Course : CSE322

Project on Network Simulator 2

An Improved Adaptive Active Queue Management Algorithm Based on Nonlinear Smoothing
Jingjun Zhang, Wenlong Xub , Liguo Wang (2011)

Supervisor:
Dr. Md. Shohrab Hossain
Professor, CSE, BUET

Presented By:
Iftekhar E Mahbub Zeeon - 1805038
Level-3 Term-2
Department of CSE
Bangladesh University of Engineering & Technology

Contents

1 Introduction

An improved Adaptive RED congestion control mechanism named Nonlinear Adaptive RED algorithm is proposed in this paper, which imposes nonlinear smooth for packet loss rate function of the RED algorithm by using the membership function of the ascend demi-cauchy of fuzzy distribution. In this project, some modifications were made to the existing RED algorithm following this paper.

2 Network Topologies Under Simulation

There are two topologies that were implemented:

- Wireless MAC type 802.11(Static) topology with random sources and random destinations
- Simple dumbbell Wired topology

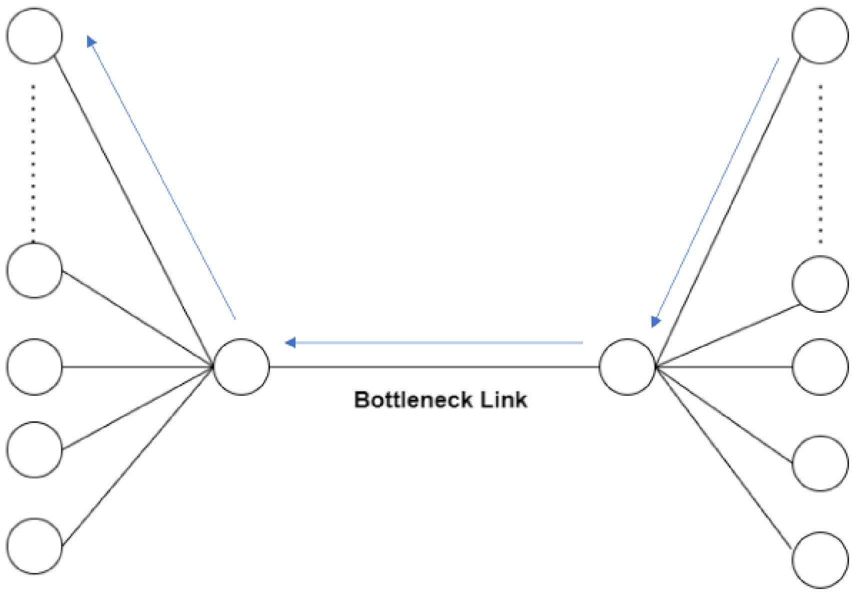


Figure 1: Wired Topology

3 Parameters Under Variation

Wireless MAC type 802.11 (Static):

- Coverage Area Size
- Number of Nodes
- Number of Flows
- Number of Packets per second

Wired:

- Number of Nodes
- Number of Flows
- Number of Packets per second

4 Metrics Under Consideration

Wireless MAC type 802.11 (Static):

- Network Throughput
- End-to-End Average Delay
- Packet Delivery Ratio
- Packet Drop Ratio
- Energy Consumption

Wired:

- Network Throughput
- End-to-End Average Delay
- Packet Delivery Ratio
- Packet Drop Ratio

5 Overview of Algorithm

NARED (Non-linear Adaptive RED) is an improved RED(Random Early Detection) algorithm.

- **Non-Linear**: The main idea is that when Q_{avg} exceeds Q_{max}, a part of the queue is idle, so the probability should not be immediately changed to 1 like RED algorithm.
- **Adaptive**: The value of P_{max} will be dynamically adjusted by using the average queue length and the target of queue length.

