CSE 322: NS2 Project Report

TCP Adaptive Reno

Md. Nazmul Islam Ananto

Roll **1805093**

Supervised by –

Dr. A. B. M. Alim Al Islam

Professor

Dept. of Computer Science & Engineering

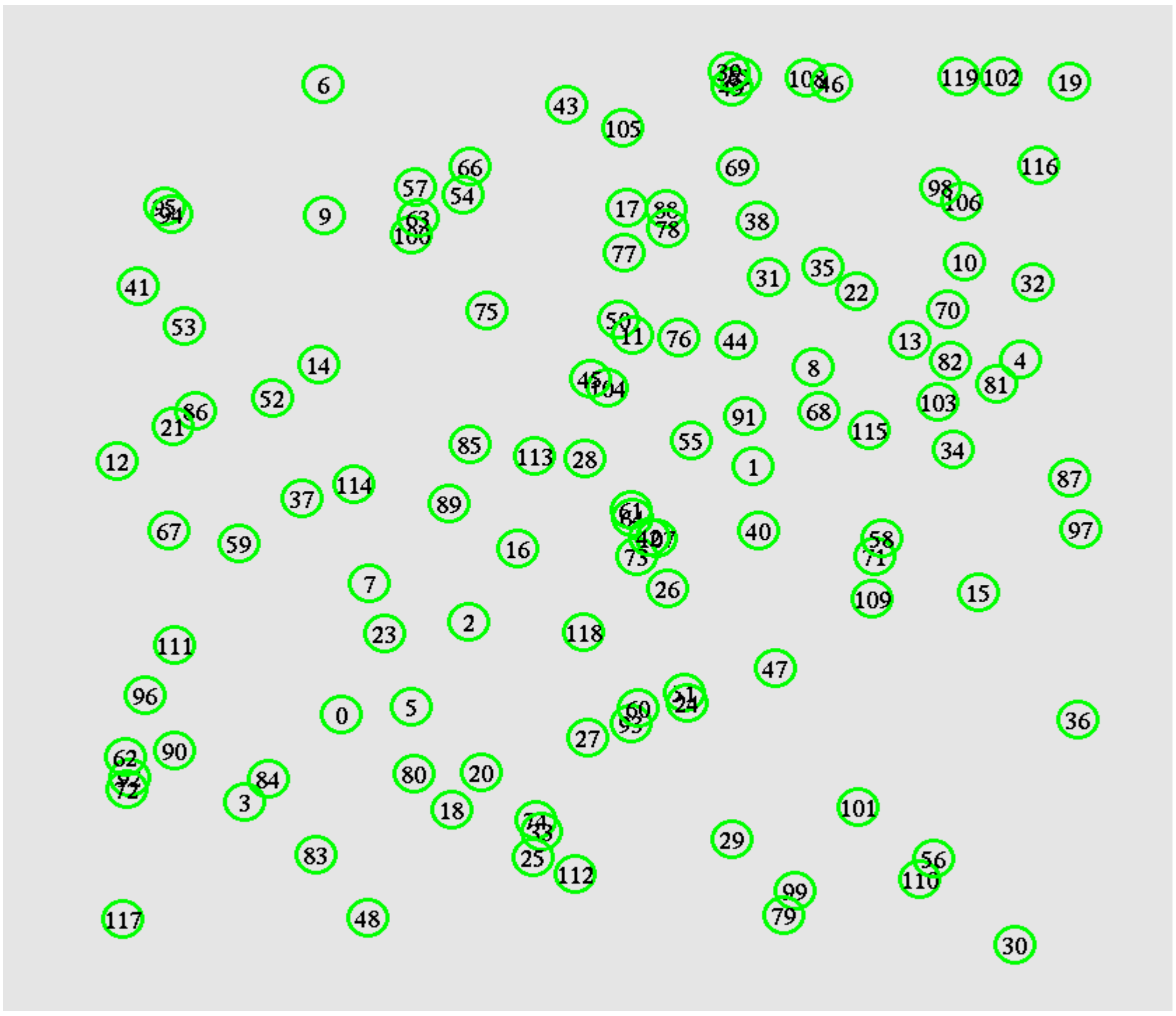
Bangladesh University of Engineering & Technology

1. Introduction

Having roll mod 8 = 5 I had to implement 802.11 (static) and 802.15.4 (mobile). I chose TCP Adaptive Reno for my project and will be explaining how this works on these networks.

