Course NO. : CSE 322

Report On NS2 Project

Project Paper:

Elastic-TCP : Flexible Congestion Control Algorithm to Adapt for High-BDP Networks.

Submitted By :

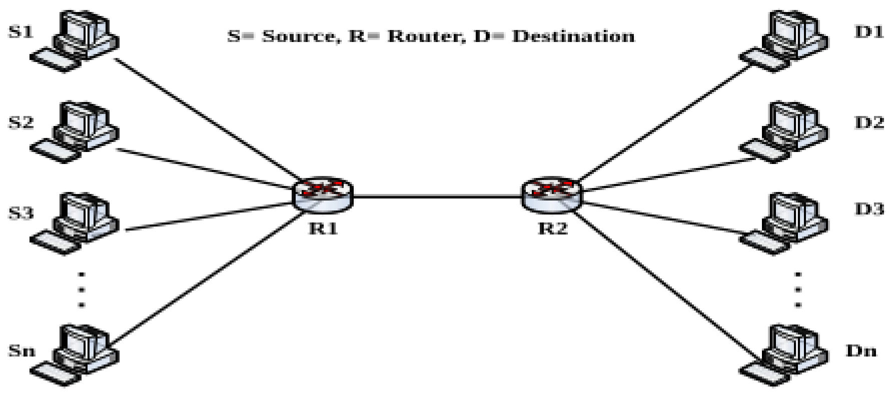
Name : Partho Kunda
Roll : 1805107
Section : B2

Network Topologies:

1. **Wireless 802.11 (mobile)**

2. **Wireless 802.15.4 (static)**

3. **Dumbbell network** topology with congested bottleneck (given in paper)



Parameters:

Packet Error Rate 0.05
Bottleneck 2-way delay between R1 and R2 100ms
Link Speed 1000Mbps

Parameters Under Variation:

For the **wireless** networks, the parameters under variation are as mentioned in the report.

1. Comparison
2. Number of **nodes (20, 40, 60, 80, and 100)**
3. Number of **flows** (**10, 20, 30, 40, and 50**)
4. Number of packets was mentioned to vary in the report. But as the project is implementing TCP congestion, and we cannot vary the number of packtes sent forcefully. So, it was skipped in my report.
5. Speed of **nodes** (**5 m/s, 10 m/s, 15 m/s, 20 m/s, and 25 m/s**) [mobile nodes only]
6. **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**) [static nodes only]

For the **Dumpbell network** Topology, as the scenario is a high BDP network with lots of congestion -

1. The number of nodes varied from **100 to 300** for both side of the network (Left and Right side of the figure).
2. Rest of the metrics were ignored because of wired network scenario.

Modifications made in the simulator:

```
class ElasticModTcpAgent : public virtual NewRenoTcpAgent{
public:
   ElasticModTcpAgent();
   virtual void recv(Packet *pkt, Handler *);
protected:
   unsigned int baseRTT_, maxRTT_;

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

   virtual void opencwnd();
   virtual void recv_newack_helper(Packet*);

   virtual double rtt_timeout();
   virtual void rtt_init();

   virtual void reset();
};
```

A new class definition was created in **tcp.h** file.

```
static class ElasticTcpClass : public TclClass {
public:
   ElasticTcpClass() : TclClass("Agent/TCP/Elastic") {}
   TclObject* create(int, const char*const*) {
       return (new ElasticTcpAgent());
   }
} class_elastic;
```

```
ElasticTcpAgent::ElasticTcpAgent() : TcpAgent(){
   baseRTT_ = __INT_MAX__;
   maxRTT_ = 0;
}
```

Constructor holds the new values needed for the algorithm.

```
if( t_rtt_ < baseRTT_ ) baseRTT_ = t_rtt_;
if( t_rtt_ > maxRTT_ ) maxRTT_ = t_rtt_;
if(t_rtt_ > 0.000001) cwnd_ = cwnd_ + ((sqrt(( ( (double)maxRTT_ / (double)t_rtt_ ) * (double)
cwnd_ ))) / cwnd_);
break;
```

In opencwnd() function, cwnd is updated based on the given algorithm. This portion is only used when new acknowledgement is received and CCA is not in slow start mode.

```
void
ElasticTcpAgent::rtt_init(){
   baseRTT_ = __INT_MAX__;
   maxRTT_ = 0;
   TcpAgent::rtt_init();
}
```

rtt_init() function is modified with the new variable.