Steps of Algorithm:

1. The following formula is the function of the ascend demi-cauchy of fuzzy distribution.

$$\mu_a(u) = \begin{cases} 0 & u \leq \alpha \\ \frac{a(u-\alpha)^\beta}{1+(u-\alpha)^\beta} & u > \alpha, a > 0, \beta > 0 \end{cases} \tag{1}$$

2. After simulation, the best value of β is 3. u is taken as Q_{avg}. α is Q_{min}.The following formula will be obtained by putting these three values into formula (1) and the function of the packet dropping probability of RED.

$$p_b = \begin{cases} 0 & Q_{avg} \leq Q_{min} \\ \frac{a(Q_{avg}-Q_{min})^3}{1+a(Q_{avg}-Q_{min})^3} & Q_{min} < Q_{avg} < bufsize \\ 1 & Q_{avg} \geq bufsize \end{cases} \tag{2}$$

3. Calculate a when Q_{avg} is Q_{max} and the value of packet loss rate is p_{max},

$$a = \frac{p_{max}}{(1-p_{max})(Q_{max}-Q_{min})^3} \tag{3}$$

4. The distribution function of the probability of packet drop will be obtained by putting the value of the parameter a into the formula(2).

$$p_b = \begin{cases} 0 & Q_{avg} \leq Q_{min} \\ \frac{p_{max}(Q_{avg}-Q_{min})^3}{(1-p_{max})(bufsize-Q_{max})^3+p_{max}(Q_{avg}-Q_{min})^3} & Q_{min} < Q_{avg} < bufsize \\ 1 & Q_{avg} \geq bufsize \end{cases} \tag{4}$$

5. The realization of packet dropping strategy of NARED algorithm is as follows:

   (a) Calculating the average queue length if the queue is not free,

   $$Q_{avg} = (1 - w_q) \times Q_{avg} + q \times w_q \tag{5}$$

   (b) If $Q_{avg} \leq Q_{min}$, then $p = 0$

   (c) If $Q_{min} < Q_{avg} < bufsize$,

   - if $(Q_{avg} < k_1)$, then $p_{max} = p_{max} \times b$
   - if $(Q_{avg} > k_2)$, then $p_{max} = p_{max} + a$
   - Calculate p_b using equation (4)
   - $p = \frac{p_b}{1 - p_b \times count}$

   (d) if $Q_{avg} > bufsize$, then $p = 1$

6. Done

6 Modifications Made in NS2

1. **File: queue/red.h**

   (a) Introducing a new integer variable named *is_modifiedRed* in struct *edp*. This indicates whether the queue management system is RED or NARED now.

```
int is_modifiedRed; /* 0 for default RED, 1 for
   modified RED (NARED) */

```

   (b) Introducing another new integer variable named *buffer_size* in struct *edp* for NARED.

```
int buffer_size; /* New buffer size, if modified red is
    on */

```

(c) In struct *edv*, declaring a function *calculate_p_modifiedRed*:

```
double calculate_p_modifiedRed(double v_ave, double
    th_max, double th_min, double count, double
    max_p_inv, double buffer_size);

```

2. **File: queue/red.cc**

(a) Initializing the variables in *REDQueue::REDQueue* constructor.

```
bind("modified_red_", &edp_.is_modifiedRed); //Is NARED
     working or not
bind("buffer_size", &edp_.buffer_size); // Buffer size
    for NARED

```

(b) Initializing the variables in the function *REDQueue::initParams*:

```
edp_.is_modifiedRed = 0;
edp_.buffer_size = 0;

```

(c) Modifying the function *REDQueue::estimator* to calculate average queue size:

```
if (edp_.is_modifiedRed) {
    new_ave = ((1.0 - q_w) * new_ave) + (q_w * nqueued)
    ;
} else {
    //Code for Default RED
}

```

(d) Adding a new function *REDQueue::calculate_p_modifiedRed* to calculate the probability:

```
/*
* Calculate the drop probability for modified RED
*/
double
REDQueue::calculate_p_modifiedRed(double v_ave, double
    th_max, double th_min, double count, double max_p,
    double buffer_size)
{
    if (v_ave < th_min) {
        p = 0.0;
    } else if (v_ave >= buffer_size) {
        p = 1.0;
    } else {
        double k1 = th_min + 0.4 * (buffer_size -
    th_min);
        double k2 = th_min + 0.6 * (buffer_size -
    th_min);
        double b = 0.9;
        double a = 0.01;
        if (a > max_p/4) {
            a = max_p/4;
        }
        if (v_ave < k1) {
            max_p = max_p * b;
        } else if (v_ave > k2) {
            max_p = max_p + a;
        }
        edv_.cur_max_p = max_p;

        double temp_p;
        temp_p = (max_p * pow((v_ave - th_min),3)) /
    ((1 - max_p) * pow((buffer_size - th_min), 3) +
    max_p * pow((v_ave - max_p), 3));

        p = temp_p / (1 - (count * temp_p));

    }

    if (p > 1.0)
        p = 1.0;
    return p;
}

