Course : CSE322

Project on Network Simulator 2

HRED: An Active Queue Management Algorithm for TCP Congestion Control

Nabhan Hamadneh, Mamoon Obiedat,
Ahmad Qawasmeh, Mohammad Bsoul

Supervisor:

Navid Bin Hasan
Lecturer, CSE, BUET

Presented By:

Fabiha Tasneem - 1805072
Level-3 Term-2
Department of CSE
Bangladesh University of Engineering & Technology

Contents

1 Introduction

Random Early Detection (RED) is an Active Queue Management strategy that keeps a history of queue dynamics by estimating an average queue size parameter and drops packets when this average exceeds preset thresholds. The parameter configuration in RED is problematic and the performance of the whole network could be reduced due to the wrong setup of these parameters.
In this project, a modification in the existing Random Early Detection(RED) algorithm for Active Queue Management (AQM) has been made in NS2 which is named as Half-Way Random Early Detection(HRED). The modifications were made following this paper.

2 Network Topologies Under Simulation

There are two topologies that were implemented:

- A random sources-random destinations Wireless MAC type 802.11 (Mobile) topology
- A simple Wired topology with one source, one router, and multiple destinations

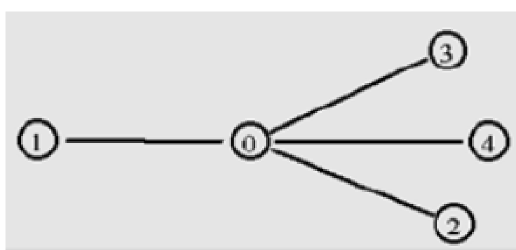


Figure 1: Wired Topology

3 Parameters Under Variation

Wireless MAC type 802.11 (Mobile):

- Area Size
- Number of Nodes
- Number of Flows
- Packet Rate
- Speed

Wired:

- Number of Nodes
- Number of Flows
- Packet Rate

4 Metrics Under Consideration

Wireless MAC type 802.11 (Mobile):

- 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

RED maintains an Exponentially Weighted Moving Average (EWMA) of the queue size on routers.The drop rate p_a is an accumulative linear value which is calculated using current drop probability p_b.
While TD uses the actual queue size, RED drops packets depending on the average queue size (avg). The modification that HRED does to existing RED, is that it introduces a new point for checking, a *midpoint*. With this *midpoint* in consideration, HRED now will just update p_a when avg is between min_{th} and *midpoint*. This is the crucial point and heart of HRED.

- If avg is between the min_{th} and *midpoint* then packets are dropped with the probability p_a.
- If avg is between the *midpoint* and max_{th} then probability p_a is just updated to 0.5.
- If the average is greater than or equal to the max_{th} then RED drops packets with probability 1.0 and the packets will be dropped forcibly. In other words, RED will drop all arriving packets similarly as TD does in case of the full buffer.

Preset min_{th}, max_{th}, max_p, w_q
Set $avg = 0$, $midpoint = min_{th} + (max_{th} - min_{th})/2$
For every packet arrival update avg (Eq. 1)
IF ($avg \geq min_{th}$ && $avg < midpoint$) THEN
Calculate p_b(Eq. 2)
Calculate p_a(Eq. 3)
Drop arriving packets with probability p_a
ELSE IF ($avg = midpoint$) THEN
p_a=0.5
ELSE IF ($avg \geq midpoint$ && $avg < max_{th}$) THEN
Update p_a(Eq. 3)
ELSE IF ($avg \geq max_{th}$) THEN
p_a=1.0

$$avg = (1 - w_q) * avg + w_q * q \qquad (1)$$

$$p_b = max_p \left(\frac{avg - min_{th}}{min_{th} - max_{th}}\right) \qquad (2)$$

$$p_a = p_b \left(\frac{1}{1 - count * p_b}\right) \qquad (3)$$

Where is:

avg : Average queue size.

w_q : A weight parameter, $0 \leq w_q \leq 1$.

q: The current queue sizes.

p_b: Immediately marking probability.

max_p: Maximum value of p_b.

min_{th}: Minimum threshold.

max_{th}: Maximum threshold.

p_a: Accumulative probability.