2. Network Topologies Under Simulation

I used a variation of nodes at random position and random number of flows. For the static part, I had to vary the coverage area and for the mobile part I had to vary the speed.



In both cases, my topology looked like this.

3. Overview of the Proposed Algorithm

The widely used congestion control algorithm TCP-Reno has some disadvantages. Its' throughput decreases in networks with large bandwidth delay product and with non-negligible packet loss. Other congestion control algorithms like TCP-Westwood / TCPHighSpeed attempts to solve this problem but their potential unfriendliness to TCPReno is one of the reasons hampering their deployment.

In order to tackle this problem, a new congestion algorithm is proposed named TCP-AReno or Adaptive Reno. This algorithm is based on TCP-Westwood-BBE. It has the following attributes:

• Estimates congestion level via RTT to determine whether a packet loss is due to congestion or not.
• Introduces a fast window expansion mechanism to quickly increase congestion window whenever it finds network underutilization. The congestion window increase will have 2 parts: The base part increases linearly in congestion avoidance phase
just like TCP-Reno whereas the probe part increase exponentially to utilize the network fully.
• Adjusts congestion window reduction based on the congestion measurement. Congestion window is halved when the network is congested and a packet loss is likely to happen, while the reduction is mitigated when the network is underutilized.

Finally the paper claims the TCP-AReno modifications results in:
• Higher throughput than TCP-Reno.
• Friendlier to TCP-Reno over a wide range of link capacities and random packet loss rates.

4. Parameters Under Variation

For the static part –

```
nodes=120
flows=80
packets=500
coverage=500
```

For the mobile part –

```
nodes=120
flows=80
packets=500
```

```
speed=15
```

These are the base value of the parameters I used. The variation of them were –

a. The number of nodes (20, 40, 60, 80, and 100)
b. Number of flows (10, 20, 30, 40, and 50)
c. Number of packets per second (100, 200, 300, 400, and 500)
d. Speed of nodes (5 m/s, 10 m/s, 15 m/s, 20 m/s, and 25 m/s)
e. Coverage area (square coverage are varying one side as Tx_range, 2 x Tx_range, 3 x Tx_range, 4 x Tx_range, and 5 x Tx_range)

5. Modifications made in the Simulator

a. Variables and Methods:

```
  double estimate_cong_level();
  double rtt_cong; /* current RTT congestion */
  double last_rtt_cong;    /* last RTT */
  double rtt_factor; /* RTT factor, a */

  int cwnd_base; /* base cwnd */
  int cwnd_probe; /* probe cwnd */
  int cwnd_inc; /* cwnd increment */

  int max_seg_size; /* maximum segment size */

  int open_congestion_window();
  int close_congestion_window();
```

b. Congestion Level Calculation:

The paper suggested that we calculate the congestion level depending on some extra variables - `rtt_cong`, `last_rtt_cong`, `rtt_factor`.

The equations are as follows –

$$rtt_{cong} = (1 - rtt_{factor})last_{rtt_{cong}} + rtt_{factor} * rtt_{estimation}$$

$$cong_{level} = \min\left(\frac{rtt_{estimation} - \min\limits_{\mathrm{rtt_{estimation}}}}{rtt_{cong} - \min\limits_{\mathrm{rtt_{estimation}}}}, 1\right)$$