```

(e) Modifying the function *REDQueue::calculate_p* to implement the algorithm:

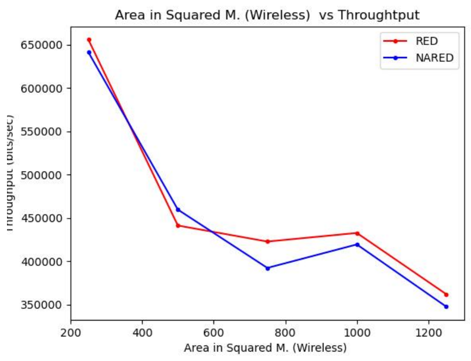
```
if (edp_.is_modifiedRed) {
    //Modified RED is running
    p = calculate_p_modifiedRed(v_ave, th_max, edp_.
   th_min, edv_.count, max_p_inv, edp_.buffer_size);
} else {
    //Code for Default RED
}

```

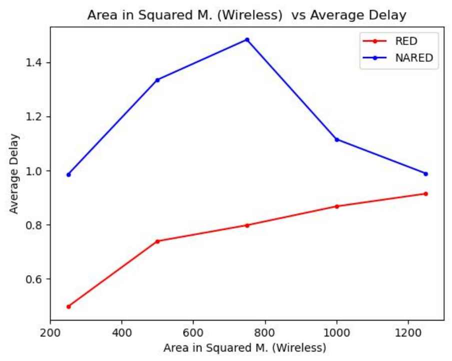
7 Results with Graphs

7.1 Wireless MAC type 802.11 (Static)

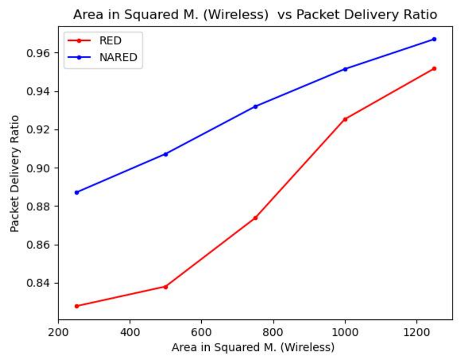
7.1.1 Varying Coverage Area



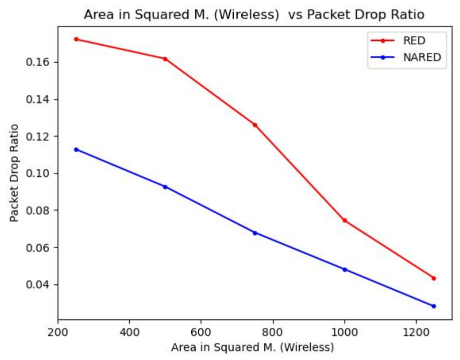
(a) Network Throughput



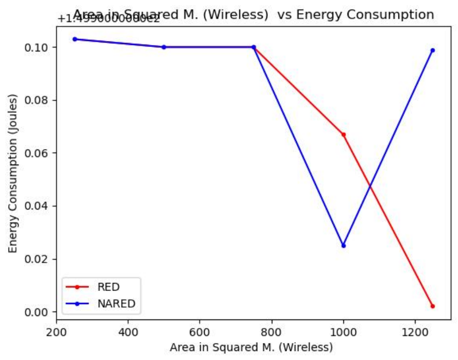
(b) End-to-End Average Delay



(c) Packet Delivery Ratio



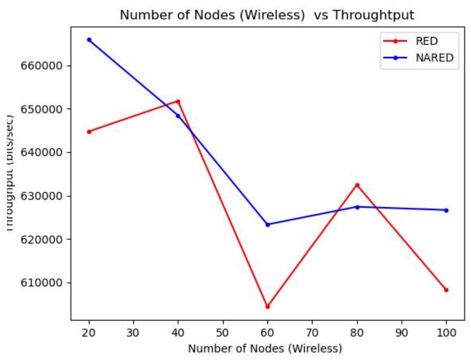
(d) Packet Drop Ratio



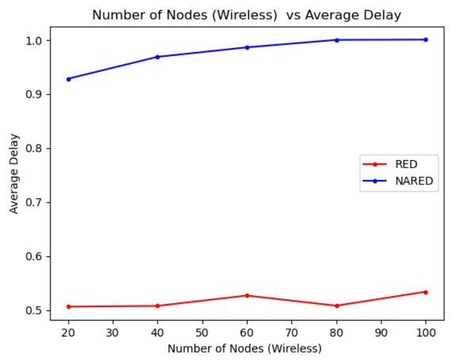
(e) Energy Consumption

Figure 2: Varying Coverage Area

