Computer Networks Sessional
CSE322
Report: TCP Congestion Control Scheme for Wireless Networks based on TCP Reserved Field and SNR Ratio in NS2

Hasan Masum
1805052

Bangladesh University of Engineering & Technology

February 28, 2023

Contents

1 Network topologies under simulation

1.1 Wired

We used the following configuration for wired simulation with varying nodes, number of flows, and number of packets per second.

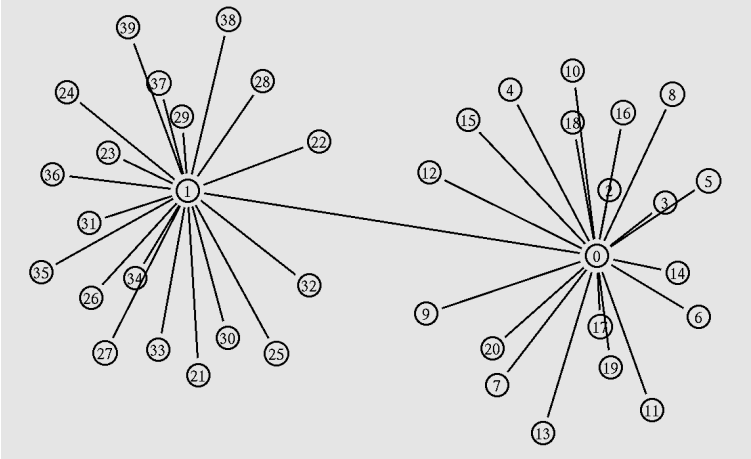


Figure 1: Wired Topology

1.2 Wireless Bluetooth(802.15.4)(Mobile)

We used the following configuration for wireless simulation with varying nodes, number of flows, number of packets per second and speed of nodes.

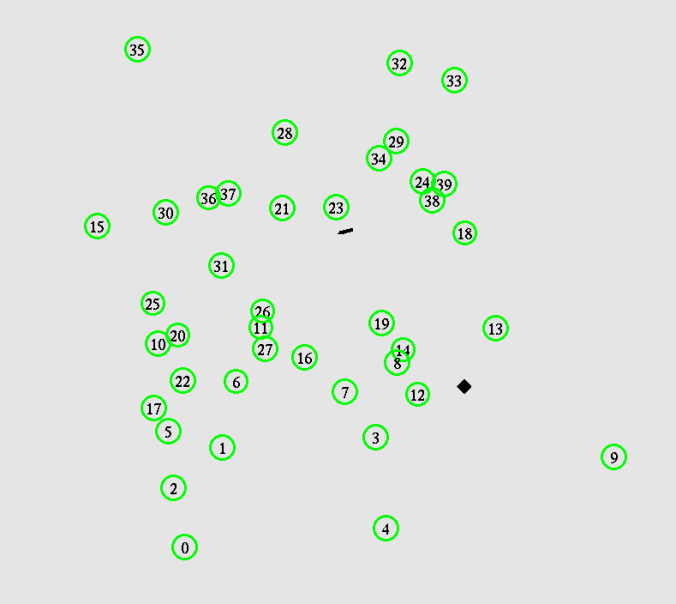


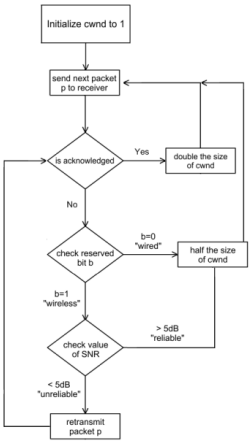
Figure 2: Wireless Topology

2 Modifications Made in NS2

2.1 Proposed Algorithm

1. Initialize congestion window *cwnd* to one segment.
2. Start sending packets to receiver.
3. For each acknowledgement received, increment congestion window *cwnd* by one segment.
4. If a timeout occurs for a particular packet *p* then check the value of the reserved bit *b* for packet *p*.
   1. If *b=0 (wired link)*, then set *cwnd* to half of its previous size (indicating a congestion)
   2. Else if *b=1 (wireless link)*, then check the SNR ratio of the link.
      1. If SNR>5dB, then set *cwnd* to half of its previous size (indicating a congestion)
      2. Else if SNR<5dB, then don't change the current size of *cwnd*, and retransmit packet *p* (indicating an error)

(a) Pseudocode



(b) Flowchart

Figure 3: TCP Congestion Control Scheme for Wireless Networks based on TCP Reserved Field and SNR Ratio.

