Bangladesh University of Engineering and Technology

Department of CSE

Course Code: CSE 322

Course: Computer Networks Sessional

Project: **INVS: TCP congestion control algorithm for heterogeneous Internet**

Submitted by:

Azizur Rahman Anik

Student Id: 1805115

Introduction

INVS is an improved congestion control protocol for heterogeneous Internet, which takes into account the path condition, diversity of communication link, lossy links and provides adaptive congestion window in congestion avoidance phase. It also tries to analyze if a packet loss is a random loss or due to congestion. INVS uses an exponential function based convex window growth function at the front of each congestion avoidance phase and a concave function for exploring available resources. It uses bandwidth estimation to adapt the path condition and buffer estimation to detect the cause of packet loss.

Proposed Algorithm

1) Growth function of congestion window is adjusted according to path condition.

$$cwnd \mathrel{+}= \begin{cases} \dfrac{cwnd_{sp} - cwnd}{k \cdot cwnd_{sp}}, & cwnd \le cwnd_{sp} - 1 \\ \dfrac{cwnd - cwnd_{sp}}{k \cdot cwnd}, & cwnd \ge cwnd_{sp} + 1 \\ \dfrac{1}{cwnd}, & cwnd_{sp} - 1 < cwnd < cwnd_{sp} + 1 \end{cases}$$

k is designed to map the variation of bandwidth and RTT

$$k = c\left(\log_2(r_{bw} r_{rtt}^{\gamma}) + 1\right), \quad 0 < \gamma < 1$$

$$\text{where } r_{bw} = \max\left[\frac{BW_{ref}}{BW_{est}}, 1\right], \; r_{rtt} = \max\left[\frac{RTT_{ref}}{RTT_{min}}, 1\right]$$

2) Action upon packet loss and loss classification scheme

$$cwnd = ssthresh = \begin{cases} \beta \cdot cwnd, \; buffer_{est} \geq \min(\delta, \; maxbuffer) \\ \min(BDP_{est}, \; cwnd), \; else \end{cases}$$

Here, BDPest is the estimated path BDP, bufferest is the estimated number of packets currently queued in the networks and δ is a predefined threshold of queue length for large buffer links whose default value is 5

Upon ACK, ABE algorithm is used in the paper, but due to the complexity of the algorithm and lack of proper clarification and being unable to find the proper variables in NS2, I have applied a simple time interval bandwidth estimation algorithm which is shown below.

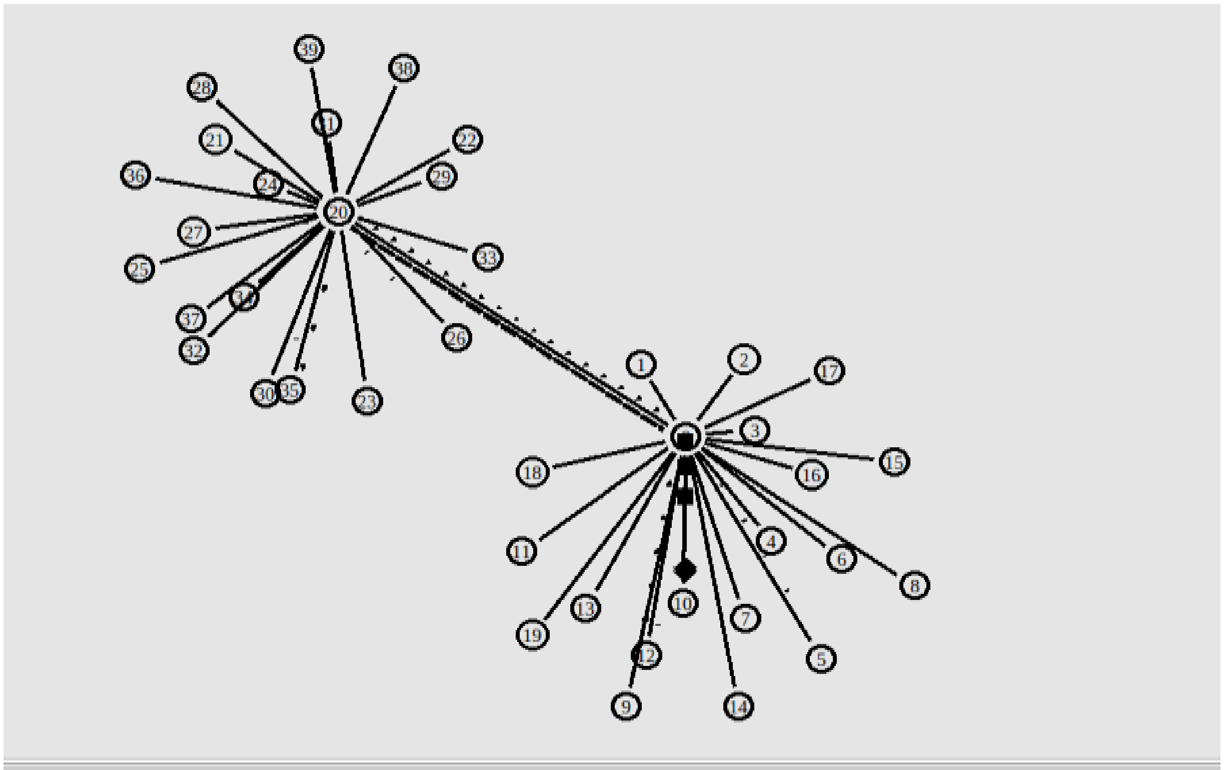
Average_packet_length[i] = α * average_packet_length[i-1]+ (1- α) * sample_packet_length

Average_interval[i] = α * average_interval[i-1] + (1- α) * sample_interval

Bwe_est = average_packet_length/average_interval

Network Topology Under Simulation

1) **Wired Topology:** For this, we have used a dumbbell topology which is suggested in the paper with bottleneck link set to 2Mbps.



2) **Wireless Topology:**

The nodes are placed randomly and for each flow, we have selected a random source and a random destination. Other details are:

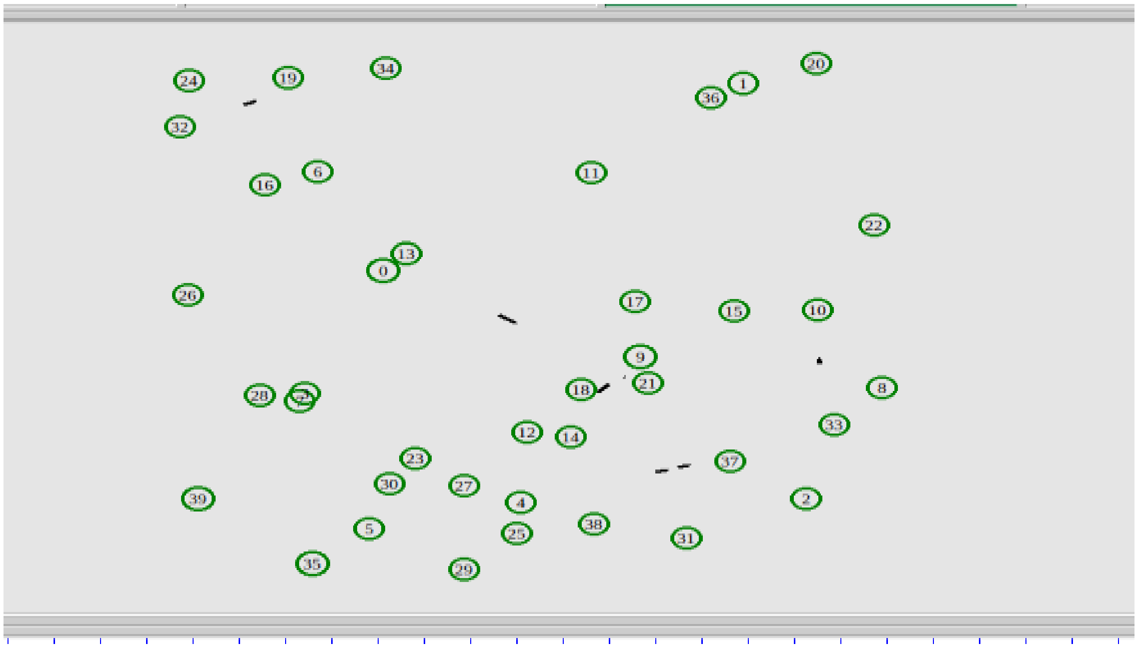
Wireless MAC Protocol: 802.11

Antenna: Omni Antenna

Routing Protocol: DSDV

Agent: TCP (Exiting) and TCP/INVS (Modified)

Application: FTP



Parameters Under Variation:

Parameter	Baseline	Range
Number of Nodes	40	20,40,60,80,100
Number of Flows	20	10,20,30,40,50
Number of Packets/sec	200	100,200,300,400,500
Speed(Mobility)	10	5,10,15,20,25

Modifications Made in the Simulator:

In tcp.h file, following codes are added

```
class INVSTcpAgent : public virtual TcpAgent {
 public:
    INVSTcpAgent();
    virtual void output(int seqno, int reason);
    virtual void recv_newack_helper(Packet *pkt);
    virtual void dupack_action();
    virtual void opencwnd();
    virtual void slowdown(int how);
    void updateMaxMinRTT();
    void updateK();
    void updateSsThresh();  //upon receipt of 3 dup acks
    void updateCwndSp();    //upon packet loss
    void estimateBandwidth();   //needed to update k
    void estimateBuffer();      //needed to update ssthresh

    double bw_ref_;
    double bw_est_;
    double rtt_est_;
    double rtt_ref_;
    double rtt_min_;
    double rtt_max_;
    double last_rtt_sample_;
    double buffer_est_;
    double buffer_max_;
    double cwnd_sp_;
    double prev_sample_length_;
    double prev_sample_interval_;
    double last_rtt_ts_;
    double average_packet_length_;
    double average_interval_;

```

In tcp.cc, following modifications are the major ones:

Bandwidth estimation:

```
void INVSTcpAgent::estimateBandwidth()
{
    double sample_length = size_*8;
    if(rtt_ts_ == 0) {
        rtt_ts_ = 10;
    }
    double interval = rtt_ts_ - last_rtt_ts_;
    last_rtt_ts_ = rtt_ts_;
    average_packet_length_ = alpha * average_packet_length_ + (1-alpha)*sample_length;
    average_interval_ = alpha * average_interval_ + (1-alpha)*interval;
    alpha = 0.5;
    bw_est_ = average_packet_length_/average_interval_;
    bw_ref_ = max(bw_est_, bw_ref_);
}
```

Buffer Estimation:

```
/*
This is called from updateSsThresh()
*/
void INVSTcpAgent::estimateBuffer()
{
    buffer_max_ = rtt_max_*bw_est_;
    buffer_est_ = (last_rtt_sample_-rtt_min_)*bw_est_;
}

/*
```

Cwndsp Update:

```
void INVSTcpAgent::updateCwndSp()
{
    if(cwnd_ < cwnd_sp_)
    {
        cwnd_sp_ = (1+beta)/2 * cwnd_;
    }else
    {
        cwnd_sp_ = cwnd_;
    }

}
```

Adaptive Path Condition variable k:

```
void INVSTcpAgent::updateK()
{
    updateMaxMinRTT();

    if(rtt_min_ == 0 || bw_est_ == 0)
    {
        k = 1;
        return;
    }
    k = c*(log2((bw_est_)*pow(rtt_min_/rtt_ref_,gamma))+1);
    if(k < 1.0) {
        k = 1.0;
    }
}
```