`rtt_estimation_` and `min_rtt_estimation` are already implemented in ns2. I just had to take care of the others.

```
double ARenoTcpAgent::estimate_cong_level()
{
  double rtt_estimate = t_rtt_ * tcp_tick_;

  // update the congestion level
  if ((rtt_cong - min_rtt_estimate <= 0))
  {
    return 0;
  }
  double probable_cong_level = (rtt_estimate - min_rtt_estimate) / (rtt_cong
- min_rtt_estimate);
  probable_cong_level = (probable_cong_level < 0) ? 0 : probable_cong_level;
  double cong_level = (probable_cong_level < 1) ? probable_cong_level : 1;
  // printf("time %f cong_level %f\n", fr_now, cong_level);
  last_rtt_cong = rtt_cong;

  return cong_level;
}
```

c. Opening window

To open the window, I had to take care of 2 new variables - `cwnd_base` and `cwnd_probe`. The former works like the windows management in Tcp Reno while the second one tries to quickly increase when the network is underutilized.

$$cwnd_{base} = cwnd_{base} + MSS/cwnd$$

$$cwnd_{probe} = \max\left(cwnd_{probe} + \frac{cwnd_{inc}}{cwnd}, 0\right)$$

$$cwnd_{inc_{max}} = \frac{B}{M} * MSS$$

$$cwnd_{inc}(c) = \frac{cwnd_{inc_{max}}}{e^{alpha*c}} + beta * c + gamma$$

The paper suggested some values of alpha, beta and gamma. So I used them. One thing to notice is that `cwnd_inc` depends on the `cong_level`.

```
int ARenoTcpAgent::open_congestion_window()
{
  double cong_level = estimate_cong_level();
  int factor_m = 10; // 10 mbps in paper

  double max_seg_size = 1460;
  double cwnd_inc_max = current_bwe_ / factor_m * max_seg_size;

  double alpha = 10;
  double beta = 2.0 * cwnd_inc_max * ((1.0 / alpha) - ((1.0 / alpha + 1.0)
/ (pow(2.71, alpha))));
  double gamma = 1.0 - (2.0 * cwnd_inc_max * ((1.0 / alpha) - ((1.0 / alpha
+ 0.5) / (pow(2.71, alpha)))));

  cwnd_inc = (int)((cwnd_inc_max / pow(2.71, alpha * cong_level)) + (beta *
cong_level) + gamma);
  cwnd_inc = (cwnd_inc > cwnd_inc_max) ? cwnd_inc_max : cwnd_inc;

  // base_window = use new renos base window
  double adder;
  if (cwnd_ > 0) {
    adder = max_seg_size / cwnd_;
    adder = (adder < 1) ? 1 : adder;
  } else {
    adder = 1;
  }
  cwnd_base += (int)adder;

  // change probe window
  if (cwnd_ > 0) {
    cwnd_probe = (int)(cwnd_probe + cwnd_inc / cwnd_);
    cwnd_probe = (cwnd_probe < 0) ? 0 : cwnd_probe;
  } else {
    cwnd_probe = 0;
  }

  // change cwnd
  return cwnd_base + cwnd_probe;
}
```

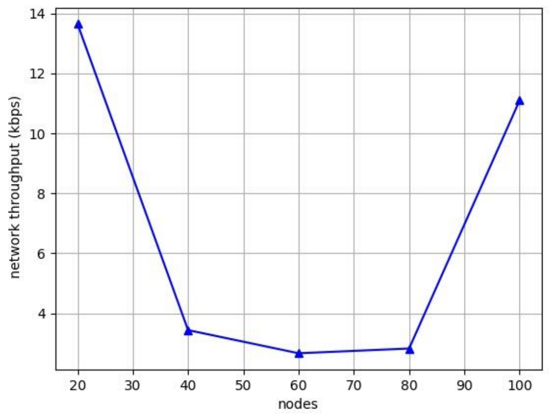
d. **Closing window**

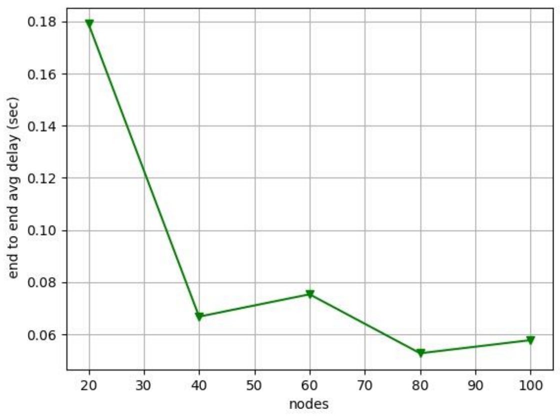
Window closing is a rather easy task.