2.2 Implementation in NS2

2.2.1 Use reserve bit of TCP header file

Step-1: Add wireless flag variable in hcp$_h drstructure of$"$tcp/tpc.h$"$ $file$.
Step-2: Init wireless flag in the output() function of "tcp/tcp.cc" file.

```
647  void TcpAgent::output(int seqno, int reason)
648  {
649      int force_set_rtx_timer = 0;
650      Packet* p = allocpkt();
651      hdr_tcp *tcph = hdr_tcp::access(p);
652      hdr_flags* hf = hdr_flags::access(p);
653      hdr_ip *iph = hdr_ip::access(p);
654      int databytes = hdr_cmn::access(p)->size();
655      tcph->seqno() = seqno;
656      tcph->ts() = Scheduler::instance().clock();
657      int is_retransmit = (seqno < maxseq_);
658+     tcph->wireless() = 0;
659      // Mark packet for diagnosis purposes if we are in Quick-Start Phase
660      if (qs_approved_) {
661          hf->qs() = 1;
662      }
663
```

Figure 5: tpc/tcp.cc

Step-3: Set wireless flag to 1 in sendUp(...) function of tcp/channel.cc

```
316 void
317 WirelessChannel::sendUp(Packet* p, Phy *tifp)
318 {
319     Scheduler &s = Scheduler::instance();
320     Phy *rifp = ifhead_.lh_first;
321     Node *tnode = tifp->node();
322     Node *rnode = 0;
323     Packet *newp;
324     double propdelay = 0.0;
325     struct hdr_cmn *hdr = HDR_CMN(p);
326+    // check if tcp packet
327+    if (hdr->ptype() == PT_TCP || hdr->ptype() == PT_ACK) {
328+        // get tcp header and set wireless flag
329+        struct hdr_tcp *tcph = hdr_tcp::access(p);
330+        tcph->wireless() = 1;
331+    }
332
```

Figure 6: tpc/channel.cc

2.2.2 Simulate packet loss and snr in ns2

Add a global static snr variable in "mac/wireless-phy.h" WirelessPhy class

```
64 //
65 class WirelessPhy : public Phy {
66 public:
67     WirelessPhy();
68+    static double snr;
69     void sendDown(Packet *p);
70     int sendUp(Packet *p);
71
```

Figure 7: mac/wireless-phy.h

Then based on the distance between the transmitter and receiver and the path-loss model(Friis) used in "mobile/tworayground.cc" Pr() function to calculate an snr value for simulating noise.

Also, drop packet randomly in to simulate packet loss in "mac/wireless-phy.cc" sendUp(...) method to retransmit it in tcp-snr.cc

```
348     if(propagation_) {
349         s.stamp((MobileNode*)node(), ant_, 0, lambda_);
350         Pr = propagation_->Pr(&p->txinfo_, &s, this);
351+        //if (Pr < CSThresh_) {
352+        double pdr = 0.9;
353+        double x = Random::uniform(0.8, MAX(1, WirelessPhy::snr /4.0) );
354+        if (x < pdr) {
355             pkt_recvd = 0;
356+            //printf("Packet dropped due to low power\n");
357             goto DONE;
358         }
```

Figure 8: mobile/tworayground.cc

2.2.3 Implementation of the TCP congestion control algorithm

A new file named tcp-snr.cc is added in tcp directory of the project which extends TcpAgent class and creates a new agent class named TCPSnr. TCPSnr class overrides the recv(...) function to implements the proposed algorithm

```
10
11  class TCPSnr : public virtual TcpAgent
12  {
13  private:
14      //double get_snr(Packet *p);
15  public:
16      void recv(Packet *pkt, Handler *);
17  };
18
19  static class TCPSnrAgentClass : public TclClass
20  {
21  public:
22      TCPSnrAgentClass() : TclClass("Agent/TCP/TCPSnr") {}
23      TclObject *create(int, const char *const *)
24      {
25          return (new TCPSnr());
26      }
27  } class_tcp_snr_agent;
28
```

