RTT-Based Congestion Control for the Internet of Things

Mamun Munshi
ID: 1805028

March 1, 2023

1 Introduction

RTT-Based Congestion Control for the Internet of Things- describes an advance method for congestion control, specially applicable for IOT devices. IOT devices demands data transmission at a low energy consumption and shorter end-to-end delay. The proposed algorithm reduce energy consumption and delay significantly. Besides, it improves overall delivery ratio and decreased drop ratio.

2 Overview of the Algorithm

1. At first, we have to declare 3 new variables namely $SRTT_S, SRTT_L$ and $RTTVAR_L$. The first two are obtained by applying exponential smoothing on RTT value and the third one is smoothed RTT Variance.

$$SRTT_S = \alpha_S r + (1 - \alpha_S) \times SRTT_S \tag{1}$$

$$SRTT_L = \alpha_L r + (1 - \alpha_L) \times SRTT_S \tag{2}$$

$$RTTVAR_L = \beta_L |SRTT_L - r| + (1 - \beta_L) \times RTTVAR_L \tag{3}$$

   Here, α_S, α_L and β_L are smoothing factors.

2. Secondly, we create the Decision Boundary function to determine the current condition of network congestion.

$$T(\gamma) = SRTT_L + \gamma \times RTTVAR_L \tag{4}$$

   Now, we can identify four characteristic regions using the following decision boundaries:

   - (**LC**): $SRTT_S < T(1)$ (low congestion). In this state we can assume that there is no congestion in the network. Thus, the sending rate can be aggressively increased.
   - (**NO**): $T(1) \leq SRTT_S < T(+1)$ (normal operating point). Packet sending rate is slightly increased
   - (**MV**): $T(+1) \leq SRTT_S < T(2)$ (medium variability). A controlled decrease of sending rate can be necessary to avoid congestion.
   - (**HV**): $SRTT_S \geq T(+2)$ (high variability). In this state, the short-term RTT estimate is significantly higher that the long-term RTT estimate. Thus, aggressive decrease of sending rate is required.

3. The pseudo code of the algorithm is given below :

Algorithm 1 RTT-CoAP CONGESTION CONTROL

function UPDATERATE(**S**, $SRTT_L$, $SRTT_S$, $RTTVAR_L$,μ)

    if (**S** $\in$ **LC**) **then**
        $R = R +$ COMPRATE$(R, SRTT_L, fast)$;
    else if (**S** $\in$ **NO**) **AND** $(\mu < L_{high})$ **then**
        $R = R +$ COMPRATE$(R, SRTT_L, slow)$;
    else if (**S** $\in$ **MV**) **AND** $(\mu \geq L_{low})$ **then**
        $R = R -$ COMPRATE$(R, SRTT_L, slow)$;
    else if (**S** $\in$ **HV**) **AND** $(\mu \geq L_{low})$ **then**
        $R = R -$ COMPRATE$(R, SRTT_L, fast)$;
    end if
end function

function COMPRATE(R, $SRTT_L$, $speed$)

    if $(speed = fast)$ **then**
        $\omega = 0.2$
    else if $(speed = slow)$ **then**
        $\omega = 0.05$
    end if

    $R_b = \frac{1}{SRTT_L}$
    if $(R < R_b)$ **then**
        $\delta = \omega \times (R_b - R)$
    else
        $\delta = \omega \times R_b$
    end if
     return δ
end function

3 Modifications in NS2

The main procedure of congestion controlled has been performed in **tcp-vegas.cc**. The main task is to update the *Rate* variable, in **tcp-vegas.cc**, it is ***v_actual_***. ***v_actual_*** is updated in legacy version as below:

```
v_actual_ = double(rttLen)/rtt;
```

Now, the modification process is as follows:

1. At first, we declare 2 functions: *Decision_Boundary()* and *CompRate()*.

```
double Decision_Boundary (int gamma, double srtt_l, double rttvar_l){
    return srtt_l + gamma*rttvar_l;
}

//  Funtion to compare rate
double CompRate(double R, double srtt_l , int speed ){
    double w , d;
    if (speed == 2)
        w = 0.2;
    else if (speed == 1)
        w = 0.05;

    double R_b = 1/srtt_l;

    if (R < R_b )
        d = w * (R_b - R);
    else
        d = w * R_b;
return d;
}
```

2. Secondly, we declare necessary variables.

```
double r = rtt;
double alpha_s = 0.999;
double alpha_l = 0.001;
double beta_l = 0.8;

double srtt_s=0;
double srtt_l=0;
double rttvar_l=0;

srtt_s = alpha_s*r + (1-alpha_s)*srtt_s;
srtt_l = alpha_l*r + (1-alpha_l)*srtt_s;
rttvar_l = beta_l*abs(srtt_l-r) + (1-beta_l)*rttvar_l;
```

3. Finally, we update the rate variable, ***v_actual_***, based on 4 conditions discussed above.

```
U   P   D   A   T   I   N   G     R   A   T   E

if(srtt_s < Decision_Boundary(-1, srtt_l, rttvar_l))
    v_actual_ += CompRate(v_actual_, srtt_l, 2);

else if(Decision_Boundary(-1, srtt_l, rttvar_l) <= srtt_s && srtt_s < Decision_Boundary(1, srtt_l, rttvar_l))
    v_actual_ += CompRate(v_actual_, srtt_l, 1);

else if(Decision_Boundary(1, srtt_l, rttvar_l) <= srtt_s && srtt_s < Decision_Boundary(2, srtt_l, rttvar_l))
    v_actual_ -= CompRate(v_actual_, srtt_l, 1);

if(srtt_s >= Decision_Boundary(2, srtt_l, rttvar_l))
    v_actual_ -= CompRate(v_actual_, srtt_l, 2);
```

4 Network Topologies Under Simulation

4.1 Wired

- **Agent:** TCP-Vegas
- **Application:** Telnet
- **Routing Protocol:** DSR
- **Node Positioning:** Random

4.2 Wireless

- **Agent:** TCP-Vegas
- **Wireless MAC type:** IEEE 802.11
  However, the assigned wireless MAC type for me was IEEE 802.15.4. But using that, the packet sizes were resulting negative. A possible reason of this is that the header size is greater than the overall packet size and after subtraction, the result becomes negative. Therefore, to reduce complexity, MAC 802.11 has been used.
- **Application:** Telnet
- **Routing Protocol:** DSR
- **Node Positioning:** Random
- **Node Movement:** Mobile

A sample wireless topology with 40 nodes is shown below:

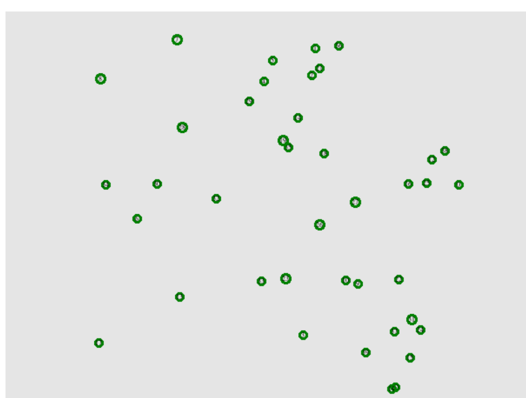


Figure 1: Sample wireless network with 40 random nodes

5 Parameters Under Variation

To study the performance comparison between our modified algorithm and previous algorithm, we have varied some parameters while simulating.

