Bangladesh University of Engineering and Technology

Department of Computer Science and Engineering

CSE 322- Project

SARED - Stochastically Adaptive RED

Submitted By:

Mohammad Abrar Nafee Akhand

1805089

Introduction

SARED(Stochastically Adaptive RED) is an improved version of RED queue management system that uses a Markov Chain to predict the average length of the queue and modifies the drop probability according to the transition probabilities.

Proposed Algorithm

This queue management system observes the queue length at discrete set of times t_0, t_1, t_2 …t, ….., where $t_i – t_{i-1}$ is very small. Let the successive observations of average queue size be denoted by X_0, X_1, …. X_t, …. and therefore it can be assumed that Xt is a random variable. The value of X_t represents the state {0, 1 or 2} at time t of the system. Therefore, the system has (i) a finite number of states, (ii) X_t is a random variable and (iii) X_t depends on X_{t-1} and thus the sequence $\{X_t\}$ is a Markov chain.
Here the system is in

State 1 when	ave <= th_min
State 2 when	th_min < ave < th_max
State 3 when	ave >= th_max

Ave is the average queue. length th_min and th_max are minimum and maximum threshold of queue length. The one step transition probabilities are:

$$p(1) = \begin{bmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \end{bmatrix}$$

These transition probabilities are calculated by using historical data of the average queue length.

Topology

For wired simulation, a dumbbell-like topology was used. A single link is used by all flows and that link acts as a bottleneck.
For wireless simulation, nodes were placed randomly and the flows were random as well. DSDV was used as the routing protocol.

Parameters Under Variation

Parameter	Base Value	Values
Number of Nodes	40	20, 40, 60, 80, 100
Number of FLows	20	10, 20, 30, 40, 50
Number of Packets/sec	200	100, 200, 300, 400, 500
Coverage Area	500	250, 500, 750, 1000, 1250

Changes made to NS2

```
if(edp_.adaptive == 2){
       edv_.saredCount++;
       if(edv_.saredCount == edp_.saredLimit){
              edv_.n00 = 1.0;
              edv_.n01 = 1.0;
              edv_.n02 = 1.0;
              edv_.n10 = 1.0;
              edv_.n11 = 1.0;
              edv_.n12 = 1.0;
              edv_.n20 = 1.0;
              edv_.n21 = 1.0;
              edv_.n22 = 1.0;
              edv_.saredCount = 0;}
       if (old_ave < edp_.th_min && new_ave < edp_.th_min)
              edv_.n00 = edv_.n00 + 1.0;
       if (old_ave < edp_.th_min && (new_ave >= edp_.th_min && new_ave <edp_.th_max))
              edv_.n01 = edv_.n01 + 1.0;
       if (old_ave < edp_.th_min && new_ave >= edp_.th_max)
              edv_.n02 = edv_.n02 + 1.0;
       if ((old_ave >= edp_.th_min && old_ave < edp_.th_max) && new_ave < edp_.th_min)
              edv_.n10 = edv_.n10 + 1.0;
       if ((old_ave >= edp_.th_min && old_ave < edp_.th_max) && (new_ave >= edp_.th_min &&
new_ave <edp_.th_max))
              edv_.n11 = edv_.n11 + 1.0;
       if ((old_ave >= edp_.th_min && old_ave < edp_.th_max) && new_ave >= edp_.th_max)
              edv_.n12 = edv_.n12 + 1.0;
       if (old_ave >= edp_.th_max && new_ave < edp_.th_min)
              edv_.n20 = edv_.n20 + 1.0;
       if (old_ave >= edp_.th_max && (new_ave >= edp_.th_min && new_ave <edp_.th_max))
              edv_.n21 = edv_.n21 + 1.0;
       if (old_ave >= edp_.th_max && new_ave >= edp_.th_max)
              edv_.n22 = edv_.n22 + 1.0;
       edv_.n0 = edv_.n00+edv_.n01+edv_.n02;
       edv_.n1 = edv_.n10+edv_.n11+edv_.n12;
       edv_.n2 = edv_.n20+edv_.n21+edv_.n22;
       edv_.p00 = edv_.n00/edv_.n0;
       edv_.p01 = edv_.n01/edv_.n0;
       edv_.p02 = edv_.n02/edv_.n0;
       edv_.p10 = edv_.n10/edv_.n1;
       edv_.p11 = edv_.n11/edv_.n1;
       edv_.p21 = edv_.n21/edv_.n1;
       edv_.p20 = edv_.n20/edv_.n2;
       edv_.p21 = edv_.n21/edv_.n2;
       edv_.p22 = edv_.n22/edv_.n2;
       edv_.p0 = edv_.p00+edv_.p01+edv_.p02;
       edv_.p1 = edv_.p10+edv_.p11+edv_.p12;
       edv_.p2 = edv_.p20+edv_.p21+edv_.p22;
       if (now > edv_.lastset + edp_.interval)
              updateMaxPSARED(new_ave, now);}
```

```
void REDQueue::updateMaxPSARED(double ave, double now)
{
	double part = 0.4*(edp_.th_max - edp_.th_min);
	// AIMD rule to keep target Q~1/2(th_min+th_max)
	int th_min = edp_.th_min;
	int th_max = edp_.th_max;
	if (ave <= th_min+part) {
		if (edv_.cur_max_p <= 0.01) {
			edp_.alpha=edv_.p01*(edv_.p0+edv_.p2); edp_.beta=edv_.p00*edv_.p1;
			edv_.cur_max_p=edv_.cur_max_p*edp_.beta+edp_.alpha; edv_.lastset=now;
		} else if (edv_.cur_max_p >= 0.5) {
			edp_.beta=(edv_.p22-edv_.p21-edv_.p10)*edv_.p1;
			edv_.cur_max_p=edv_.cur_max_p*edp_.beta; edv_.lastset=now;
		} else {
			edp_.beta=(edv_.p11-edv_.p10)*edv_.p1;
			edv_.cur_max_p=edv_.cur_max_p*edp_.beta; edv_.lastset=now;
		}
	} else if (ave>th_min+part && ave<th_max-part){
		if (edv_.cur_max_p <= 0.01){
			edp_.alpha=edv_.p01*(edv_.p0+edv_.p2);
			edv_.cur_max_p=edv_.cur_max_p+edp_.alpha; edv_.lastset=now;
		} else if (edv_.cur_max_p >= 0.5){
			edp_.beta=(edv_.p22-edv_.p21)*edv_.p1;
			edv_.cur_max_p=edv_.cur_max_p*edp_.beta; edv_.lastset=now;
		} else {
			edp_.alpha=edv_.p12*(edv_.p0+edv_.p2); edp_.beta=edv_.p11*edv_.p1;
			edv_.cur_max_p=edv_.cur_max_p*edp_.beta+edp_.alpha; edv_.lastset=now;
		}
	} else if (ave >= th_max-part){
		if (edv_.cur_max_p <= 0.01){
			edp_.alpha=(edv_.p01+edv_.p12)*(edv_.p0+edv_.p2);
			edv_.cur_max_p=edv_.cur_max_p+edp_.alpha; edv_.lastset=now;
		} else if (edv_.cur_max_p >= 0.5) {
			edp_.alpha=edv_.p21*(edv_.p0+edv_.p2); edp_.beta=edv_.p22*edv_.p1;
			edv_.cur_max_p=edv_.cur_max_p*edp_.beta+edp_.alpha; edv_.lastset=now;
		} else {
			edp_.alpha=edv_.p12*(edv_.p0+edv_.p2);
			edv_.cur_max_p=edv_.cur_max_p+edp_.alpha; edv_.lastset=now;
		}
	}
}
```