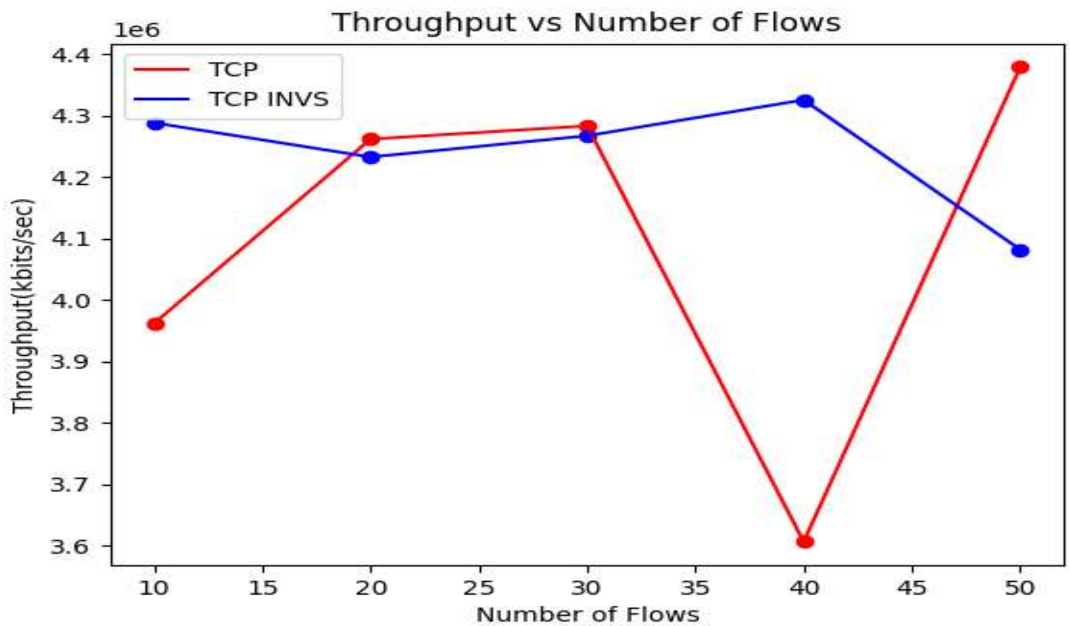
Modified opencwn():

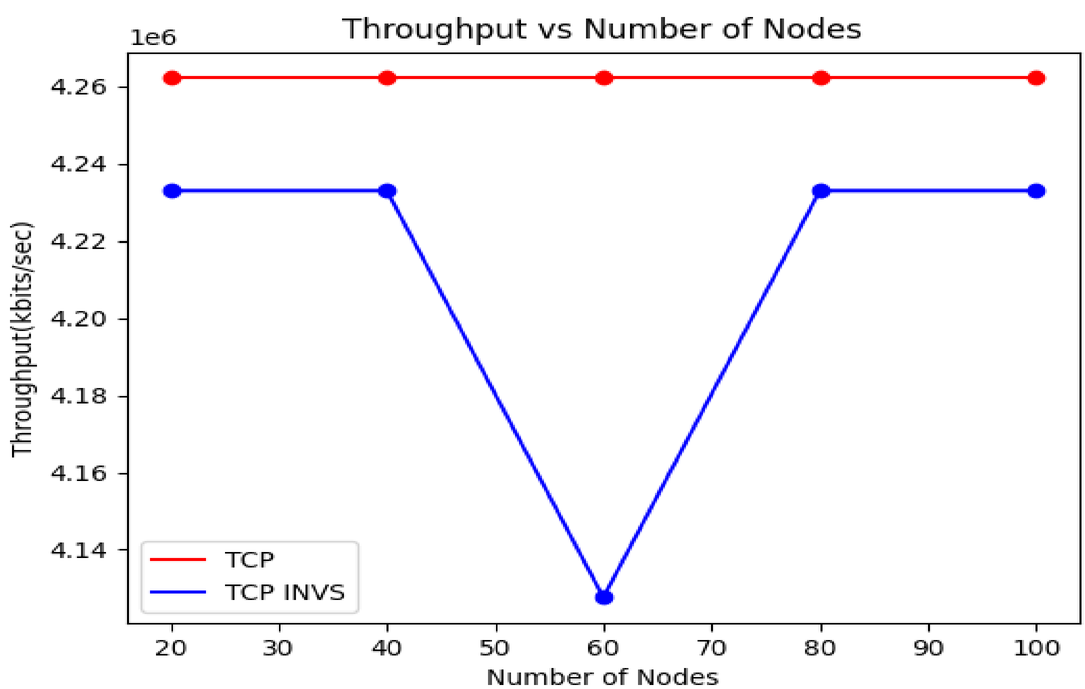
```
void INVSTcpAgent::opencwnd()
{
    //slow start phase remains as it is
    if(cwnd_ < ssthresh_)
    {
        cwnd_ += 1;
    }
    else    //congestion avoidance phase is modified
    {
        updateK();
        if(cwnd_ <= cwnd_sp_-1)
        {
            cwnd_ += (cwnd_sp_ - cwnd_)/(k*cwnd_sp_);
        }else if(cwnd_ >= cwnd_sp_ +1)
        {
            cwnd_ += (cwnd_-cwnd_sp_)/(k*cwnd_);
        }else if(cwnd_sp_-1 < cwnd_ && cwnd_ < cwnd_sp_+1)
        {
            cwnd_ += 1/cwnd_;
        }
    }

    if(maxcwnd_ && (int(cwnd_) > maxcwnd_))
    {
        cwnd_ = maxcwnd_;
    }
}
```

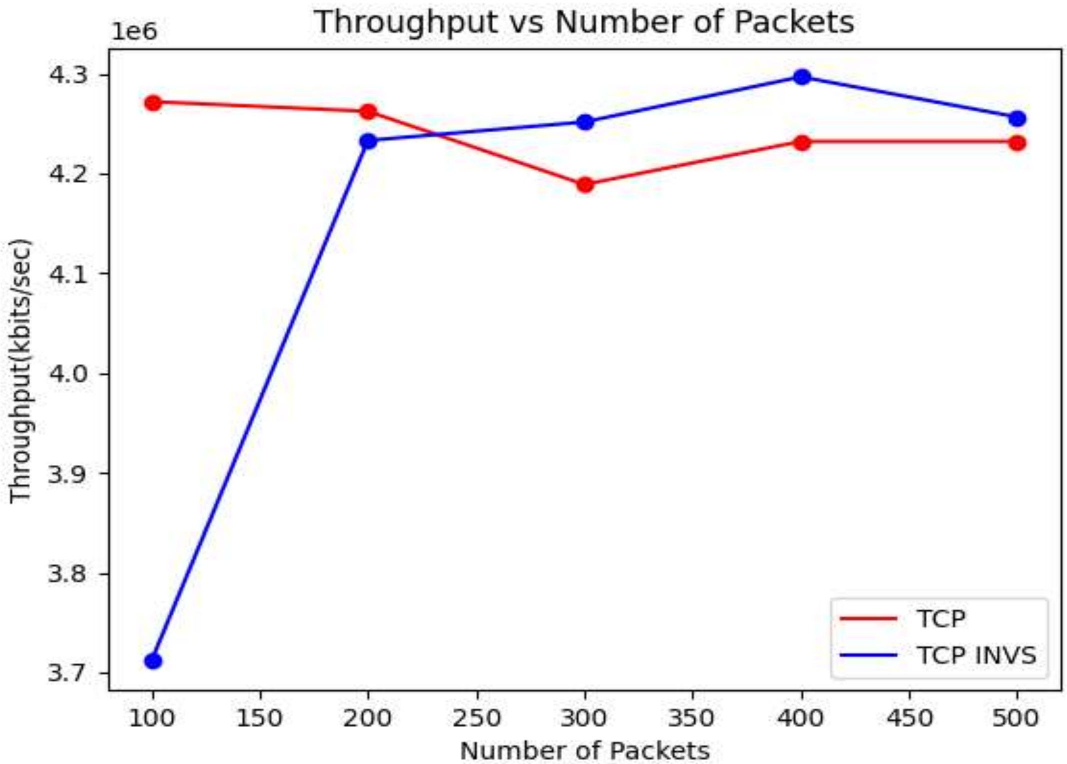
Graphical Analysis: Existing vs Modification (i.e, TCP vs INVS)

Throughput:

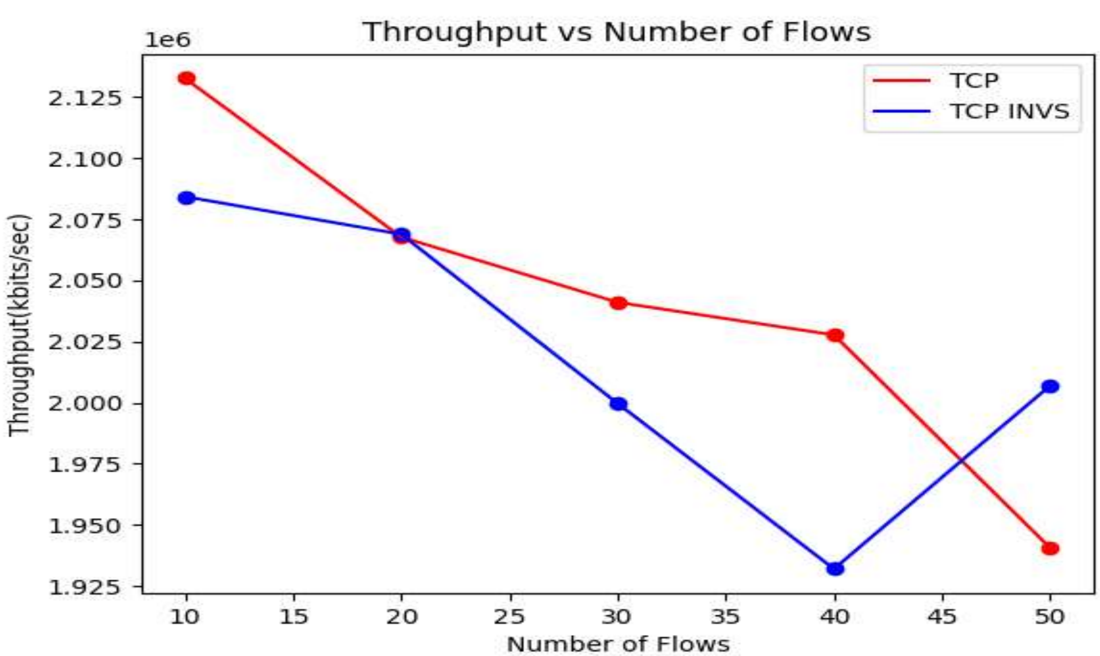
Wired Topology:

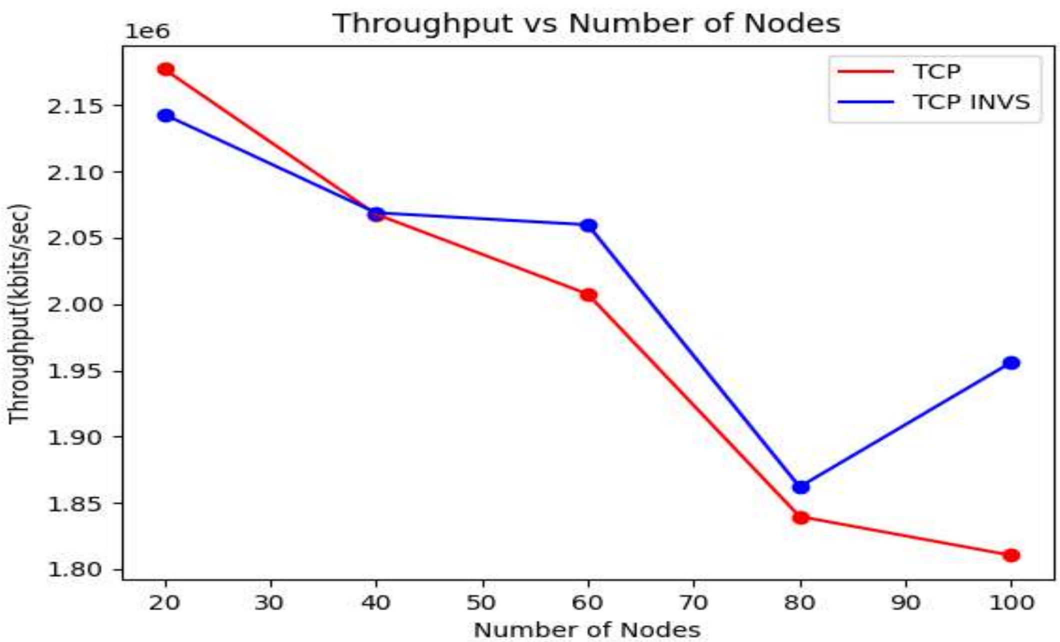




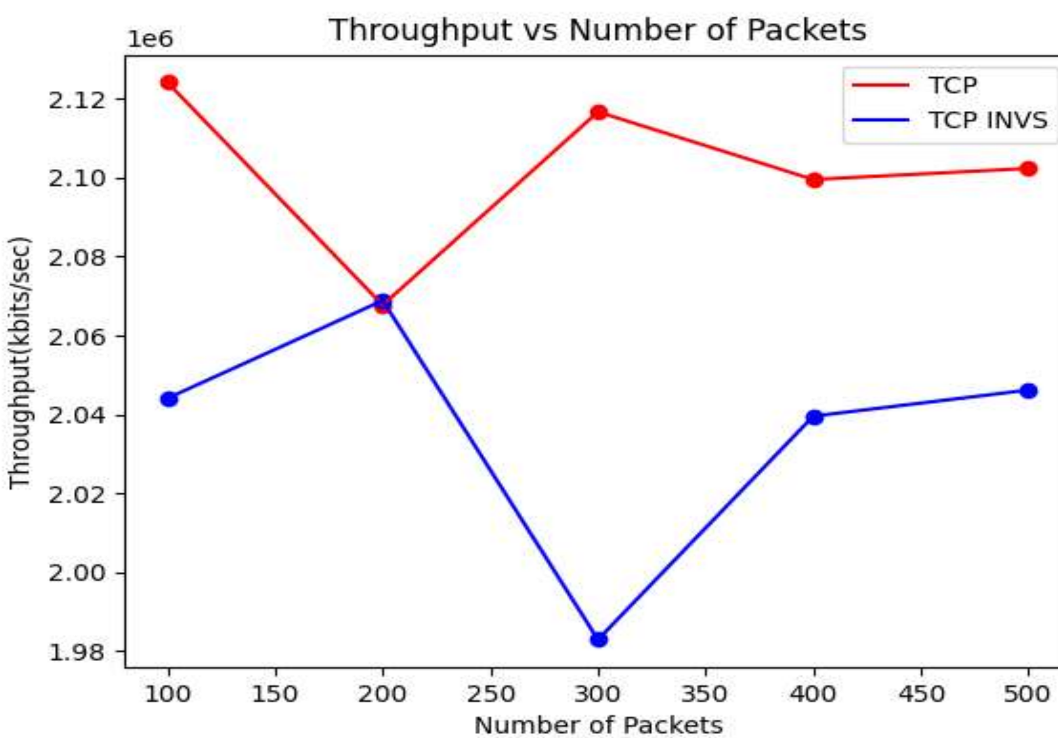


Wireless Topology:

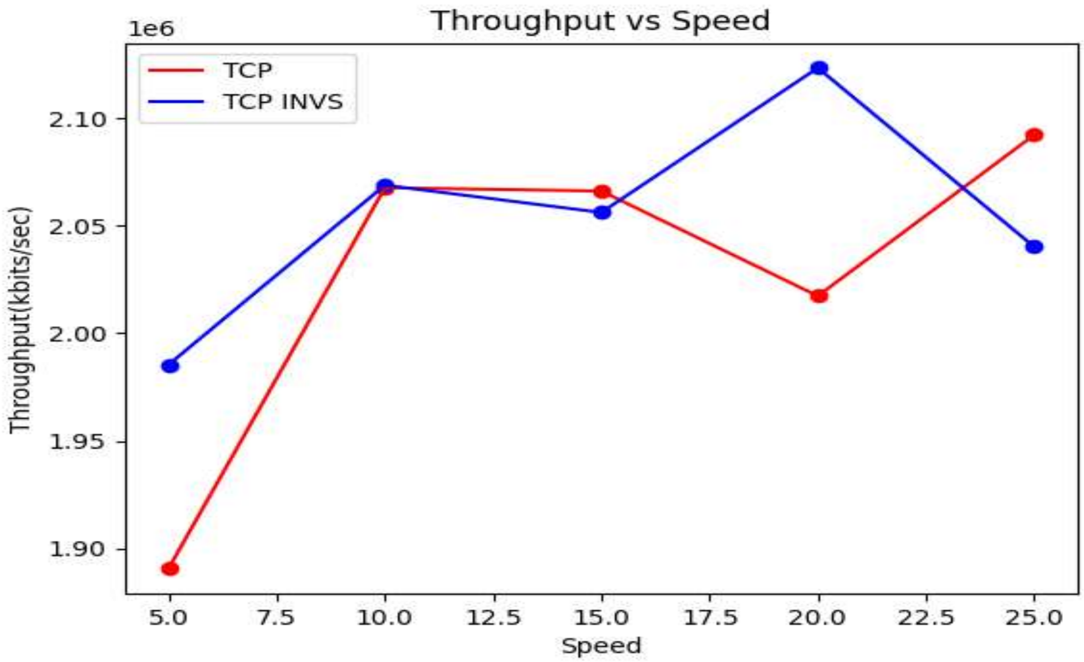




Throughput vs Number of Nodes
1e6
TCP
TCP INVS
Throughput(kbits/sec)
2.15
2.10
2.05
2.00
1.95
1.90
1.85
1.80
20
30
40
50
60
70
80
90
100
Number of Nodes

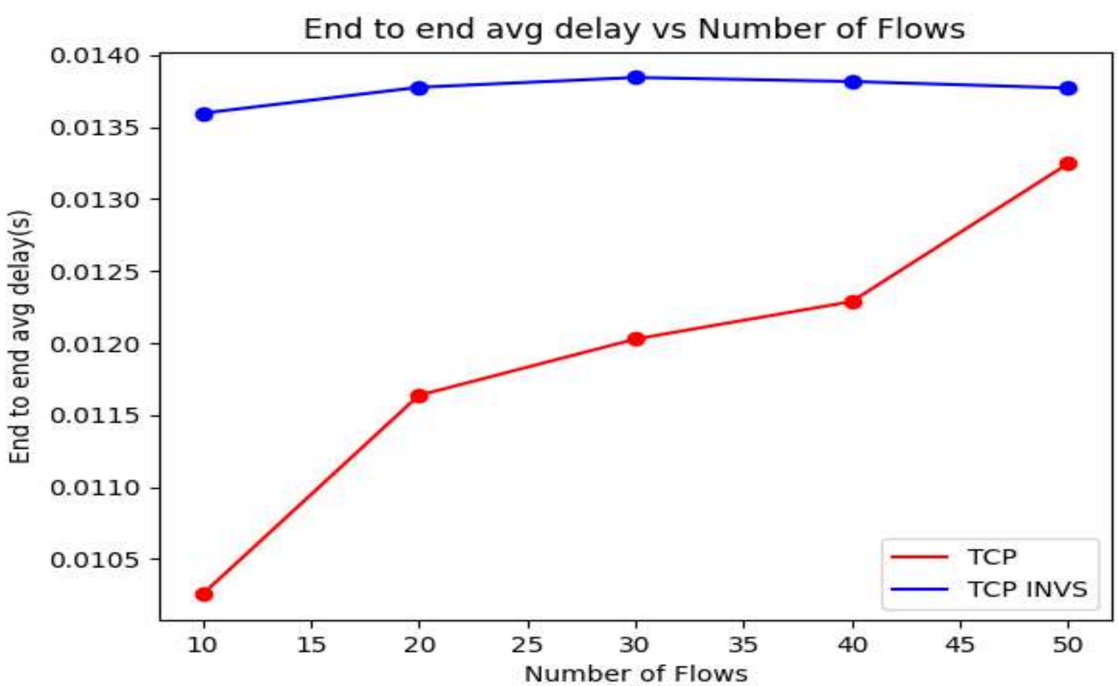


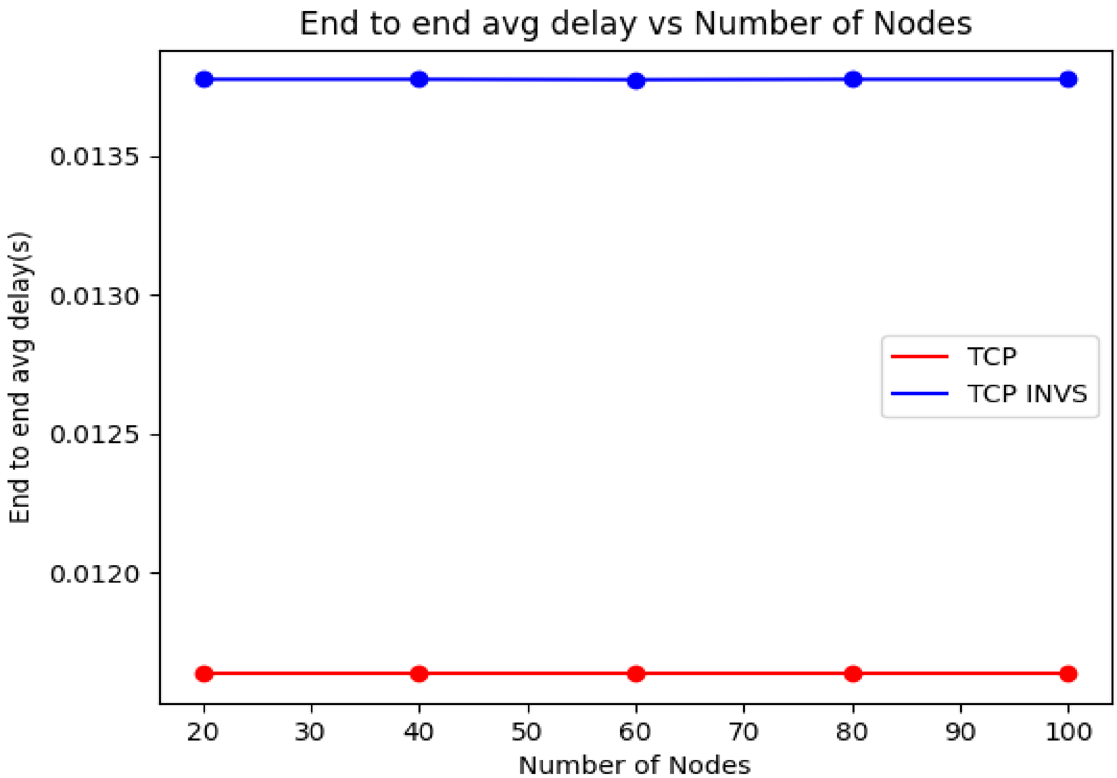
Throughput vs Number of Packets
1e6
TCP
TCP INVS
Throughput(kbits/sec)
2.12
2.10
2.08
2.06
2.04
2.02
2.00
1.98
100
150
200
250
300
350
400
450
500
Number of Packets