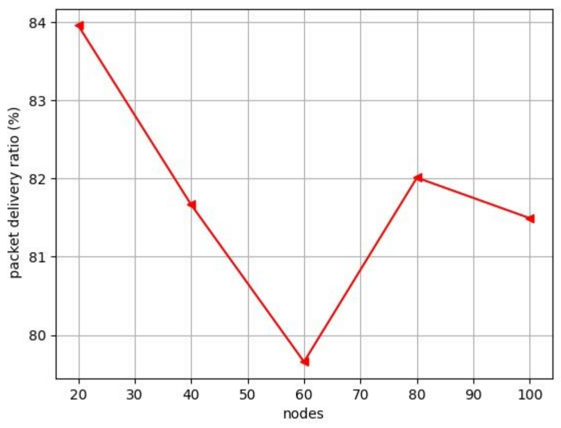
```
int ARenoTcpAgent::close_congestion_window()
{
  double max_seg_size = 1460;
  double cong_level = estimate_cong_level();
  int ssthresh_ = (int)(cwnd_ / (1.0 + cong_level));
  ssthresh_ = (2 * max_seg_size > ssthresh_) ? 2 * max_seg_size : ssthresh_;
  ssthresh_ = (ssthresh_ > (int)cwnd_) ? (int)cwnd_ : ssthresh_;

  // reset calculations
  cwnd_base = ssthresh_;
  cwnd_probe = 0;

  return cwnd_base + cwnd_probe;
}
```

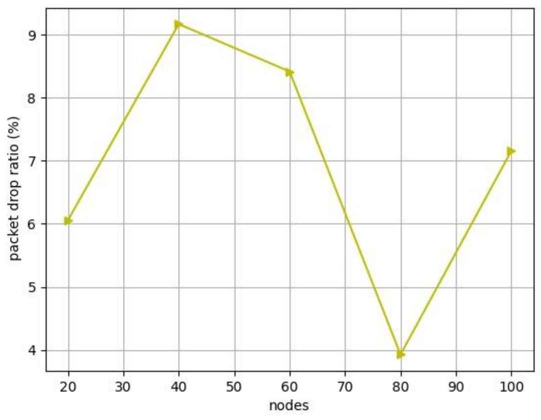
6. Results with Graphs

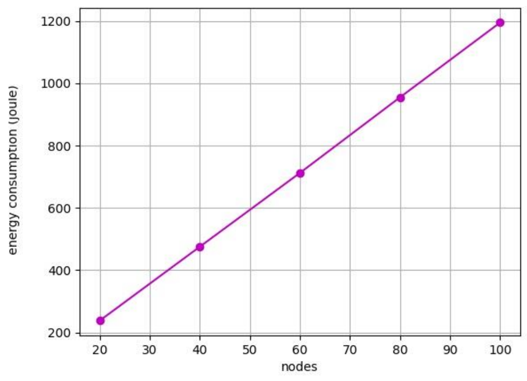
Varying number of nodes –











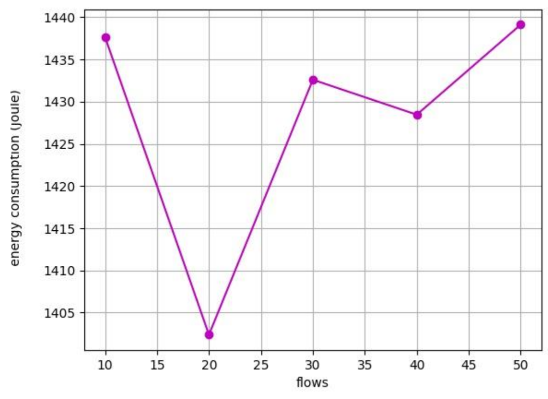
As number of nodes increase, throughput increases slightly. The more the number of nodes increase, the number of flow increases and the bottleneck link gets utilized. It should be noted that the link capacity is set at 1Mbps, so even from the beginning (node=20) it was used to its full capacity. That is why there is only a slight increase as the nodes grow.