These changes were made according to the algorithm from the paper.

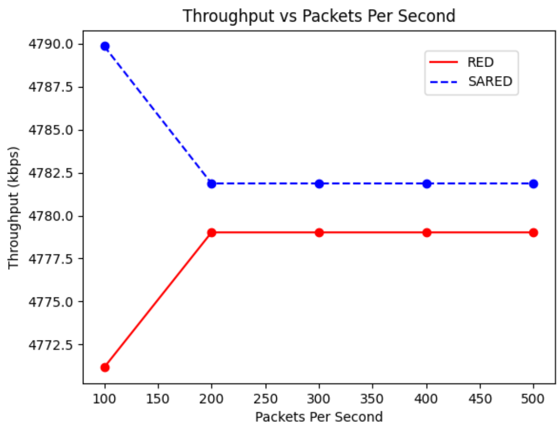
The following are further changes made for this project.

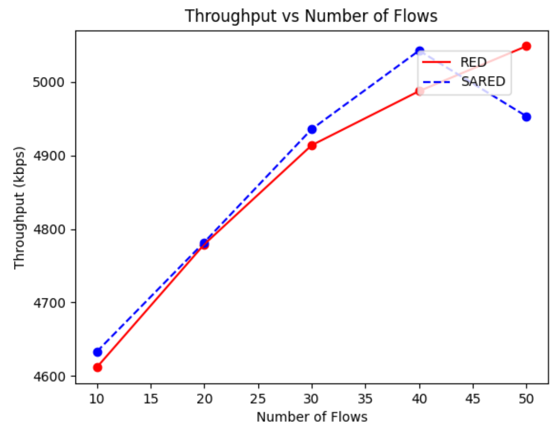
```
if(edp_.adaptive == 2){
	if(v_ave < edp_.th_min)
		p = 1 - edv_.p00;
	else if(v_ave >= edp_.th_max)
		p = v_c * v_ave + v_d;
	else {
		if(edv_.p10 > edv_.p11 && edv_.p10 > edv_.p12)
			p = 1 - edv_.p10;
		else{
			p = v_a * v_ave + v_b;
			p *= max_p;
		}

	}
}
```

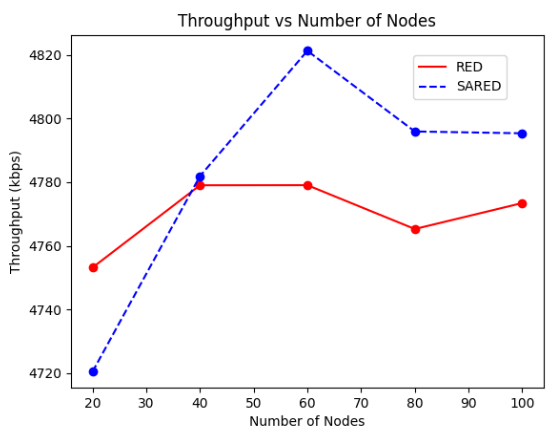
```
if(edv_.saredCount == edp_.saredLimit){
		edv_.n00 = 1.0;
		edv_.n01 = 1.0;
		edv_.n02 = 1.0;
		edv_.n10 = 1.0;
		edv_.n11 = 1.0;
		edv_.n12 = 1.0;
		edv_.n20 = 1.0;
		edv_.n21 = 1.0;
		edv_.n22 = 1.0;
		edv_.saredCount = 0;
}
```

