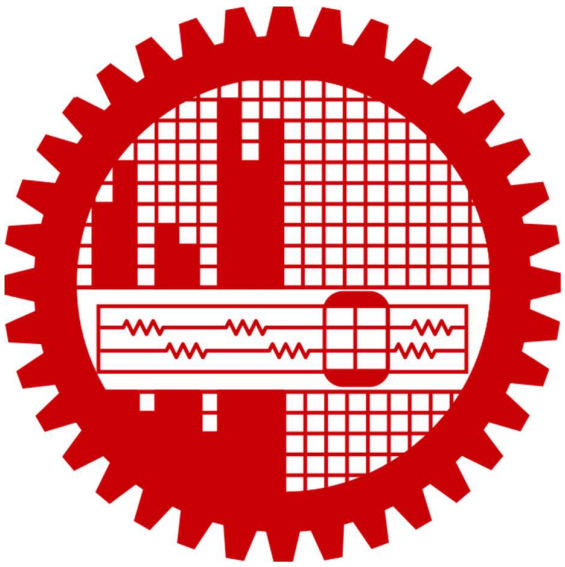
NS2 Project Report

Md. Shahrukh Islam

Student ID: 1805098



Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology

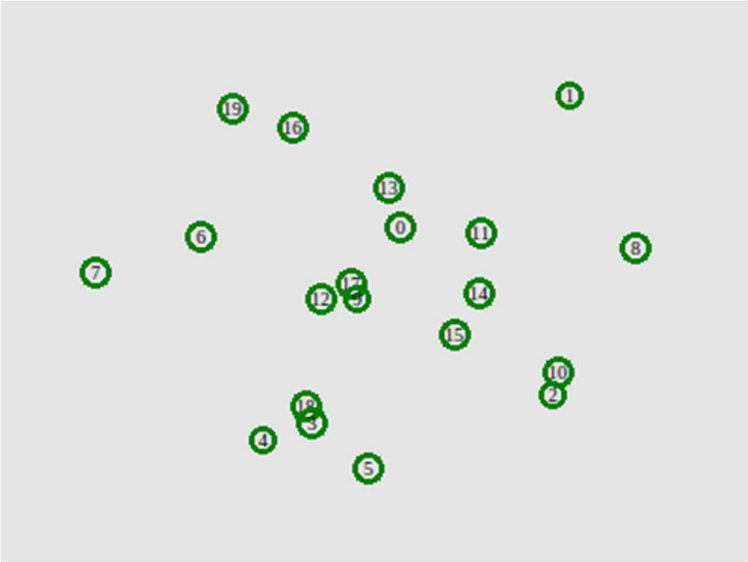
Introduction

NS2 is an open-source event-driven simulator designed specifically for research in communication networks. In this project, I have to vary different parameters such as number of nodes, number of flows, number of packets per second, speed of nodes and measure some metrics and plot them in some graphs. I also had to modify some mechanisms of Retransmission Timeout calculation and compare the modified version with the original implementation of NS2.The paper from which I have taken idea from is [The Peak-Hopper: A New End-to End Retransmission Timer for Reliable Unicast Transport]

Simulated Networks

• Wireless 802.11 (dynamic) network using AODV routing in Random topology

• Wireless 802.15.4 (dynamic) network using DSDV routing in Random Topology



Parameters and Metrics

Parameters

1. Number of nodes (20, 40, 60, 80, 100)
2. Number of flows (10, 20, 30, 40, 50)
3. Number of packets per second (100, 200, 300, 400, 500)
4. Speed of nodes (5, 10, 15, 20, 25)

Metrics

1. Network Throughput
2. End-to-End Delay
3. Packet Delivery Ratio

4. Packet Drop Ratio
5. Energy Consumption

Overview of the proposed algorithm

The PeakHopper algorithm essentially runs two RTO algorithms in parallel. One algorithm (Short-Term History RTO) monitors the present and short-term history in order to respond to RTT increases. The other algorithm (Long-Term History RTO) simply decays the current value of RTO, and can therefore be said to represent the long-term history.

We describe the details of how the PH-RTO is calculated in five steps.

$$\delta = \frac{RTT_{sample} - RTT_{previous}}{RTT_{previous}} \quad \text{(Step 1)}$$

$$D = 1 - \frac{1}{F * S} \quad \text{(Step 2)}$$

$$B \leftarrow \min(\max(2*\delta, D*B), B_{\max}) \quad \text{(Step 3)}$$

$$RTT_{\max} = \max(RTT_{sample}, RTT_{previous}) \quad \text{(Step 4)}$$

$$RTO \leftarrow \max(D*RTO, (1+B)*RTT_{\max}, RTO_{\min}) \quad \text{(Step 5)}$$

1. Having collected a new RTT sample, RTTsample, we compare this value to the previous RTT sample collected, RTTprevious, as shown in Step 1. We call the normalized change between these two samples delta. This is the measure of the short-term changes in RTT.
2. In Step 2, we further define a decay factor, D. D determines how rapidly the RTO is decayed. We also introduce a fader variable, F, which controls the speed of this decay (a high F gives a slow decay and vice versa).
3. In Step 3, we calculate a booster variable B. The booster variable determines how high the RTO should hop when a large RTT increase has been detected.
4. In Step 4, we set RTTmax to the maximum of the new RTT sample, RTTsample, and the previous RTT sample, RTTprevious. RTTmax is used to represent the short-term history of the RTT.
5. In step 5, We set RTO to the maximum of a long-term history (represented by the term D*RTO) and the short-term history (represented by the term ((1+B)*RTTmax).
6. In final step (Step 6), we ensure that the RTO does not fall below the minimum allowed RTO.

Modifications:

I have modified the RTO calculation algorithm in NS2. The relevant files were tcp.h and tcp.cc.

In tcp.h new variables are declared:

```
293     double minrto_;          /* min value of an
294     //Shahrukh Changes
295     double B; //save rakhar jonno
296
297     double RTTsample;
298     double RTTprev;
299     double F;
300     double S;
301     double RTO;
302
303     //Shahrukh Change done
```

In rtt_timeout() function new mechanism is implemented.

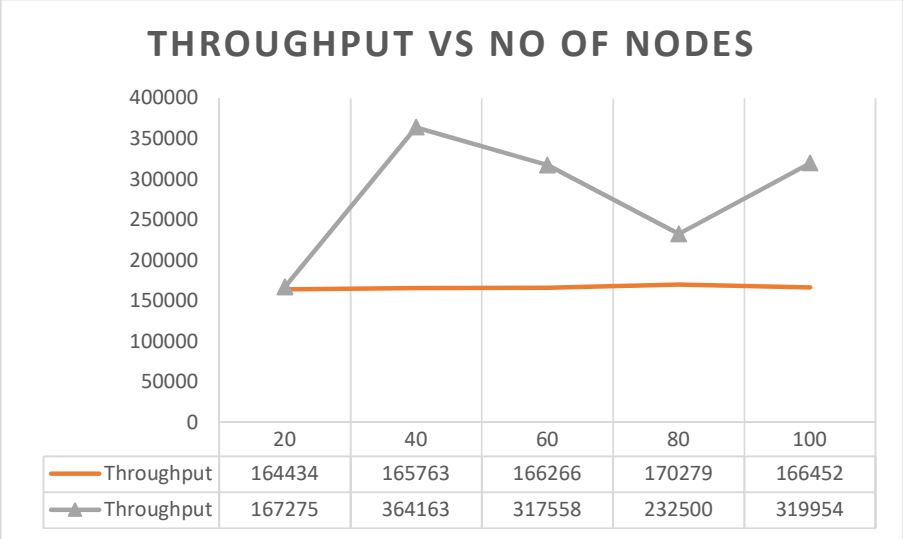
```
double TcpAgent::rtt_timeout()
{
    //Shahrukh Changes
    int peakhopper=1;
    if(peakhopper==1)
    {
        double delta=0;
        if (RTTprev!=0)
            delta=(RTTsample-RTTprev)/RTTprev;
        if(ts_option_)
        {
            S=t_seqno_ - highest_ack_;
        }
        else
            S=1;
        double D;
        if (S==0)
        {
            D =1;
        }

        else
```

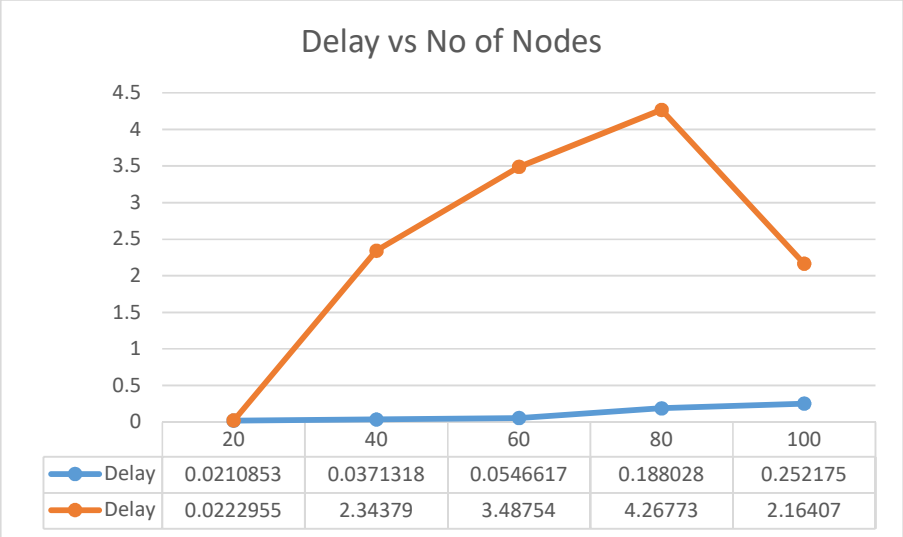
```
551             {
552                 D =1;
553             }
554
555             else
556                 D = 1- (1/(F*S));
557
558
559             B=fmax(delta, D*B);
560             double RTTmax=fmax(RTTsample, RTTprev);
561             RTO=fmax(D*RTO, (1+B)*RTTmax);
562             RTO=fmax(RTO, minrto_);
563             return RTO;
564         }
565
566         //Change Done by Shahrukh
567
568
569
```

