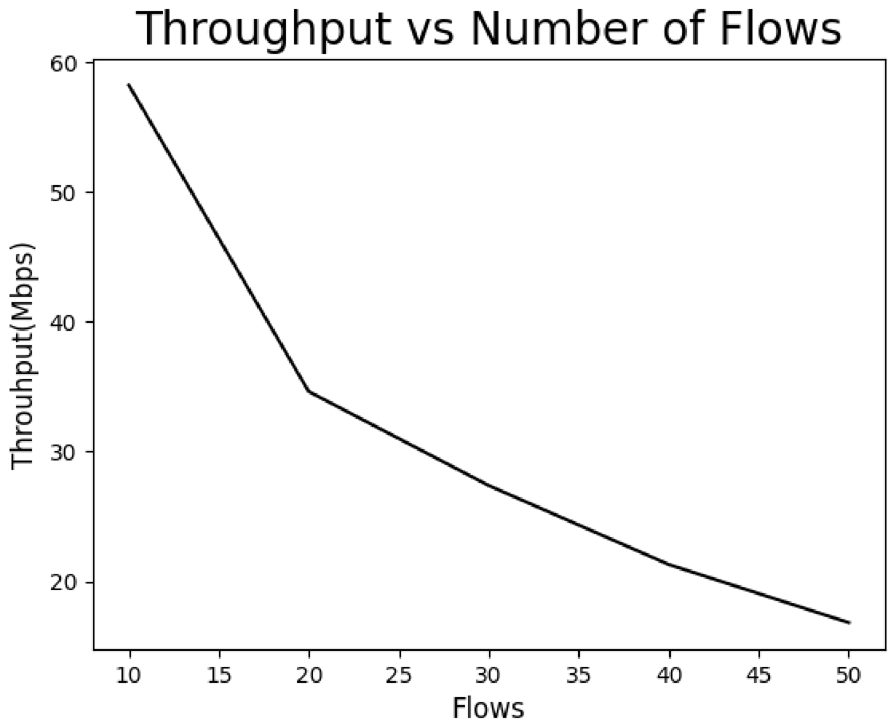
Variation in number of nodes,flows,packet per second and coverage was made to plot graphs.

