Report

NS2 Term Project : Modifications of Mechanisms for Congestion Control in TCP (TCP-Constant)

Date of Submission: 27th February 2023

Submitted by:

Kazi Ababil Azam
1805077
L-3/T-2
CSE-B1

Introduction:

Congestion control is a crucial aspect of network performance, especially in wired and wireless networks, where resources are limited, and network traffic is high. To improve TCP throughput in such networks, researchers have proposed various congestion control algorithms over the years. In this project, we aim to implement a new congestion control algorithm proposed in the paper "Updated Congestion Control Algorithm for TCP: Improvement in Wired and Wireless Network" by Prof. K. Srinivas, Dr. A. A. Chari and Prof. N. Kasiviswanath.

The proposed congestion control algorithm aims to improve TCP throughput in wired and wireless networks by minimising the number of dropped packets and ensuring a fair share of network resources to each TCP flow. The algorithm achieves this by dynamically adjusting the congestion window size of each TCP flow based on network conditions, and keeping it constant as long as the network conditions do not change.

To evaluate the performance change, we have compared the metrics of the modified algorithm to the existing algorithm of TCP Westwood, and analysed the corresponding graphs subsequently.

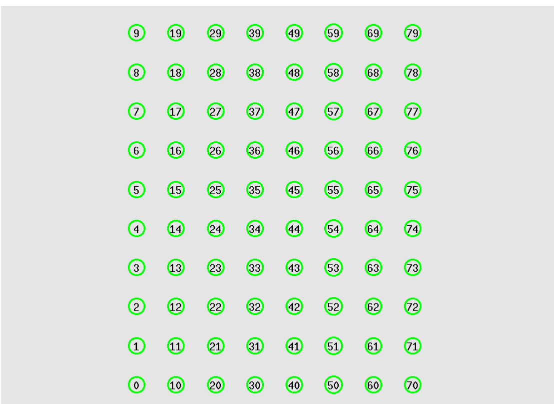
Network topologies:

The network topology assigned to this project were:

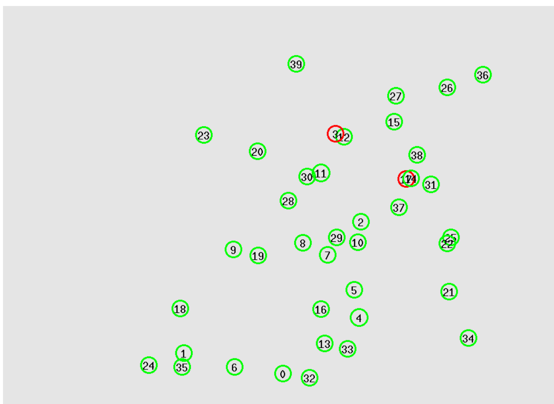
1. Wireless static nodes with MAC 802.11
2. Wireless mobile nodes with MAC 802.15.4

We have performed necessary simulations and derived results accordingly of these two kinds of networks. Due to some difficulties, the topology referred to in the source paper could not be replicated and thus the improvements in the wired-cum-wireless network could not be emulated by the project. Visualisations of both the networks are as follows:

Wireless static nodes:



Wireless mobiles nodes:



Parameters under variation:

The parameters varied for the topologies in the simulation are as follows:

1. Number of nodes: 20, 40, 60, 80, 100
2. Number of flows: 10, 20, 30, 40, 50
3. Number of packets sent per second: 100, 200, 300, 400, 500
4. Coverage area: square of Transmission Range times 1, 2, 3, 4, 5 (applicable for the simulation involving static nodes)
5. Speed of mobile nodes: 5 m/s, 10 m/s, 15 m/s, 20 m/s, 25 m/s (applicable for the simulation involving mobile nodes)

Modifications made:

The modification of this project is regarding the congestion control mechanism of TCP. The reference paper indicates for the congestion window to be constant for the duration the network scenario is constant. That is, no stray packet losses will be taken as the cause of congestion, rather the network scenario should be monitored with rtt measurements. The optimal congestion window calculated on the basis of those measurements will be the constant window value for the TCP agent.

The congestion control in TCP Agents are divided into three phases:

1. Slow start: The initial phase of TCP connection establishment in which the sender starts transmitting data to the receiver with a conservative transmission rate. During this phase, the sender increases its transmission rate exponentially until it detects congestion in the network.
2. Congestion Avoidance: The slow start threshold (ssthresh) is reached, and we increase the congestion window according to the protocol.
3. Congestion Control: Congestion is detected, and similar to avoidance, a procedure is followed to decrease the window to decrease the loss due to congestion.

These phases are handled by two functions in ns2, namely opencwnd() and slowdown().

The modifications I made were on the TCP Westwood protocol, as the bandwidth calculation of TCPW was necessary for the calculations of fair share. But ns-2.35 did not entail TCP Westwood in cpp format, so I imported the necessary files from here. The protocol was inherited from the default TCP Agent (Tahoe). The slowdown() function is overridden but the opencwnd() function is not.