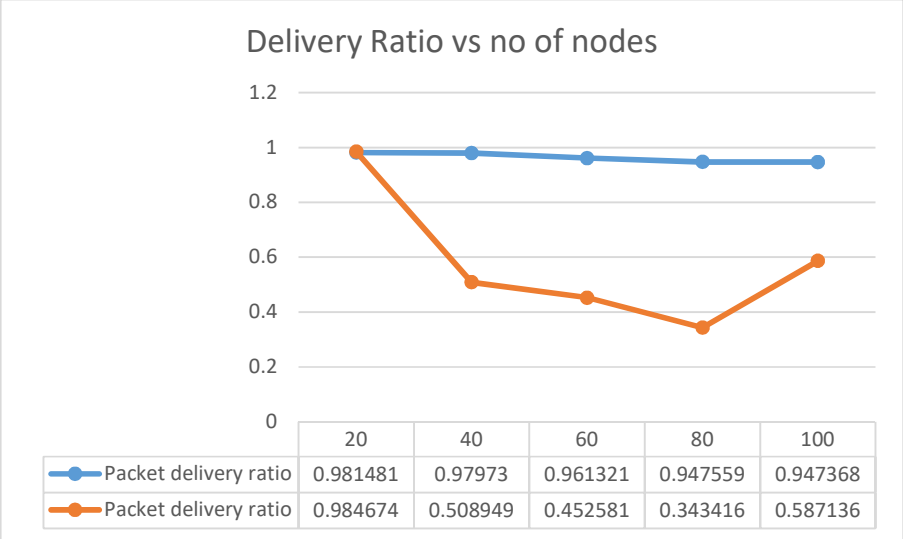
Graphs:

802.11 network:

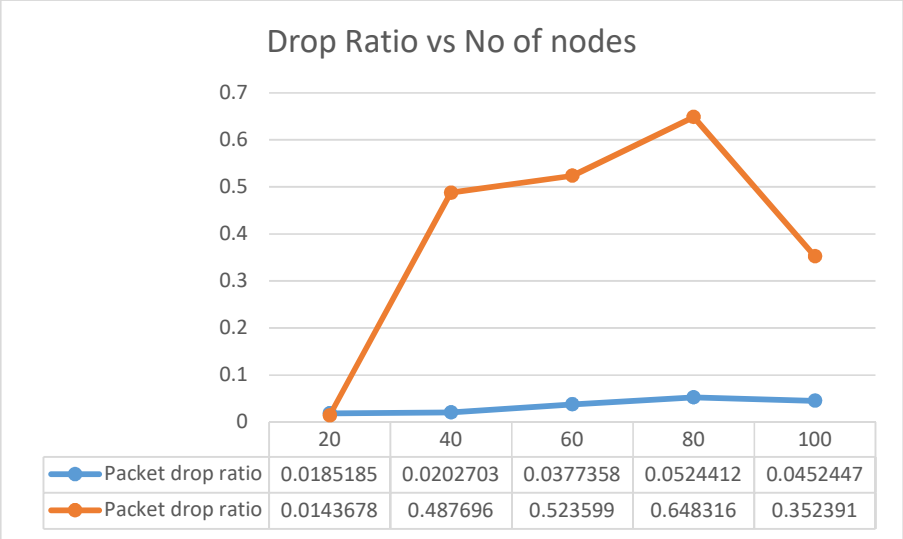
Varying no of nodes:



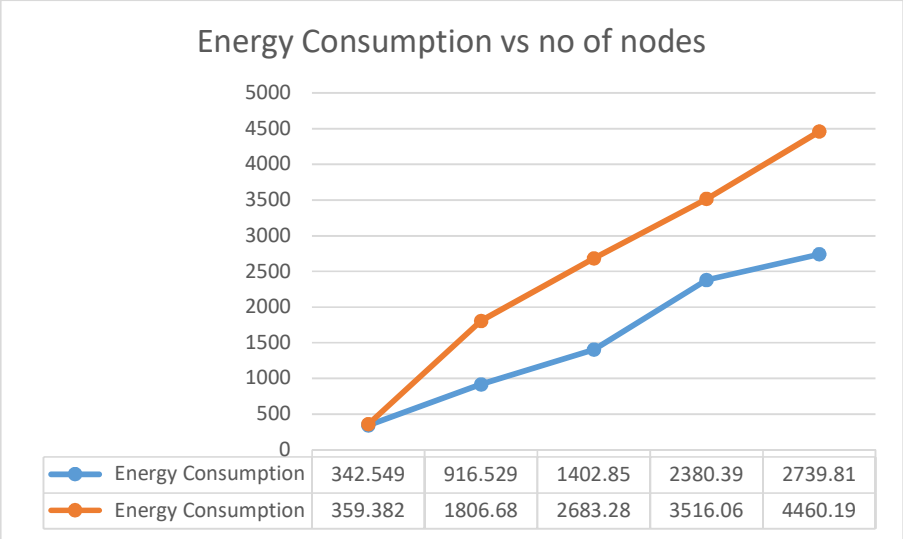




	20	40	60	80	100
Packet delivery ratio	0.981481	0.97973	0.961321	0.947559	0.947368
Packet delivery ratio	0.984674	0.508949	0.452581	0.343416	0.587136

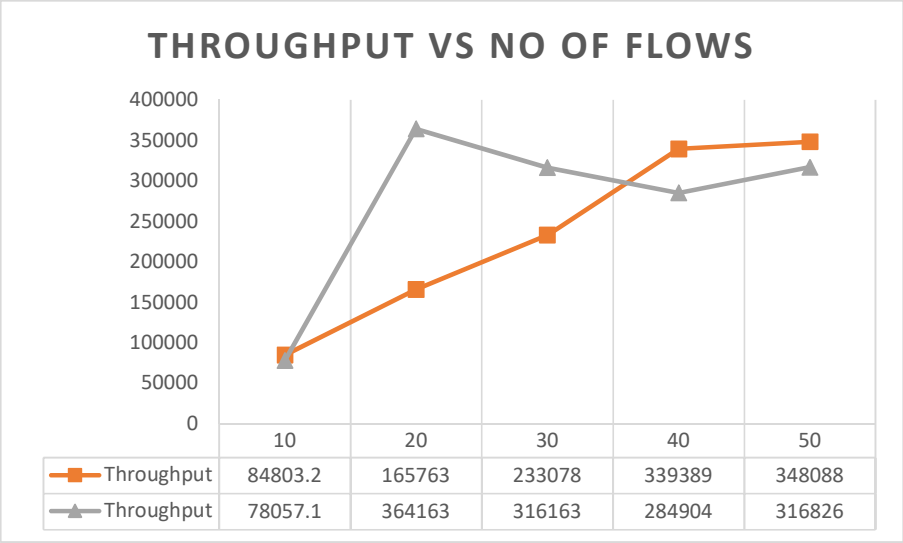


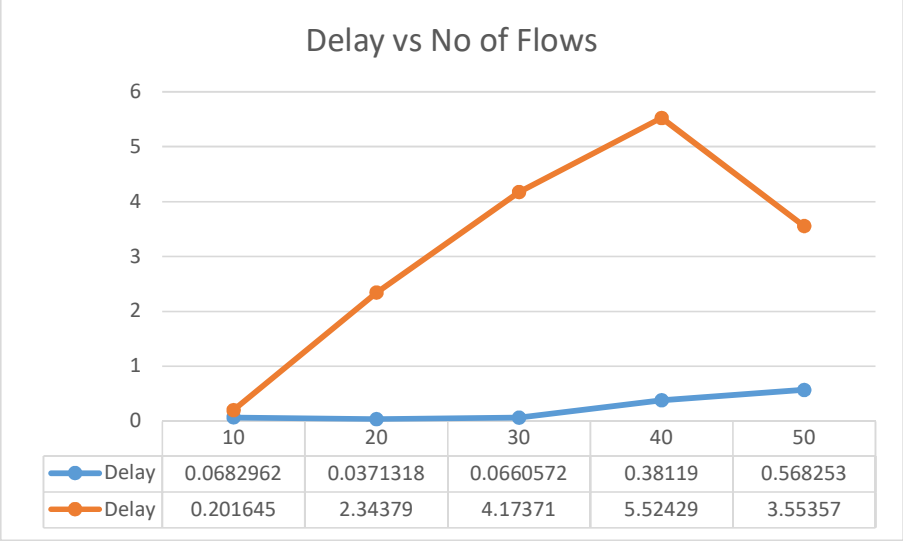
	20	40	60	80	100
Packet drop ratio	0.0185185	0.0202703	0.0377358	0.0524412	0.0452447
Packet drop ratio	0.0143678	0.487696	0.523599	0.648316	0.352391

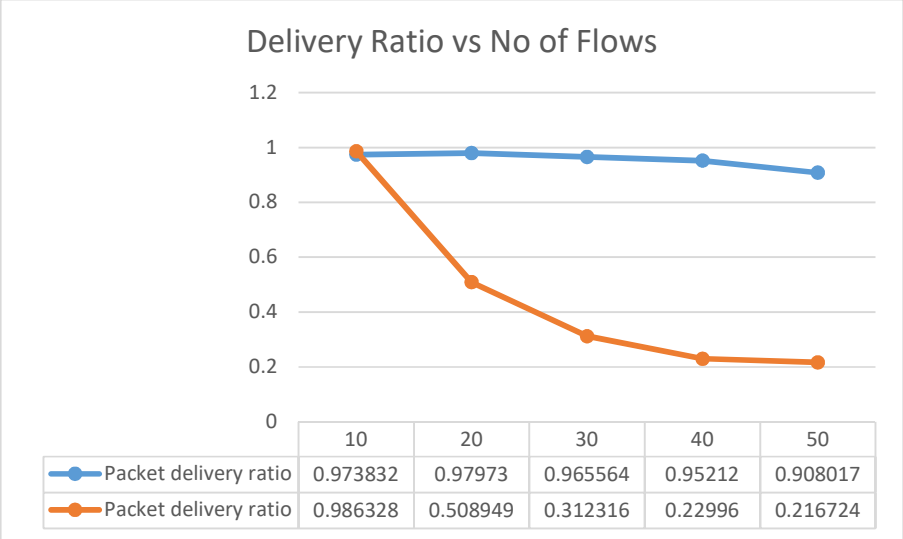


Energy Consumption	342.549	916.529	1402.85	2380.39	2739.81
Energy Consumption	359.382	1806.68	2683.28	3516.06	4460.19