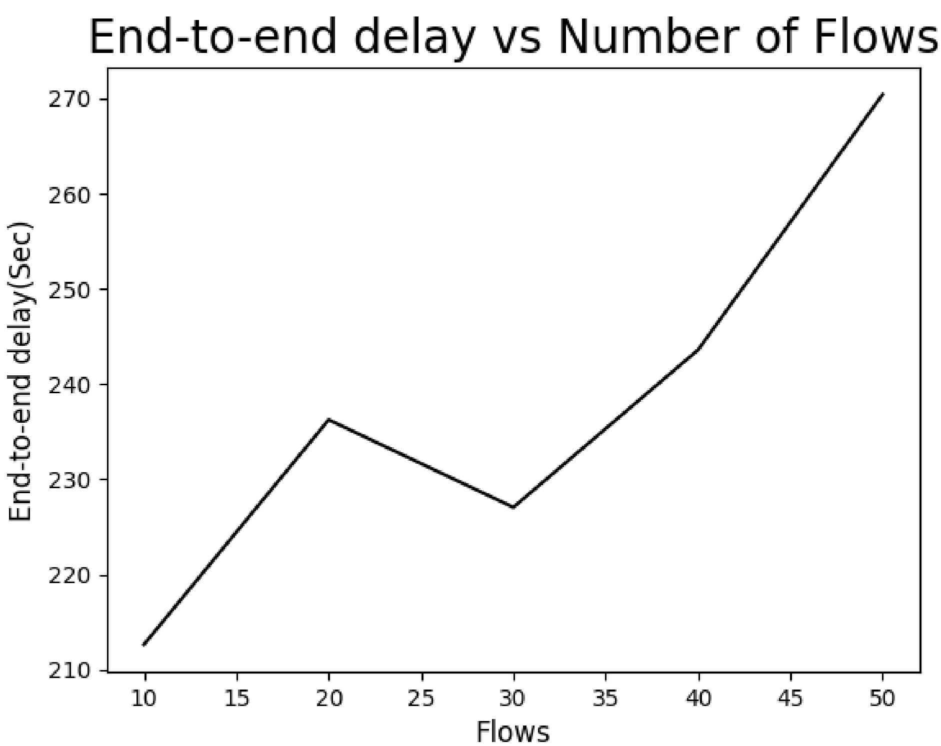


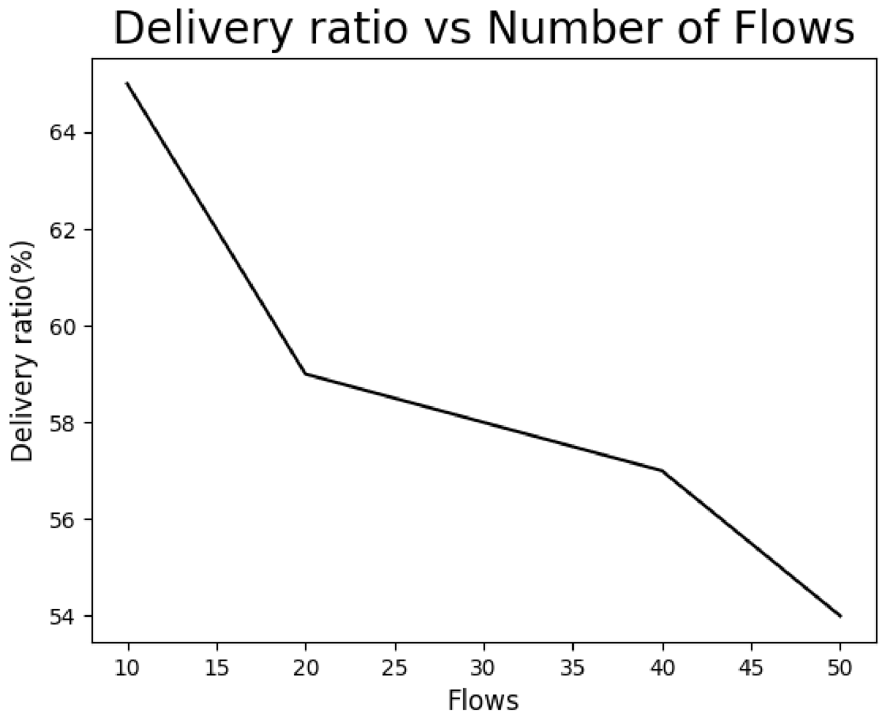


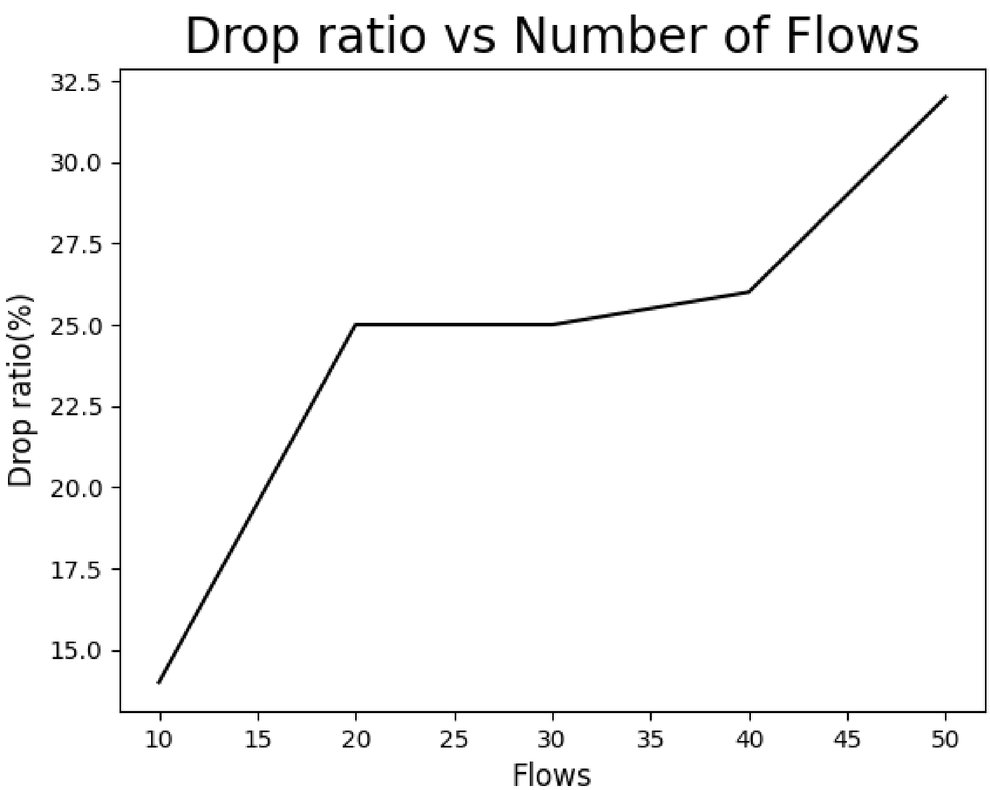

Delivery ratio vs Number of Nodes
Delivery ratio(%)
64
62
60
58
56
54