Wired Results

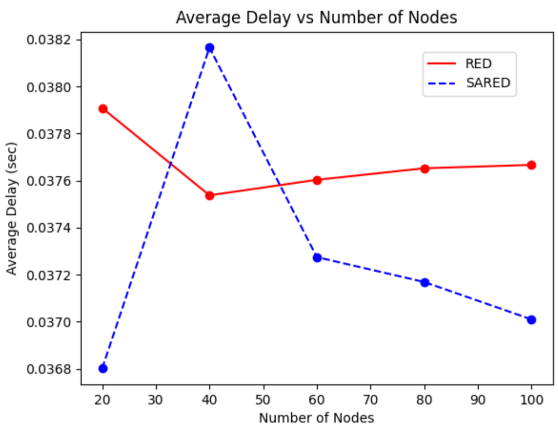
Throughput

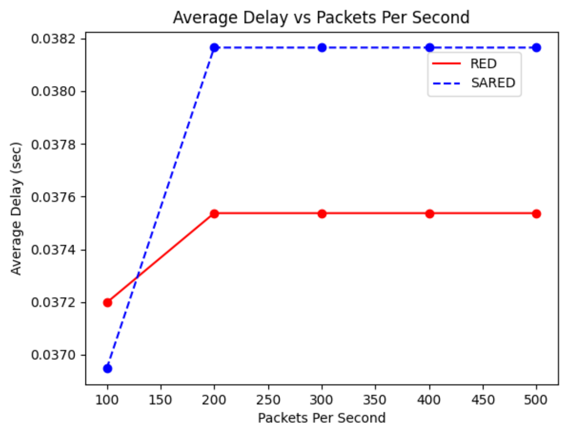


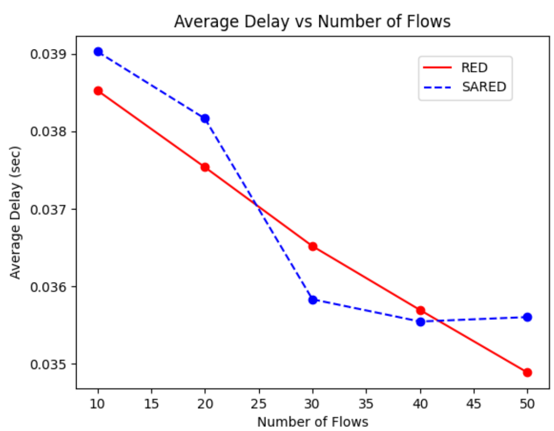




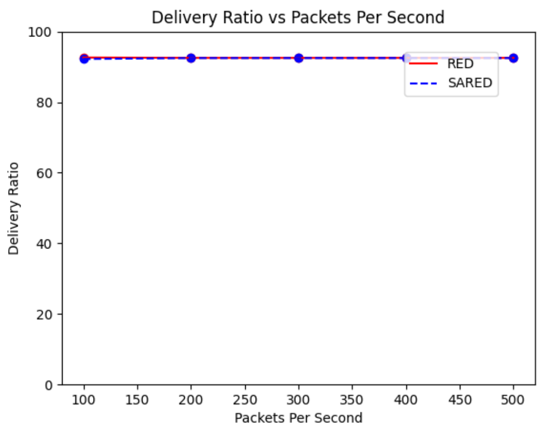
Average Delay

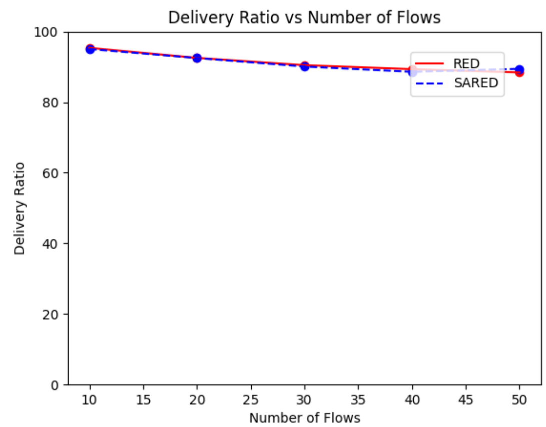


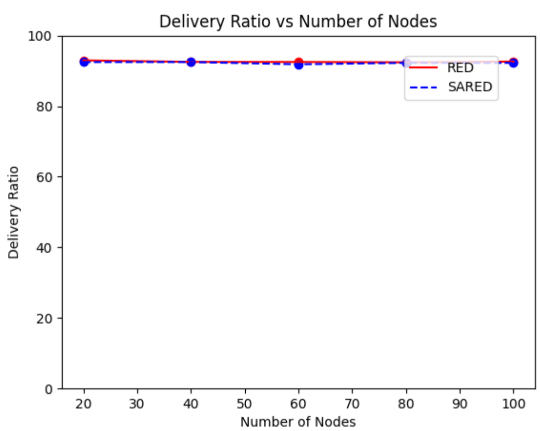




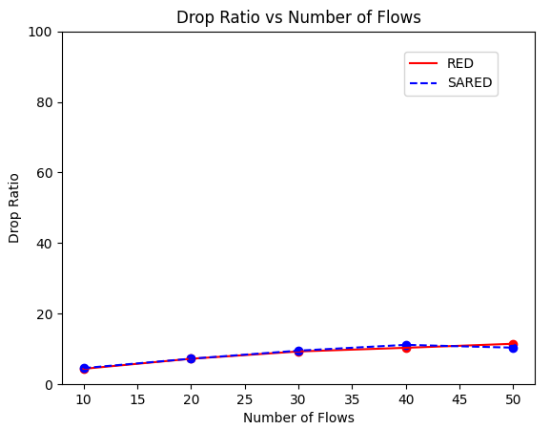
Delivery Ratio

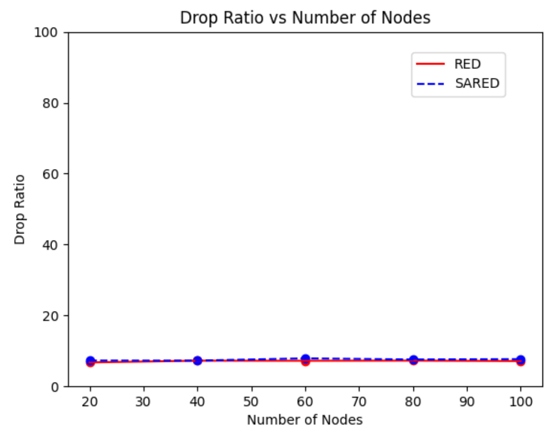


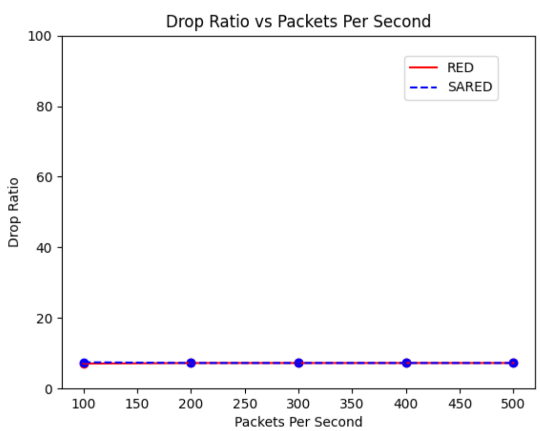