Base Parameters:

1. Node Count : 40
2. Flow Count : 20
3. Packets per second : 300
4. Speed (for wireless nodes) : 10 m/s

Varying Parameters:

1. Node Count : 20 , 40 , 60 , 80 , 100
2. Flow Count : 10 , 20 , 30 , 40 , 50
3. Packets per second : 100 , 200 , 300 , 400 , 500
4. Speed (for wireless nodes) : 5 m/s, 10 m/s, 15 m/s, 20 m/s, 25 m/s

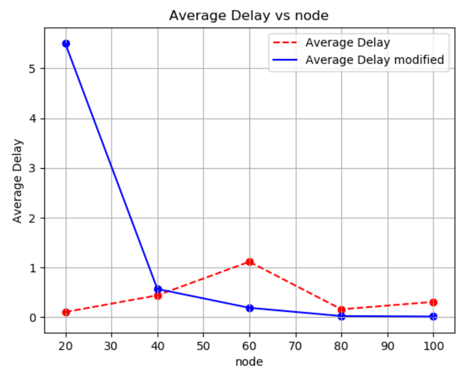
While varying these parameters, following outcomes have been evaluated:

- Average Throughput
- End-to-End Delay
- Delivery Ratio
- Drop Ratio
- Energy Consumption (for wireless topology)

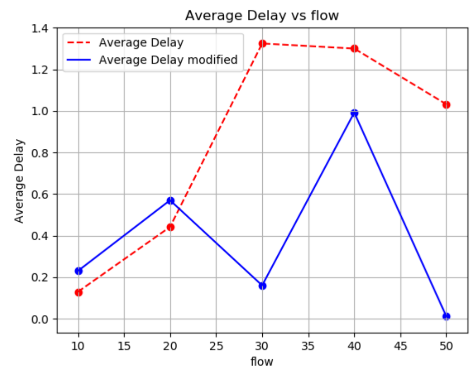
6 Results in Graph

6.1 Wireless Network

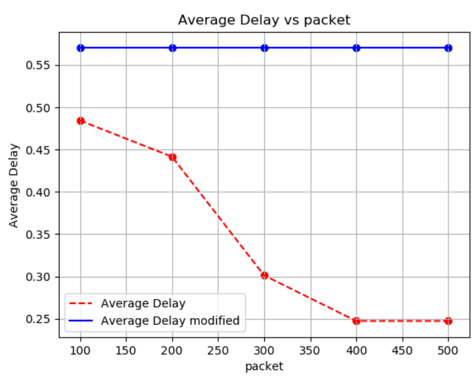
6.1.1 Average Delay



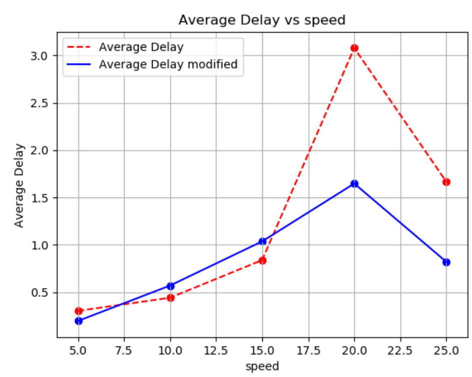
(a) Average Delay vs Node count



(b) Average Delay vs Flow count



(c) Average Delay vs Packet-rate

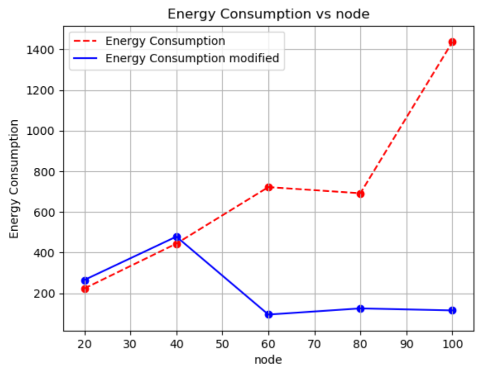


(d) Average Delay vs Speed

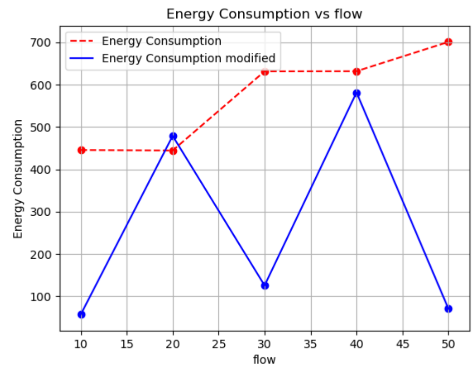
Figure 2: Comparison of Average Delay for wireless network