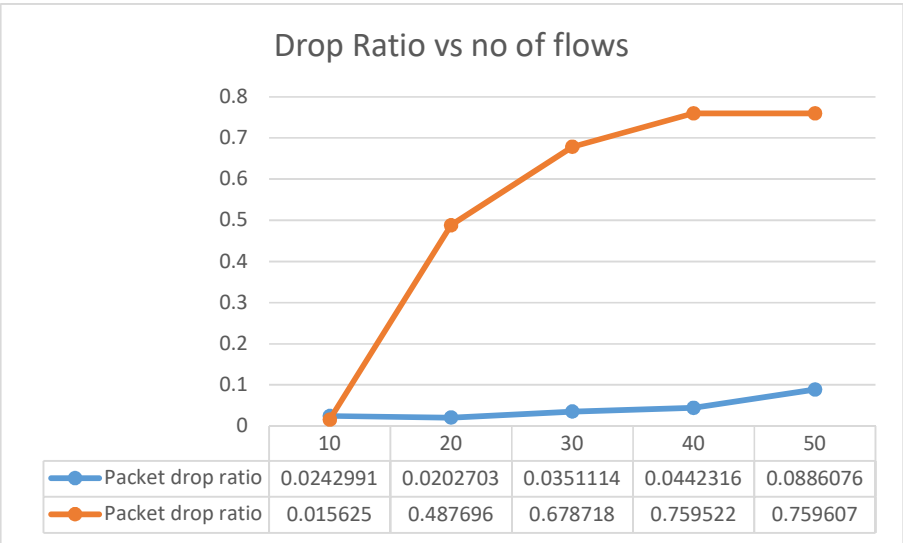
Varying no of flows:



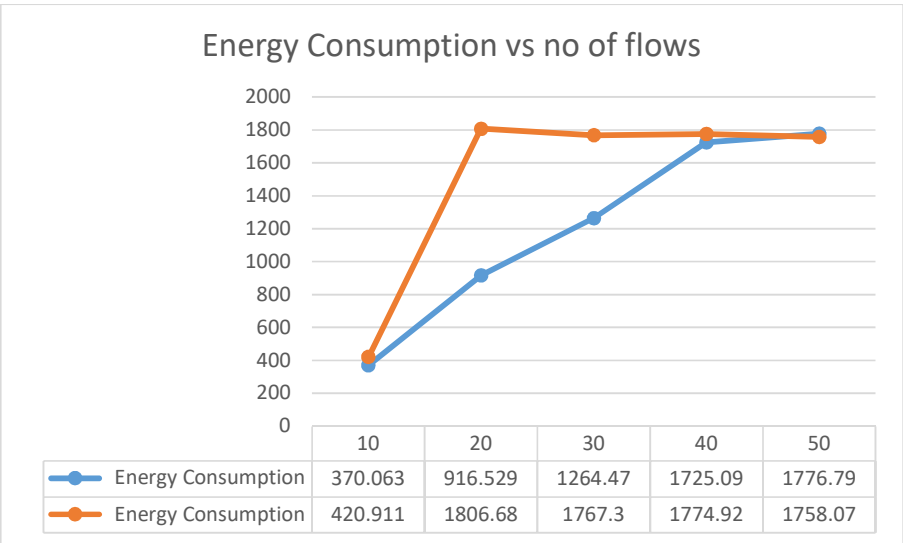




	10	20	30	40	50
Packet delivery ratio	0.973832	0.97973	0.965564	0.95212	0.908017
Packet delivery ratio	0.986328	0.508949	0.312316	0.22996	0.216724

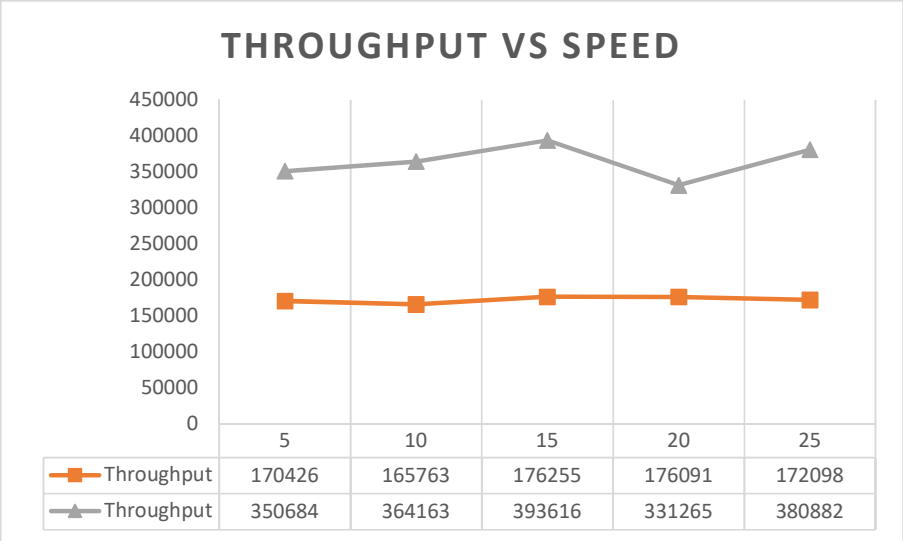


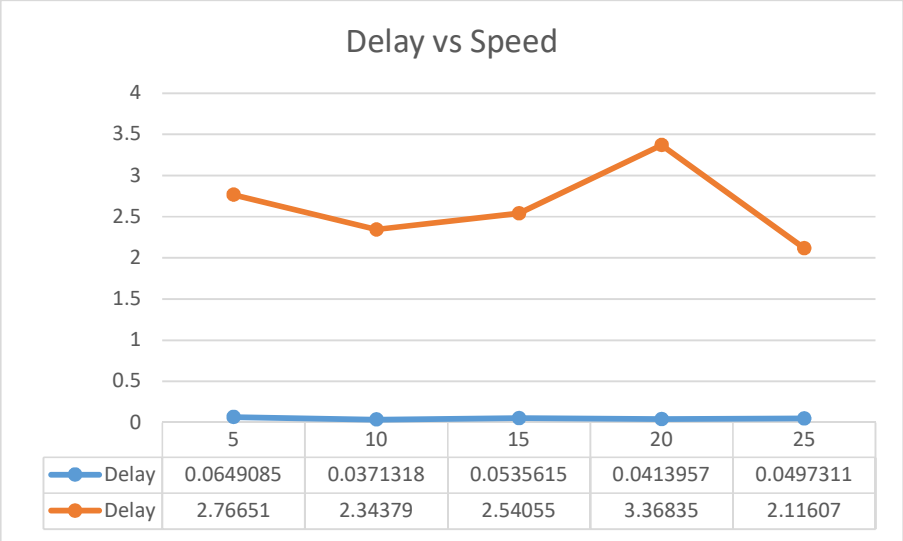
	10	20	30	40	50
Packet drop ratio	0.0242991	0.0202703	0.0351114	0.0442316	0.0886076
Packet drop ratio	0.015625	0.487696	0.678718	0.759522	0.759607

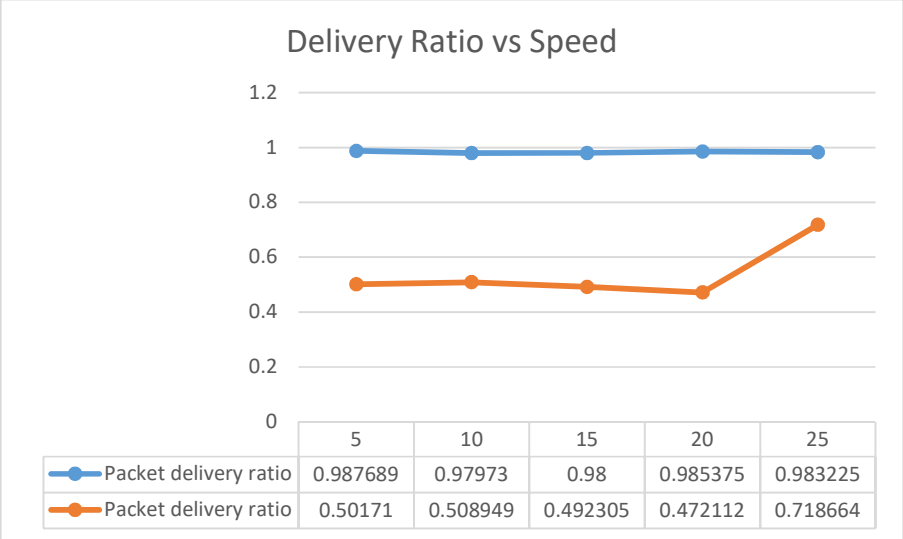


	10	20	30	40	50
Energy Consumption	370.063	916.529	1264.47	1725.09	1776.79
Energy Consumption	420.911	1806.68	1767.3	1774.92	1758.07