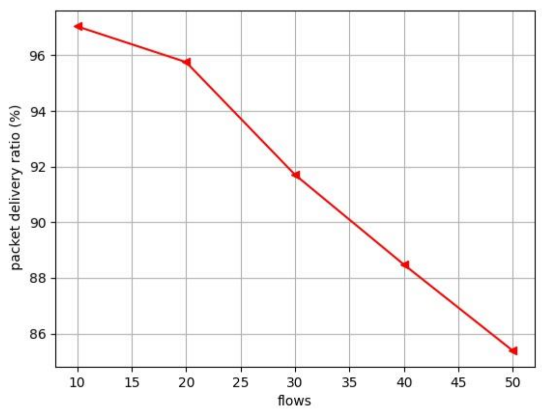
End to end delay decreases as more nodes are added because of the congestion control algorithm. This makes packets wait until congestion is below a suitable level to send the packets. Thus the more nodes/flow there are, the more the packets have to wait and the delay increases.

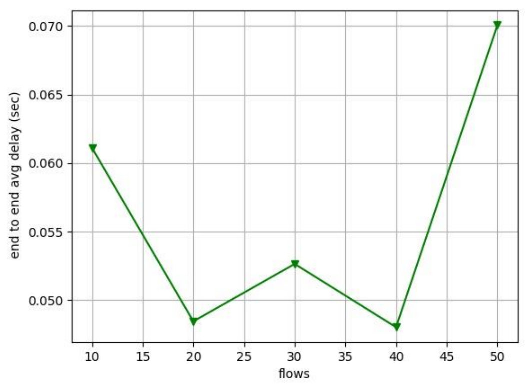
Delivery ratio decreases as more nodes means more congestion and the ratio of received and sent packets will continue to decrease.

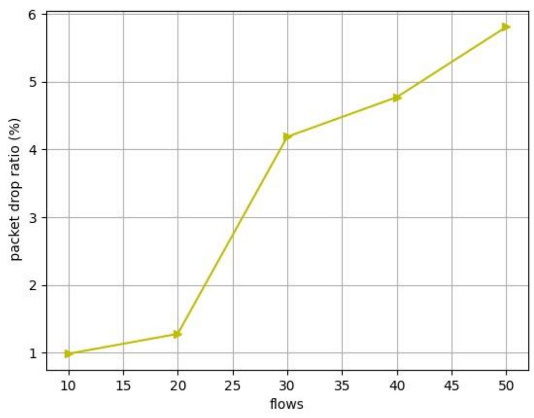
For the same reason, that means the drop ratio will increase as more packets are being dropped due to increased congestion.

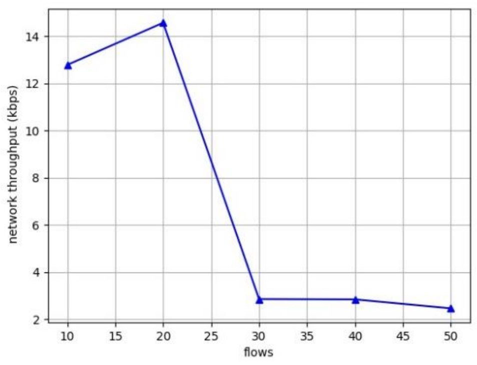
Varying number of flows –











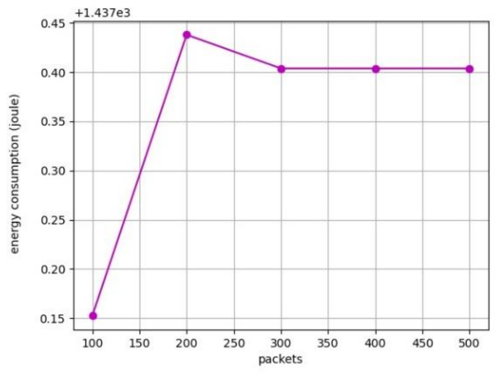
Here nodes are kept constant (at 120) and flows are being varied. This shows the effect of flow increase more clearly. Throughput increases slightly as more flows utilize the bottleneck layer but again it was already running up to capacity from the beginning.