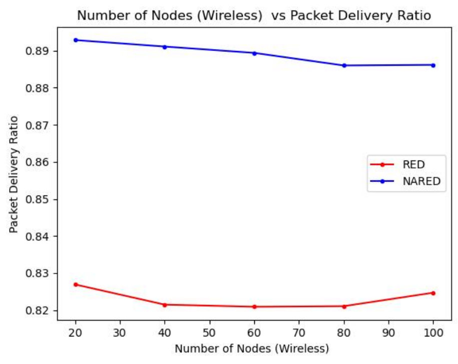
7.1.2 Varying Number of Nodes



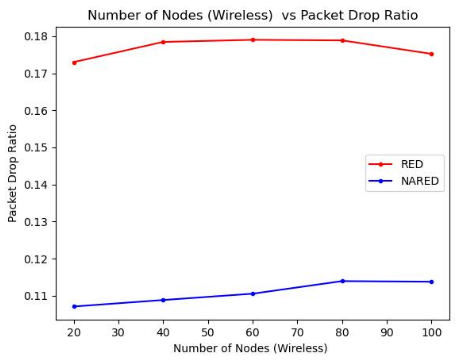
(a) Network Throughput



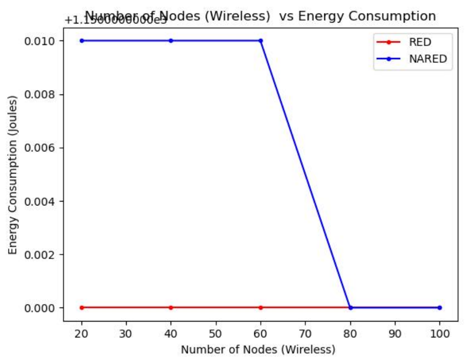
(b) End-to-End Average Delay



(c) Packet Delivery Ratio



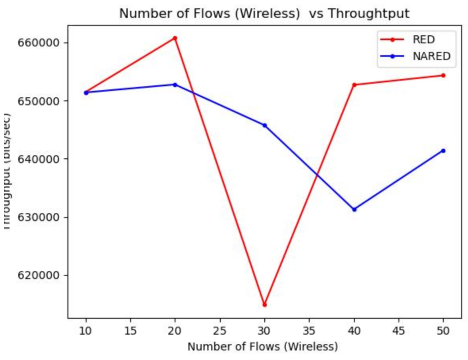
(d) Packet Drop Ratio



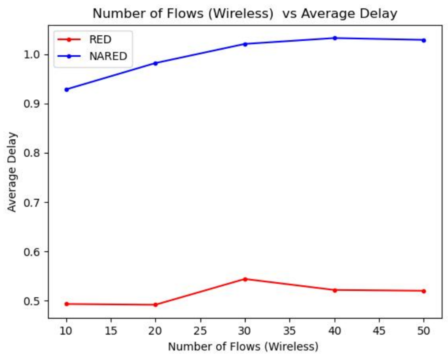
(e) Energy Consumption

Figure 3: Varying Number of Nodes

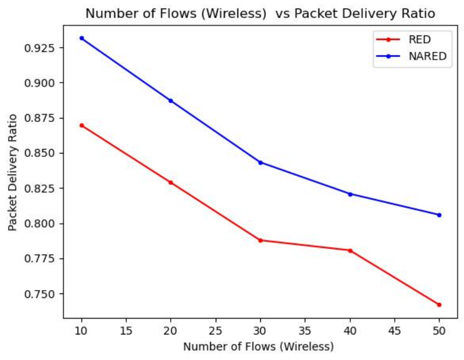
7.1.3 Varying Number of Flows



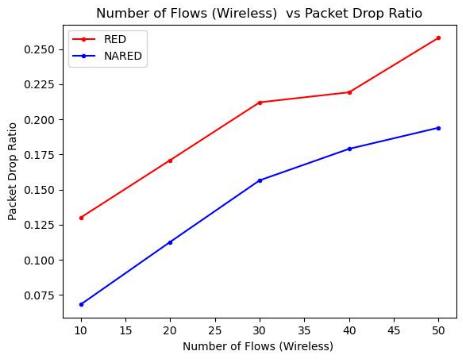
(a) Network Throughput



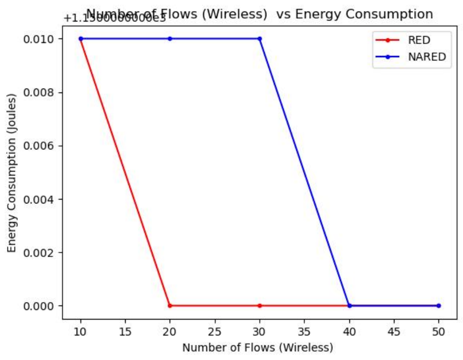
(b) End-to-End Average Delay



(c) Packet Delivery Ratio



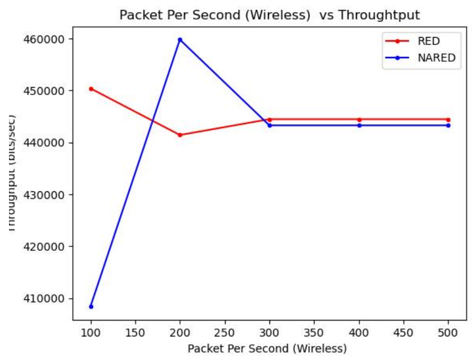
(d) Packet Drop Ratio



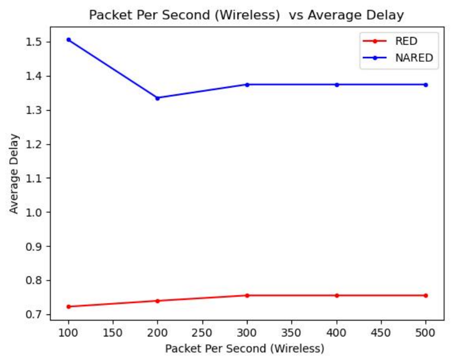
(e) Energy Consumption

Figure 4: Varying Number of Flows

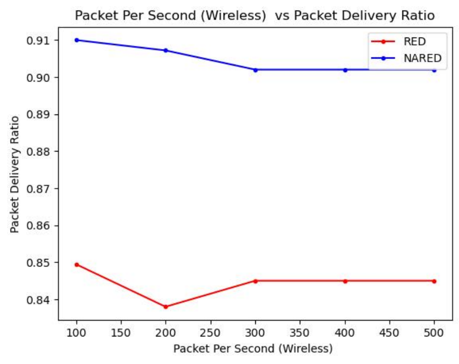