Observation: In case of average delay, now our modified algorithm outperforms the previous one in most of the cases. As the graphs show, average end-to-end delay decrease a lot in modified algorithm, which was one of core purposes of our algorithm.

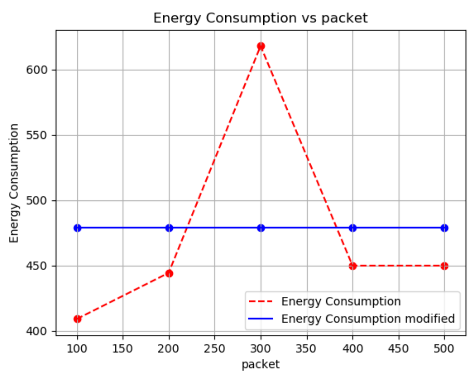
6.1.2 Energy Consumption



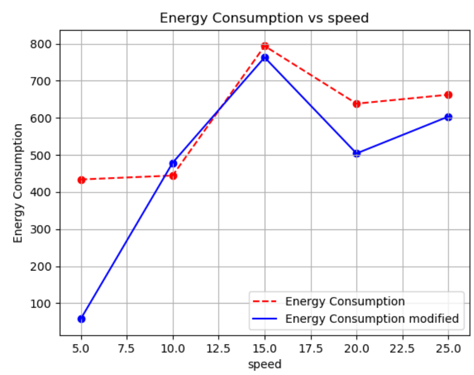
(a) Energy Consumption vs Node count



(b) Energy Consumption vs Flow count



(c) Energy Consumption vs Packet-rate

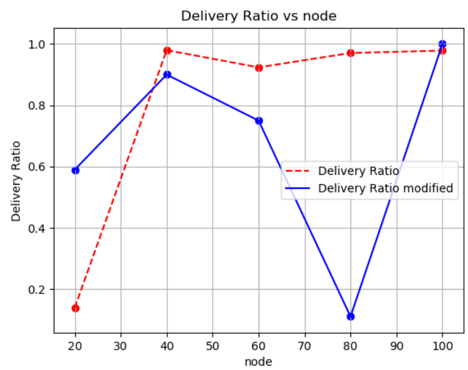


(d) Energy Consumption vs Speed

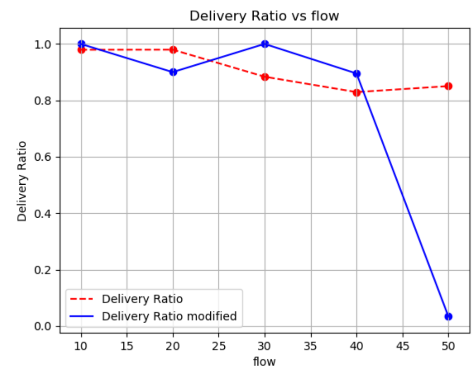
Figure 3: Comparison of Energy consumption for wireless network

Observation: The modified congestion control algorithm outperforms the previous one in case of energy consumption. As the graphs show, average energy consumption decrease a lot in modified algorithm, which was one of core purposes of our algorithm.