I implemented the following changes:

In tcp-westwood-nr.h:

1. New variables for optimal congestion window calculation:

```
    double current_ts_;
    double current_echoed_ts_;
    int last_seq_;
    double last_cwnd_;
    int openadd_;

    /* these where originally in class TcpAgent (file: tcp.h) */
    int mss_;            /* Maximum Segment Size - MGM+CC 31/08/2000 */
    double current_bwe_;    /* Current Bandwidth estimation */
        double last_bwe_sample_;/* Last sample used to compute BWE */
        int unaccounted_;       /* unaccounted ACKs already received */
        double fr_a_;           /* bandwidth reduction factor */
        double min_rtt_estimate;/* smaller recorded RTT estimate */
        //! modifications
        double rtt_archive;     // used to store the last rtt value for which the cwnd was modified
        int k_rounds;           // used to count the number of rounds of slowstart
        int modified;           // used to indicate if the protocol is modified or not
        TracedInt myseqno_;     /* my own exportable copy of seqno */

        virtual void bwe_computation(Packet *pkt);

};

#endif /* tcp_w_nr_h */
```

2. Modified functions to run when modifications are enabled:

```
class WestwoodNRTcpAgent : public virtual NewRenoTcpAgent {
public:
    WestwoodNRTcpAgent();
    virtual void recv(Packet *pkt, Handler*);
    virtual void dupack_action();
    virtual void timeout (int tno);

    /* these where originally in class TcpAgent (file: tcp.h) */
    //! modifications
    virtual void opencwnd();
    virtual void modifiedopencwnd(); // overriden opencwnd to control slow start
    virtual void modifiedslowdown(int how); // overriden slowdown to keep constant cwnd
    virtual void slowdown(int how);
    virtual void newack(Packet* pkt);

    virtual int  delay_bind_dispatch(const char *varName, const char *localName, TclObject *tracer);

protected:
    double lastackno_;  /* Last ACK number */
    double lastackrx_;  /* Time last ACK was received */
    double fr_alpha_;   /* exponential averaging coefficient */
    int filter_type_;   /* exponential filter type */
    double tau_;        /* time constant used in filter 3 */
```

In tcp-westwood-nr.cc:

1. Initialization of the new variables:

tcp-westwood-nr.cc 9+ | tcp-westwood-nr.h 2 | ns-default.tcl | tcp-newreno.cc

ns-2.35 > tcp > tcp-westwood-nr.cc > WestwoodNRTcpAgent()

```cpp
} class_westwoodnr;

/////
// WestwoodNRTcpAgent()
WestwoodNRTcpAgent::WestwoodNRTcpAgent() : NewRenoTcpAgent(),
  // these where originally in TcpAgent()
  current_bwe_(0), last_bwe_sample_(0), unaccounted_(0),
  fr_a_(0), min_rtt_estimate(100.0), rtt_archive(5.0), k_rounds(20), modified(0), myseqno_(1),last_ts_(0),last_echoed_ts_(0),last_s
  lastackrx_(0.0), fr_alpha_(0.9), filter_type_(1), tau_(1.0), total_time_(0.0), total_size_(0.0), fr_prev_(20.0)

{
    // Read defaults variables from ns-defaults.tcl

    // these where originally in TcpAgent()

    bind("current_bwe_", &current_bwe_);
    bind("last_bwe_sample_", &last_bwe_sample_);
    bind("unaccounted_", &unaccounted_);
    bind("fr_a_", &fr_a_);
    bind("fr_amin_", &fr_amin_);
    bind("fr_amax_", &fr_amax_);
    bind("fr_prev_", &fr_prev_);
    bind("min_rtt_estimate", &min_rtt_estimate);
```

2. Change in slowdown process:

tcp-westwood-nr.cc 9+ | tcp-westwood-nr.h 2 | ns-default.tcl | tcp-newreno.cc

ns-2.35 > tcp > tcp-westwood-nr.cc > modifiedopencwnd()

```cpp
//! modifications
void WestwoodNRTcpAgent::modifiedopencwnd()
{
    double increment;
    if (cwnd_ < ssthresh_ && k_rounds > 0) {
        /* slow-start (exponential) */
        cwnd_ += 1;
        // printf("modified: %d\n", modified);
        //! modifications
        if(modified){
            double rtt_estimate = t_rtt_ * tcp_tick_;
            rtt_archive = rtt_estimate;
            // printf("rtt_estimate=%f, t_rtt_=%d\n", rtt_estimate,(int) t_rtt_);
            k_rounds--;
        }
    }
//  else {
//      /* linear */
//      double f;
//      switch (wnd_option_) {
//      case 0:
//          if (++count_ >= cwnd_) {
//              count_ = 0;
```

3. Congestion window calculation:

tcp-westwood-nr.cc 9+ | tcp-westwood-nr.h 2 | ns-default.tcl | tcp-newreno.cc

ns-2.35 > tcp > tcp-westwood-nr.cc > modifiedopencwnd()