7.1.4 Varying Packet Rate



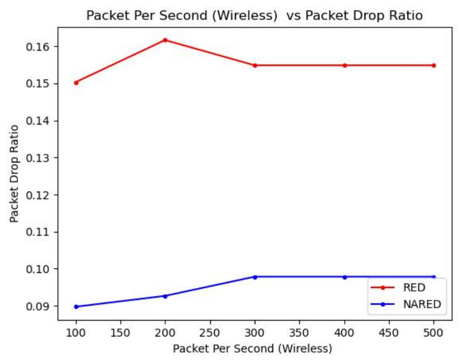
(a) Network Throughput



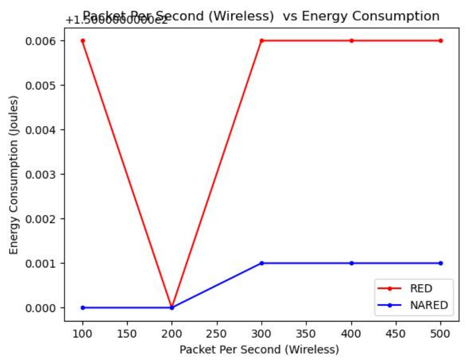
(b) End-to-End Average Delay



(c) Packet Delivery Ratio



(d) Packet Drop Ratio

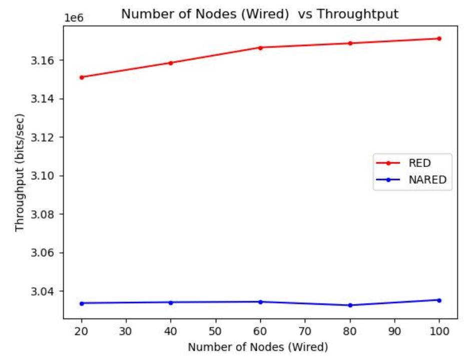


(e) Energy Consumption

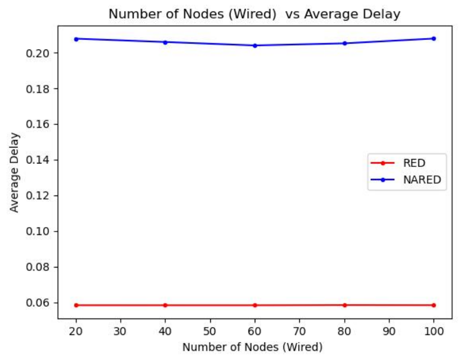
Figure 5: Varying Packet Rate

7.2 Wired

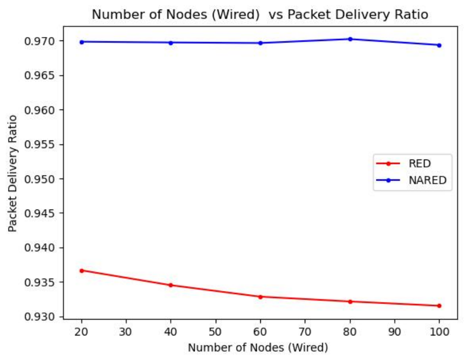
7.2.1 Varying Number of Nodes



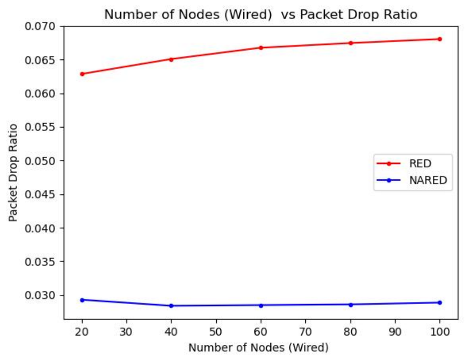
(a) Network Throughput



(b) End-to-End Average Delay



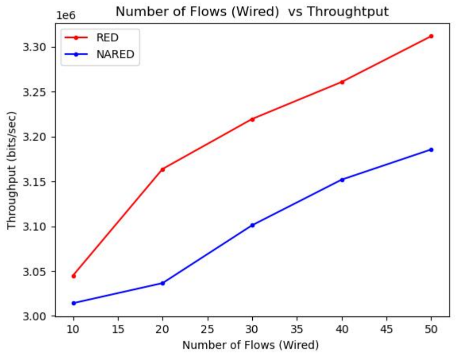
(c) Packet Delivery Ratio



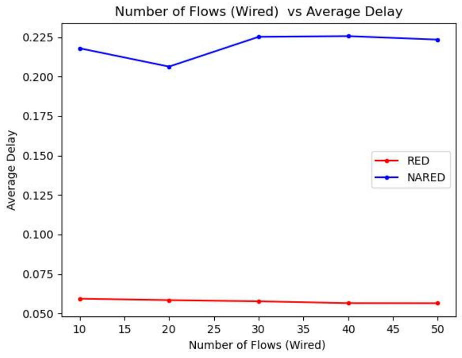
(d) Packet Drop Ratio

Figure 6: Varying Number of Nodes

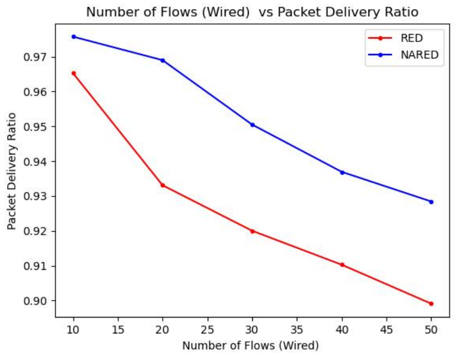
7.2.2 Varying Number of Flows



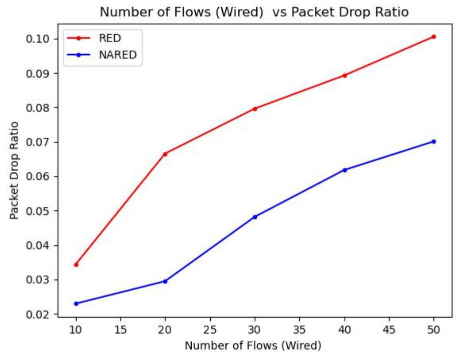
(a) Network Throughput



(b) End-to-End Average Delay