$count$: number of undraped packets since the last dropped one.

6 Modifications made in NS2

1. **File: queue/red.h**
   Introducing a new integer variable named *isHRED* in struct ***edp***. This indicates whether the queue management system is RED or HRED now. $isHRED = 0$, means normal RED and $isHRED = 1$ means HRED.

```
        int isHRED;    /* 1 if HRED, 0 if RED */

```

   Declaring a function ***calculate_p_a*** to calculate the value of p_a and another function ***calculate_p_b*** to calculate the value of p_b:

```
double calculate_p_b(double v_ave, double th_max, double
    th_min, double max_p); //for HRED
double calculate_p_a(double p_b); //for HRED

```

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

   Initializing the variable *isHRED* in ***REDQueue::REDQueue*** constructor.

```
    bind("isHRED_", &edp_.isHRED);

```

   Adding a function ***calculate_p_b*** to calculate the value of p_b. Here some modifications were brought from the paper. The paper provides the following formula.

$$p_b = max_p * (v_{ave} - th_{min})/(th_{min} - th_{max}) \tag{1}$$

   But as this formula provides a negative probability, hence we changed the equation to

$$p_b = max_p * (v_{ave} - th_{min})/(th_{max} - th_{min}) \tag{2}$$

   :

```
double REDQueue::calculate_p_b(double v_ave, double th_max,
    double th_min, double max_p)
{
    double p;
    p = max_p * (v_ave - th_min) / (th_max - th_min);
    return p;
}
```

Adding a function ***calculate_p_a*** to calculate the value of p_a. Here we again modified a little bit from the paper and took the absolute value of p_a so that the probability is always positive:

```
double REDQueue::calculate_p_a(double p_b)
{
    double p;
    p = p_b * (1 / (1 - (edv_.count * p_b)));
    if(p < 0)
         p = p * -1;
    return p;
}
```

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

```
if(edp_.isHRED == 0)       // normal RED
{
    while (--m >= 1)
    {
        new_ave *= 1.0 - q_w;
    }
    new_ave *= 1.0 - q_w;
    new_ave += q_w * nqueued;
}
else if(edp_.isHRED == 1)
{              // HRED
    new_ave *= 1.0 - q_w;
    new_ave += q_w * nqueued;
}
```

Introducing a new variable named ***midpoint*** to implement the algorithm:

```
double midpoint = edp_.th_min + ((edp_.th_max - edp_.th_min)
    / 2);

```

Changing the function ***REDQueue::enque*** to implement the algorithm:

```
else if (edp_.isHRED == 1 && edp_.th_min <= qavg && qavg <
   midpoint)
{
     if (drop_early(pkt))
     {
         droptype = DTYPE_UNFORCED;
     }
}
else if (edp_.isHRED == 1 && qavg == midpoint)
{
     edv_.v_prob = 0.5;
}
else if (edp_.isHRED == 1 && qavg >= midpoint && qavg <=
   edp_.th_max)
{
     edv_.v_prob = calculate_p_a(edv_.v_prob1);
}
else if (edp_.isHRED == 1 && qavg >= edp_.th_max)
{
     edv_.v_prob = 1.0;
}

```

Modifying the function ***REDQueue::drop_early*** to implement the algorithm:

```
int REDQueue::drop_early(Packet *pkt)
{
hdr_cmn *ch = hdr_cmn::access(pkt);

if (edp_.isHRED == 0)      //calculate this only if it's
   normal RED
```