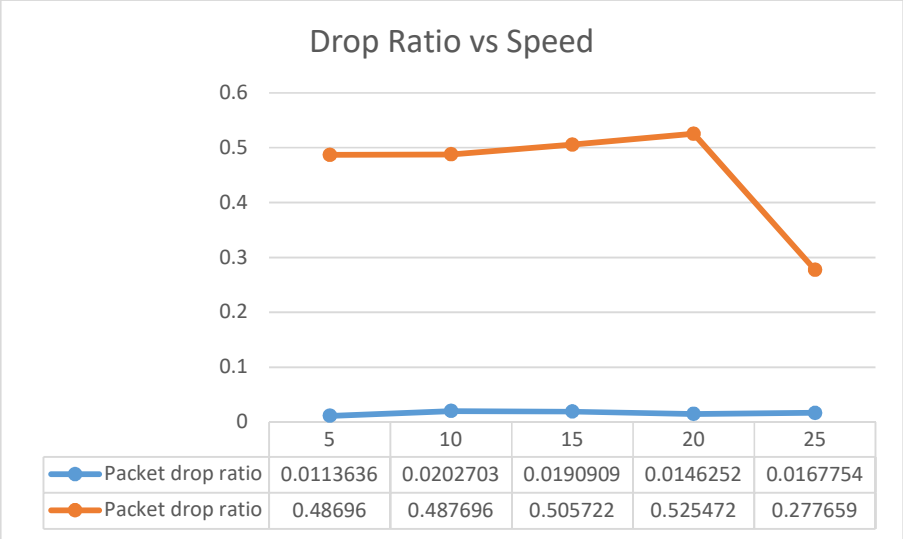
Varying speed:



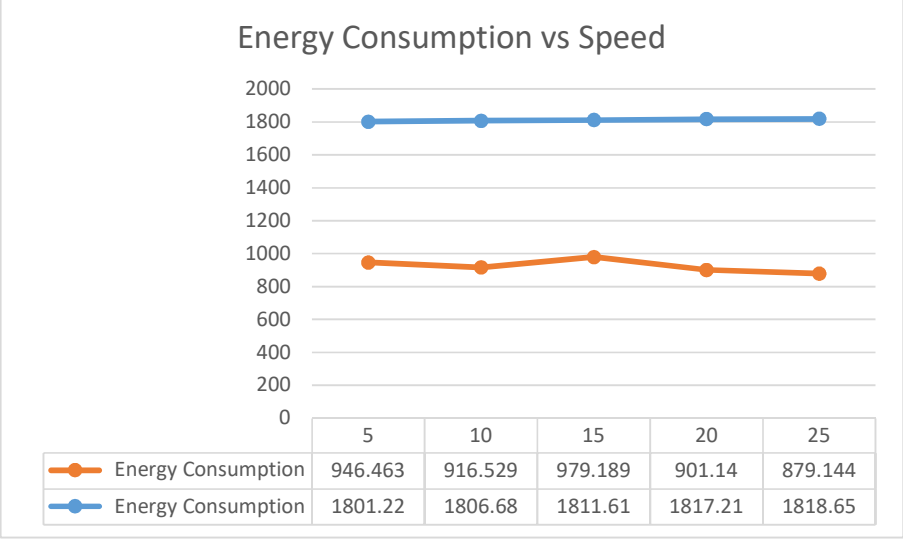




	5	10	15	20	25
Packet delivery ratio	0.987689	0.97973	0.98	0.985375	0.983225
Packet delivery ratio	0.50171	0.508949	0.492305	0.472112	0.718664

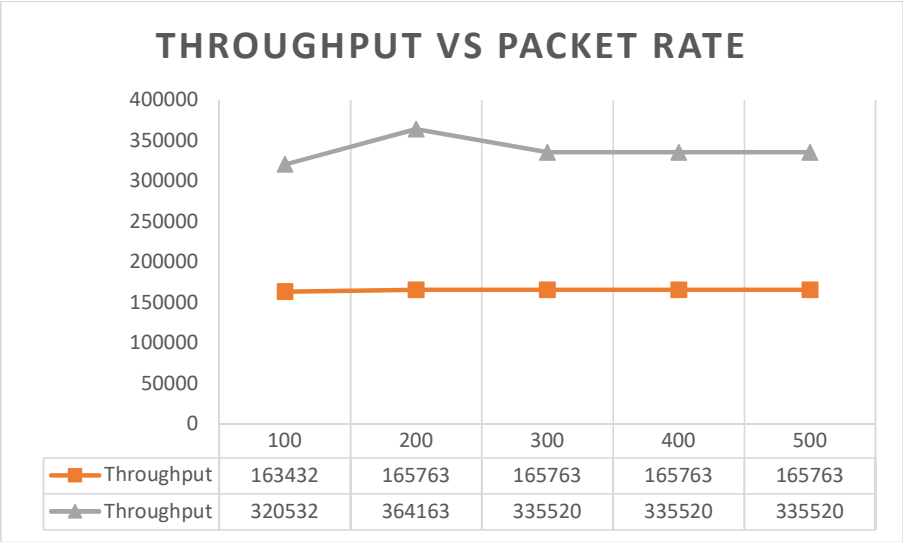


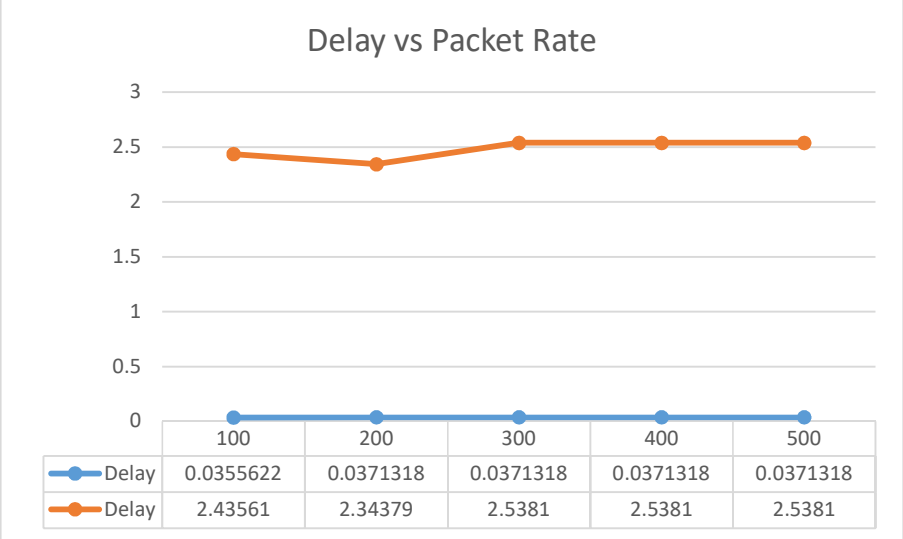
	5	10	15	20	25
Packet drop ratio	0.0113636	0.0202703	0.0190909	0.0146252	0.0167754
Packet drop ratio	0.48696	0.487696	0.505722	0.525472	0.277659

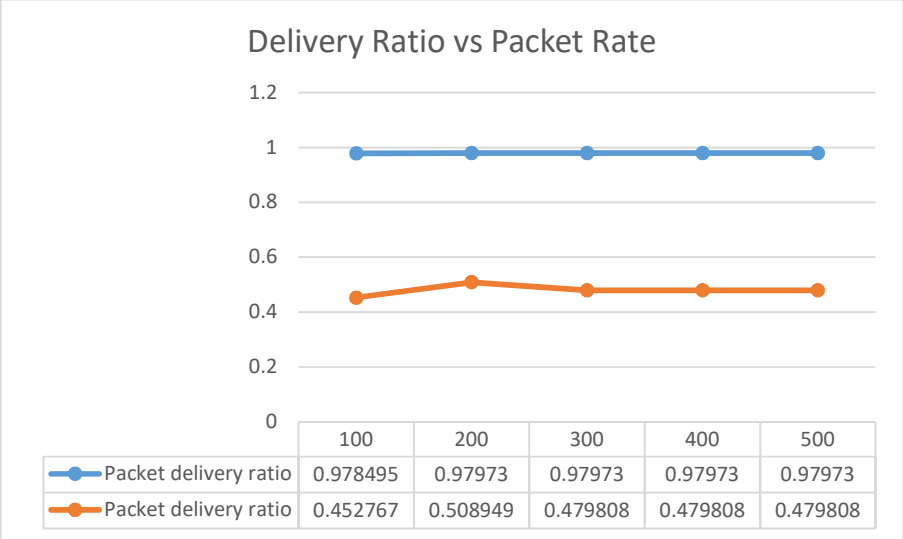


	5	10	15	20	25
Energy Consumption	946.463	916.529	979.189	901.14	879.144
Energy Consumption	1801.22	1806.68	1811.61	1817.21	1818.65

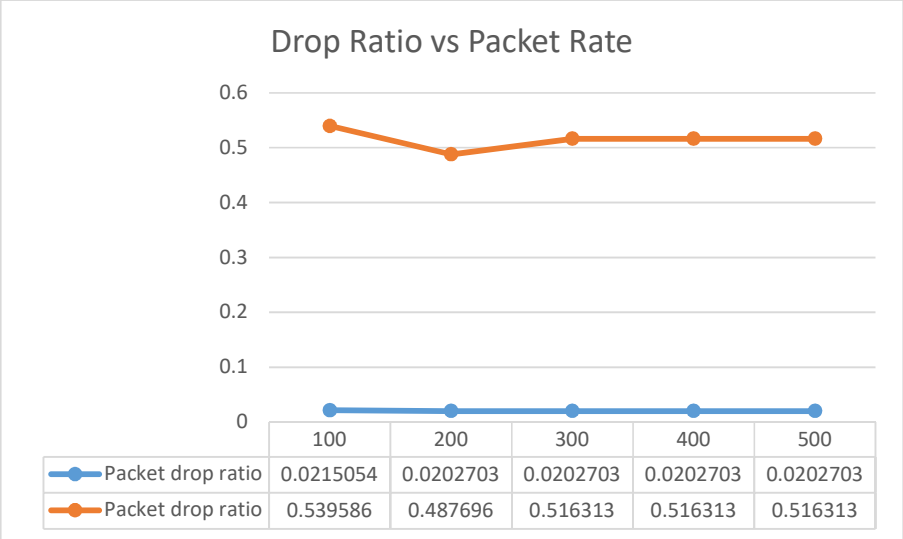
Varying packet per second:



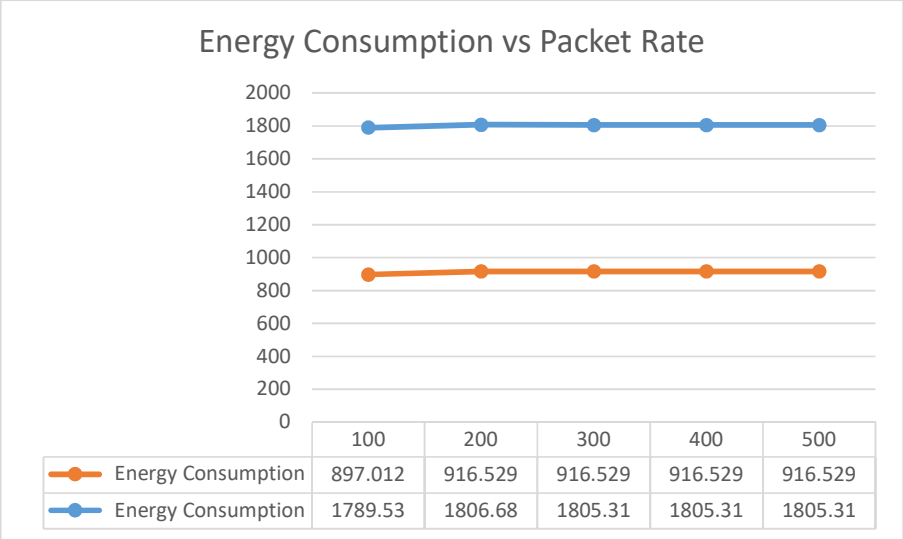




	100	200	300	400	500
Packet delivery ratio	0.978495	0.97973	0.97973	0.97973	0.97973
Packet delivery ratio	0.452767	0.508949	0.479808	0.479808	0.479808



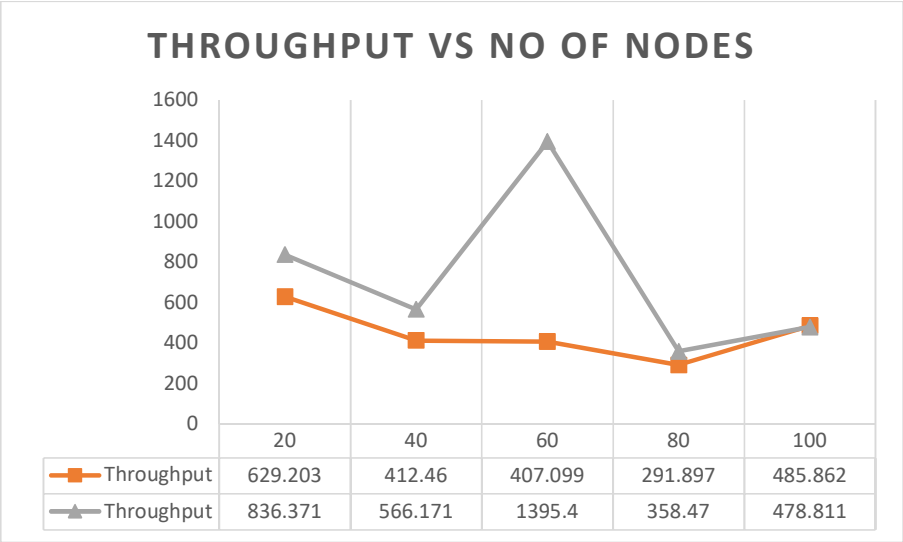
	100	200	300	400	500
Packet drop ratio	0.0215054	0.0202703	0.0202703	0.0202703	0.0202703
Packet drop ratio	0.539586	0.487696	0.516313	0.516313	0.516313

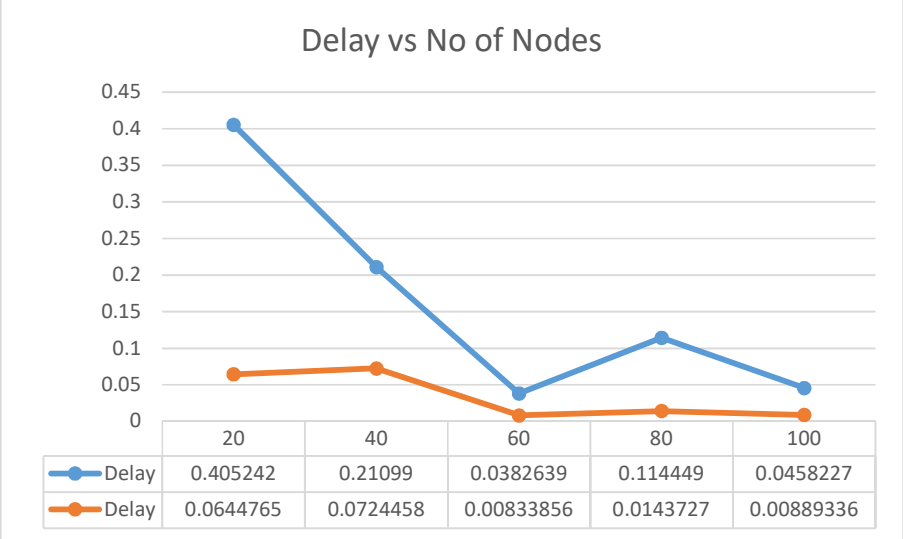


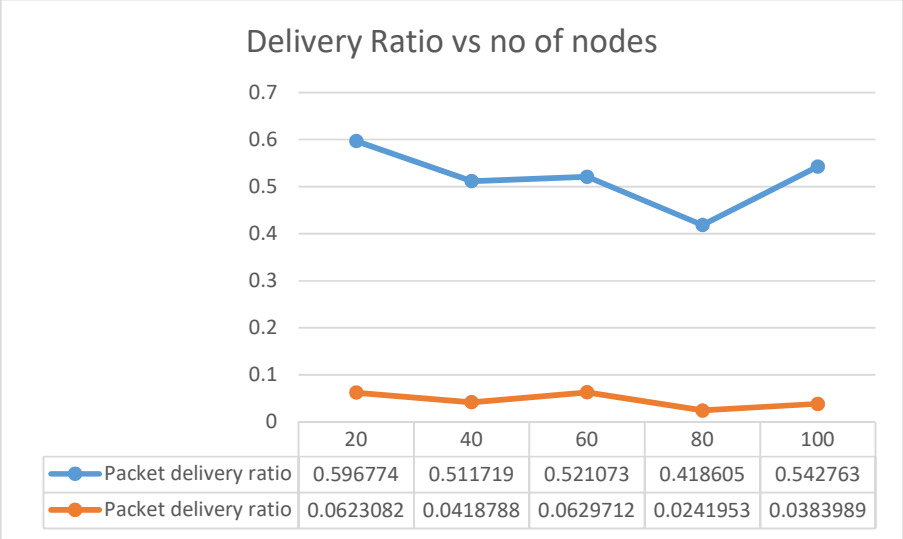
	100	200	300	400	500
Energy Consumption	897.012	916.529	916.529	916.529	916.529
Energy Consumption	1789.53	1806.68	1805.31	1805.31	1805.31

802.11 network:

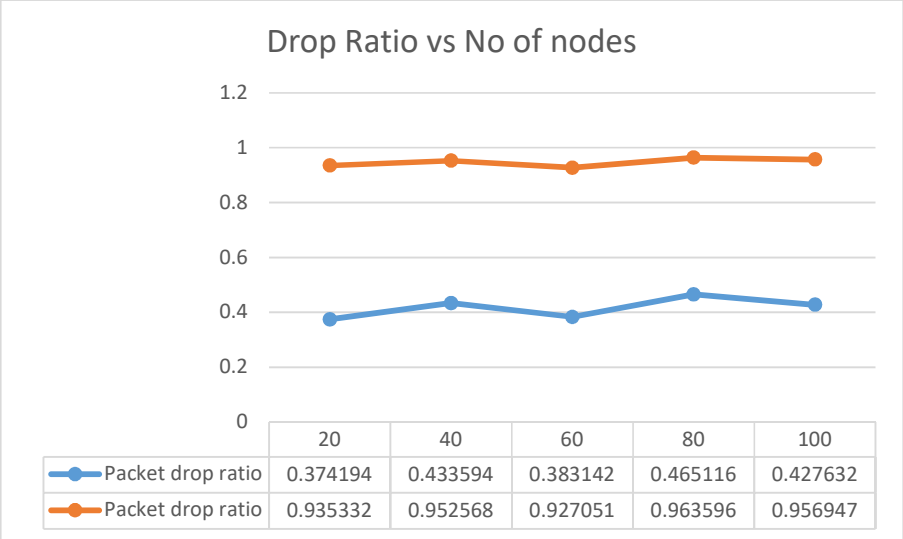
Varying no of node:



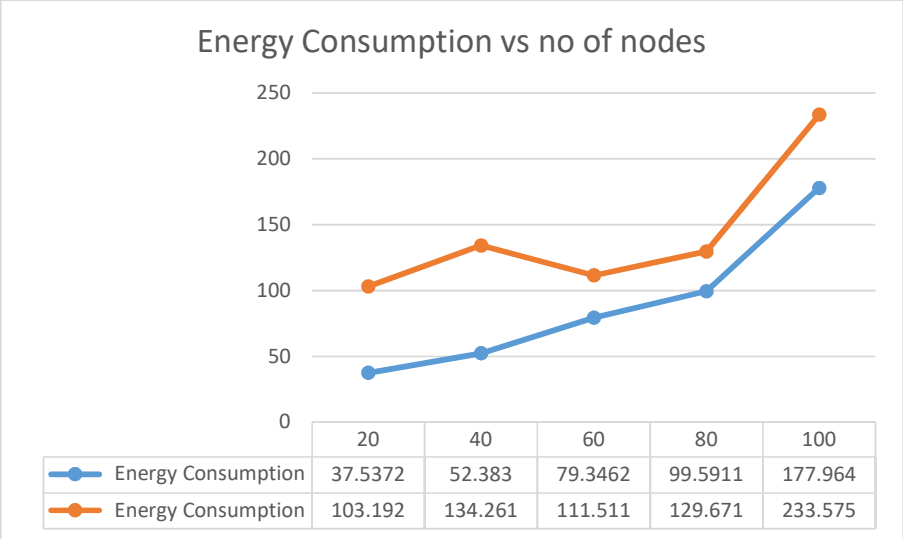




	20	40	60	80	100
Packet delivery ratio	0.596774	0.511719	0.521073	0.418605	0.542763
Packet delivery ratio	0.0623082	0.0418788	0.0629712	0.0241953	0.0383989