Drop Ratio

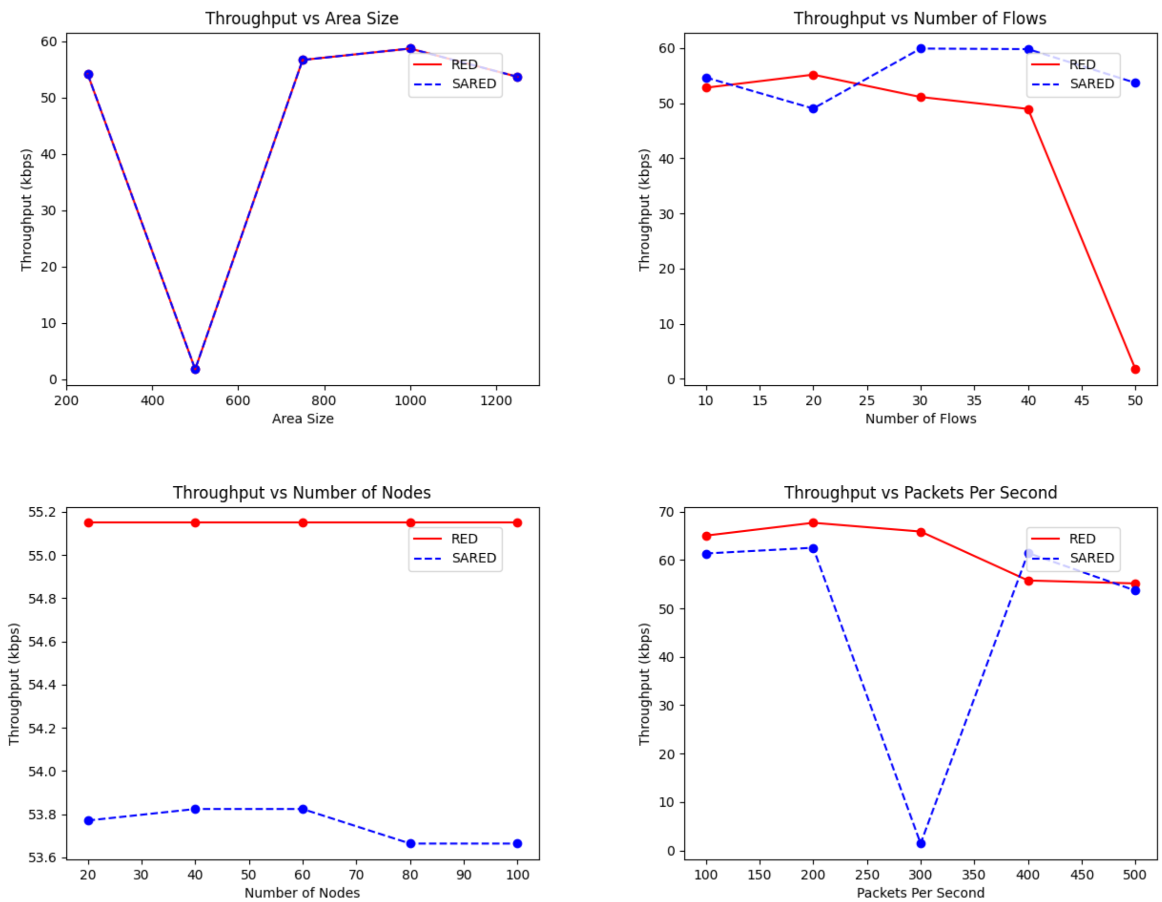

Drop Ratio vs Number of Flows
RED
SARED
Drop Ratio
Number of Flows
0
20
40
60
80
100
10
15
20
25
30
35
40
45
50


Drop Ratio vs Number of Nodes
RED
SARED
Drop Ratio
Number of Nodes
0
20
40
60
80
100
20
30
40
50
60
70
80
90
100

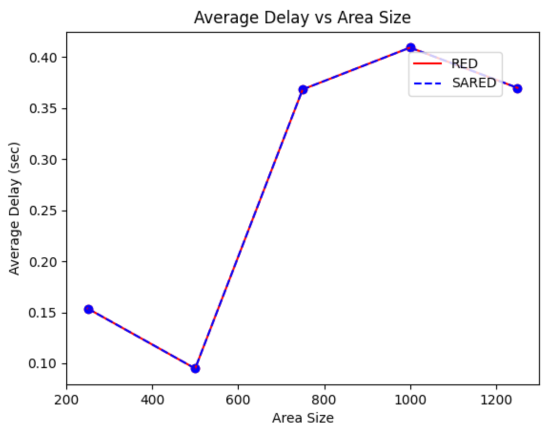

Drop Ratio vs Packets Per Second
RED
SARED
Drop Ratio
Packets Per Second
0
20
40
60
80
100
100
150
200
250
300
350
400
450
500