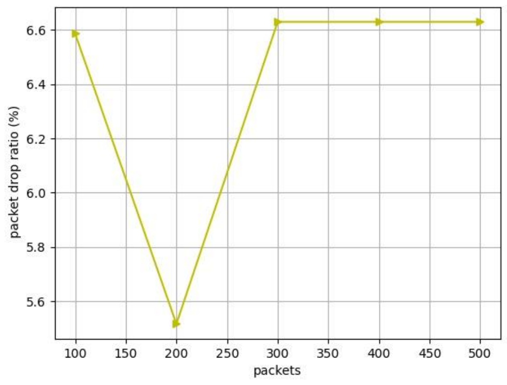
End to end delay increases as an increase of flow creates congestion.

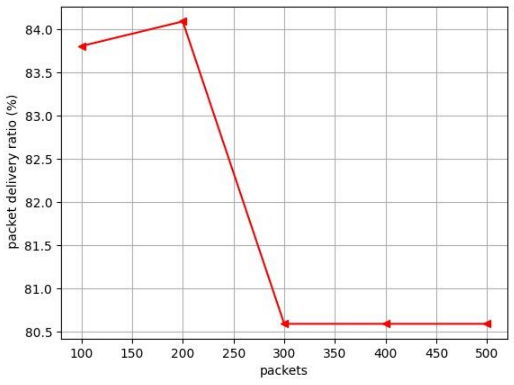
This means that delivery ratio will continue to decline.

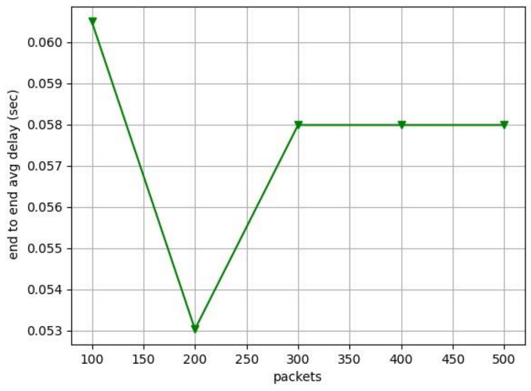
And the drop ratio will continue to increase.