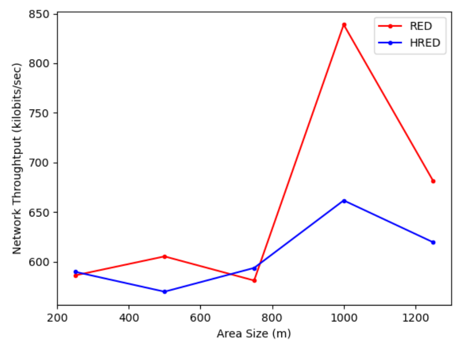
```
{
    edv_.v_prob1 = calculate_p_new(edv_.v_ave, edp_.th_max, edp_.gentle, edv_.v_a, edv_.v_b, edv_.v_c, edv_.v_d, edv_.cur_max_p);
    edv_.v_prob = modify_p(edv_.v_prob1, edv_.count, edv_.count_bytes,
    edp_.bytes, edp_.mean_pktsize, edp_.wait, ch->size());
}
else
{
    //HRED
    edv_.v_prob1 = calculate_p_b(edv_.v_ave, edp_.th_max, edp_.th_min, edv_.cur_max_p);//(step 4)
    edv_.v_prob = calculate_p_a(edv_.v_prob1);    //(step 5)
}

```

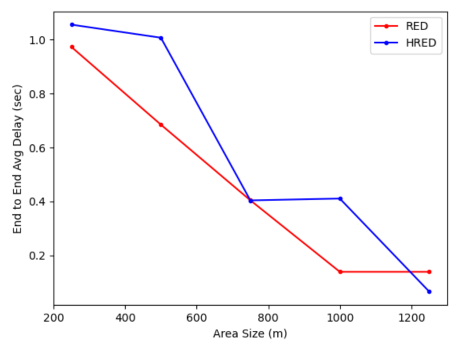
7 Results with Graphs

7.1 Wireless MAC type 802.11 (Mobile)

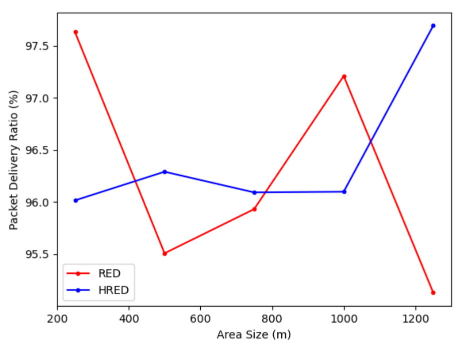
7.1.1 Varying Area Size



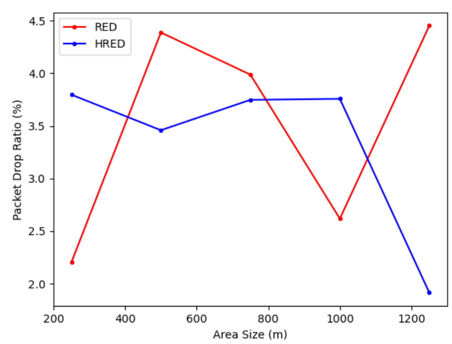
(a) Network Throughput



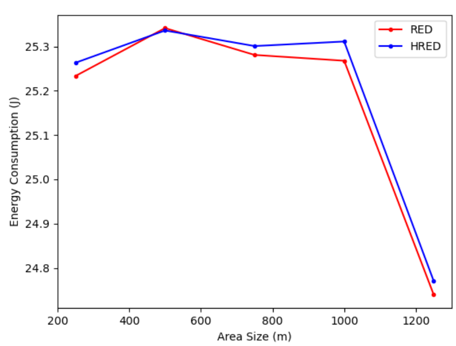