Figure 9: TCPSnr and TCPSnrAgent class implementation

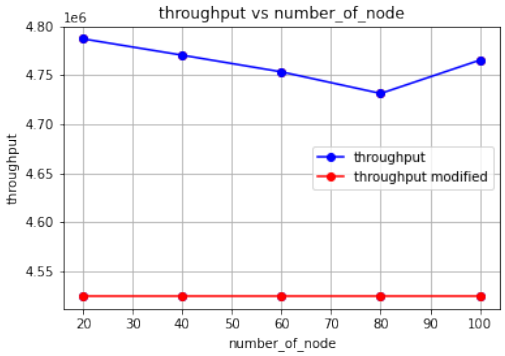
```
33  void TCPSnr::recv(Packet *pkt, Handler *){
34      hdr_tcp *tcph = hdr_tcp::access(pkt);
35      int valid_ack = 0;
36      ++nackpack_;
37      if(tcph->seqno() > last_ack_){
38          // new ack
39          newack(pkt);
40      }else if(tcph->seqno() == last_ack_){
41          // duplicate ack
42          ++dupacks_;
43          if(dupacks_ == 3){
44              // fast retransmit
45              if(tcph->wireless()){
46                  // half congestion window size
47                  if(WirelessPhy::snr < 5){
48                      // retransmit the packet lost
49                      dupack_action();
50                  }else {
51                      cwnd_ = max(cwnd_ / 2, 1.0);
52                  }
53              }else{
54                  cwnd_ = max(cwnd_ / 2, 1.0);
55              }
56          }
57      }
58      if(tcph->seqno() >= last_ack_){
59          valid_ack = 1;
60      }
61      // deallocate the
62      Packet::free(pkt);
63      if(valid_ack){
64          send_much(0, 0, maxburst_);
65      }
66  }
67
```