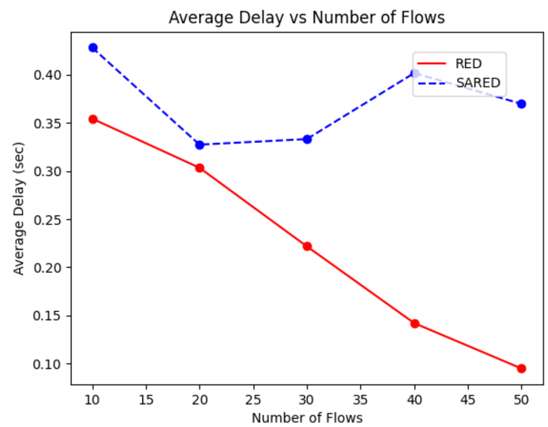
Wireless Results

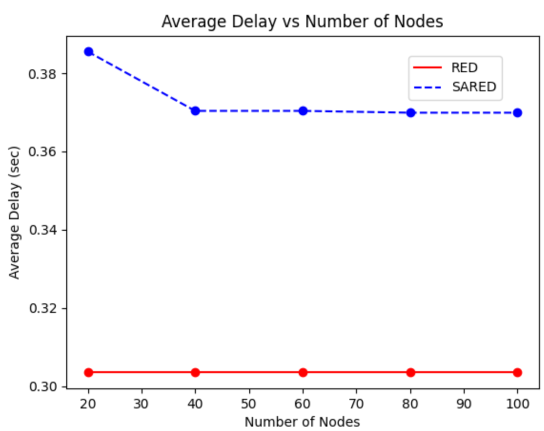
Throughput

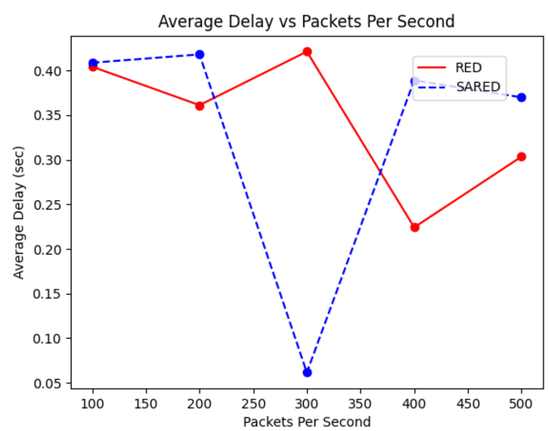


Average Delay

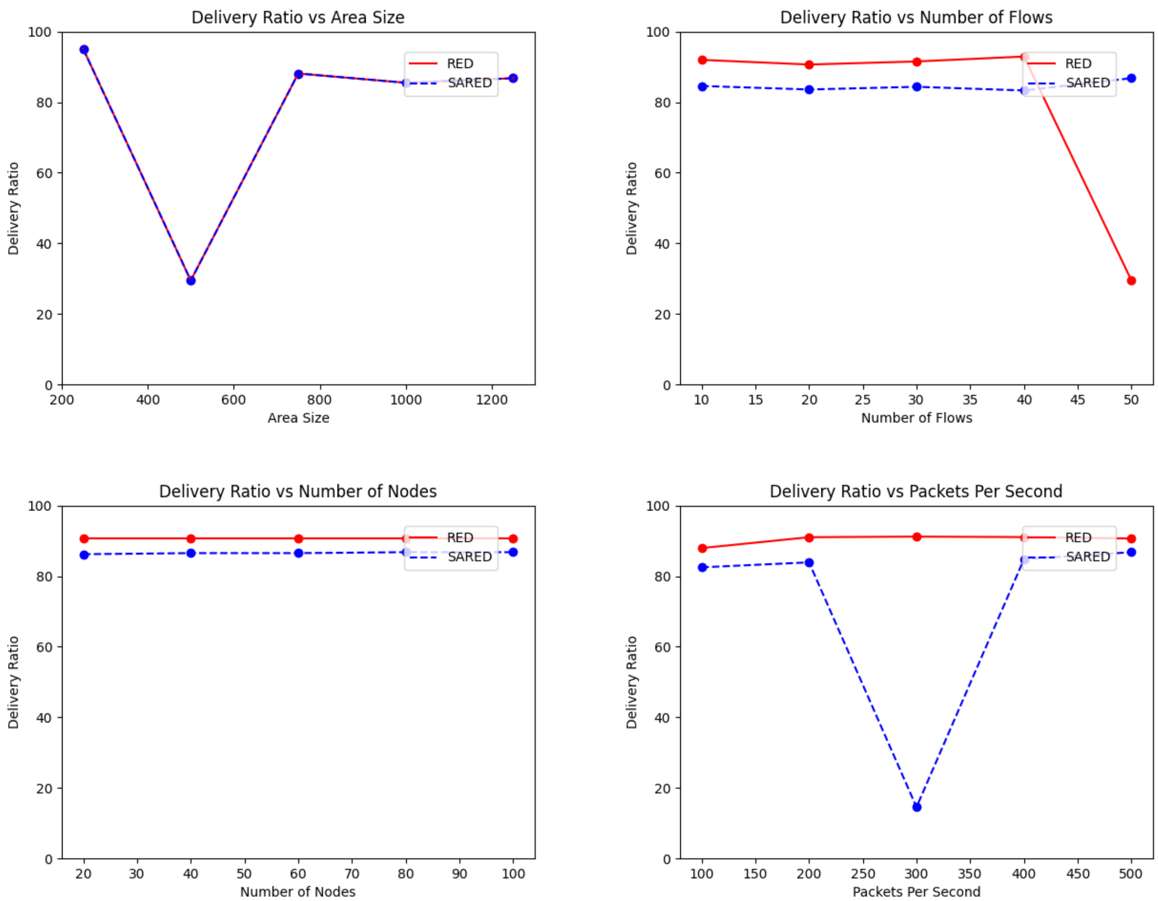




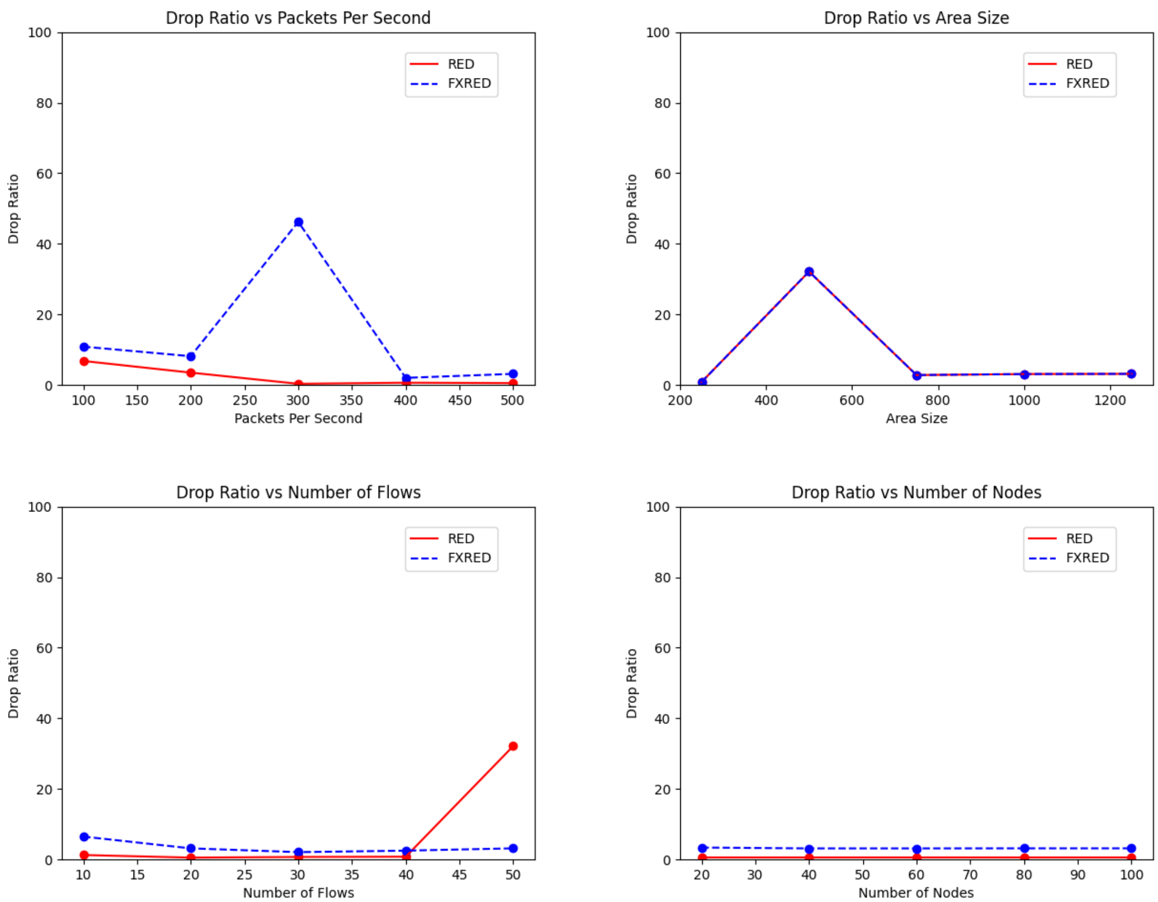




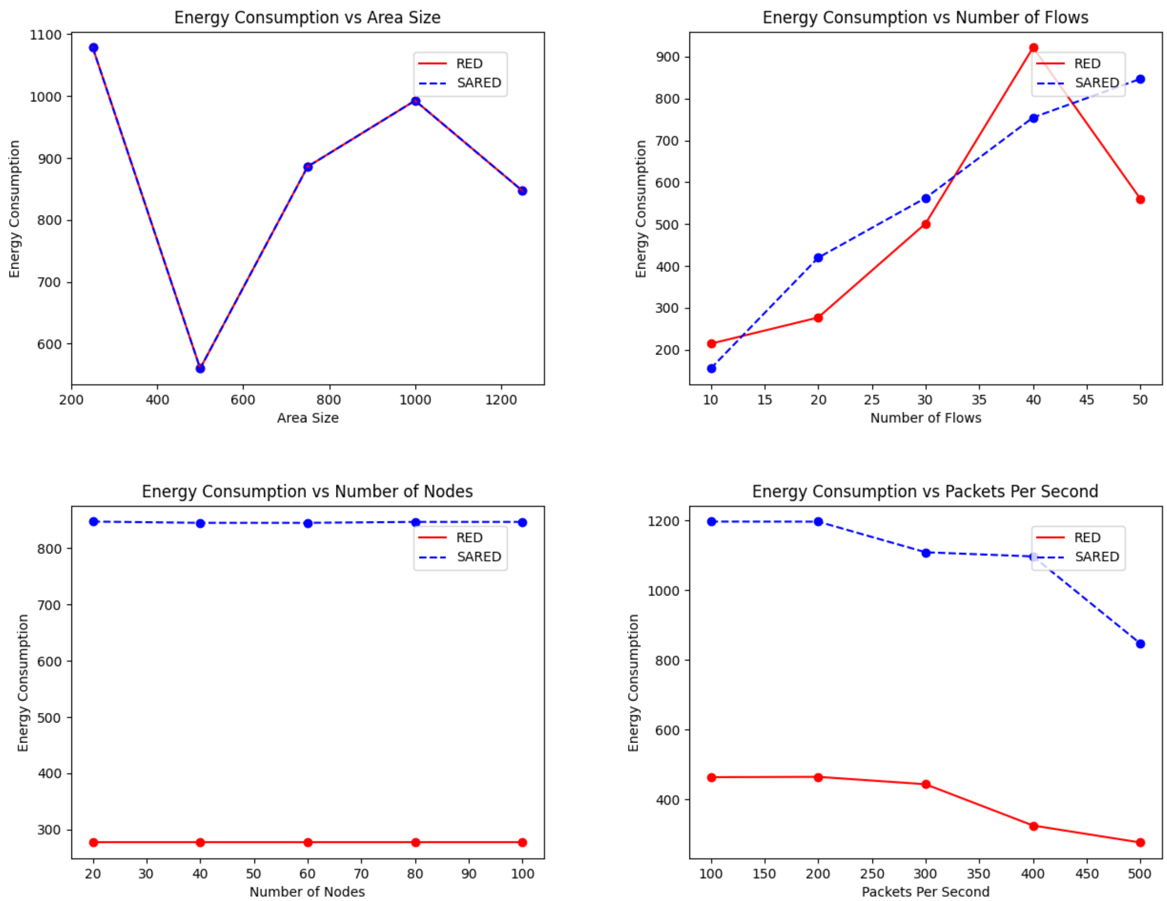
Delivery Ratio



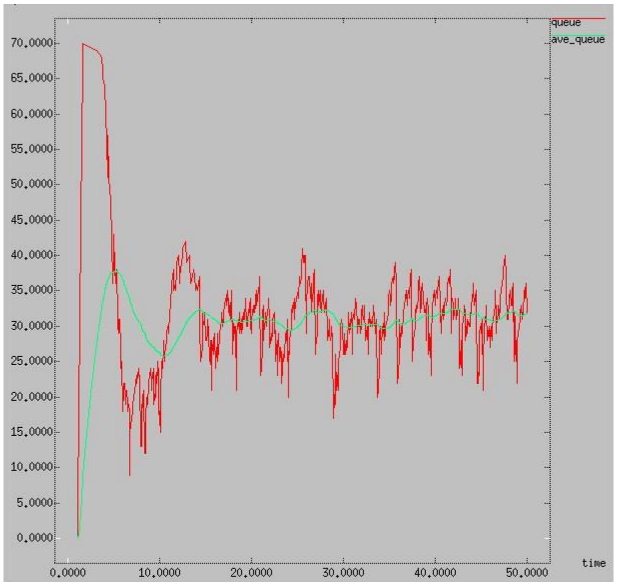