20
30
40
50
60
70
80
90
100
Nodes

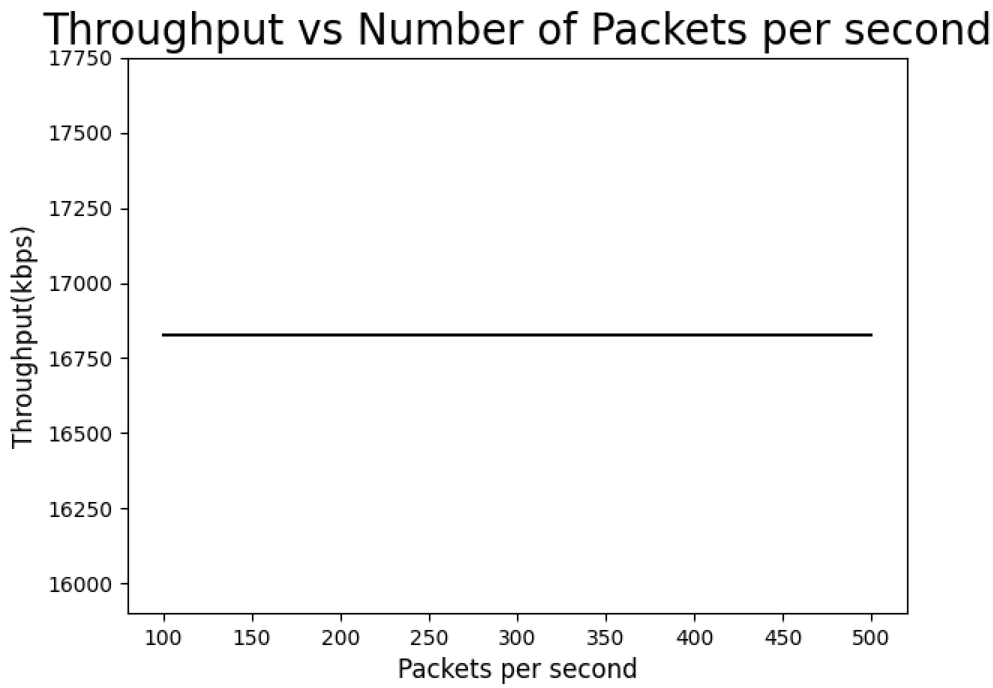

Drop ratio vs Number of Nodes
Drop ratio(%)
32.5
30.0
27.5
25.0
22.5
20.0
17.5
15.0
20
30
40
50
60
70
80
90
100
Nodes