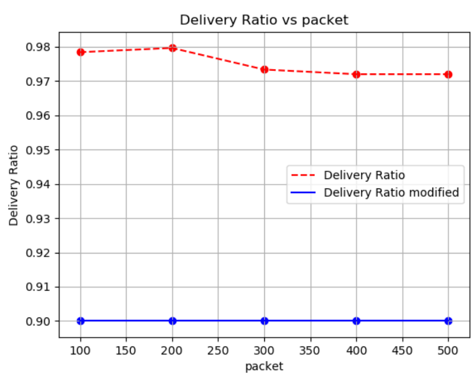
6.1.3 Delivery Ratio



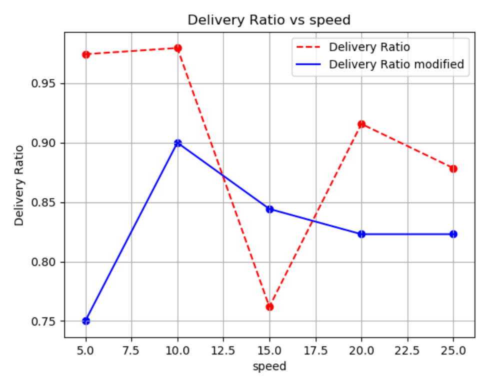
(a) Delivery Ratio vs Node count



(b) Delivery Ratio vs Flow count



(c) Delivery Ratio vs Packet-rate

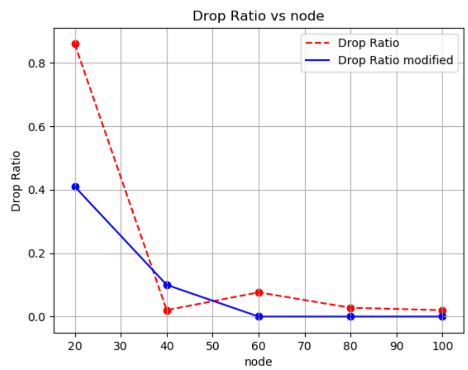


(d) Delivery Ratio vs Speed

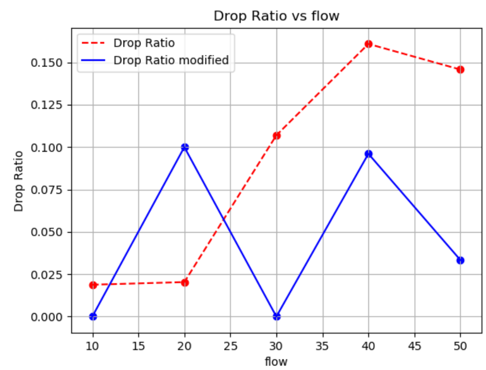
Figure 4: Comparison of Delivery Ratio for wireless network

Observation: In our modified algorithm, delivery ratio is better in some cases. However, in some random simulation, our algorithm gives awful outcomes in terms of delivery ratio, which is considerable.