Drop Ratio


Drop Ratio vs Packets Per Second
Drop Ratio
Packets Per Second
RED
FXRED
Drop Ratio vs Area Size
Drop Ratio
Area Size
RED
FXRED
Drop Ratio vs Number of Flows
Drop Ratio
Number of Flows
RED
FXRED
Drop Ratio vs Number of Nodes
Drop Ratio
Number of Nodes
RED
FXRED

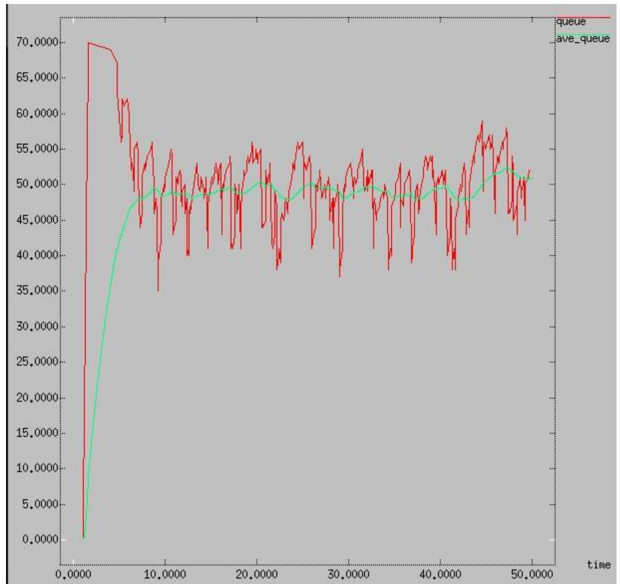
Energy Consumption