Figure 10: recv() function implementation

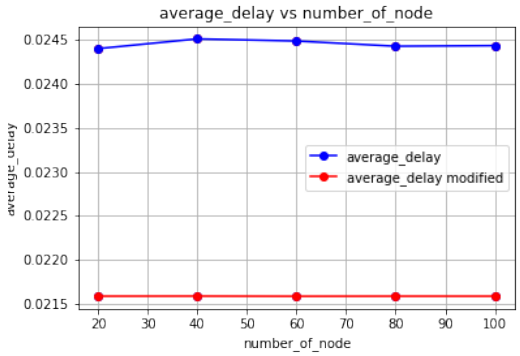
3 Results with varying parameters

3.1 Wired

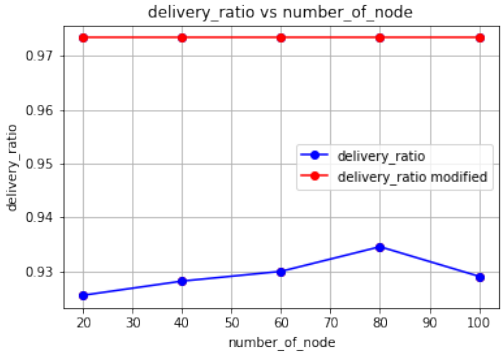
3.1.1 Varying Nodes



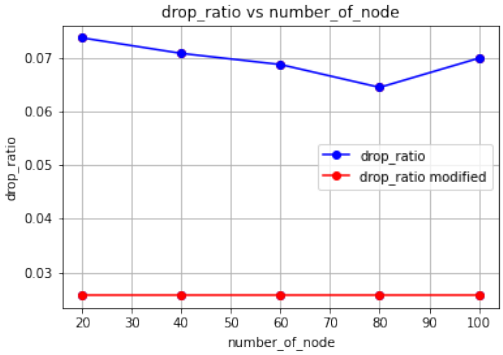
(a) Throughput vs #Nodes



(b) Average Delay vs #Nodes



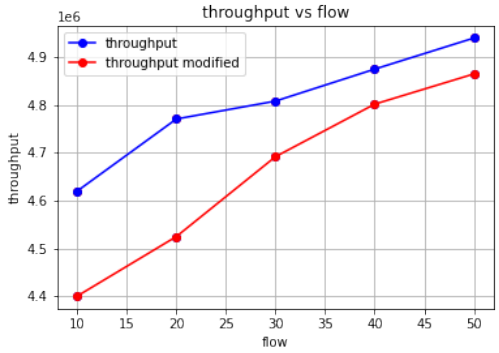
(c) Delivery Ratio vs #Nodes



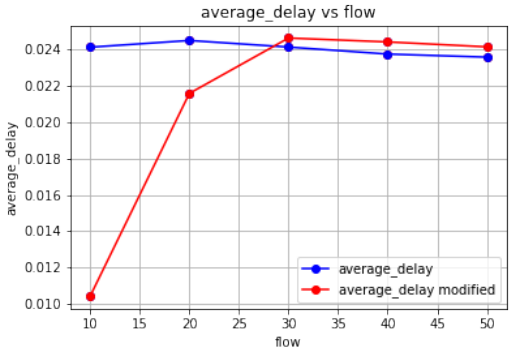
(d) Drop Ratio vs #Nodes

Figure 11: Measuring Against Nodes.

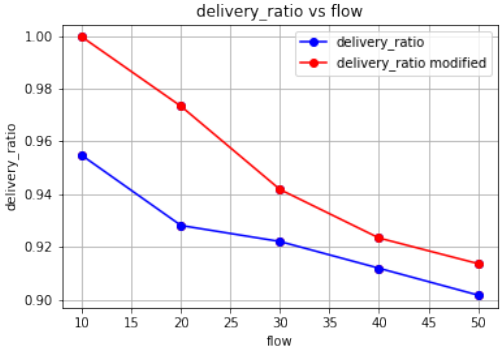
3.1.2 Varying Flows



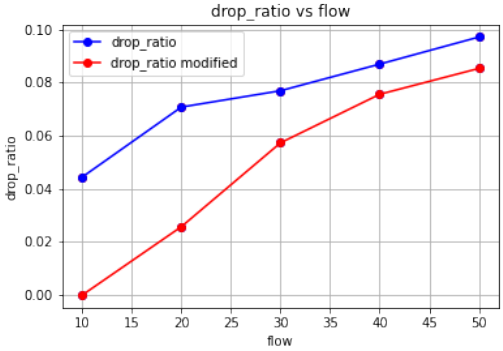
(a) Throughput vs #Flows



(b) Average Delay vs #Flows



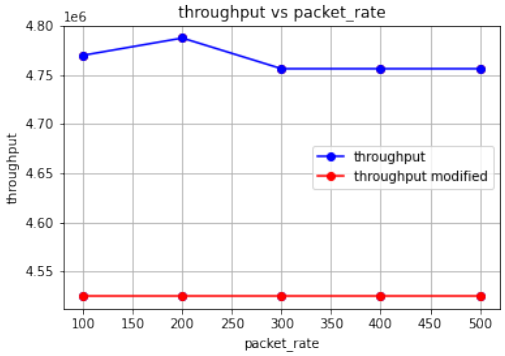
(c) Delivery Ratio vs #Flows



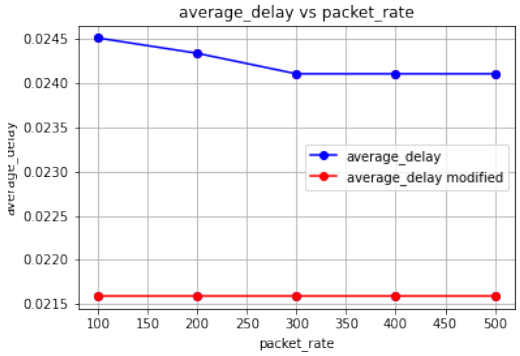
(d) Drop Ratio vs #Flows

Figure 12: Measuring Against Flows.