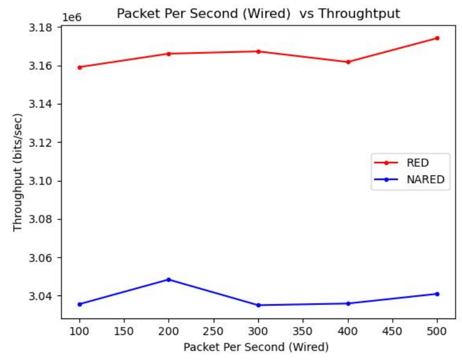
(c) Packet Delivery Ratio



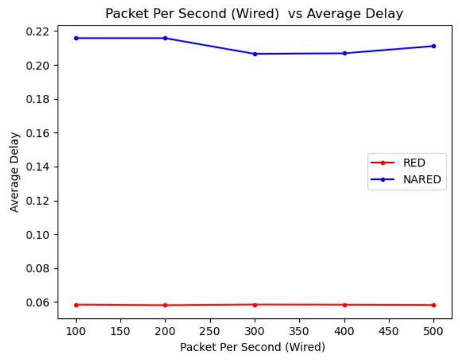
(d) Packet Drop Ratio

Figure 7: Varying Number of Flows

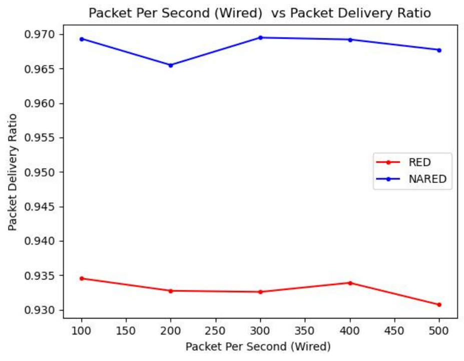
7.2.3 Varying Packet Rate



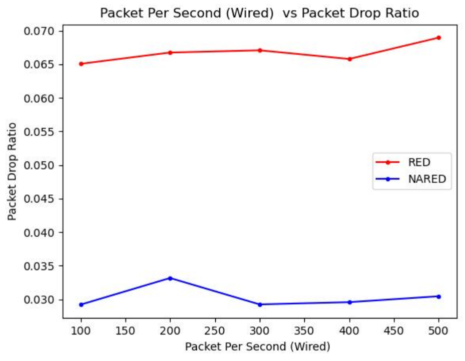
(a) Network Throughput



(b) End-to-End Average Delay



(c) Packet Delivery Ratio



(d) Packet Drop Ratio

Figure 8: Varying Packet Rate

8 Observation

The main difference between the modified RED I implemented and the default RED consists of that during the default RED, the probability of dropping becomes 1 whenever the average queue length reaches the maximum threshold. Here I have implemented such an algorithm where the average length does not move linearly, but rather non-linearly, and the maximum threshold is now changed to a new parameter buffer size. The paper I took as a reference mainly focused on the wired topology and how to decrease the drop ratio. In my both wired and wireless cases, the drop ratio was improved. However, in the case of other parameters, it was pretty random. In general, throughput was improved a bit as can be seen from the graphs. I ran all the simulations for 150 seconds and used 1000 J as the initial energy for the nodes. In TCP, it is not recommended to change the packet rate, hence it was not easy to vary the packet rate. I tried to change it by setting the window size, but the result was not that much noticeable. The topology I used for wired was suitable for my project as the router nodes were tending to drop packets. Overall, I am happy with my outcome that the modification improved the drop ratio significantly for both wired and wireless for every parameter.