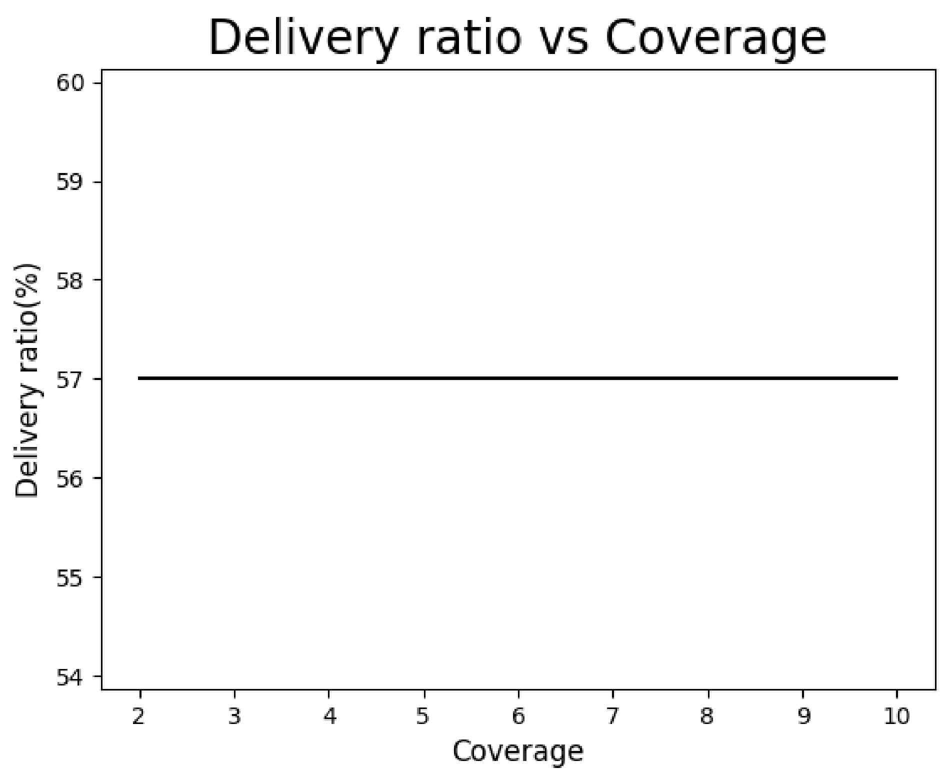

Throughput vs Number of Flows
Throuhput(Mbps)
60
50
40
30
20
10
15
20
25
30
35
40
45
50
Flows


End-to-end delay vs Number of Flows
End-to-end delay(Sec)
270
260
250
240
230
220
210
10
15
20
25
30
35
40
45
50
Flows


Delivery ratio vs Number of Flows
Delivery ratio(%)
64
62
60
58
56
54
10
15
20
25
30
35
40
45
50
Flows