(b) End-to-End Average Delay



(c) Packet Delivery Ratio



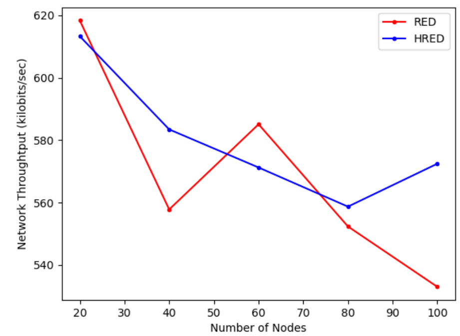
(d) Packet Drop Ratio



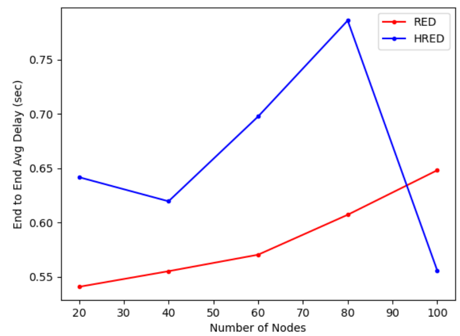
(e) Energy Consumption

Figure 3: Varying Area Size

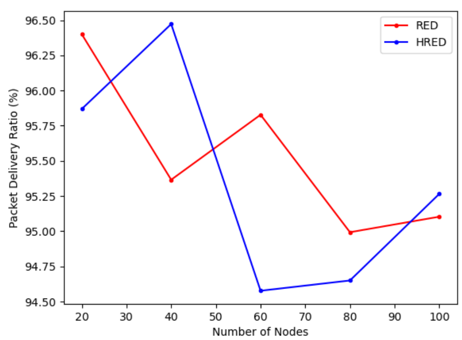
7.1.2 Varying Number of Nodes



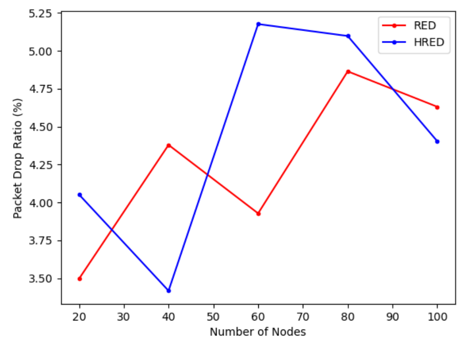
(a) Network Throughput



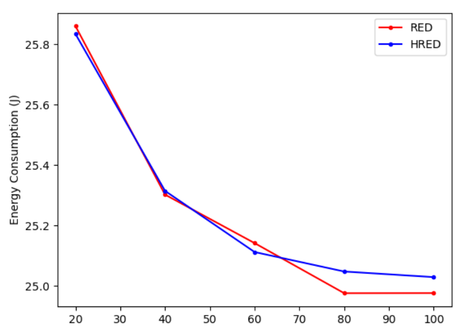
(b) End-to-End Average Delay



(c) Packet Delivery Ratio



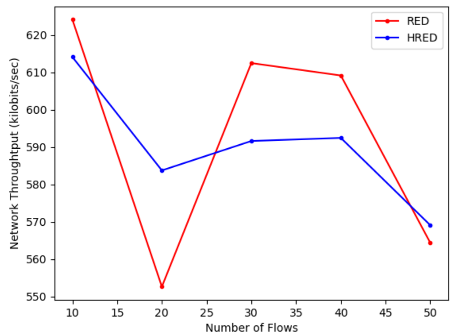
(d) Packet Drop Ratio



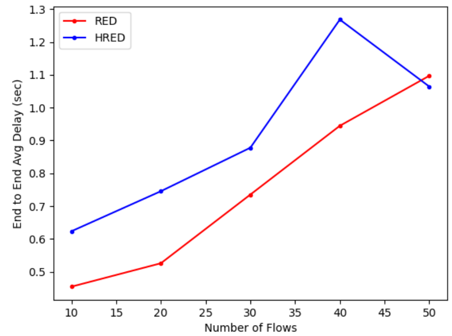
(e) Energy Consumption

Figure 4: Varying Area Size

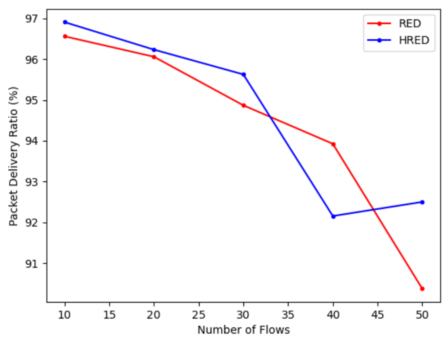
7.1.3 Varying Number of Flows



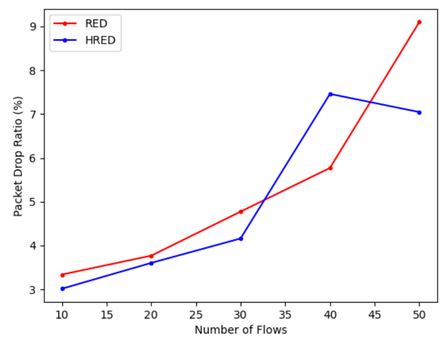
(a) Network Throughput