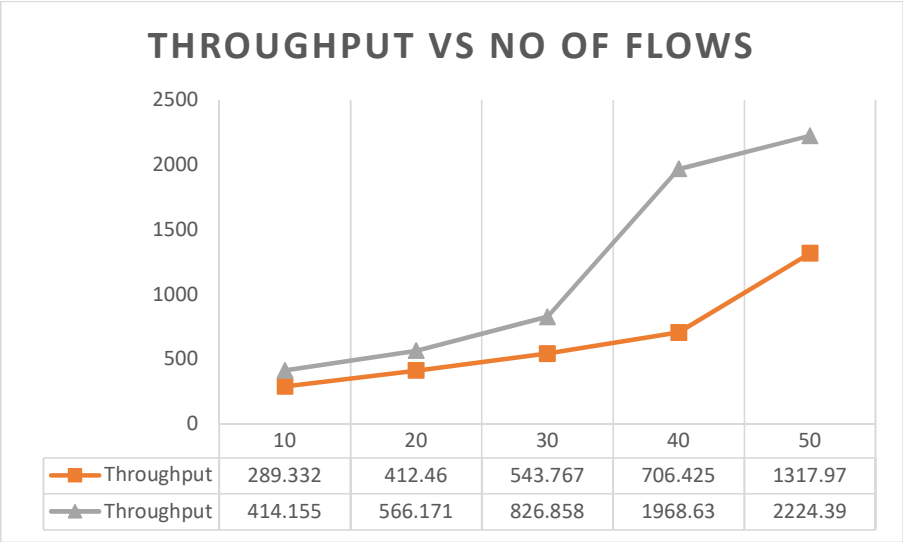


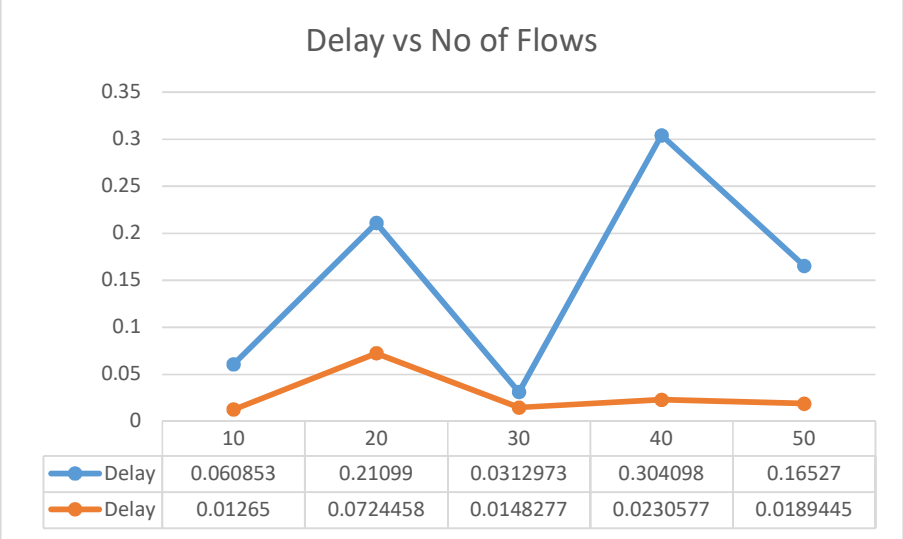
	20	40	60	80	100
Packet drop ratio	0.374194	0.433594	0.383142	0.465116	0.427632
Packet drop ratio	0.935332	0.952568	0.927051	0.963596	0.956947

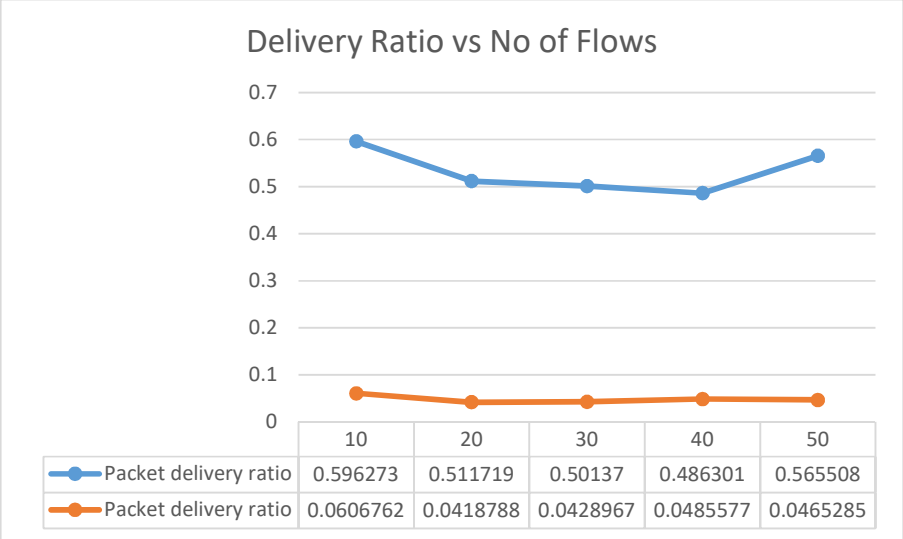


	20	40	60	80	100
Energy Consumption	37.5372	52.383	79.3462	99.5911	177.964
Energy Consumption	103.192	134.261	111.511	129.671	233.575

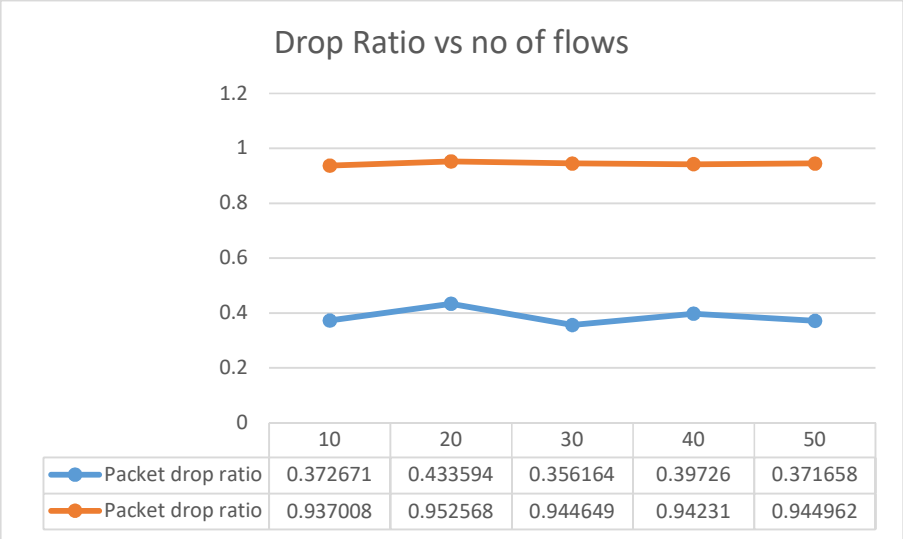
Varying no of flows:



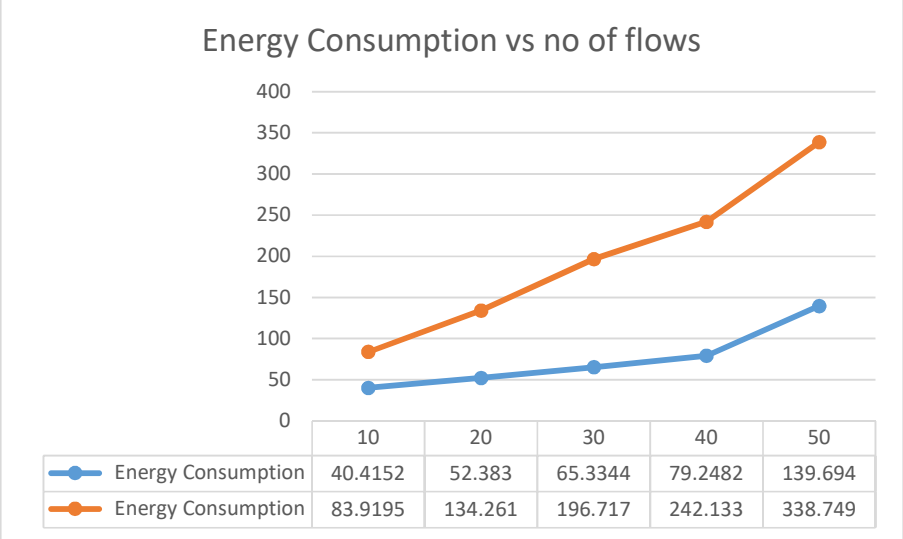




	10	20	30	40	50
Packet delivery ratio	0.596273	0.511719	0.50137	0.486301	0.565508
Packet delivery ratio	0.0606762	0.0418788	0.0428967	0.0485577	0.0465285