Queue Length and Average Queue Length



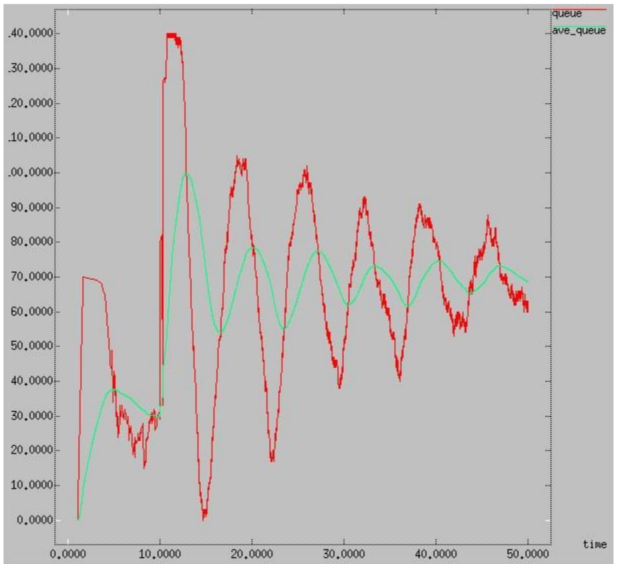

```
Throughput:         960656.98
AverageDelay:          0.29976
SentPackets:          6031.00
ReceivedPackets:          5889.00
DropPackets:          106.00
PacketDeliveryRatio:       97.65
PacketDropRatio:        1.76
Time:                48.99980
TotalDelay:        1765.28294
```