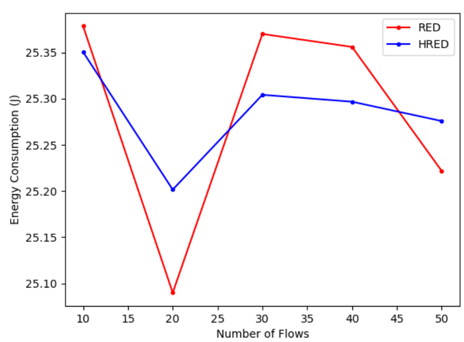
(b) End-to-End Average Delay



(c) Packet Delivery Ratio



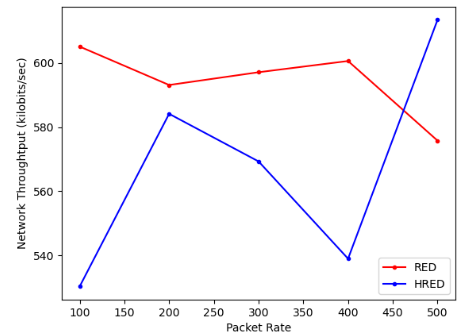
(d) Packet Drop Ratio



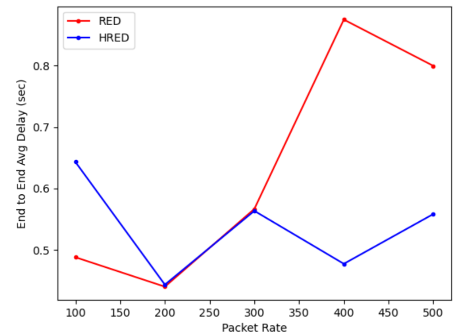
(e) Energy Consumption

Figure 5: Varying Area Size

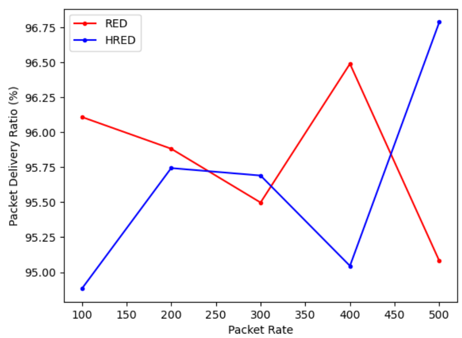
7.1.4 Varying Packet Rate



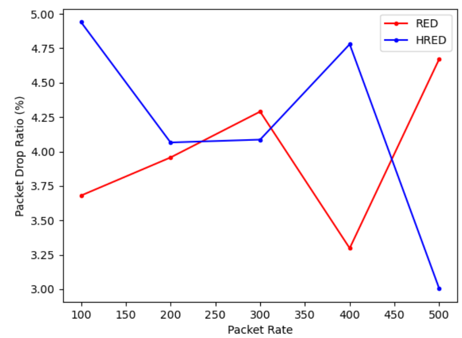
(a) Network Throughput



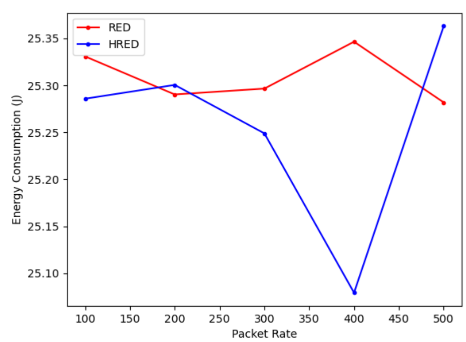
(b) End-to-End Average Delay



(c) Packet Delivery Ratio



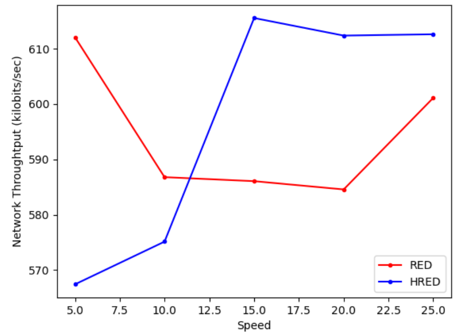
(d) Packet Drop Ratio



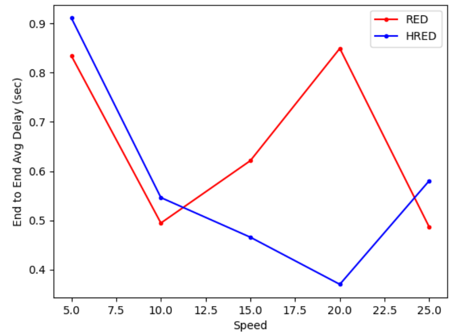
(e) Energy Consumption

Figure 6: Varying Area Size

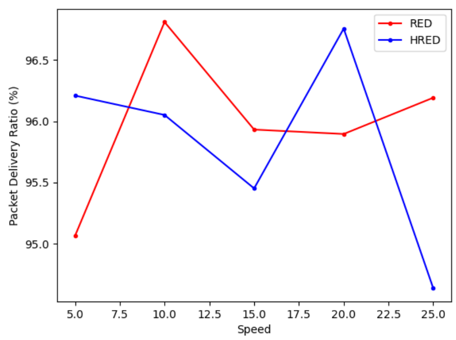
7.1.5 Varying Speed of Mobile Nodes