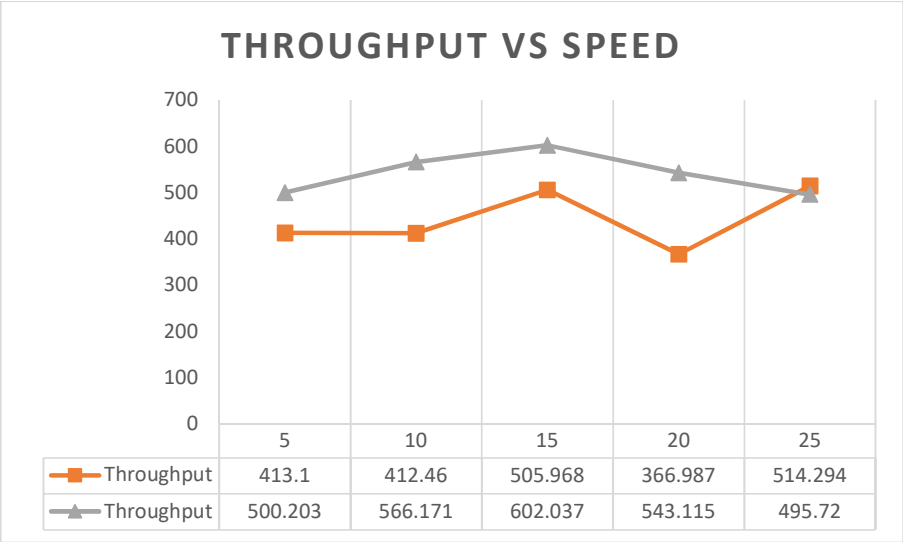


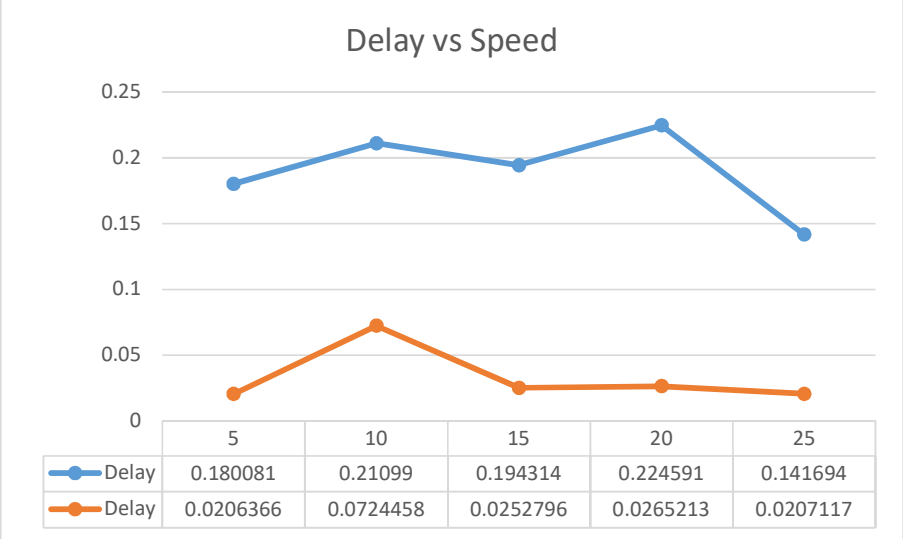
	10	20	30	40	50
Packet drop ratio	0.372671	0.433594	0.356164	0.39726	0.371658
Packet drop ratio	0.937008	0.952568	0.944649	0.94231	0.944962

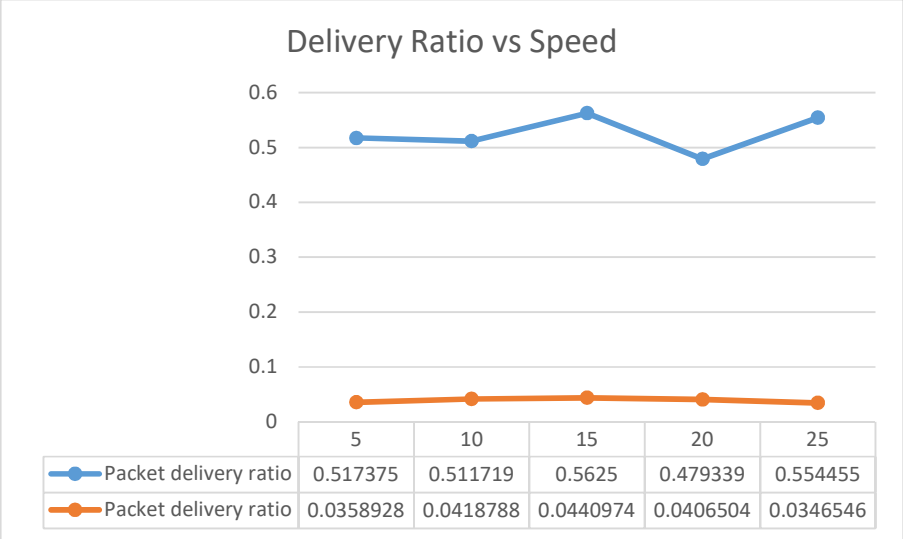


	10	20	30	40	50
Energy Consumption	40.4152	52.383	65.3344	79.2482	139.694
Energy Consumption	83.9195	134.261	196.717	242.133	338.749

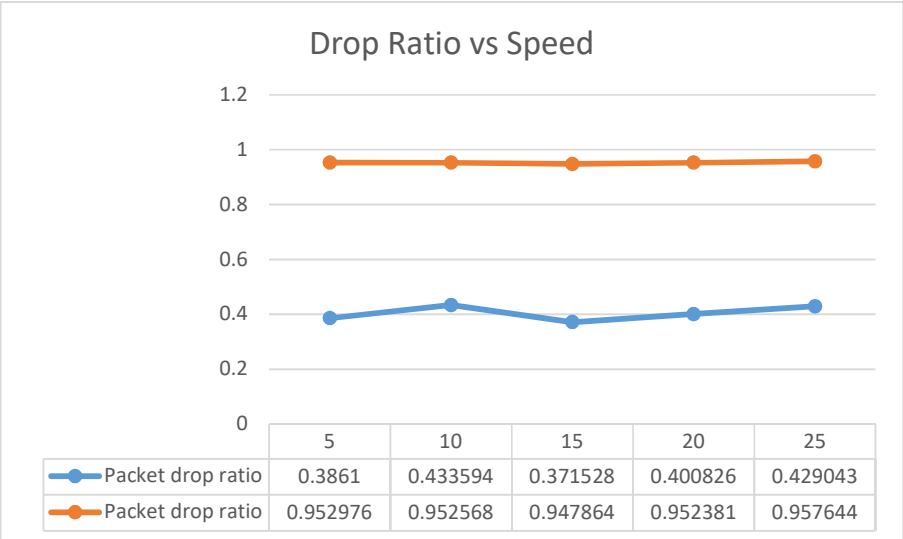
Varying speed of nodes:



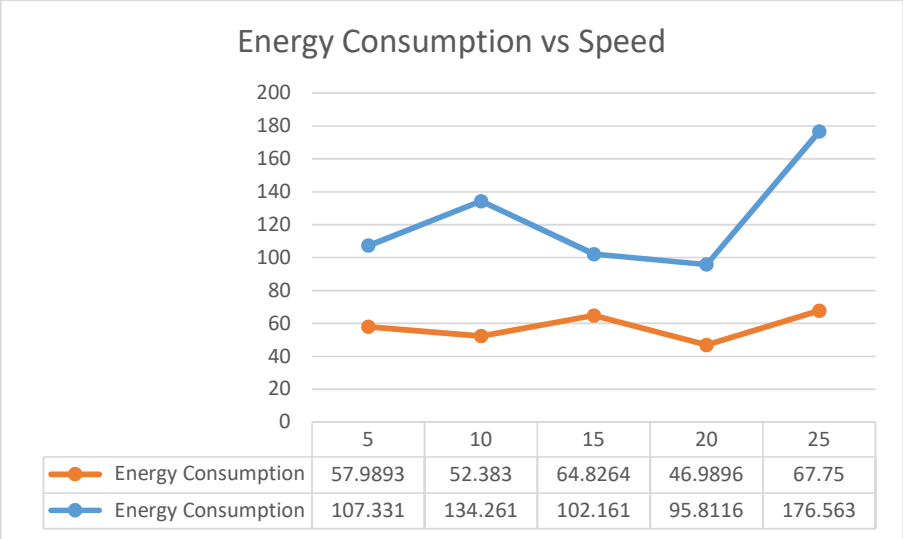




	5	10	15	20	25
Packet delivery ratio	0.517375	0.511719	0.5625	0.479339	0.554455
Packet delivery ratio	0.0358928	0.0418788	0.0440974	0.0406504	0.0346546

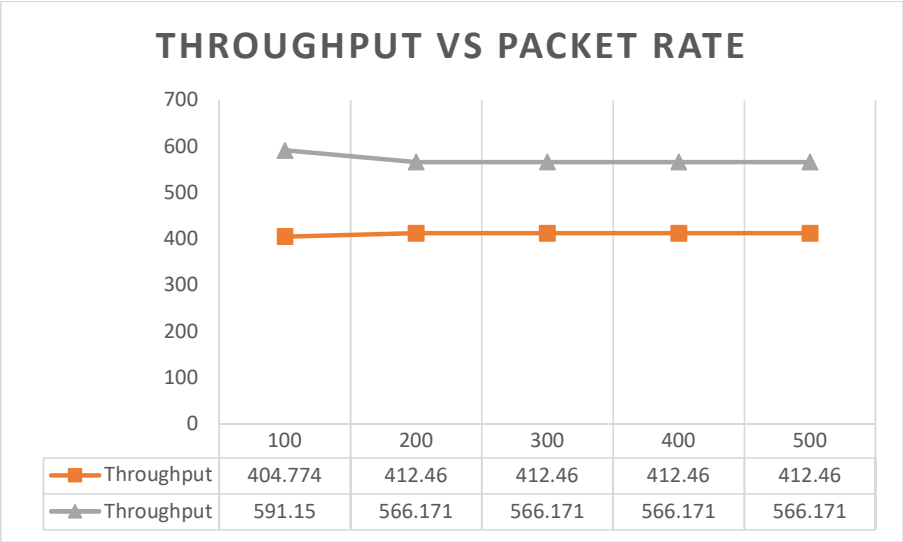


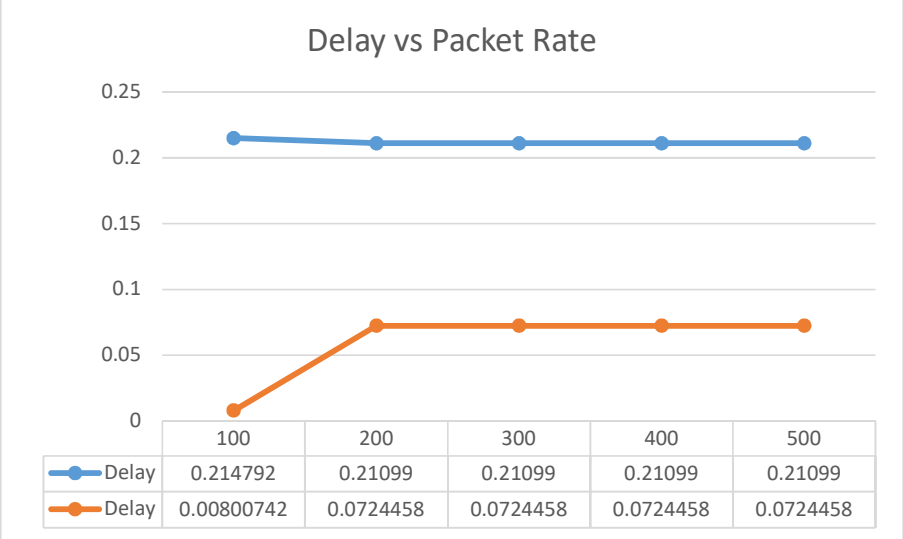
	5	10	15	20	25
Packet drop ratio	0.3861	0.433594	0.371528	0.400826	0.429043
Packet drop ratio	0.952976	0.952568	0.947864	0.952381	0.957644