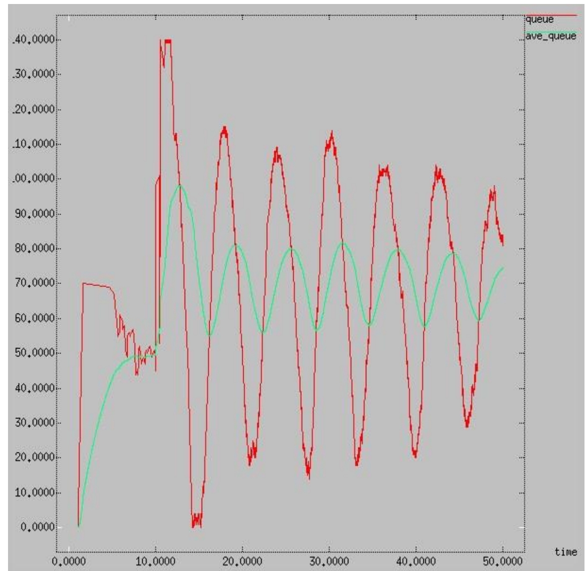

```
Throughput:         960656.98
AverageDelay:          0.45147
SentPackets:          5986.00
ReceivedPackets:          5889.00
DropPackets:           41.00
PacketDeliveryRatio:       98.38
PacketDropRatio:        0.68
Time:                48.99980
TotalDelay:        2658.70222
```

RED **SARED**

5 flows from beginning to end



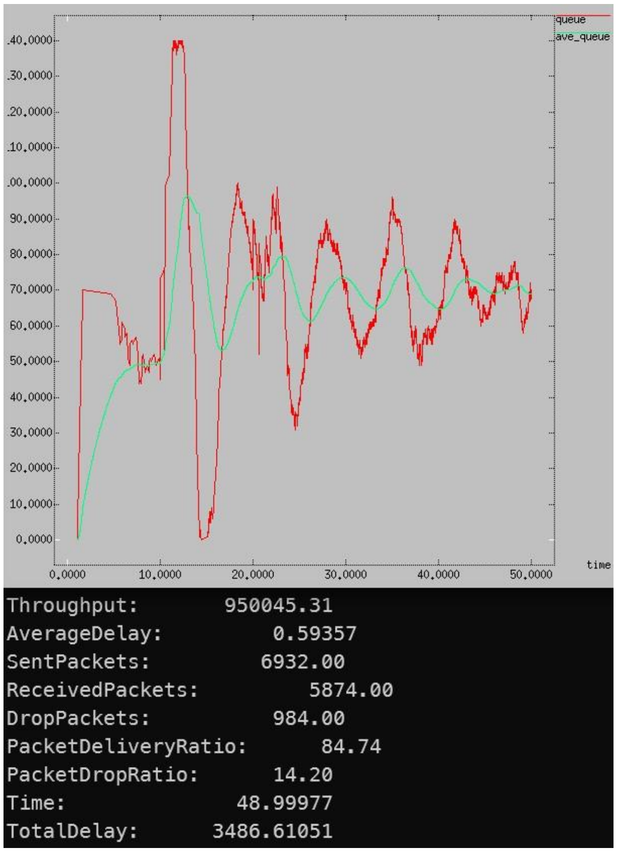
Throughput: 960333.27
AverageDelay: 0.56100
SentPackets: 7098.00
ReceivedPackets: 5937.00
DropPackets: 1097.00
PacketDeliveryRatio: 83.64
PacketDropRatio: 15.46
Time: 48.99966
TotalDelay: 3330.63050

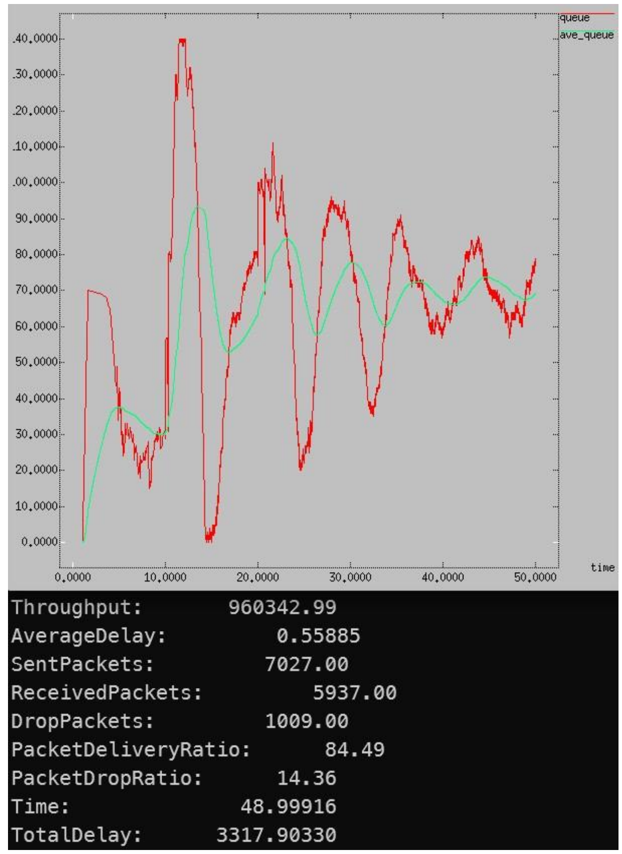


Throughput: 959030.27
AverageDelay: 0.58529
SentPackets: 7039.00
ReceivedPackets: 5929.00
DropPackets: 1025.00
PacketDeliveryRatio: 84.23
PacketDropRatio: 14.56
Time: 48.99950
TotalDelay: 3470.19717

RED **SARED**

5 flows at the beginning and 50 flows at 10 seconds





SARED **RED**

5 flows at the beginning and 25 more flows at 10 seconds then 25 more flows at 20 seconds

Observations

From the first wired simulations we can see that SARED improves over RED in throughput. But no definite answer can be given about average delay. The changes in drop ratio and delivery ratio is miniscule.
From the wireless simulations we can see that the energy consumption increases in SARED. No definite answer can be given about the other parameters as there is much variation in the results.
From the last queue length graphs, we can see that SARED works better than RED when the data rate is relatively low by letting the queue grow larger when data rate is low. But works about the same as RED when data rate is higher as the queue length quickly grows too close to th_max. But the improvements come at a cost as the average delay is much higher in SARED. So the negatives might outweigh the positives.