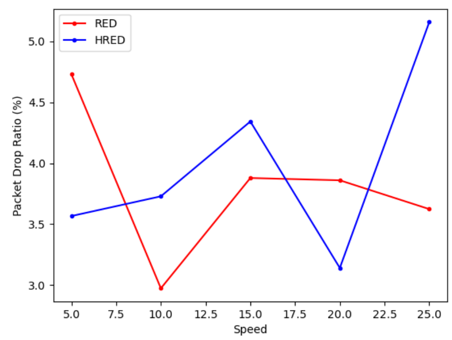
(a) Network Throughput



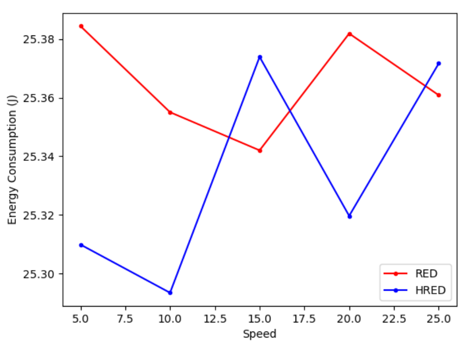
(b) End-to-End Average Delay



(c) Packet Delivery Ratio



(d) Packet Drop Ratio

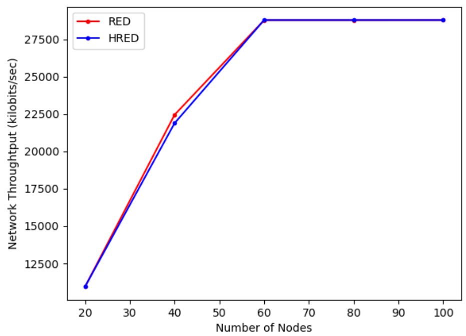


(e) Energy Consumption

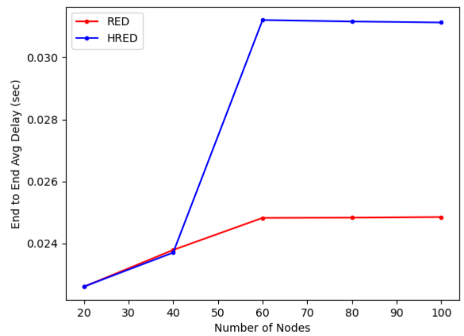
Figure 7: Varying Area Size

7.2 Wired

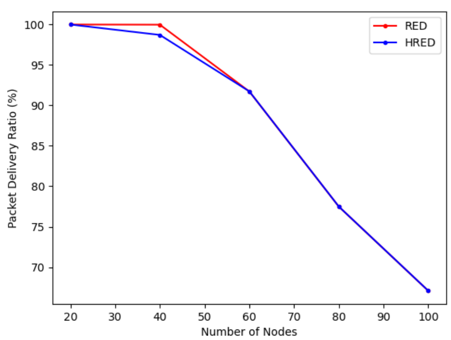
7.2.1 Varying Number of Nodes



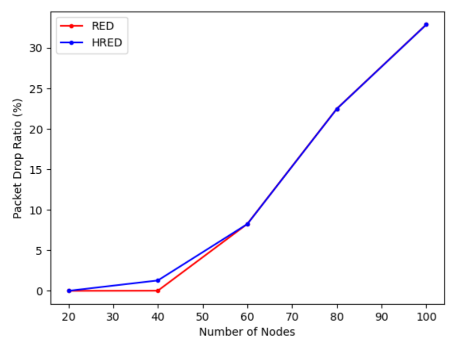
(a) Network Throughput



(b) End-to-End Average Delay



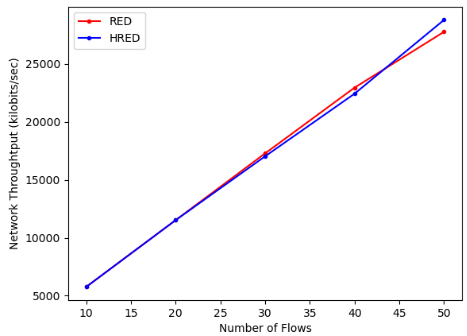
(c) Packet Delivery Ratio



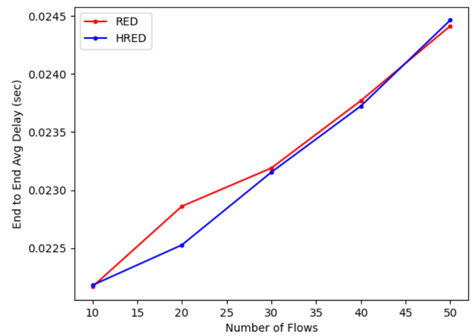
(d) Packet Drop Ratio

Figure 8: Varying Number of Nodes

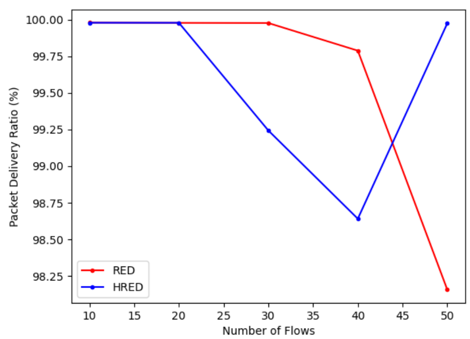
7.2.2 Varying Number of Flows



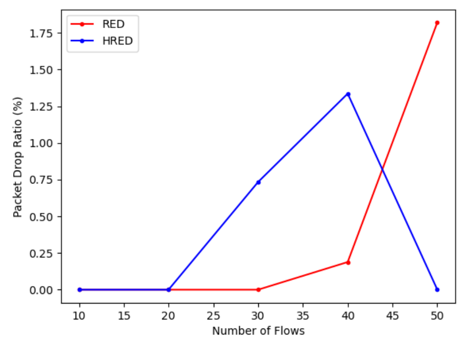