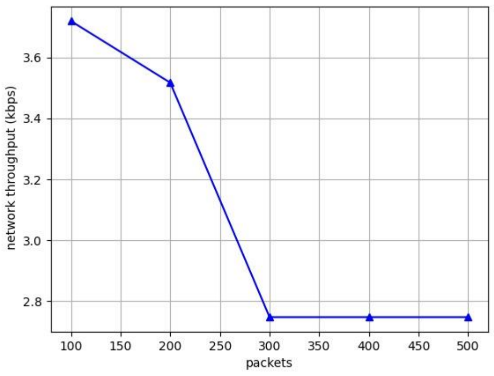
Varying packets per second –





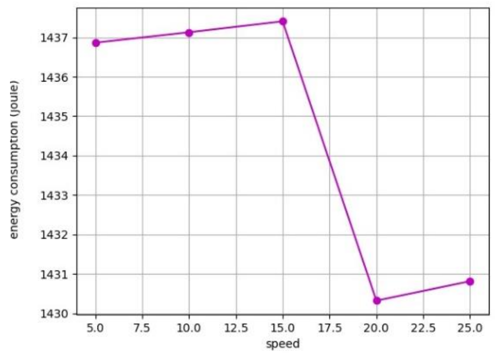


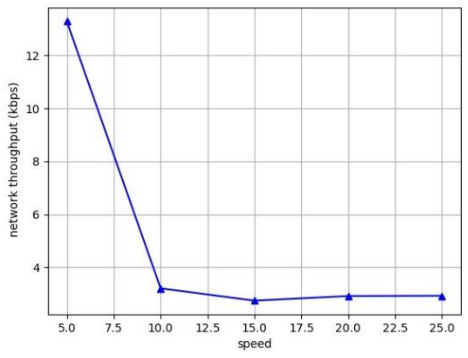


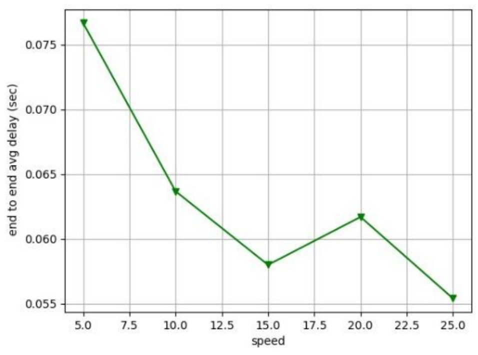


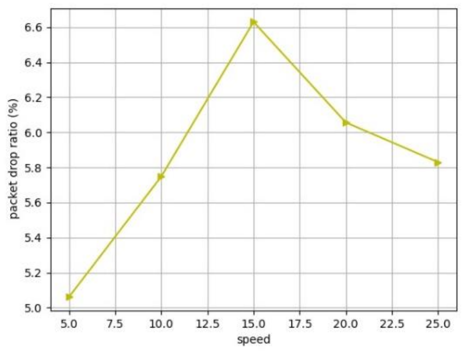
Here only transmission number of packets per second is being varied. We can see that there is a very slight increase of throughput, end to end delay, delivery ratio and slight decrease of drop ratio. These increase and decrease are too small to be substantial. This points to the fact that, as packets per second is being increased, the congestion control algorithm properly does its job and synchronises the transmission of packets with congestion by delaying some packet transmission until a suitable congestion level. That is why all the metrics remain kind of constant across the increase of packets per second transmission.