There are some other bookkeeping functions.

```
void
ElasticTcpAgent::delay_bind_init_all()
{
   delay_bind_init_one("baseRTT_");
   delay_bind_init_one("maxRTT_");
   TcpAgent::delay_bind_init_all();
   reset();

}
```

```
int
ElasticTcpAgent::delay_bind_dispatch(const char *varName, const char *localName, TclObject
*tracer)
{
   /* init vegas var */
       if (delay_bind(varName, localName, "baseRTT_", &baseRTT_, tracer))
       return TCL_OK;
       if (delay_bind(varName, localName, "maxRTT_", &maxRTT_, tracer))
       return TCL_OK;
       return TcpAgent::delay_bind_dispatch(varName, localName, tracer);
}
```

```
void
ElasticTcpAgent::reset()
{
   baseRTT_ = __INT_MAX__;
   maxRTT_ = 0;

   if(DEBUG) printf("RTT RESET!!!\n");

   TcpAgent::reset();
}
```

```
Agent/TCP/Elastic set maxRTT_ 0
Agent/TCP/Elastic set baseRTT_ 0x7fffffff
```

My Modifications:

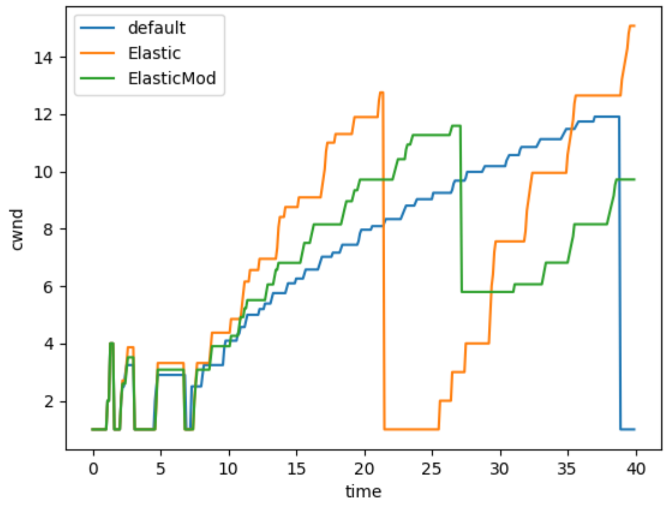
Based on the given algorithm, the congestion window for every new packet acknowledgement is increased proportional to **window correlated weighting function(wwf)** :

$$WWF = \sqrt{RTT_{max} \div RTT_{current} * cwnd}$$

This increase factor causes the cwnd to reach congestion much faster. So, I have decreased the congestion rate wwf by a another factor of sqrt.

$$WWF_{new} = \sqrt{WWF}$$

At first I wanted to make the reduction factor root of 1.5. But, I found that complex calculation can slow down the TCP transmission. So, another square root seemed like a good fit. Also, Elastic TCP doesn't mention about how to decrease the congestion window. I have borrowed code from NewReno in ns-2.35 for that. I have generated a **cwnd vs time graph** to compare my task:



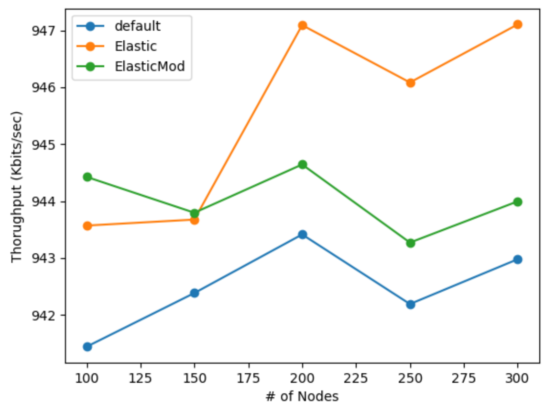
Here, we can see the ElasticMod cwnd is reaching the peak at a slower rate and the decrease is not as sharper as the ElasticTCP.