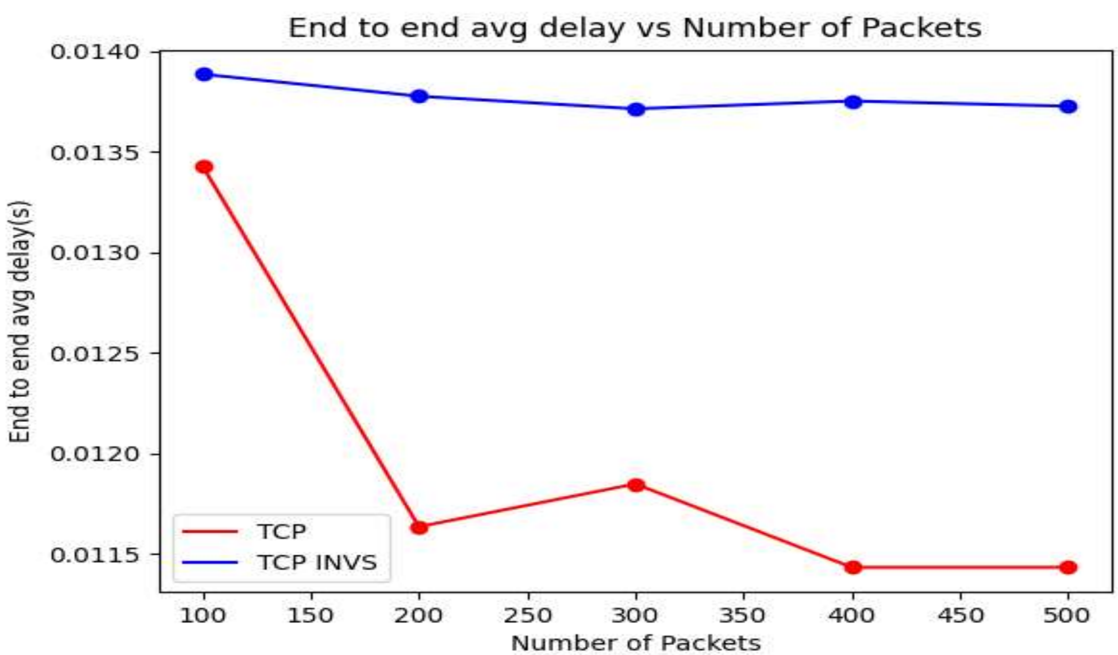


End-to-end delay

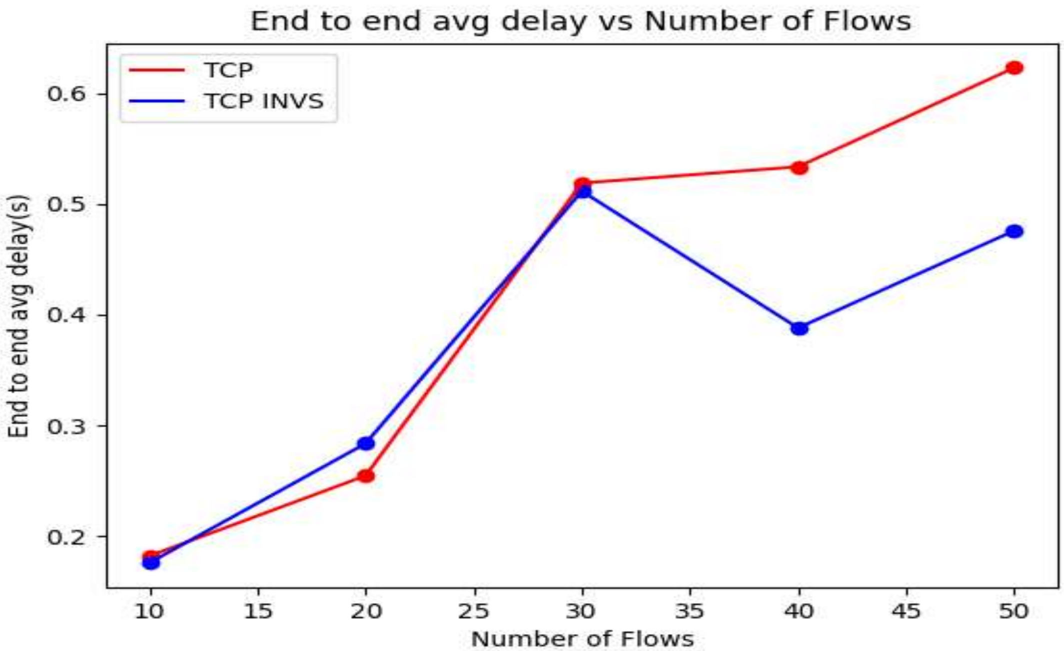
Wired Topology

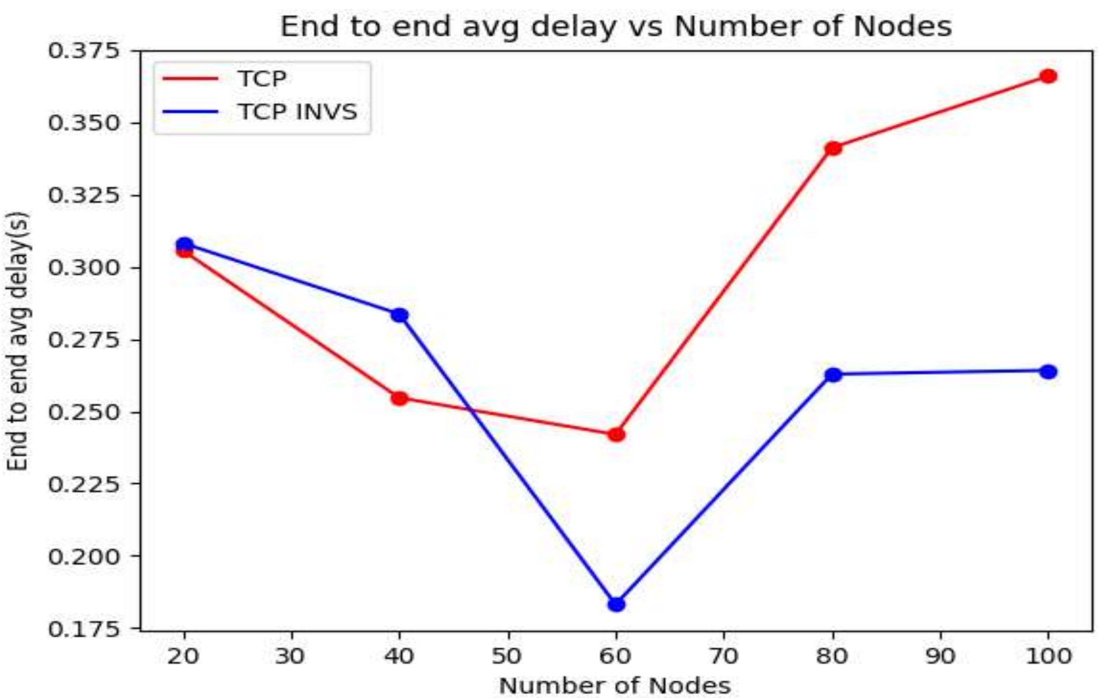


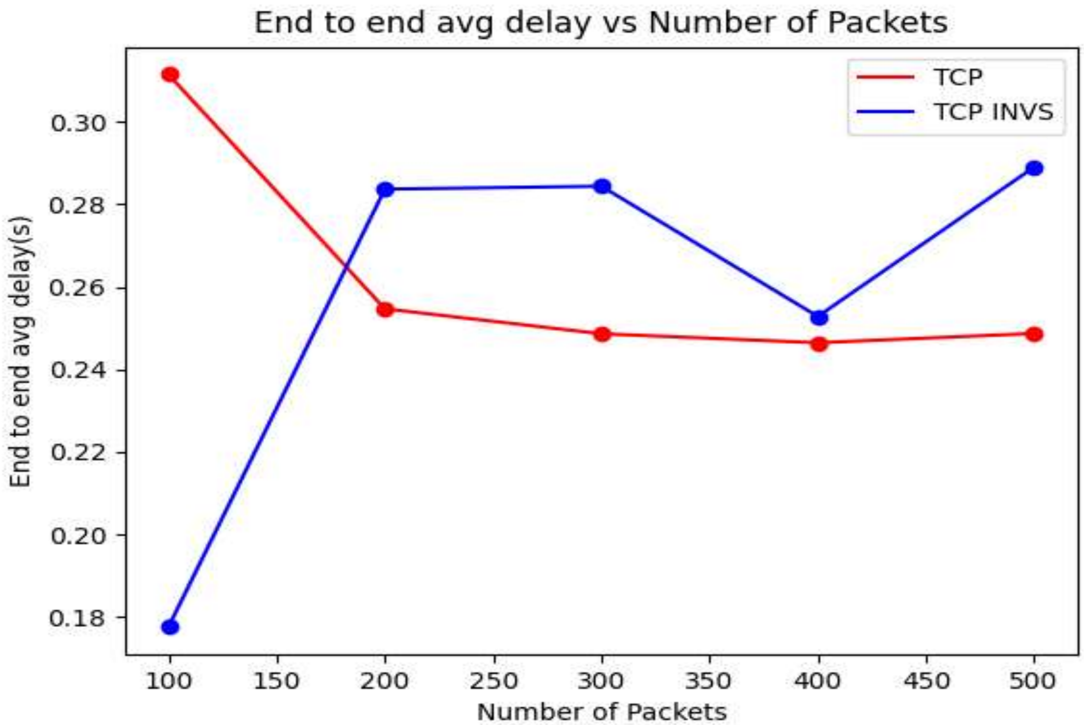

End to end avg delay vs Number of Nodes
0.0135
0.0130
0.0125
0.0120
End to end avg delay(s)
TCP
TCP INVS
20
30
40
50
60
70
80
90
100
Number of Nodes

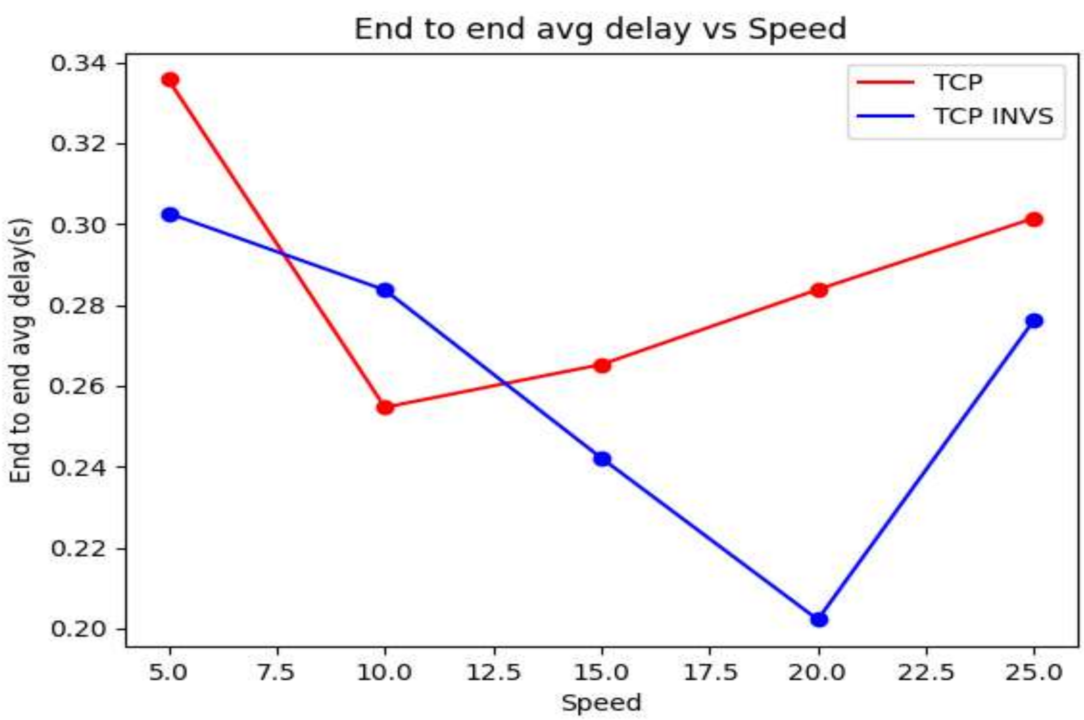

End to end avg delay vs Number of Packets
0.0140
0.0135
0.0130
0.0125
0.0120
0.0115
End to end avg delay(s)
TCP
TCP INVS
100
150
200
250
300
350
400
450
500
Number of Packets

Wireless Topology:



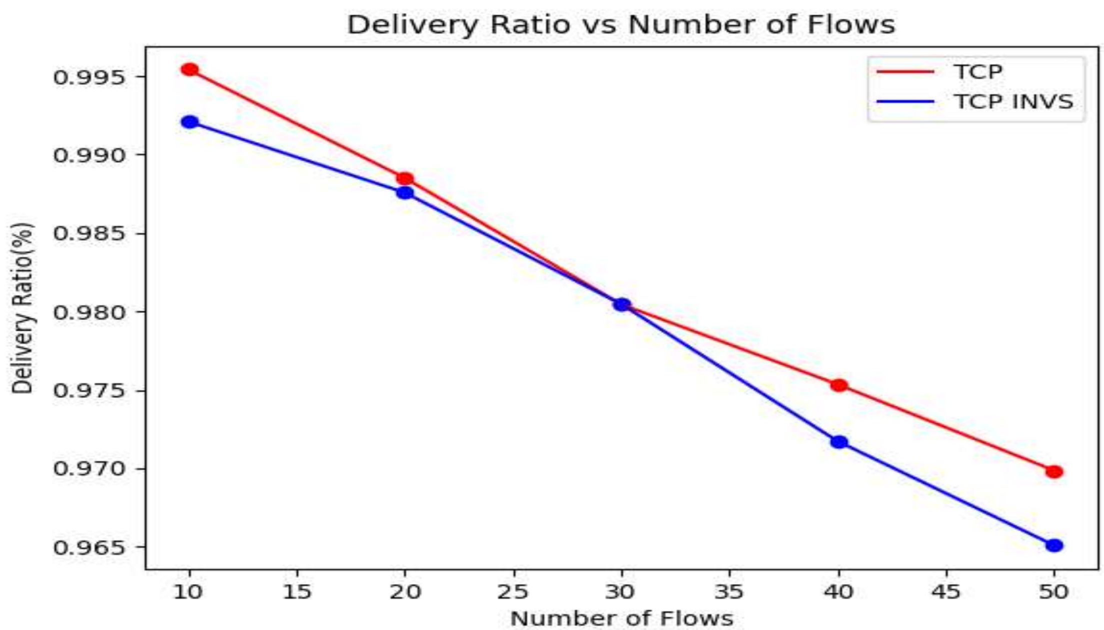


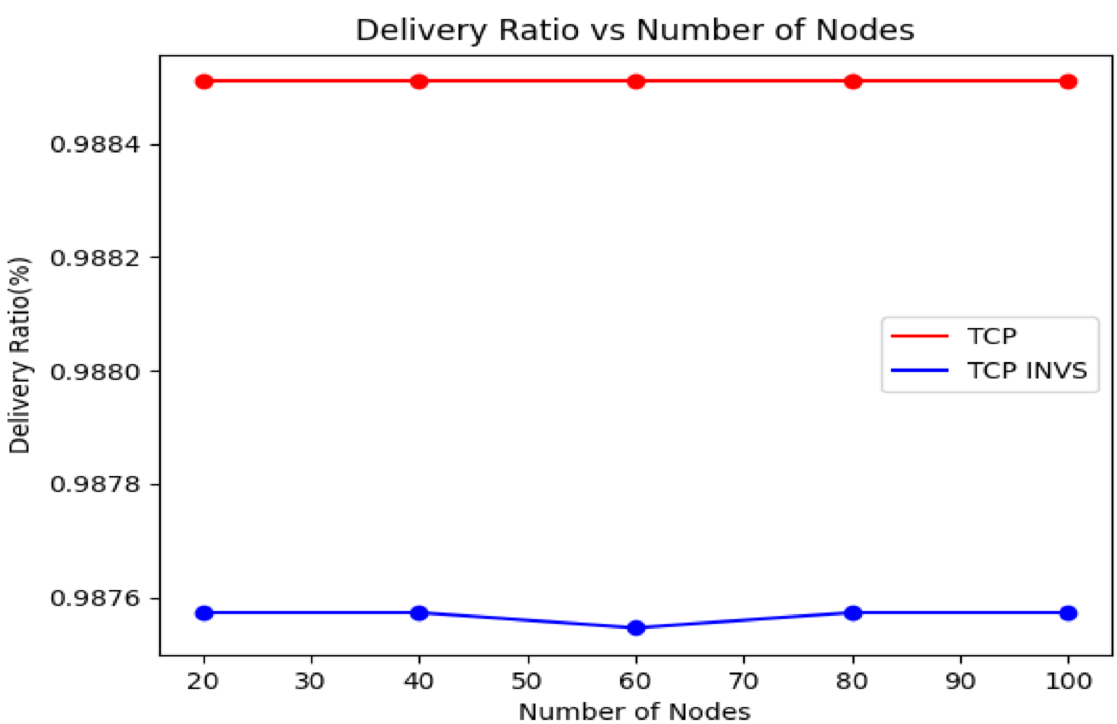

End to end avg delay vs Number of Packets
TCP
TCP INVS
End to end avg delay(s)
0.30
0.28
0.26
0.24
0.22
0.20
0.18
100
150
200
250
300
350
400
450
500
Number of Packets

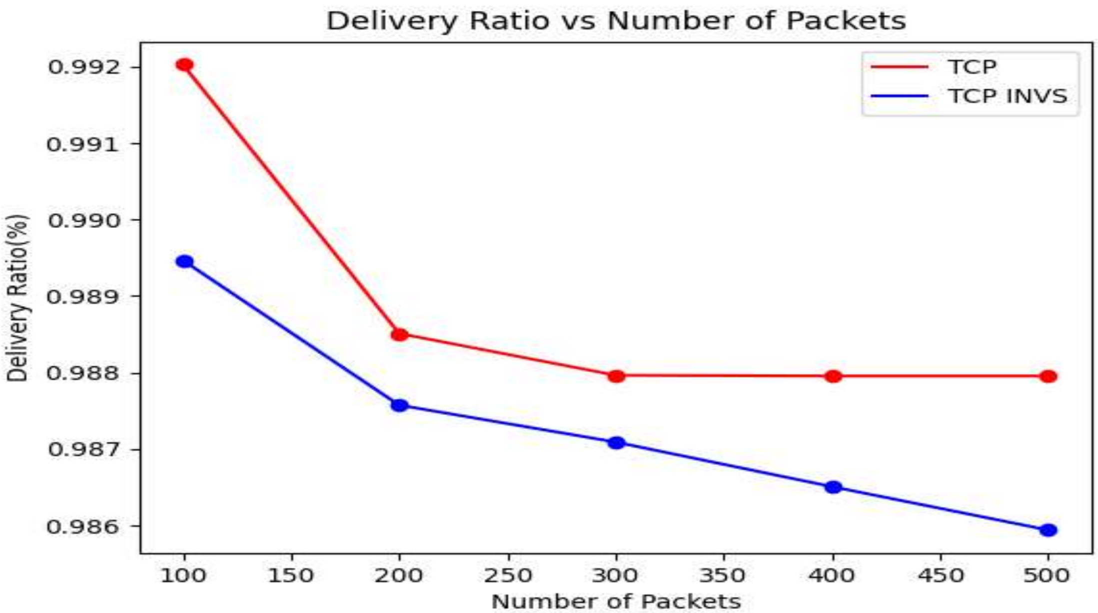

End to end avg delay vs Speed
TCP
TCP INVS
End to end avg delay(s)
0.34
0.32
0.30
0.28
0.26
0.24
0.22
0.20
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
Speed