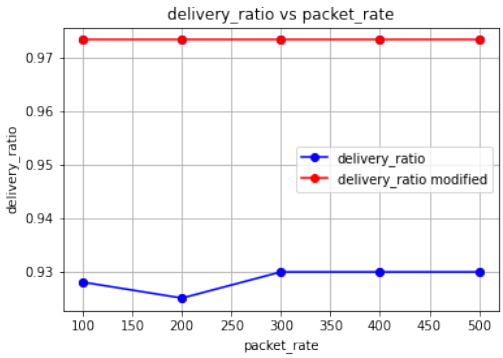
3.1.3 Varying Packets Per Second



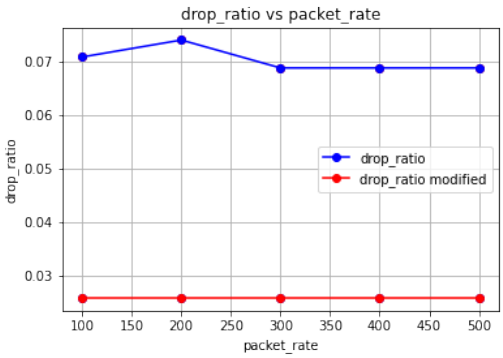
(a) Throughput vs #Packets Per Second



(b) Average Delay vs #Packets Per Second



(c) Delivery Ratio vs #Packets Per Second

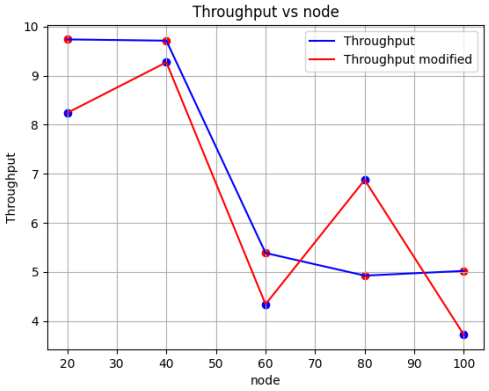


(d) Drop Ratio vs #Packets Per Second

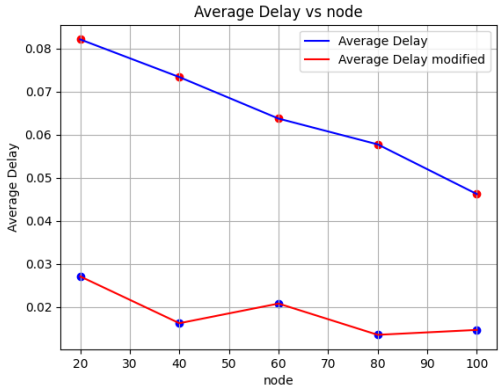
Figure 13: Measuring Against Packets Per Second.