(a) Network Throughput



(b) End-to-End Average Delay



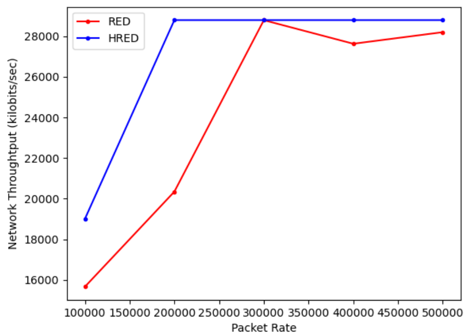
(c) Packet Delivery Ratio



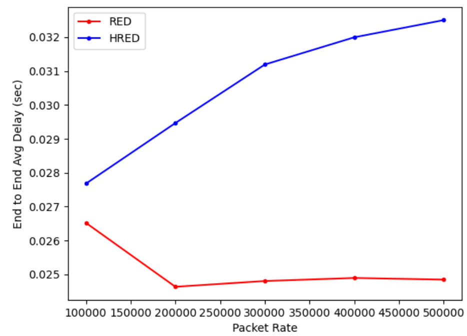
(d) Packet Drop Ratio

Figure 9: Varying Number of Flows

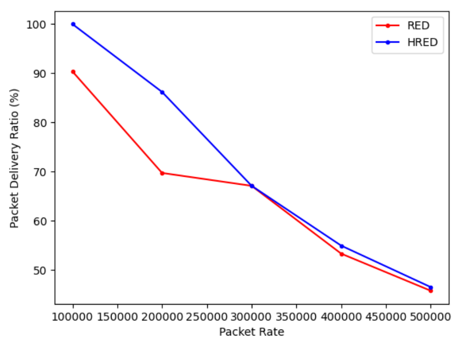
7.2.3 Varying Packet Rate



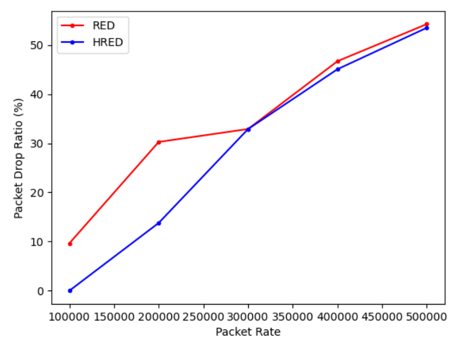
(a) Network Throughput



(b) End-to-End Average Delay



(c) Packet Delivery Ratio



(d) Packet Drop Ratio

Figure 10: Varying Packet Rate

8 Observation

The most critical area of the RED queue is the area between the minimum and maximum thresholds; because RED is always trying to stabilize the average queue size in this area. For this reason, the midpoint between the minimum and maximum thresholds was chosen to study the impact of modifying the drop probability on the performance of RED.
The average queue size at this point is an indicator of accumulative-aggressive traffic. Hence, increasing the drop probability to 0.5 was supposed to reduce this aggressive traffic and smoothen it before it reaches the most aggressive

drop probability. But we can see this in varying data rate figures, where RED and HRED both spike up or down suddenly. To solve this issue, for each parameter, 5 runs were taken and then their average value was chosen for the graph.
After taking this measure, graphs for the Wireless random sources-random destinations still show visible ups and downs. But graphs for the Wired one source-multiple destinations topology showed a smoothened structure.