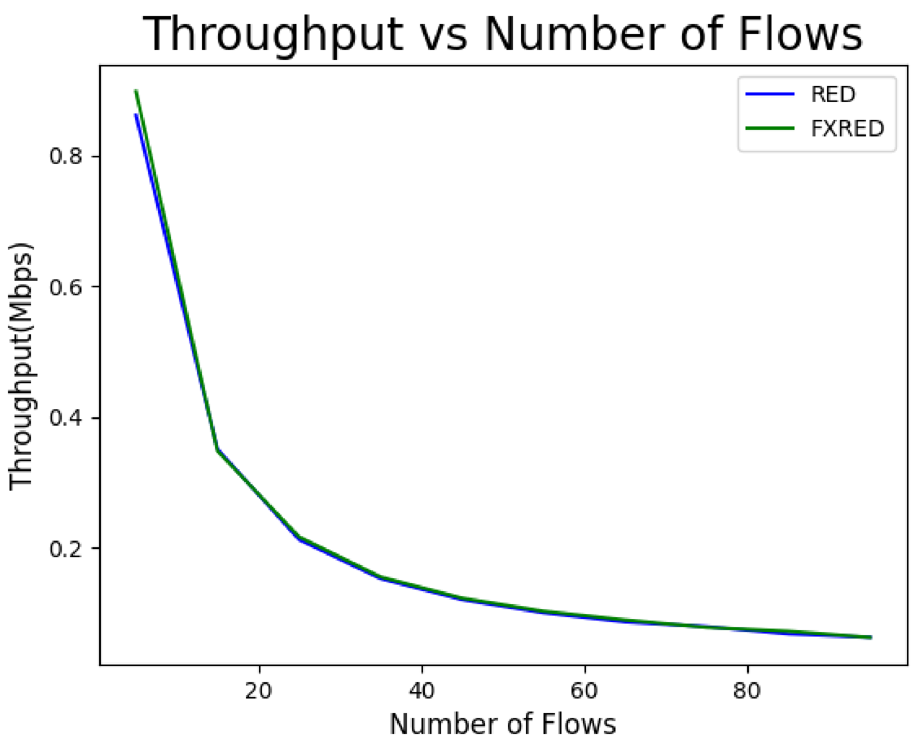

Drop ratio vs Number of Flows
Drop ratio(%)
32.5
30.0
27.5
25.0
22.5
20.0
17.5
15.0
10
15
20
25
30
35
40
45
50
Flows

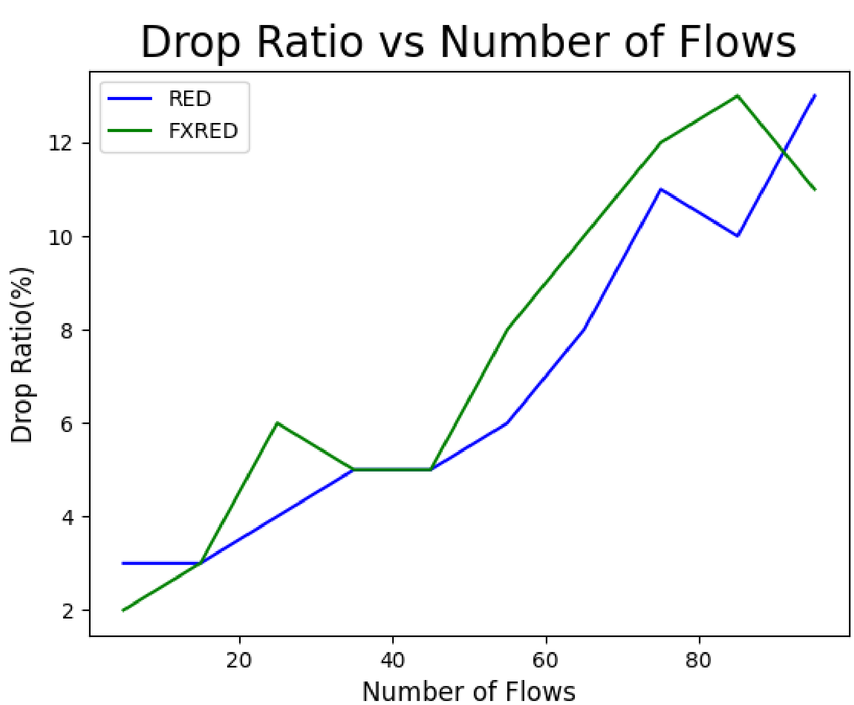

Throughput vs Number of Packets per second
17750
17500
17250
17000
16750
16500
16250
16000
Throughput(kbps)
100
150
200
250
300
350
400
450
500
Packets per second


Delivery ratio vs Coverage
60
59
58
57
56
55
54
Delivery ratio(%)
2
3
4
5
6
7
8
9
10
Coverage

Task B :

Variation of number of flows





Summary :

Task A :

As we can see the average throughput is decreasing with respect to increasing flows and nodes. The reason behind it is the packet error probability increases with an increase in the number of nodes.

End to end delay is increasing. Propagation delay, transmission delay,queuing delay are the main factors behind its increment.

Task B:

From the graph we can see the average throughput of the proposed algorithm and classical RED algorithm is almost same. The proposed algorithm has a very insignificant amount of betterment.
As for the drop ratio, in a small number(0-20) of flows the proposed algorithm works better than RED algorithm. And again the algorithm works better in a large(>90) number of flows.

The simulated results are not identical with the ones in the paper. A major reason behind this can be that the network topology used in the paper and in the simulation are not completely same.