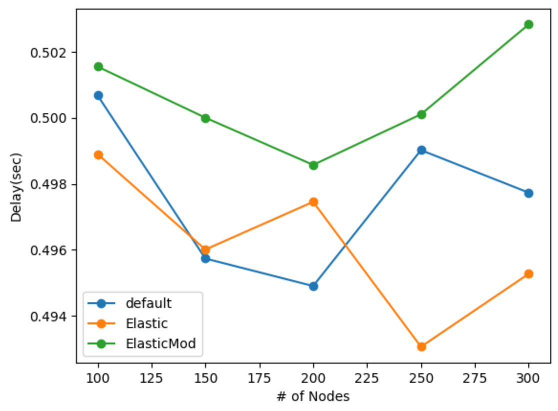
Results with graphs:

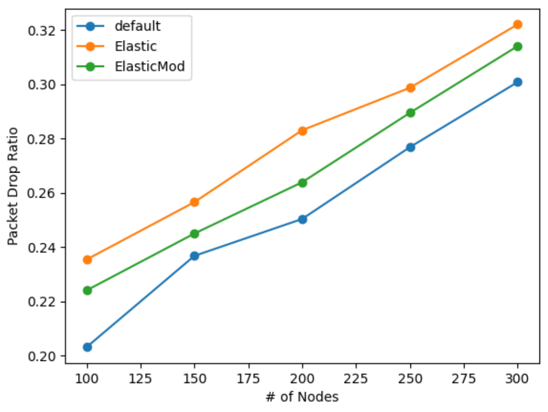
For comparison of **Elastic TCP**, and my modified protocol **ElasticMod**, default **TCP Tahoe (Agent/TCP)** was used. The graphs mention the default CCA as **default** in the legend.

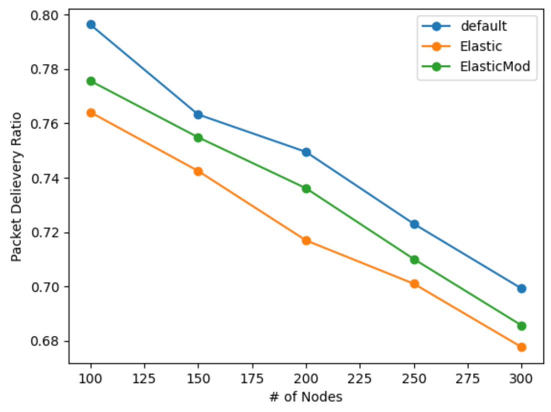
In this section, first we will look at the given scenario in the paper Dumbbell Network Topology with congested network. Then we will look at the scenarios mentioned in our assignment for both 802.11(mobile) and 802.15(static)

Dumbbell Topology:







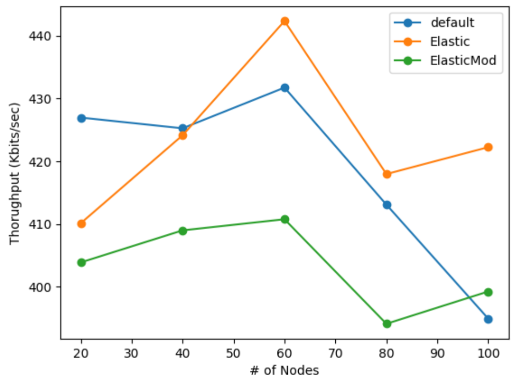


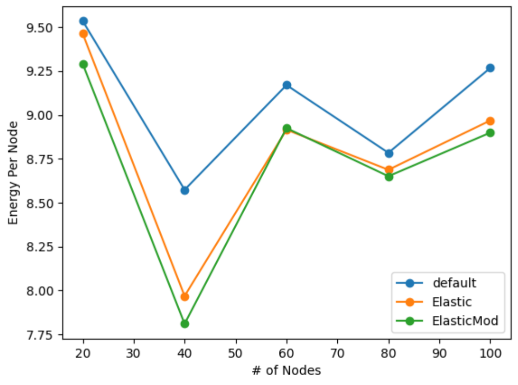
Result:

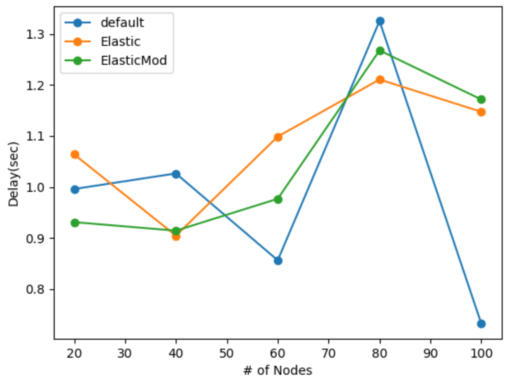
Elastic TCP is performing better in all scenario as mentioned in the paper. ElasticMod is performing average except that it has overall more delay than even TCP Tahoe.