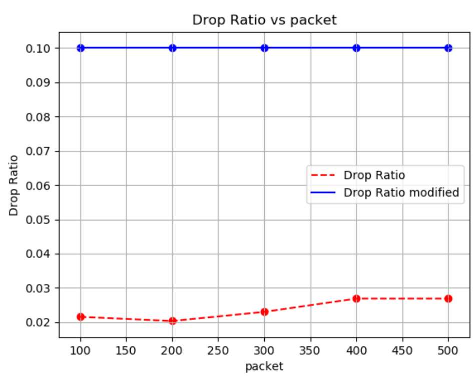
6.1.4 Drop Ratio



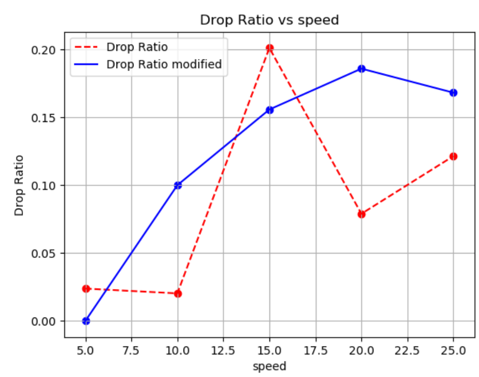
(a) Drop Ratio vs Node count



(b) Drop Ratio vs Flow count



(c) Drop Ratio vs Packet-rate

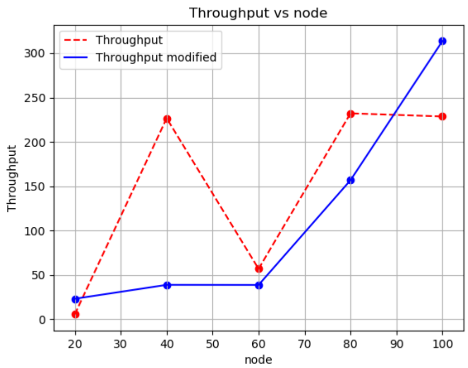


(d) Drop Ratio vs Speed

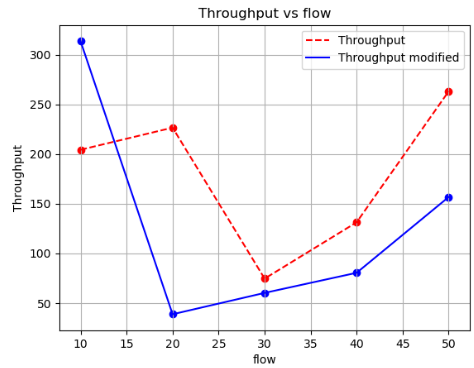
Figure 5: Comparison of Drop Ratio for wireless network

Observation: In our modified algorithm, without some exceptions, drop ratio decreases which is a positive aspect of our algorithm.

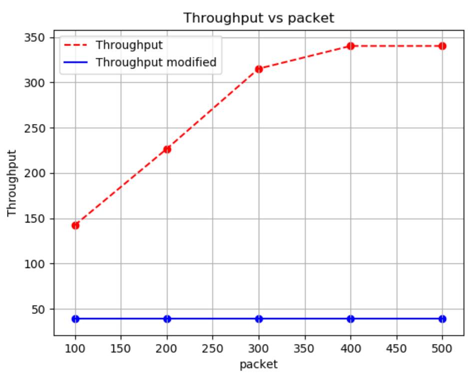
6.1.5 Throughput



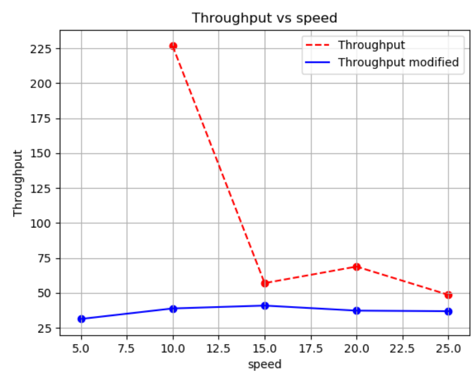
(a) Throughput vs Node count



(b) Throughput vs Flow count



(c) Throughput vs Packet-rate



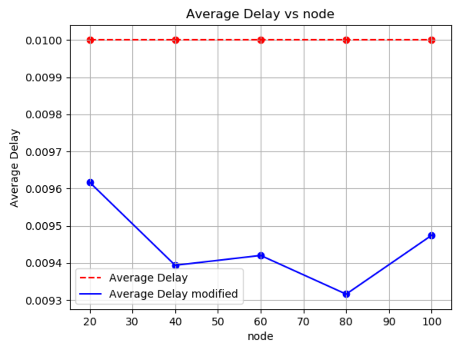
(d) Throughput vs Speed

Figure 6: Comparison of Throughput for wireless network

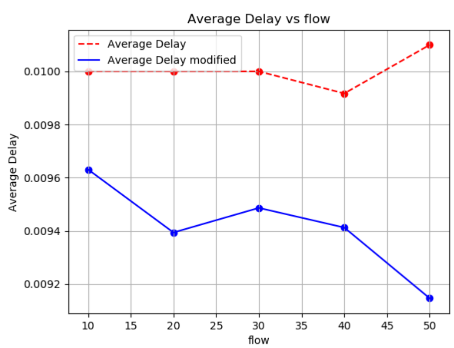
Observation: Throughput is the only aspect where our modified algorithm performs little bad. However, improving throughput is not our main concern.