3.2 Wireless Bluetooth(802.15.4)(Mobile)

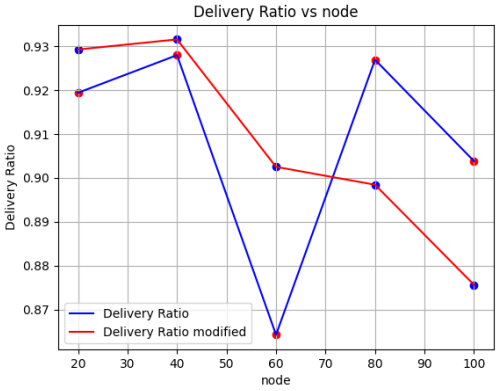
3.2.1 Varying Nodes



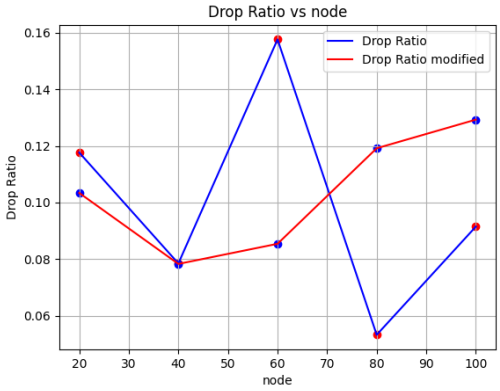
(a) Throughput vs #Nodes



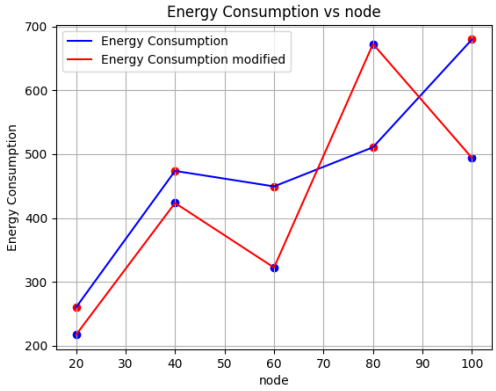
(b) Average Delay vs #Nodes



(c) Delivery Ratio vs #Nodes



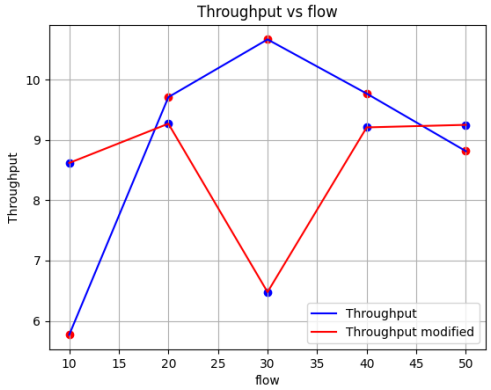
(d) Drop Ratio vs #Nodes



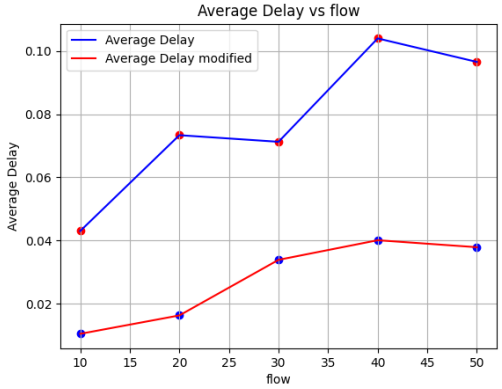
(e) Energy Consumption vs #Nodes

Figure 14: Measuring Against Nodes.