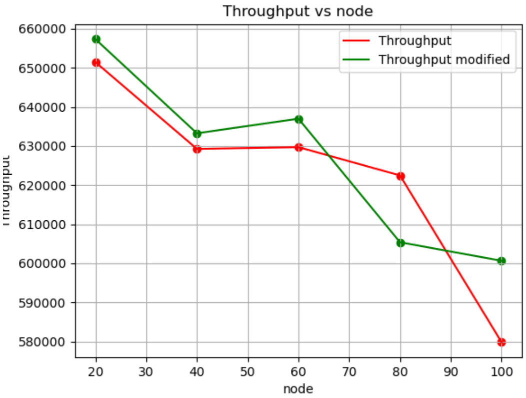
```cpp
    // //             (#of ticks) * (tick duration)
    // // We need to do away with the coarseness...


    double rtt_estimate = t_rtt_ * tcp_tick_;
    // printf("before min");
    if ((rtt_estimate <= min_rtt_estimate)&&(rtt_estimate > 0)) {
        min_rtt_estimate = rtt_estimate;
        // printf("in min\n");
    }


    double rtt_change = rtt_archive - rtt_estimate;
    double beta_ = 0.015;
    // printf("rtt_change: %f\n", rtt_change);
    // printf("rtt_archive: %f\n", rtt_archive);
    if ((rtt_archive == 0 || abs(rtt_change)/rtt_archive) > beta_) {
        cwnd_ = (((current_bwe_*(min_rtt_estimate))/((double)(size_*8.0))));
        if (cwnd_ < 1) cwnd_ = 1;
        rtt_archive = rtt_estimate;
        // printf("cwnd: %d\n", (int) cwnd_);
        // printf("rtt_archive after cwnd change: %f\n", rtt_archive);
```

Results with graphs:

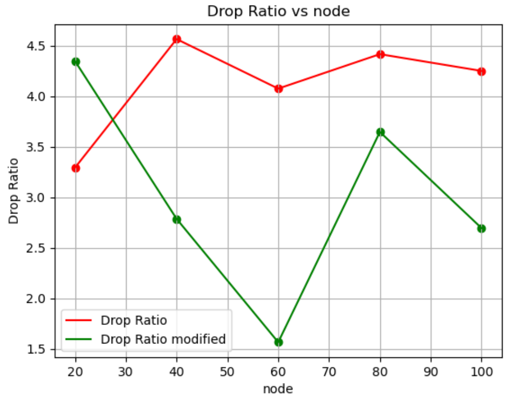
1. Wireless 802.11 static nodes:

With respect to Number of Nodes:

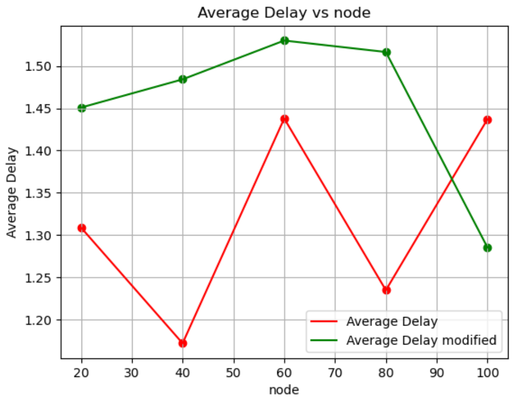
Network Throughput:



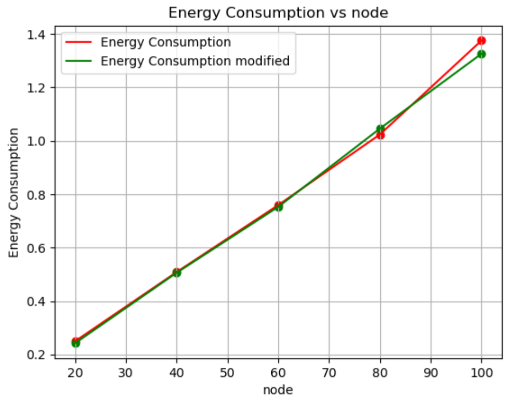
Packet Drop Ratio:



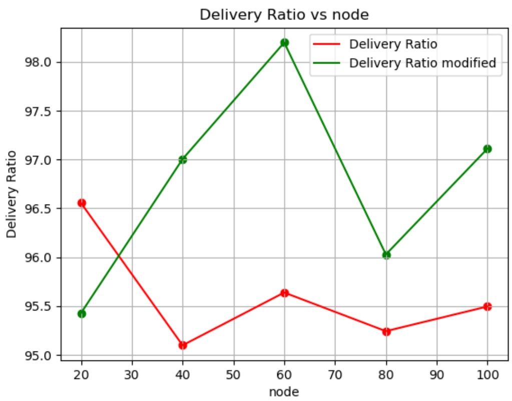
End-to-End Delay:



Energy Consumption:

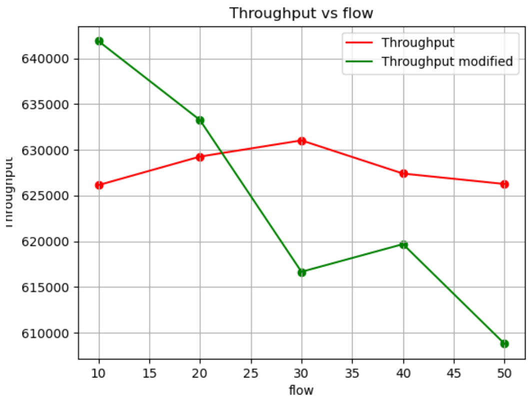


Packet Delivery Ratio:

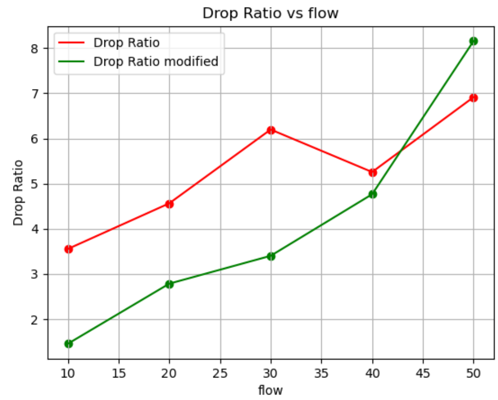


With respect to Number of Flows:

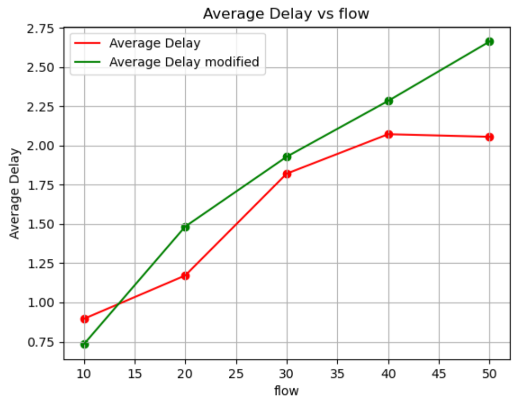
Network Throughput:



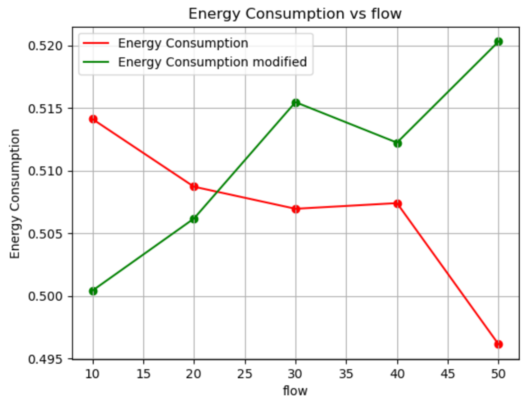
Packet Drop Ratio:



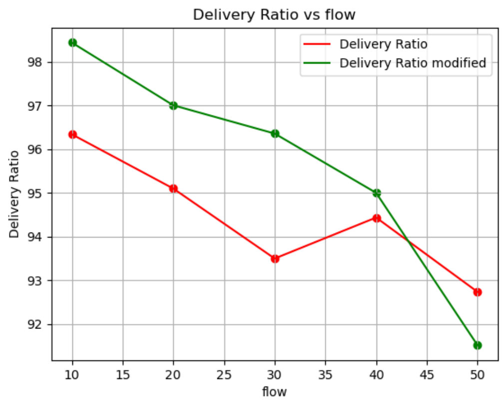
End-to-End Delay:



Energy Consumption:

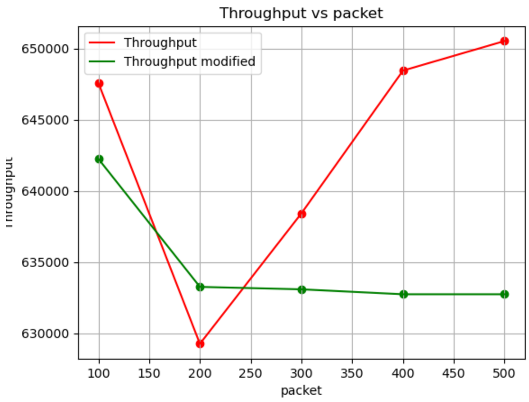


Packet Delivery Ratio:

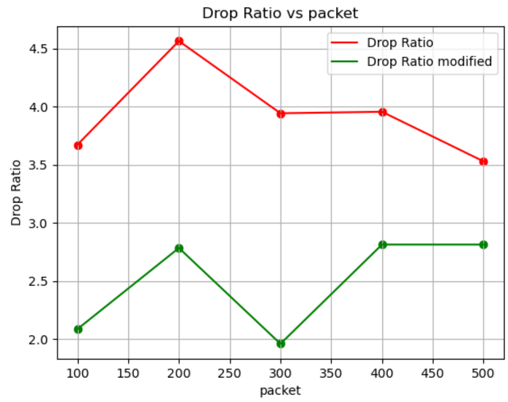


With respect to Number of Packets per second:

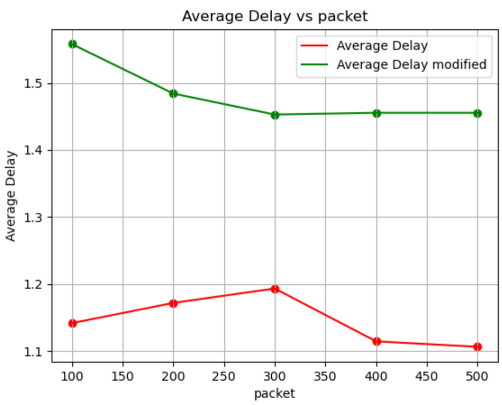
Network Throughput:



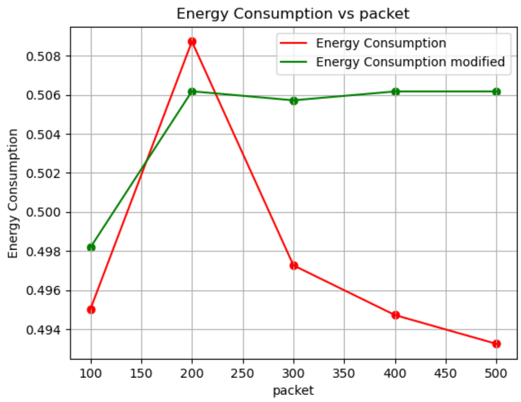
Packet Drop Ratio:



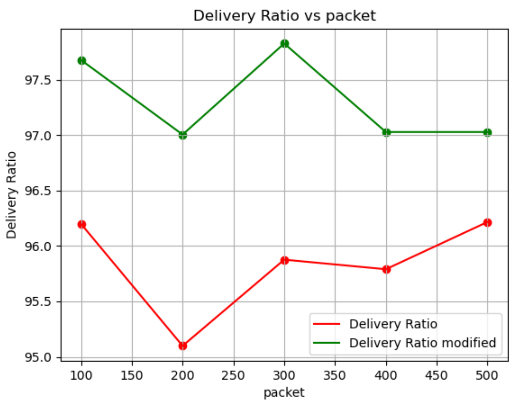
End-to-End Delay:



Energy Consumption:

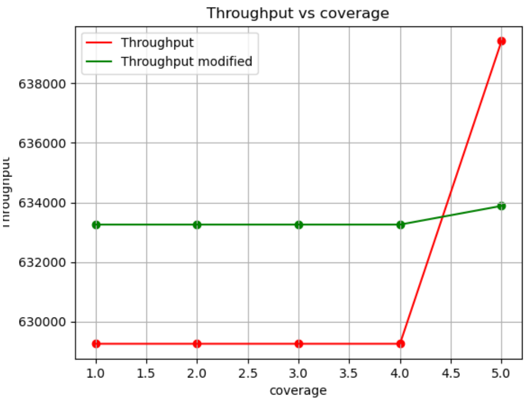


Packet Delivery Ratio:

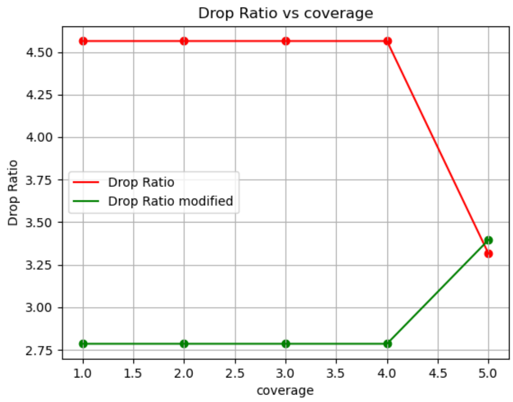


With respect to Coverage Area:

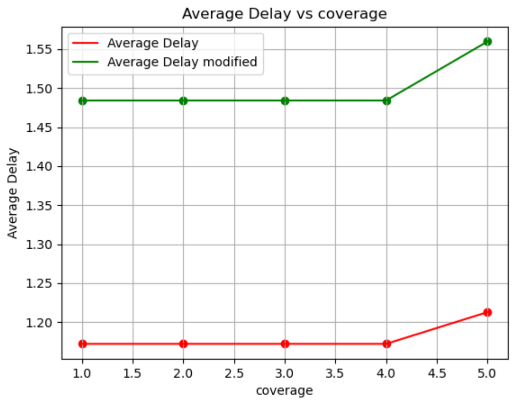
Network Throughput:



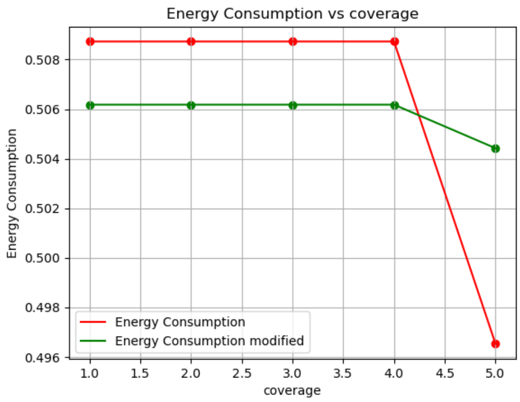
Packet Drop Ratio:



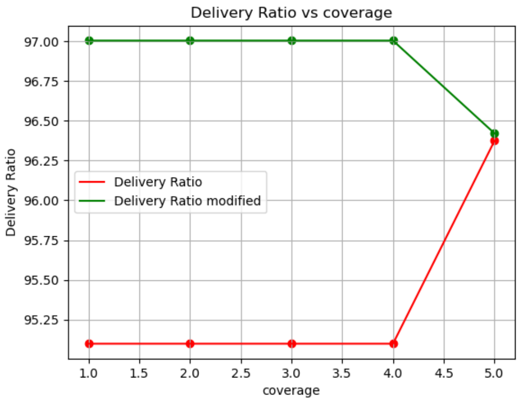
End-to-End Delay:



Energy Consumption:



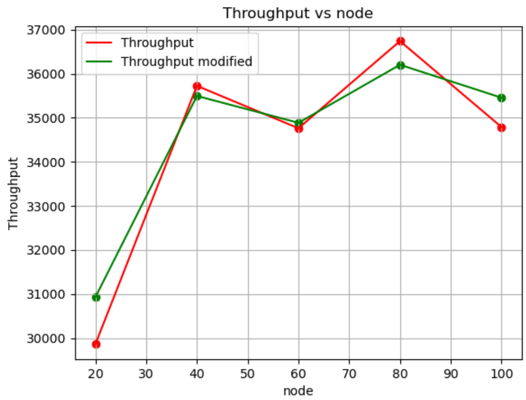
Packet Delivery Ratio:



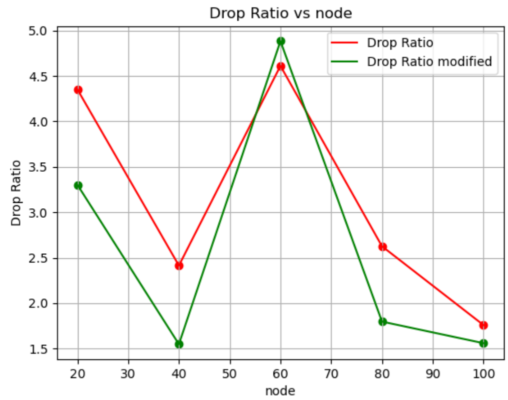
2. Wireless 802.15.4 mobile nodes:

With respect to Number of Nodes:

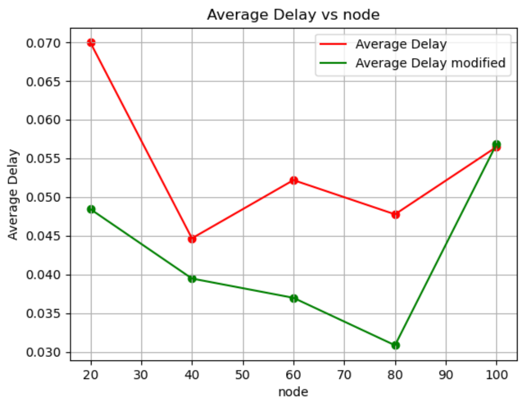
Network Throughput:



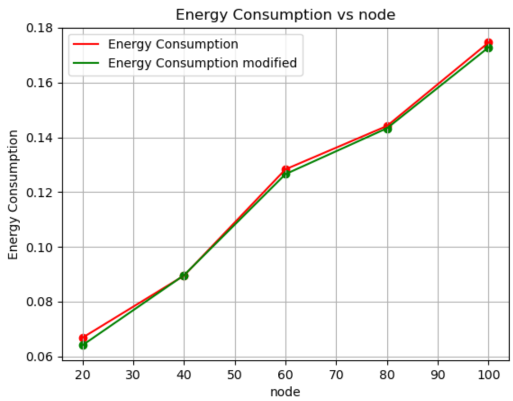
Packet Drop Ratio:



End-to-End Delay:



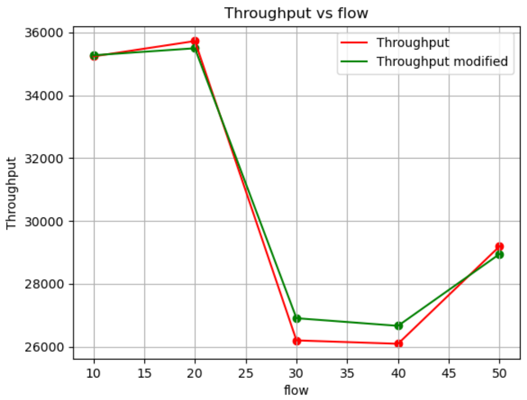
Energy Consumption:



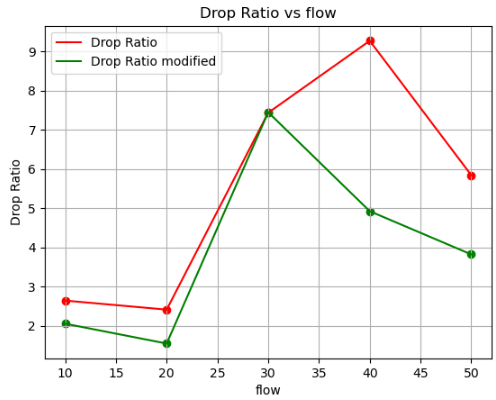
Packet Delivery Ratio:

With respect to Number of Flows:

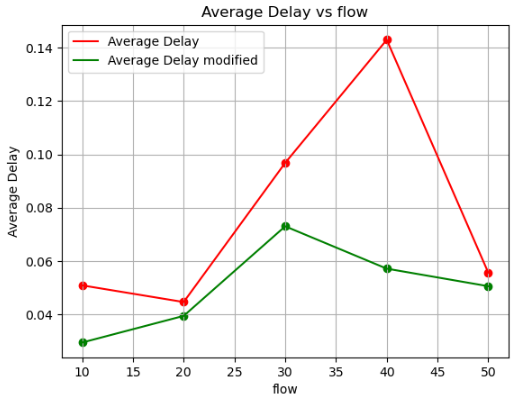
Network Throughput:



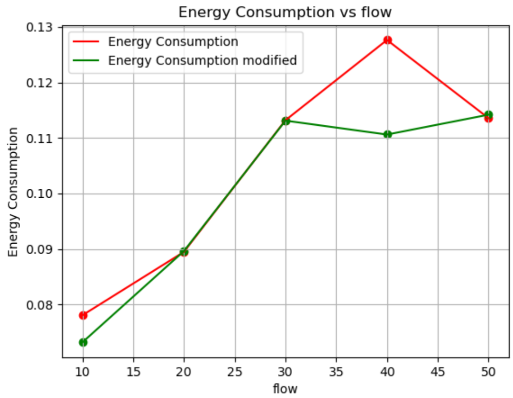
Packet Drop Ratio:



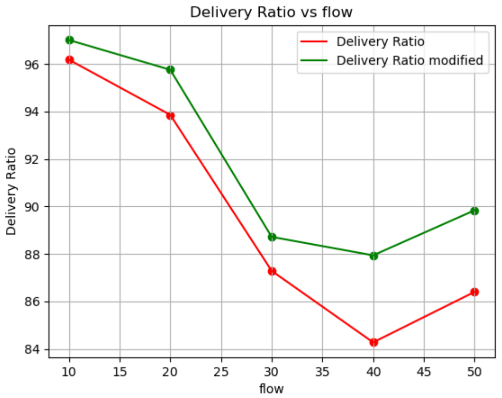
End-to-End Delay:



Energy Consumption:

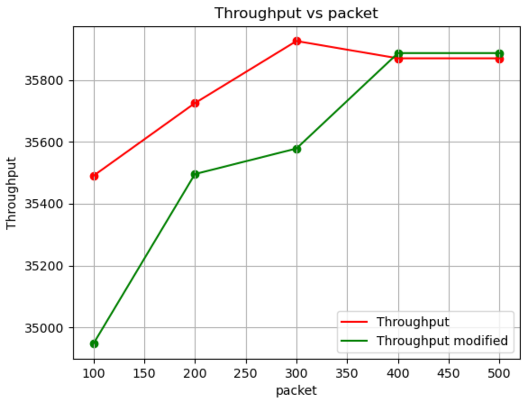


Packet Delivery Ratio:

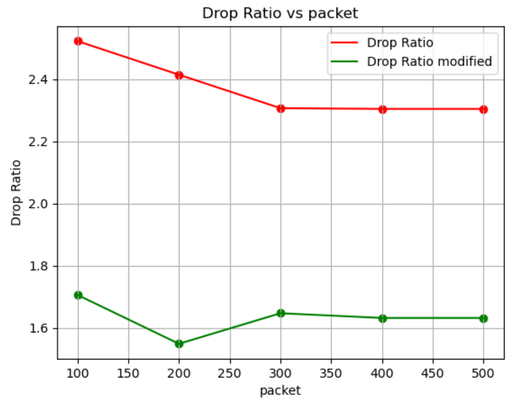


With respect to Number of Packets per second:

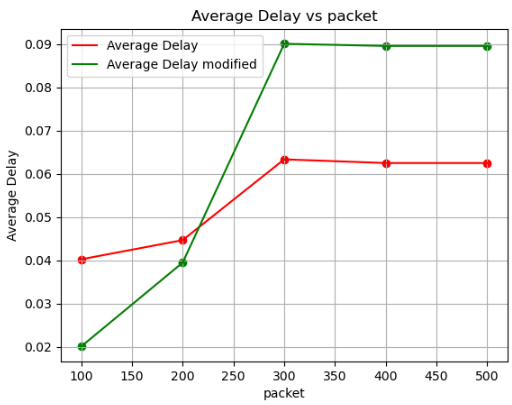
Network Throughput:



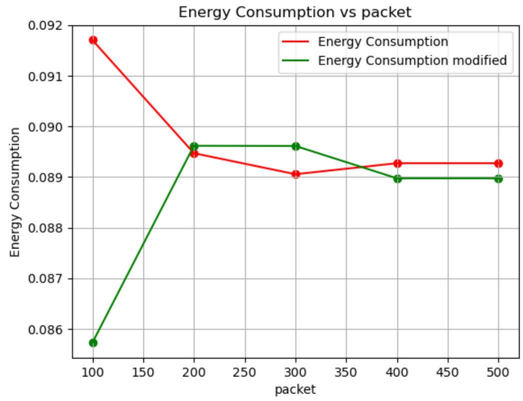
Packet Drop Ratio:



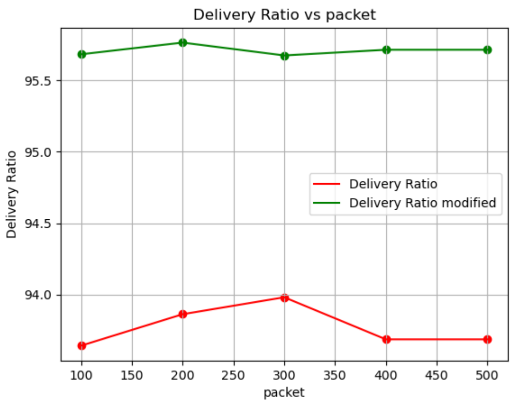
End-to-End Delay:



Energy Consumption:

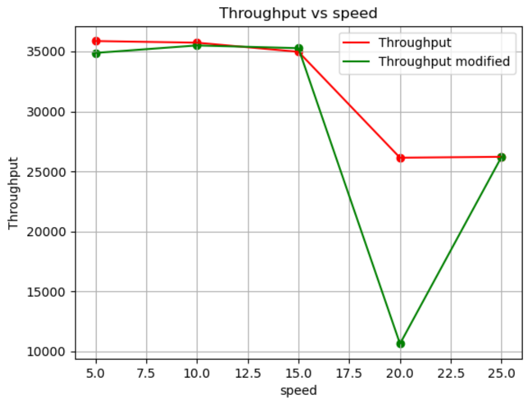


Packet Delivery Ratio:

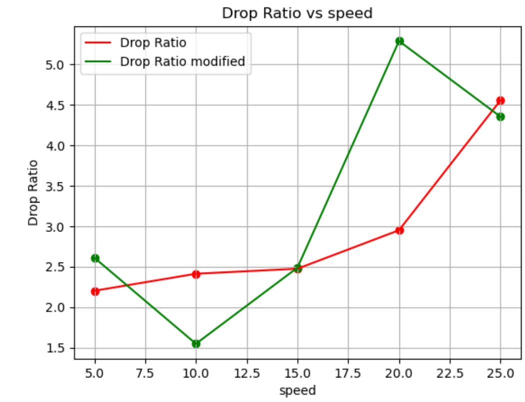


With respect to Node Speed:

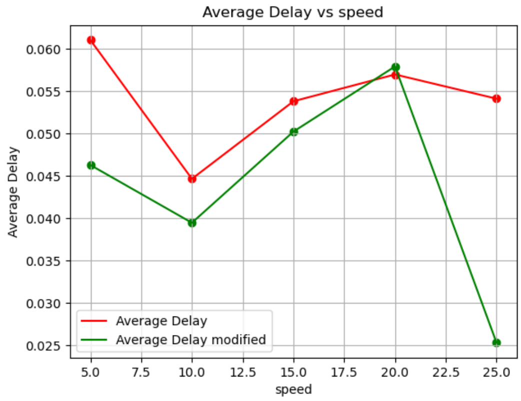
Network Throughput:



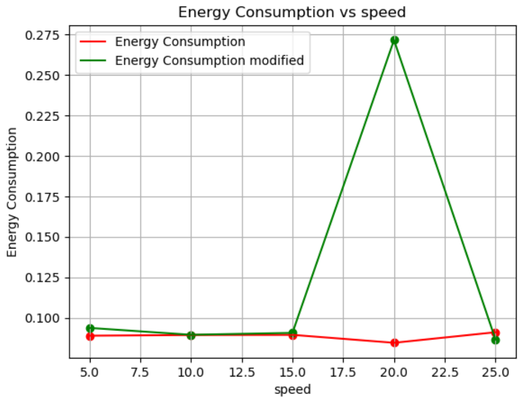
Packet Drop Ratio:



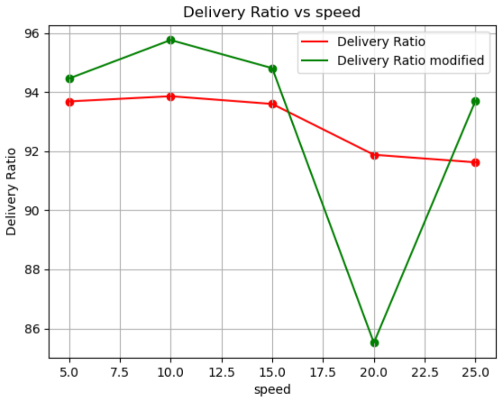
End-to-End Delay:



Energy Consumption:



Packet Delivery Ratio:



Summary findings:

The graphs do not show the improvements expected from the implementation of the algorithm. The reason behind that may be the limitations of ns2 in providing stray packet losses. It is to be noted that a uniform error model was also used in the simulation to generate the stray error in receiving the files by the TCP Sink. But even after applying such models to introduce stray packet loss during transmission, the results remain the same.

Both the assigned topologies are wireless, where there is slight to no change of energy consumption. Other than that, the values of the metrics after compared to the coverage area are very very close, similar at times. We may infer that coverage area does not hold much impact on the wireless simulations, and study the graphs accordingly. The end-to-end average delay changes unpredictably in most cases. Packet delivery ratios show improvements, but only in specific cases. Throughput remains worse for most part of the simulation models, and it

The modification of slow start could be made better for the reaching of the initial optimal congestion window. The bandwidth estimation algorithm could also be better for a better estimate of the cwnd.

The topology explained in the reference paper would be a benchmark to understand the difference of the other used protocols against this modified protocol, but the simulation model could not be created.

To conclude, the implementation of TCP constant cwnd has not provided enough improvement in the simulations to prove the eligibility of the protocol as an improved version than Westwood.