6.2 Wired Network

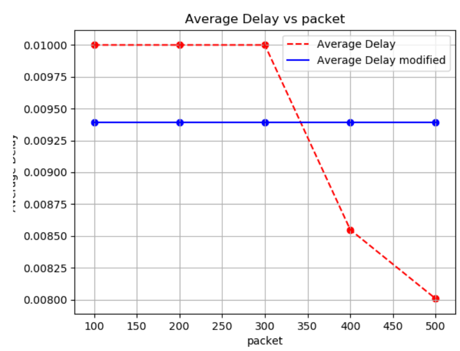
6.2.1 Average Delay



(a) Average Delay vs Node count



(b) Average Delay vs Flow count

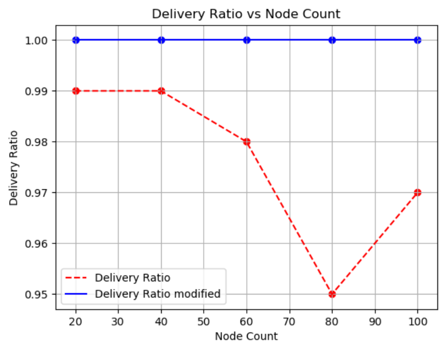


(c) Average Delay vs Packet-rate

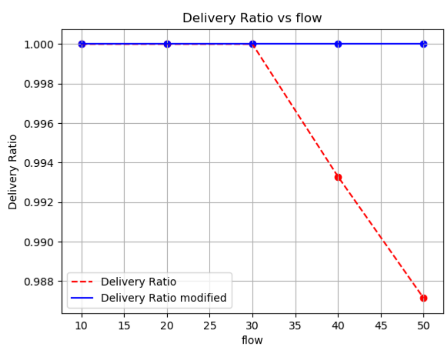
Figure 7: Comparison of Average Delay for wired network

Observation: In case of average delay, now our modified algorithm outperforms the previous one. As the graphs show, average end-to-end delay decrease a lot in modified algorithm, which was one of core purposes of our algorithm.

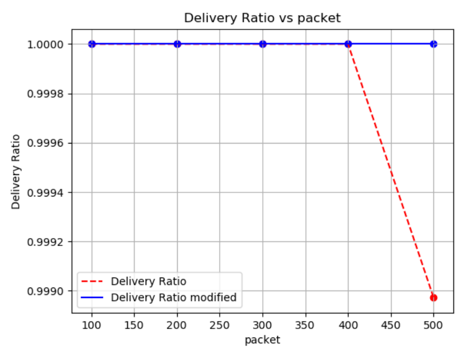
6.2.2 Delivery Ratio



(a) Delivery Ratio vs Node count



(b) Delivery Ratio vs Flow count

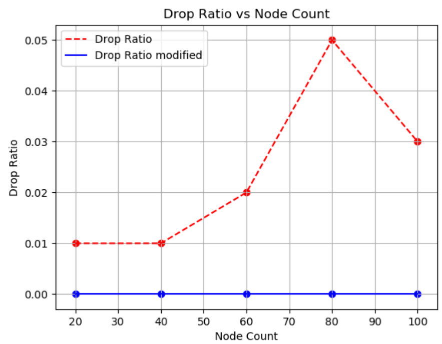


(c) Delivery Ratio vs Packet-rate

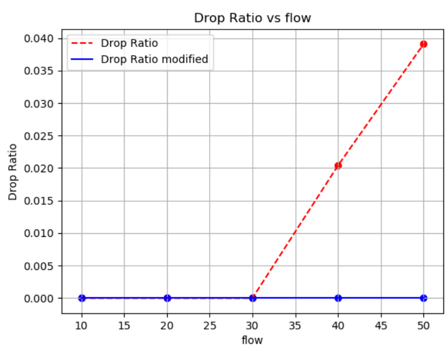
Figure 8: Comparison of Delivery Ratio for wired network