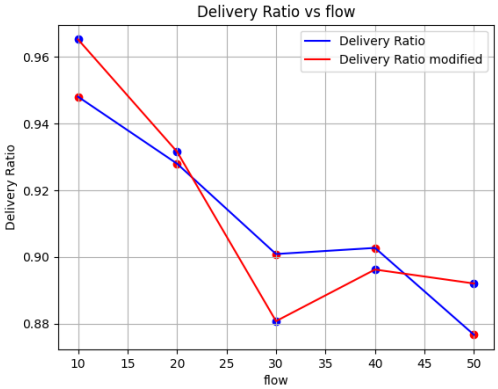
3.2.2 Varying Flows



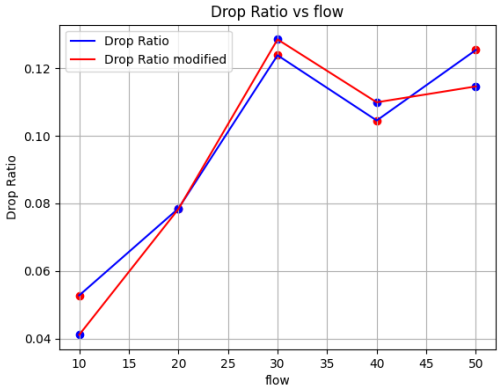
(a) Throughput vs #Flows



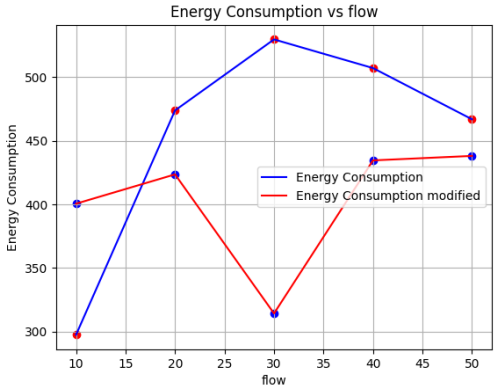
(b) Average Delay vs #Flows



(c) Delivery Ratio vs #Flows



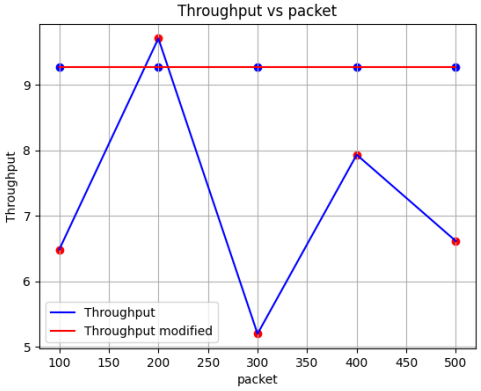
(d) Drop Ratio vs #Flows



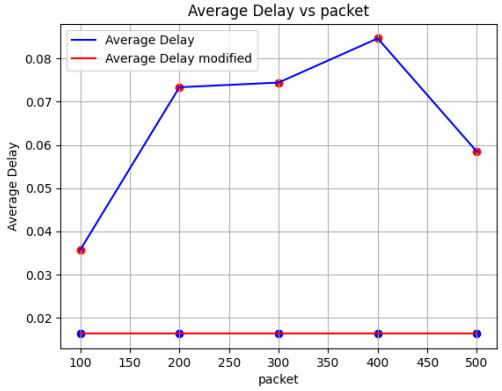
(e) Energy Consumption vs #Flows

Figure 15: Measuring Against Flows.

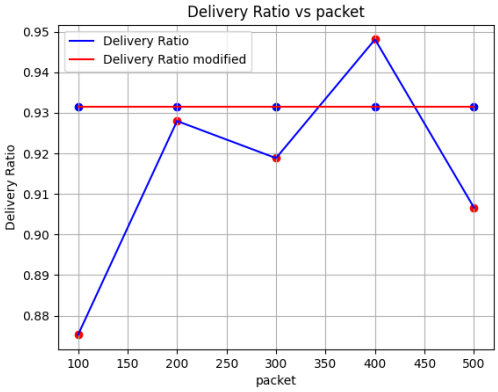
3.2.3 Varying Packets Per Second



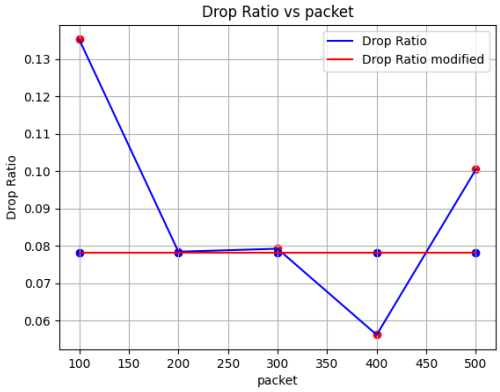
(a) Throughput vs #Packets Per Second



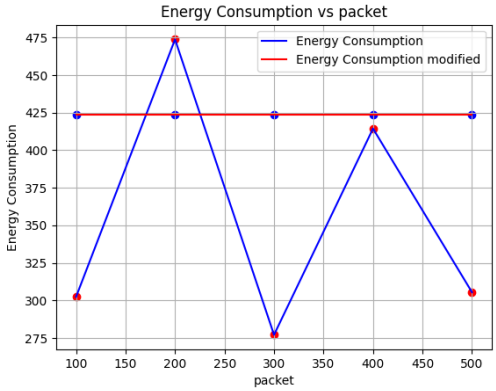
(b) Average Delay vs #Packets Per Second