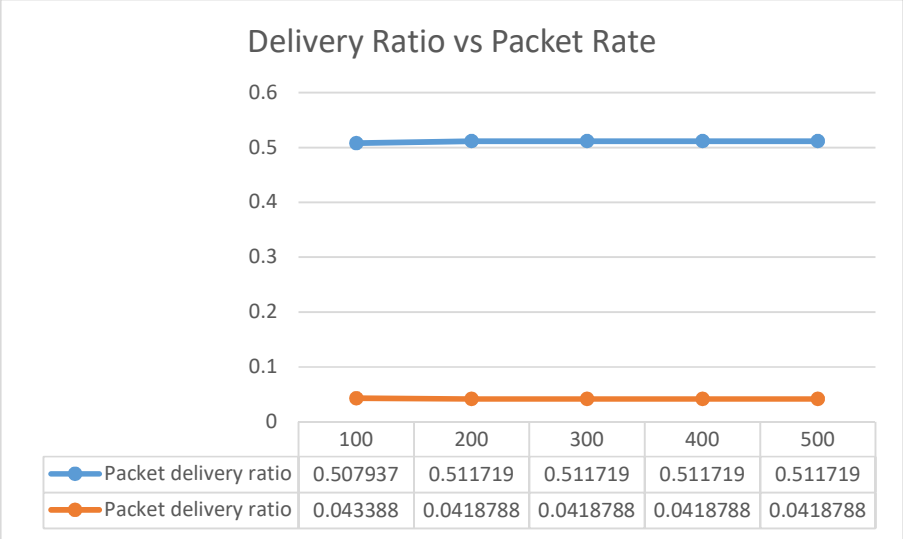


	5	10	15	20	25
Energy Consumption	57.9893	52.383	64.8264	46.9896	67.75
Energy Consumption	107.331	134.261	102.161	95.8116	176.563

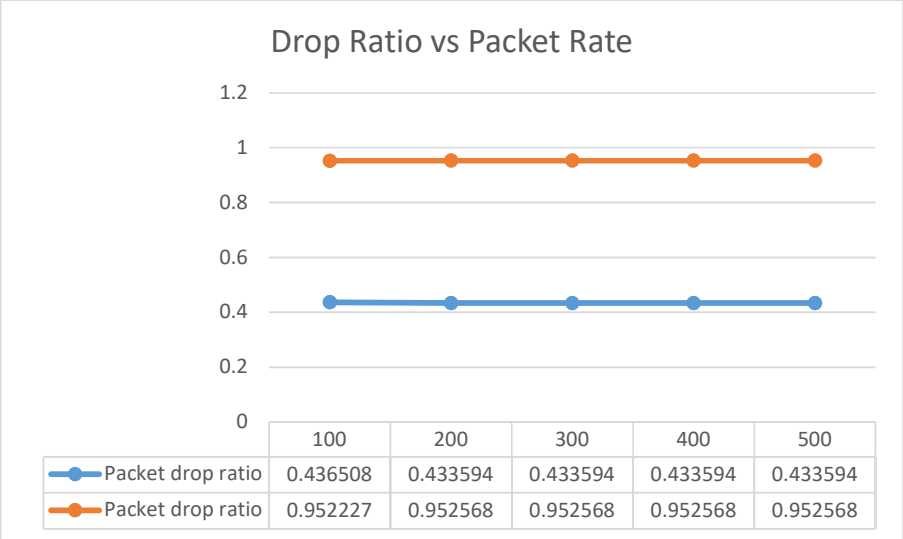
Varying packet rate:



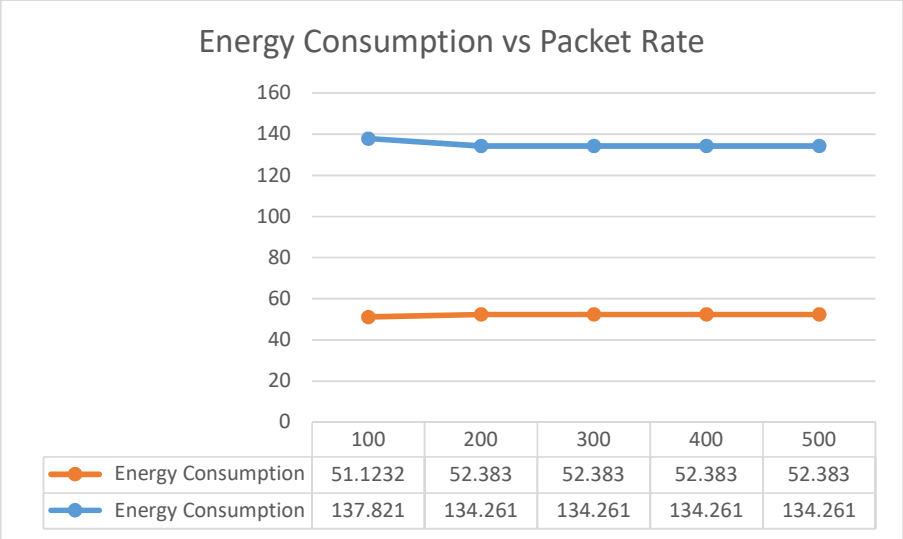




	100	200	300	400	500
Packet delivery ratio	0.507937	0.511719	0.511719	0.511719	0.511719
Packet delivery ratio	0.043388	0.0418788	0.0418788	0.0418788	0.0418788



	100	200	300	400	500
Packet drop ratio	0.436508	0.433594	0.433594	0.433594	0.433594
Packet drop ratio	0.952227	0.952568	0.952568	0.952568	0.952568



	100	200	300	400	500
Energy Consumption	51.1232	52.383	52.383	52.383	52.383
Energy Consumption	137.821	134.261	134.261	134.261	134.261

Summary:

Explanation of Results found in 802.11 static network

Varying Number of Nodes

- Throughput: Decreases as increase of node congesting the network flow more
- End-to-End Delay: Increases because of more congested network
- Delivery Ratio: Decreases with increase of more nodes congested in same network
- Drop Ratio: Starts to increase a little bit because of network congestion.
- Energy Consumption: Increases because of more congested network.

Varying Number of Flows

- Throughput: Increases due to more flow in the network
- End-to-End Delay: Increases due to more flow in the network
- Delivery Ratio: Shows a trend of decrease with increasing flows
- Drop Ratio: Increases because of network congestion.
- Energy Consumption: Increases because of more congested network.

Varying Speed of Nodes

- Throughput: Increases due to more movement of the nodes.
- End-to-End Delay: Decreases due to more movement of the nodes.
- Delivery Ratio: Shows a trend of decrease.
- Drop Ratio: Increases because of randomness in a large network.
- Energy Consumption: Remains almost same.

Varying Number of Packets/Second

- Throughput: Increases and then becomes flat.
- End-to-End Delay: Increases and then becomes flat.
- Delivery Ratio: Increases and then becomes flat.
- Drop Ratio: Decreases and then becomes flat.
- Energy Consumption: Increases and then becomes flat.

Explanation of Results found in 802.15.4 static network

Varying Number of Nodes

- Throughput: Decreases as increase of node congesting the network flow more
- End-to-End Delay: Decreases because of more nodes
- Delivery Ratio: Decreases with increase of more nodes congested in same network
- Drop Ratio: Starts to increase a little bit because of network congestion.
- Energy Consumption: Increases because of more congested network.

Varying Number of Flows

- Throughput: Increases due to more flow in the network
- End-to-End Delay: Increases due to more flow in the network
- Delivery Ratio: Shows a trend of decrease with increasing flows
- Drop Ratio: Increases because of network congestion.
- Energy Consumption: Increases because of more congested network.

Varying Speed of Nodes

- Throughput: Increases and then decreases due to speed increment.
- End-to-End Delay: Increases and then decreases due to more movement of the nodes.
- Delivery Ratio: Shows a trend of increase.
- Drop Ratio: Decreases because of randomness in a large network.
- Energy Consumption: Increases slightly.

Varying Number of Packets/Second

- Throughput: Increases and then becomes flat.
- End-to-End Delay: Increases and then becomes flat.
- Delivery Ratio: Increases and then becomes flat.
- Drop Ratio: Decreases and then becomes flat.
- Energy Consumption: Increases and then becomes flat.

Explanation of Performance Metrics Result in RTT/RTO modification algorithm:

From the mean retransmission graph in the paper, it's seen that modified algorithm provides a lower mean retransmission than default Jacobson's implementation. It was expected because a lower mean indicates that retransmission timer follows a narrow waiting time and a higher mean indicates to a longer waiting time.

Another metric, known as the mean of RTO_N/RTT_L where RTO_N is the newly calculated RTO and RTT_L is the last RTT sample prior to the RTO update. We then calculated P as the mean value of this

fraction over an entire simulation run. This mean is found to be lower in modified algorithm than in default implementation which is expected too. A retransmission timer with lower mean of this ratios is expected to perform better.

In graphs we noticed that throughput has been increased after this modification that means per second number of received bits has been increased. That means a smaller RTO has occurred and packet drop is early detected. And for this reason throughput has been increased after modification.