Packet Delivery Ratio

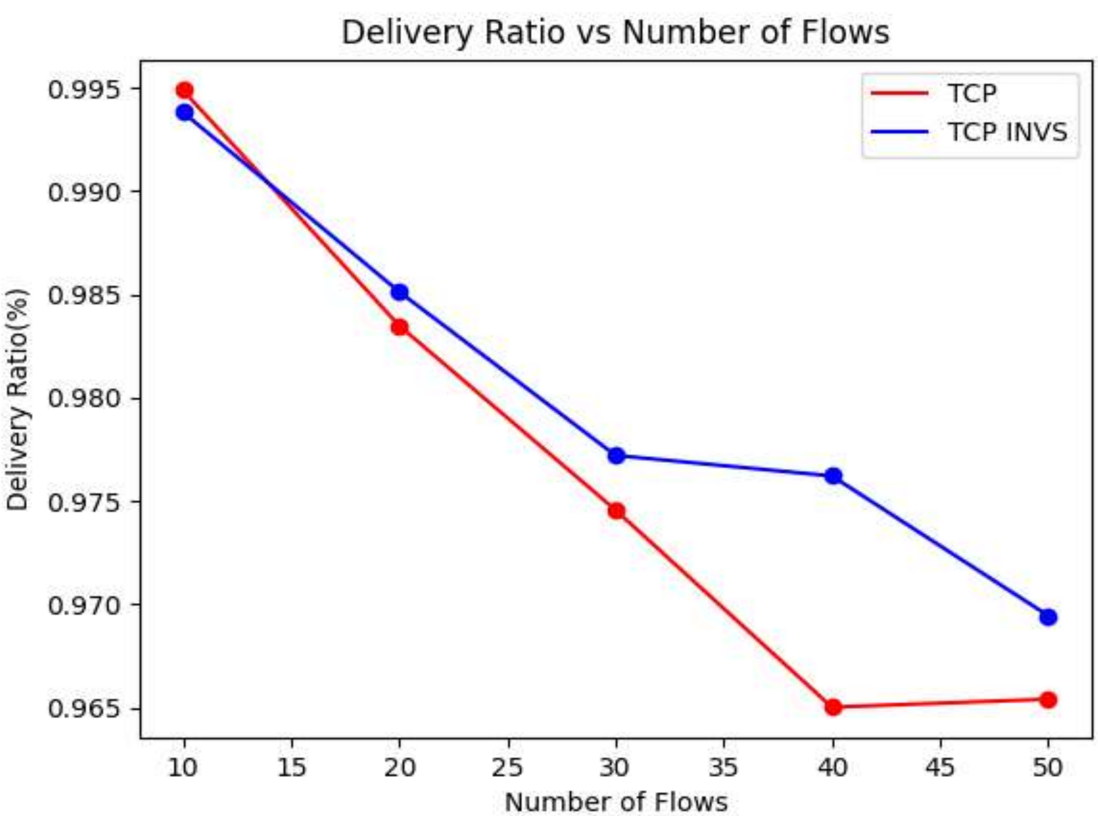
Wired Topology

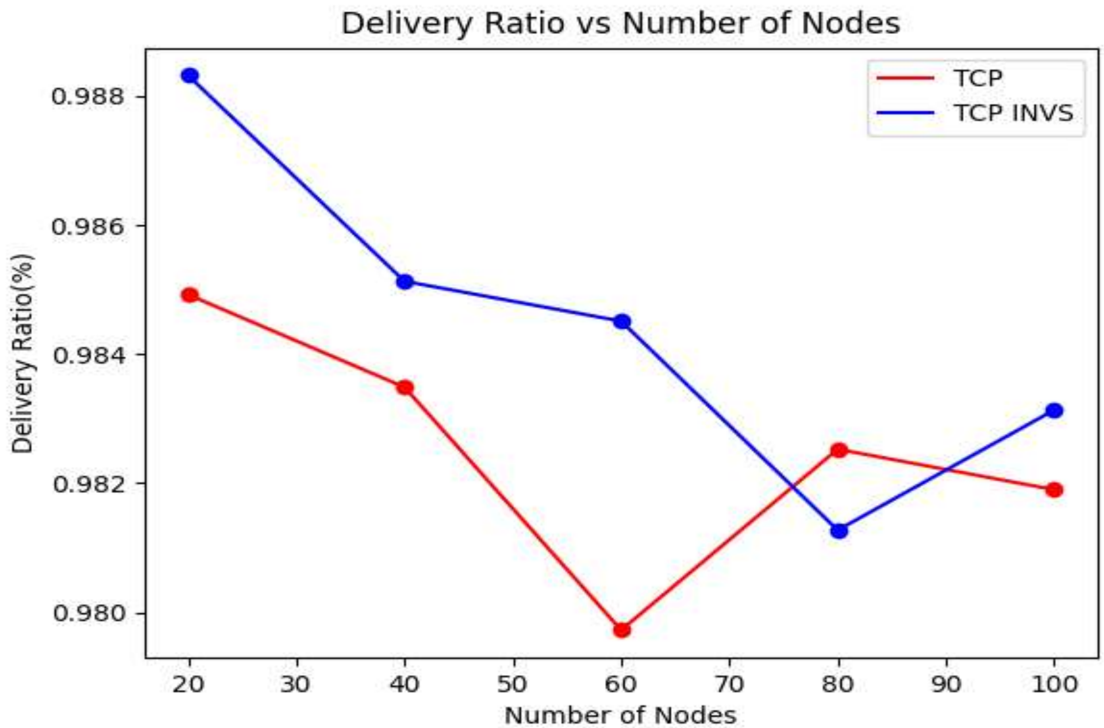


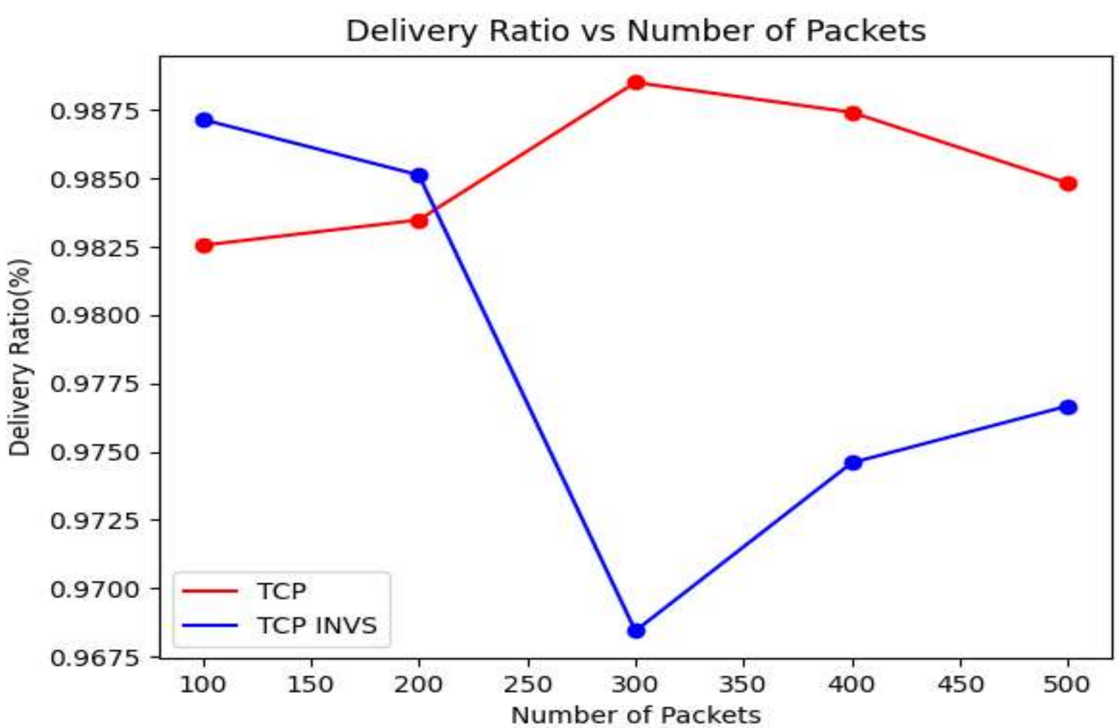


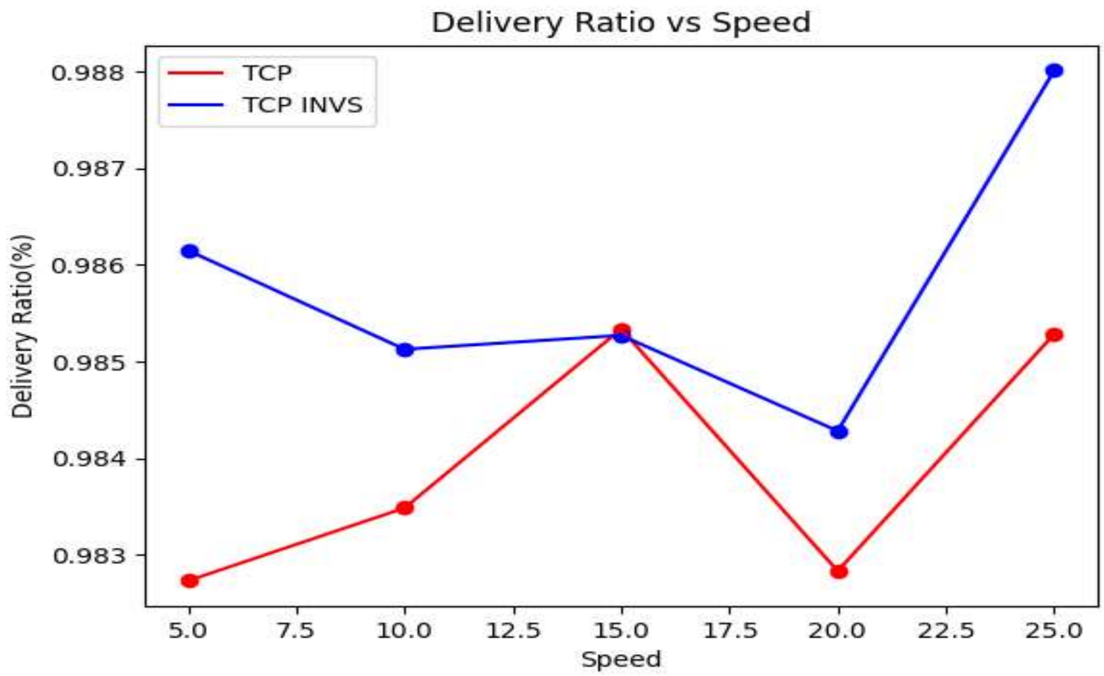


Wireless Topology



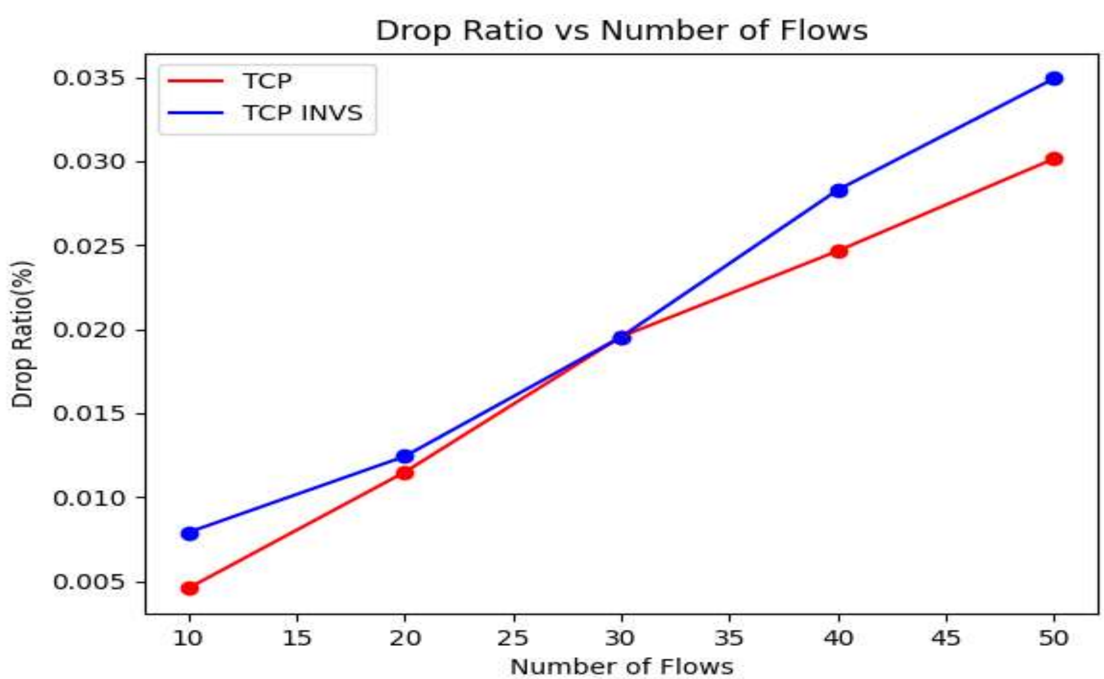

Delivery Ratio vs Number of Nodes
TCP
TCP INVS
Delivery Ratio(%)
0.988
0.986
0.984
0.982
0.980
20
30
40
50
60
70
80
90
100
Number of Nodes

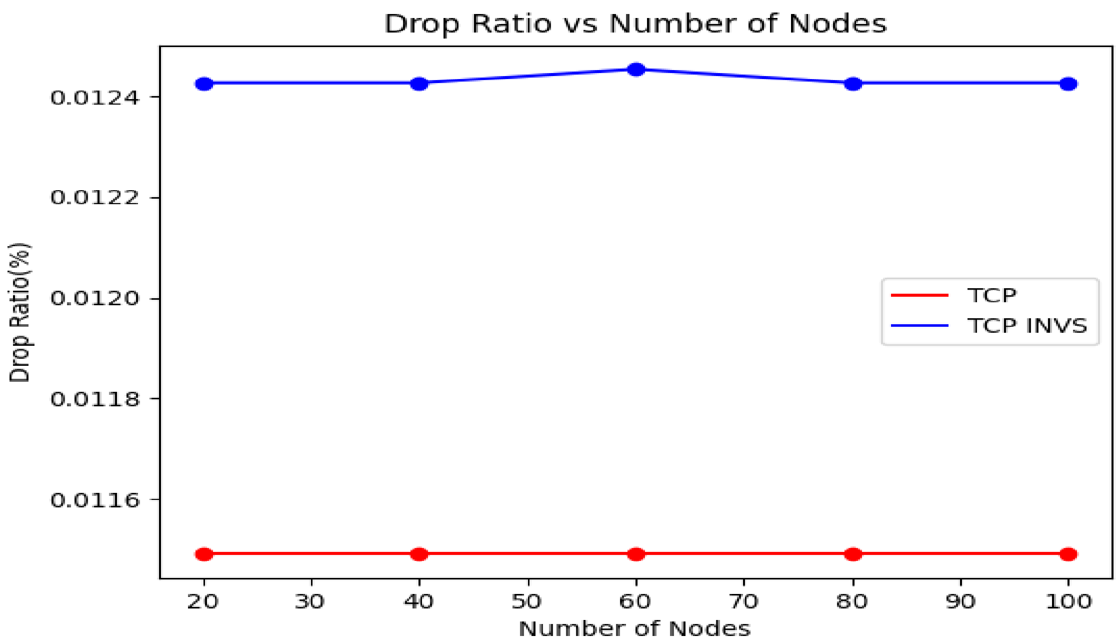

Delivery Ratio vs Number of Packets
TCP
TCP INVS
Delivery Ratio(%)
0.9875
0.9850
0.9825
0.9800
0.9775
0.9750
0.9725
0.9700
0.9675
100
150
200
250
300
350
400
450
500
Number of Packets