(c) Delivery Ratio vs #Packets Per Second



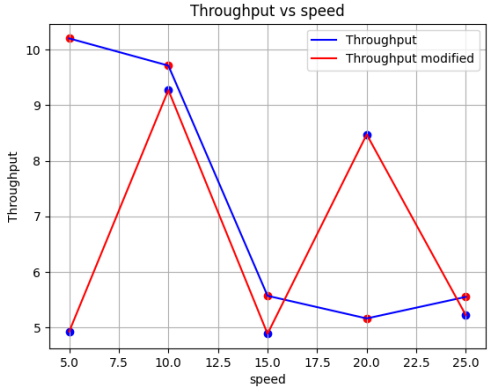
(d) Drop Ratio vs #Packets Per Second



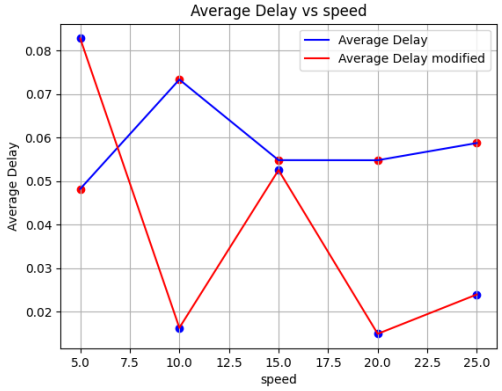
(e) Energy Consumption vs #Packets Per Second

Figure 16: Measuring Against Packets Per Second.

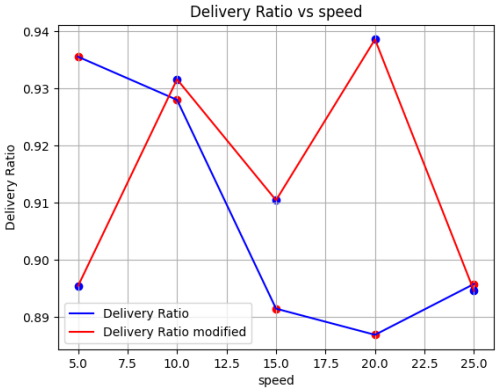
3.2.4 Varying Speed



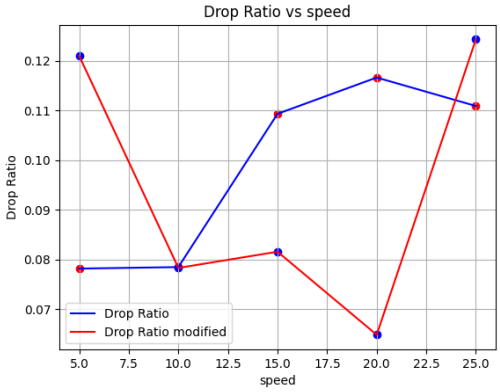
(a) Throughput vs Speed



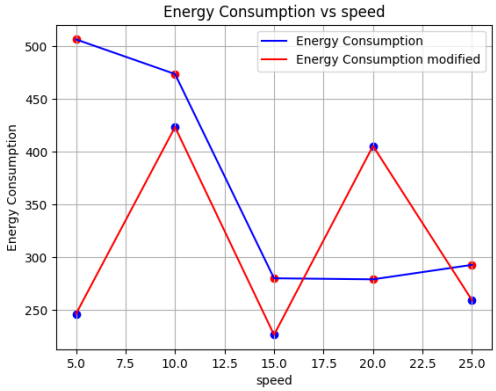
(b) Average Delay vs Speed



(c) Delivery Ratio vs Speed



(d) Drop Ratio vs Speed



(e) Energy Consumption vs Speed

Figure 17: Measuring Against Speed.

4 Conclusion

The result from the above experiments shows that TCP Snr has little impact on wired networks. But it improves performance a bit for the wireless network. But the change is a bit random as ns2 doesn't have any noise simulation and snr calculation method by default and we implemented our own randomized snr calculation and packet loss method.