Observation: In case of delivery ratio, now our modified algorithm performs way better than the previous one. Delivery ratio increases significantly in modified algorithm.

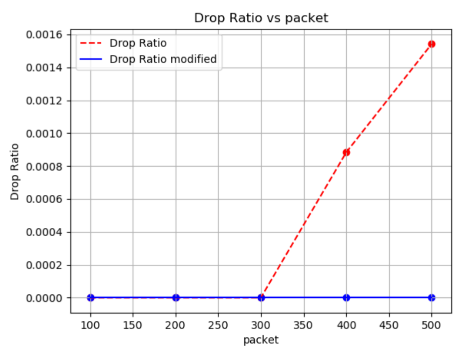
6.2.3 Drop Ratio



(a) Drop Ratio vs Node count



(b) Drop Ratio vs Flow count

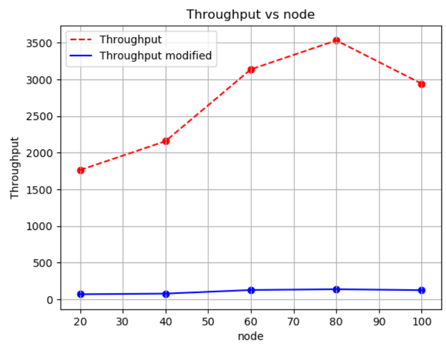


(c) Drop Ratio vs Packet-rate

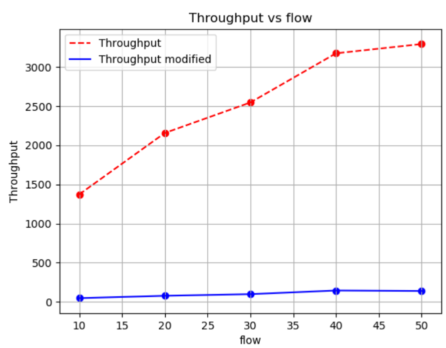
Figure 9: Comparison of Drop Ratio for wired network

Observation: In modified algorithm, packet drop ratio decreases noticeably which is a good side of our algorithm.

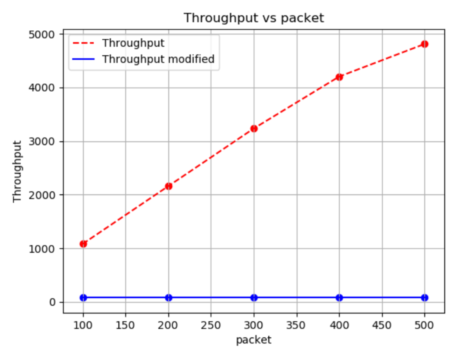
6.2.4 Throughput



(a) Throughput vs Node count



(b) Throughput vs Flow count



(c) Throughput vs Packet-rate

Figure 10: Comparison of Throughput for wired network

Observation: As we can see, our new algorithm doesn't perform so well in case of throughput. Average throughput has been decreased noticeably after simulating under modified version.

7 Conclusion

Providing a better congestion control algorithm for IOT devices was our main motivation behind replacing the existing algorithm with a newer one. As we have already known, IOT devices demand low energy consumption and low end-to-end delay. Analyzing the attached graphs, it is quite clear that we have successfully improved in those two sectors. Moreover, the packet delivery ratio has increased significantly in our modified algorithm. In terms of throughput, our modified algorithm performs little bad, however, this can't suppressed other key improvements.