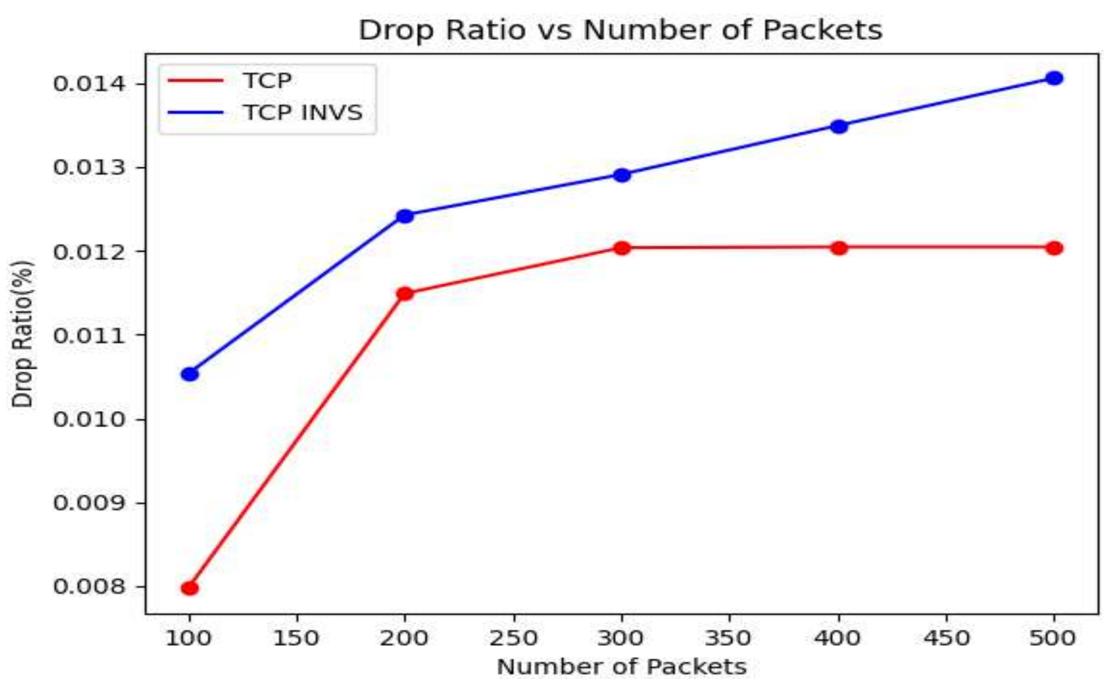
Packet Drop Ratio

Wired Topology



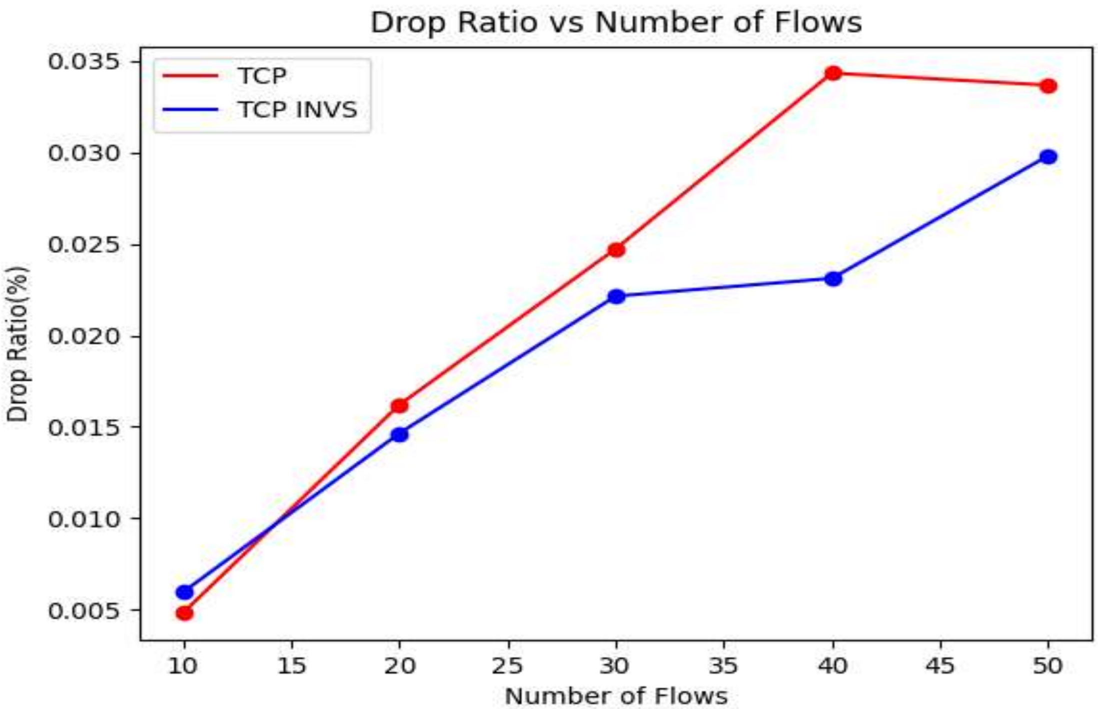


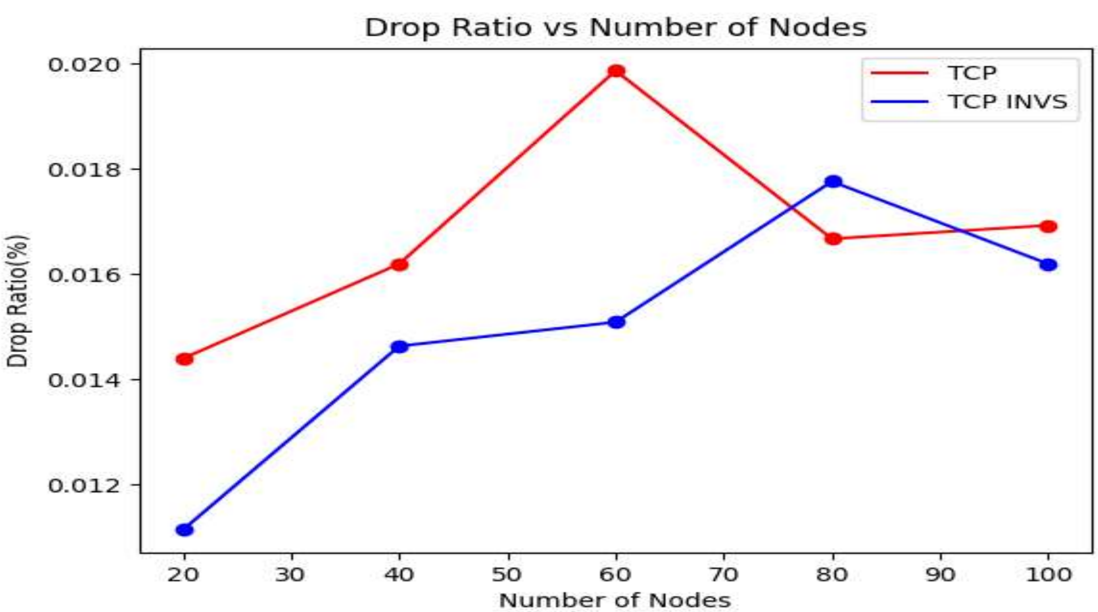
Drop Ratio vs Number of Nodes
Drop Ratio(%)
0.0124
0.0122
0.0120
0.0118
0.0116
20
30
40
50
60
70
80
90
100
Number of Nodes
TCP
TCP INVS

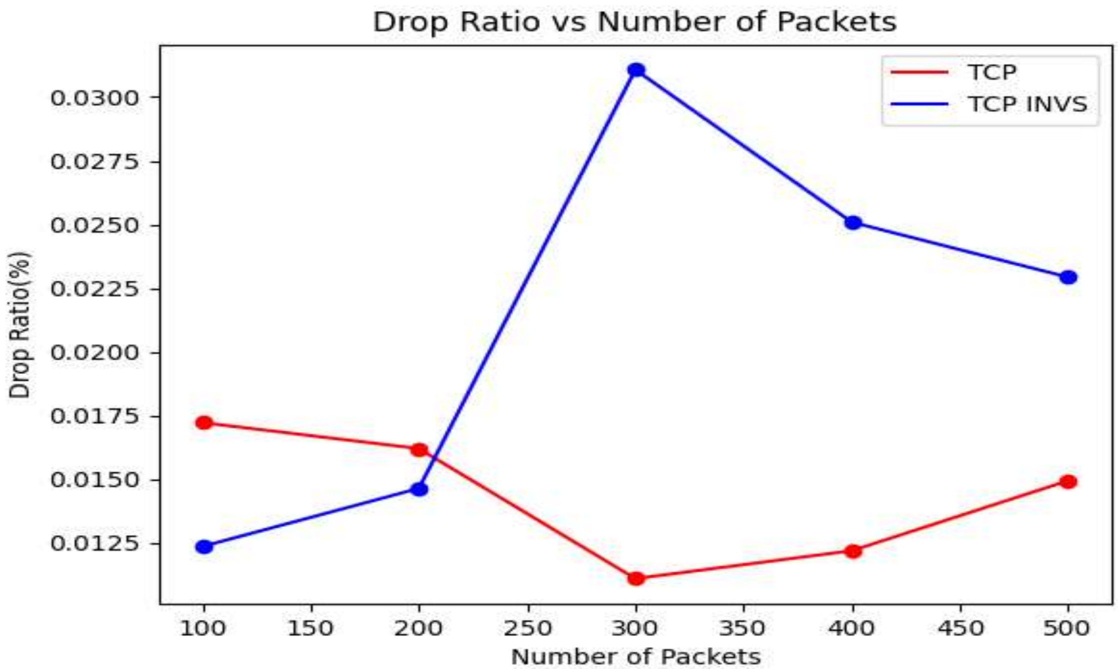


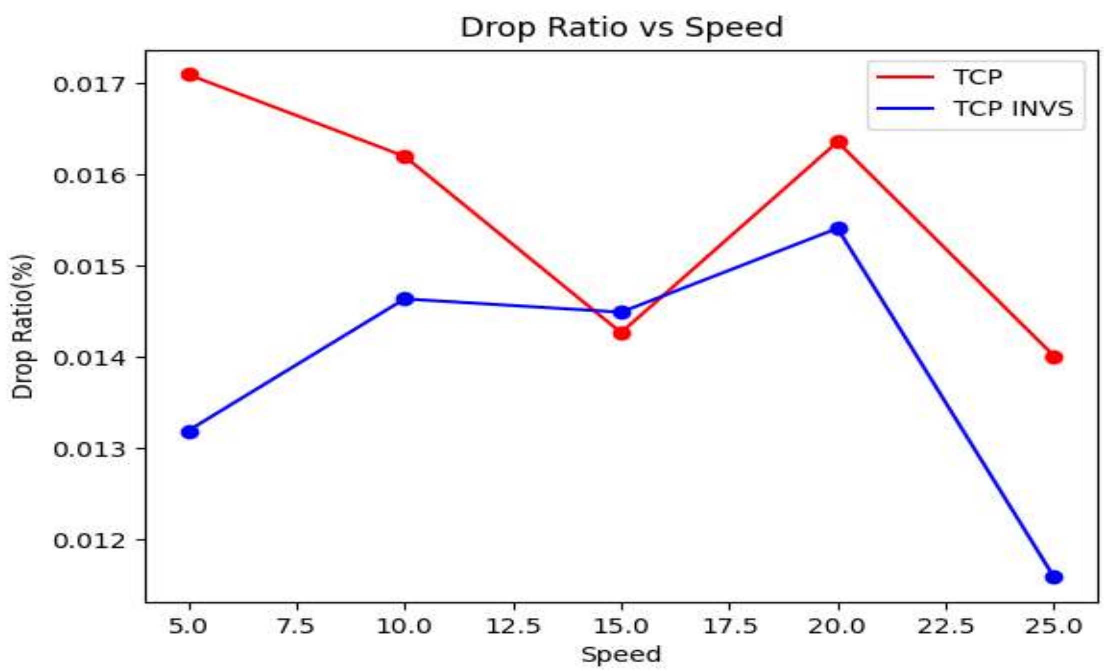
Drop Ratio vs Number of Packets
Drop Ratio(%)
0.014
0.013
0.012
0.011
0.010
0.009
0.008
100
150
200
250
300
350
400
450
500
Number of Packets
TCP
TCP INVS

Wireless Topology:



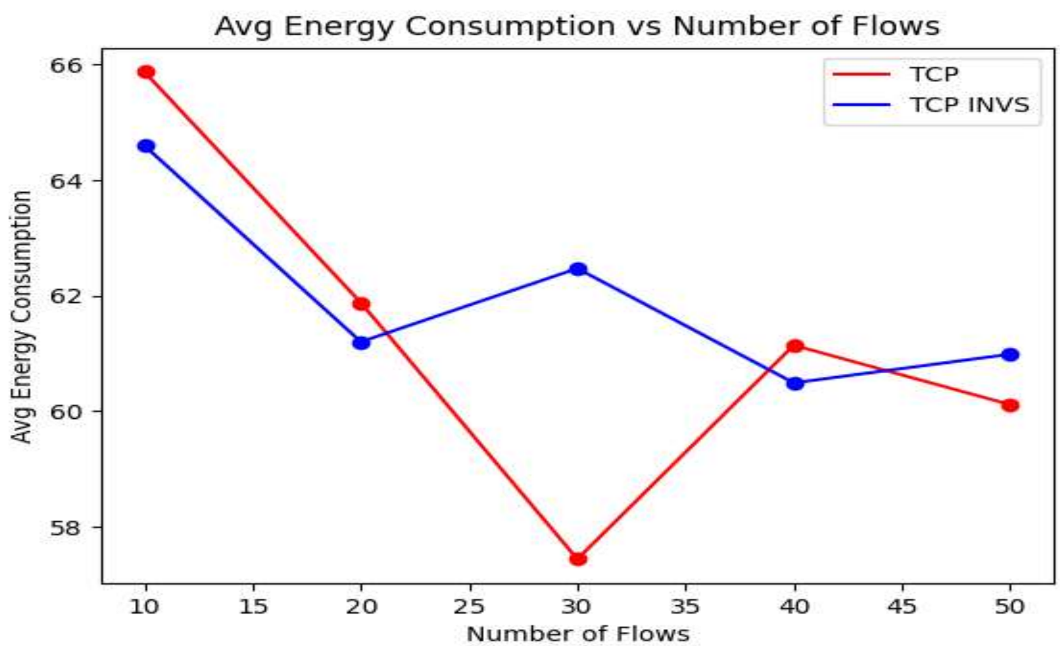


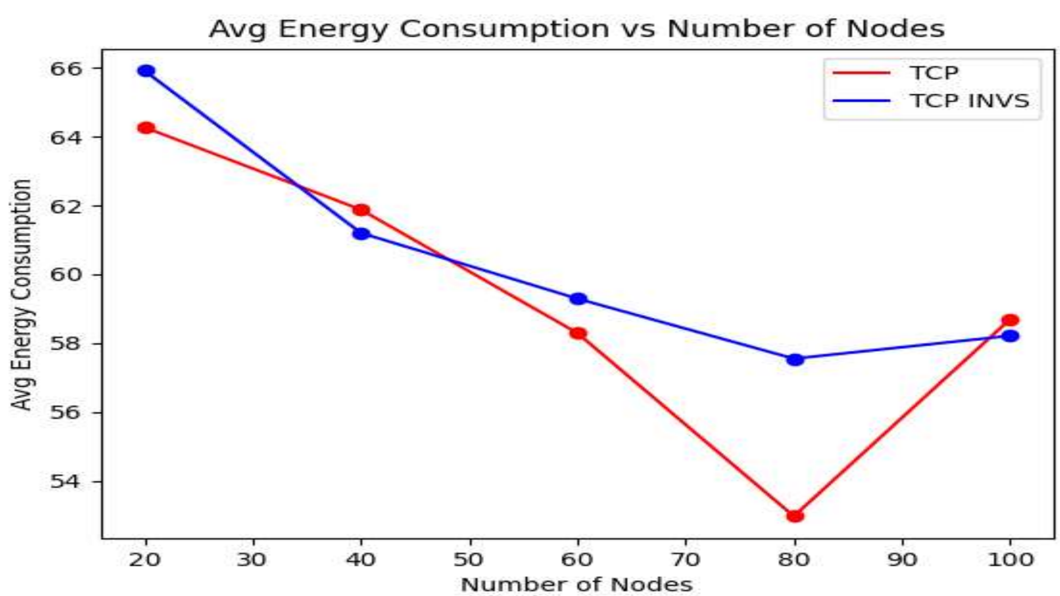

Drop Ratio vs Number of Packets
TCP
TCP INVS
Drop Ratio(%)
0.0300
0.0275
0.0250
0.0225
0.0200
0.0175
0.0150
0.0125
100
150
200
250
300
350
400
450
500
Number of Packets

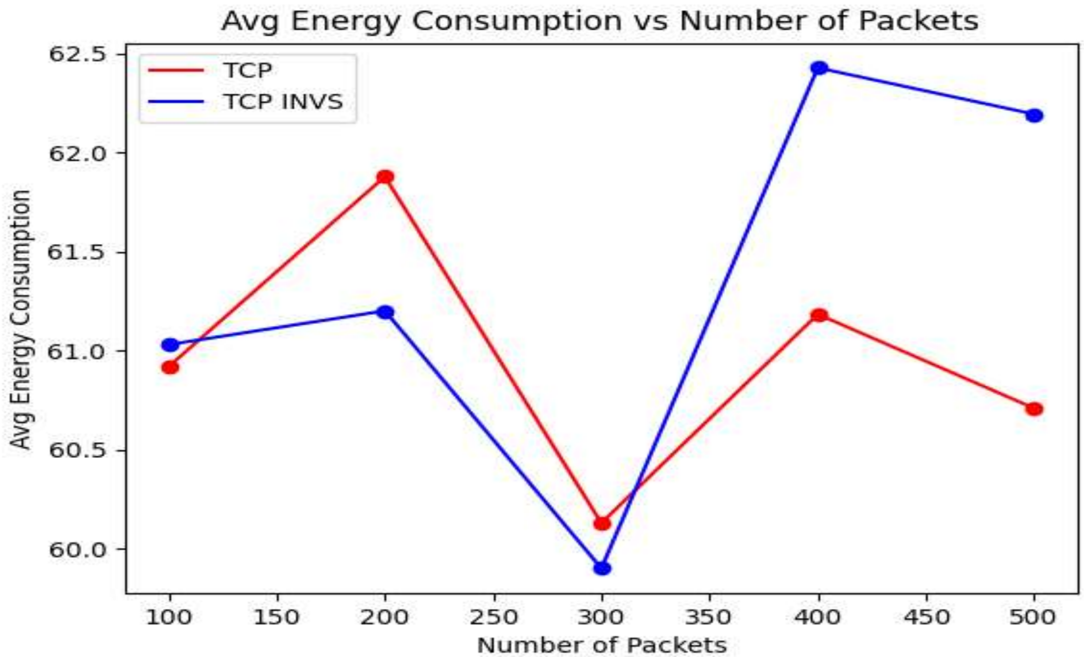

Drop Ratio vs Speed
TCP
TCP INVS
Drop Ratio(%)
0.017
0.016
0.015
0.014
0.013
0.012
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
Speed

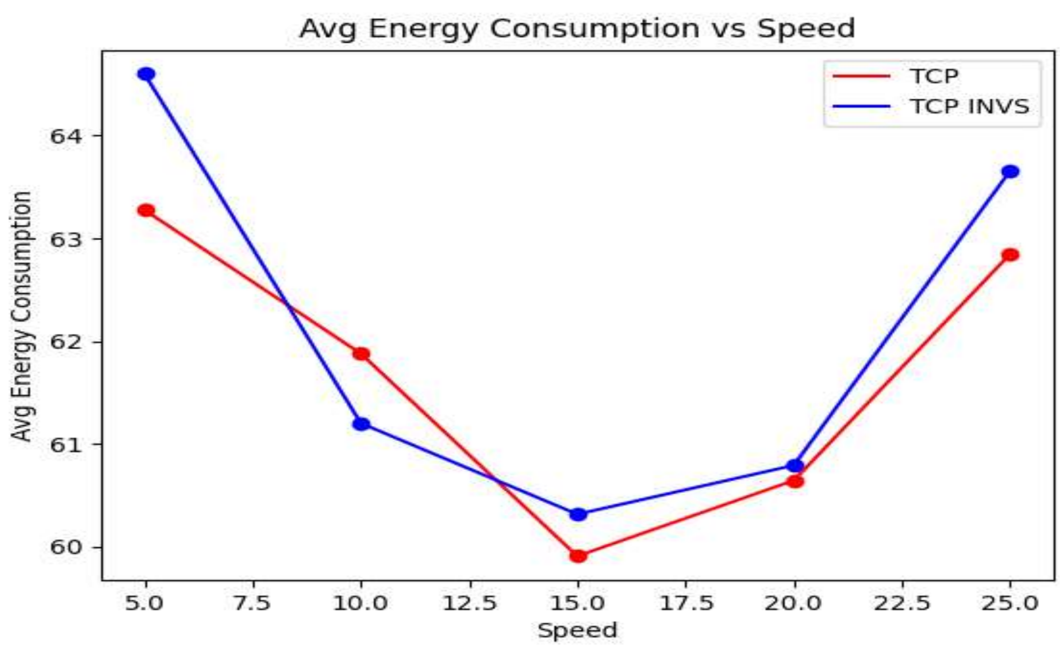
Energy Consumption

Wireless Topology






Avg Energy Consumption vs Number of Packets
TCP
TCP INVS
Avg Energy Consumption
62.5
62.0
61.5
61.0
60.5
60.0
100
150
200
250
300
350
400
450
500
Number of Packets


Avg Energy Consumption vs Speed
TCP
TCP INVS
Avg Energy Consumption
64
63
62
61
60
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
Speed

Summary and Findings

Analyzing the graphs, it is clearly seen that INVS (slightly modified from the original one) has given a little bit poor performance than the existing TCP for wired topology but it has slightly given a little bit better performance in wireless topology.

As INVS has an adaptive path condition based congestion window control in congestion avoidance phase, its **throughput** increases with increasing flows and number of packets per second. However, with increasing number of nodes, it doesn't improve as there is a bottleneck link which has limited bandwidth. It also depends on channel delay, queue etc.

End to end delay is increased a little bit. The reason behind it can be the fact that we are estimating the buffer and bandwidth and according to that sending packets to Network. Therefore the waiting time in queue increases a little bit.

Packet Delivery Ratio gives more or less similar performance to existing algorithm. According to INVS, it is adaptive to packet loss ratio but in this case, we cannot see any improvement as claimed in the paper. This can be due to applying a simple bandwidth estimation algorithm which simply works on packet size and time interval of sending packets. Another reason can be that, it tries to utilize the available bandwidth of the network in CA phase but if the Bandwidth itself is not good enough, then it cannot provide the expected outcome of INVS. Also in the topology, **Queue type is used DropTail** which simply drops the incoming packets when its buffer is full.

For **Wireless**, throughput increases with number of nodes and speed of nodes. Packet delivery ratio gives better performance for almost every variation. End to end delay is less comparative to wired topology as all the packets don't wait in the same queue. Energy consumption is similar to existing TCP.

In wireless communication, the performance of the metrics depends on various factors such as **transmission range**, **interference** from other wireless networks, **node mobility**, **routing protocols**, **traffic load** and **power consumption**. Here, packet loss can be due to direction of antenna (we have used **omni antenna**) as if the antenna does not send the packet in right direction, the signal strength can be weaker. Moreover, node mobility can cause changes in signal strength or interference. Therefore, packet loss cannot cause due to only congestion, it depends on lots of other factors too in wireless communication. INVS tries to detect the loss classification and based on that manipulates the congestion window. Therefore, the performance improvement in wireless topology for INVS is expected as we see from the graphs.

However, **in some graph, we see the performance fluctuates** for both existing TCP and modified INVS. The reasons can depend on various factors described above.

An observable point is that, **the end to end delay has increased a lot in wireless communication than wired communication**, this extra delay is introduced due to processing of packets in each node; as the hop count increases to reach the destination, this delay increases too.

Another observable fact is that **the throughput in wired medium is 2x times the throughput in wireless medium**. As wireless mediums are usually lossy links (due to mobility, interference, antenna direction etc), packets are dropped more often which causes the throughput to decrease than that of wired medium.

INVS tries to utilize the available resources in CA phase. In congestion avoidance (CA) phase, existing tcp increases the window size linearly to avoid congestion. But during this time, available network resources are unutilized. Here INVS provides a growth function that helps to utilize the available resources and send more packets during that period. This fact can be observed from our simulated result. Sent and Received Packets are more in INVS than in existing TCP which is shown below.

	A	B	C	D	E	F
1	Nodes	Flows	Packets P	Sent Packets	Received Packets	Dropped Packets
2	40	10	200	199430	198517	913
3	40	20	200	215928	213447	2481
4	40	30	200	220756	216440	4316
5	40	40	200	224613	219075	5538
6	40	50	200	227545	220685	6860
7						

Figure: Sent Packets in existing TCP of NS2

Nodes	Flows	Packets P	Sent Packets	Received Packets	Dropped Packet
40	10	200	217325	215608	1717
40	20	200	218717	215999	2718
40	30	200	222193	217852	4341
40	40	200	226716	220304	6412
40	50	200	230240	222200	8040

Figure: Sent Packets in modified INVS

Conclusion

INVS is mainly introduced for heterogeneity of real Networks and it tries to adapt with different path conditions and lossy links. As we could not introduce much heterogeneity in topologies while simulating INVS, all of its performance could not be observed properly. However, based on the simulated data, we can conclude that, in normal scenarios, INVS does not show greater improvement over existing TCP. But in wireless networks, it starts to improve over the existing mechanism. And the fact that it utilizes network resources in CA phase is acceptable based on our simulation.