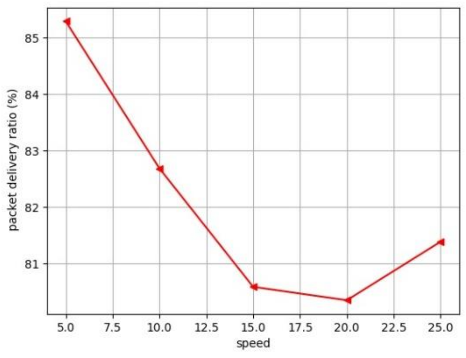
Varying speed –





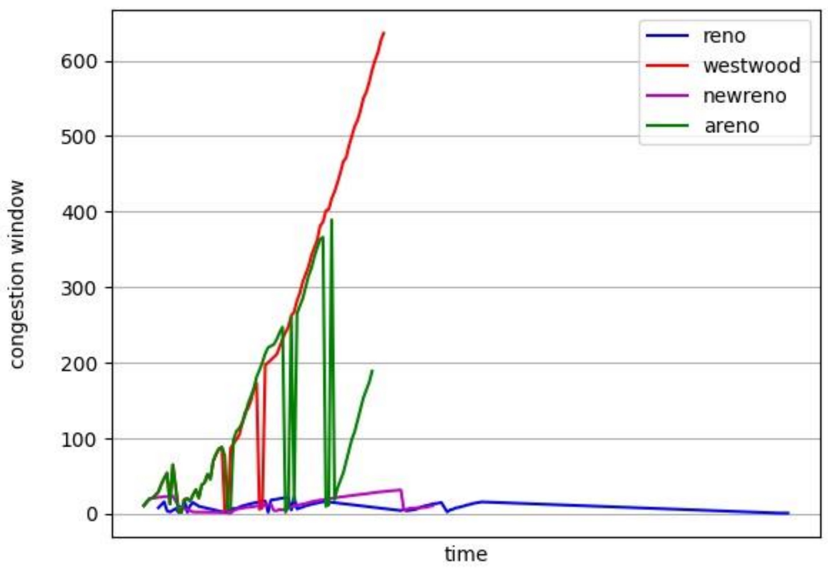






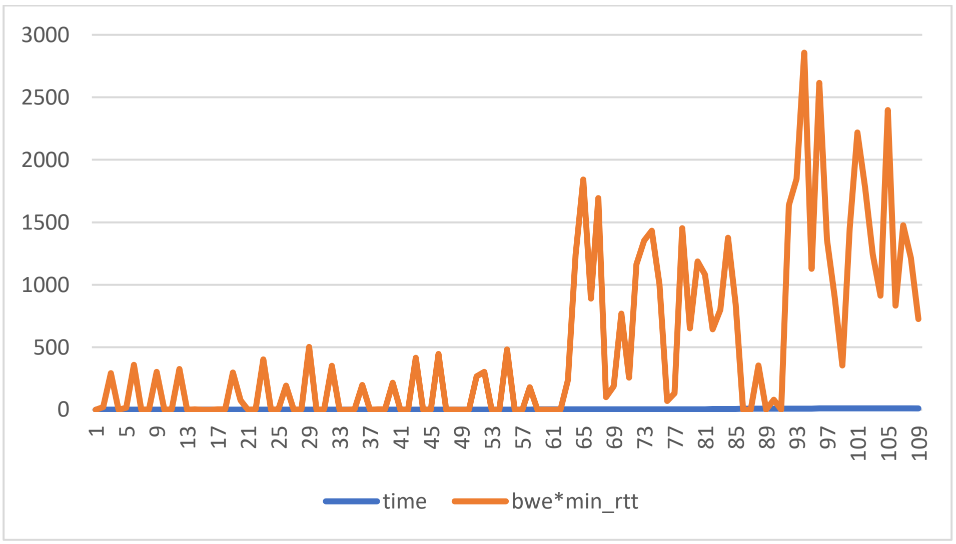
7. Comparison with Other Implementations

Congestion Window –

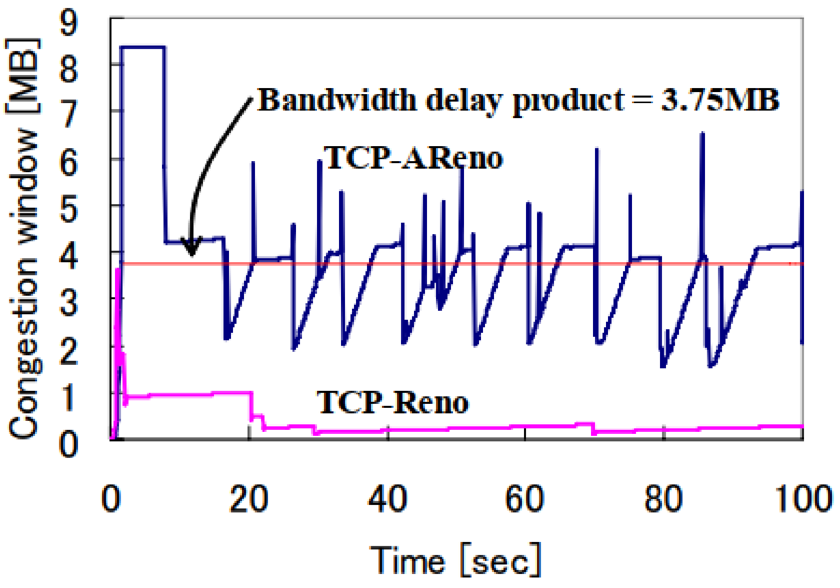


To understand how Tcp Adaptive Reno works, this graph is the key to that. Here we can see some aggressive yet effective change in the congestion window calculation comparison with other implementations. Tcp Westwood is the only one close to AReno and that's because AReno was initially built on Westwood BBE, a variation of Westwood.

Bandwidth delay product –



Here we have plotted the bandwidth estimation vs time graph to have a better understanding of what's happening under the hood. The one from the paper anticipated the similar behavior –



8. Summary

From the simulations we can say that TCP-Adapative Reno efficient in high network environments and is friendly towards TCP-NewReno and its variants. Thus my results agree with these claims from the paper.