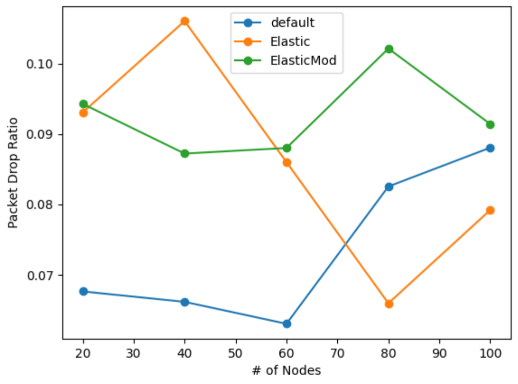
Wireless 802.11 Mobile (Mobile):

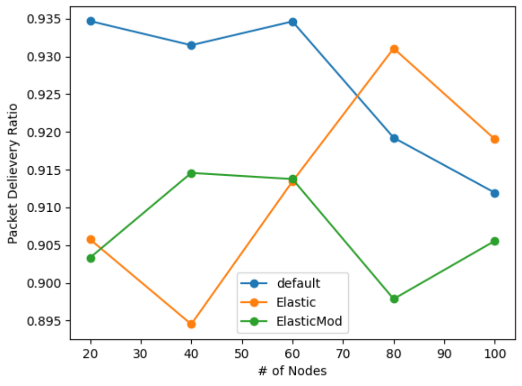
Number of Nodes:







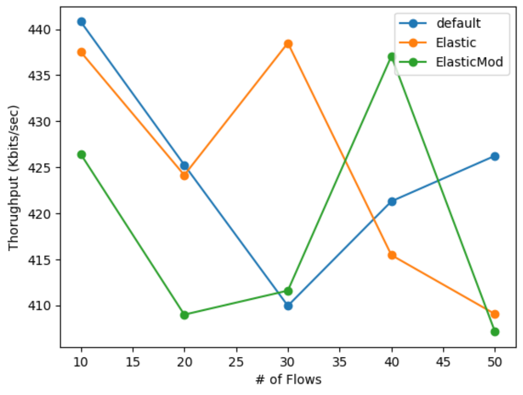


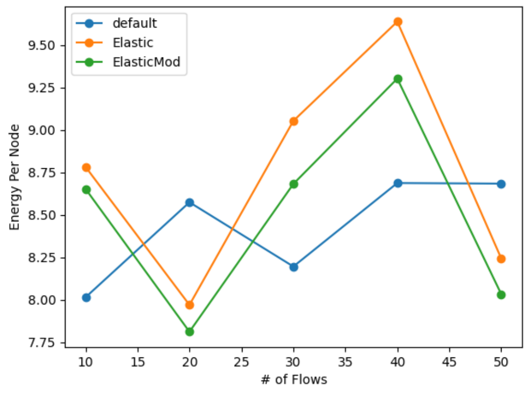


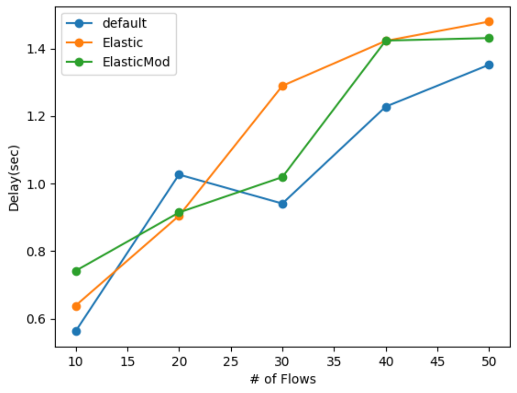
Result :

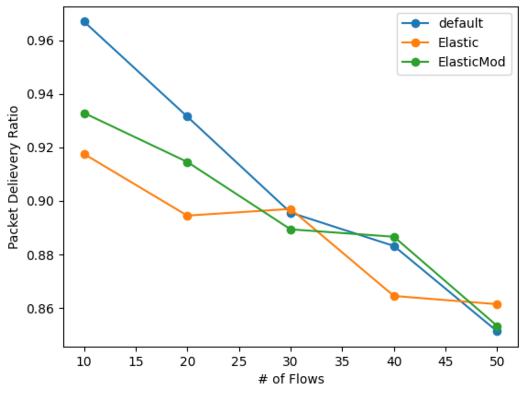
Throughput is quite same for TCP Tahoe and Elastic TCP. ElasticMod performs better in energy since it throughput is less. Delay is almost same for everyone. Tahoe has better delivery ratio.

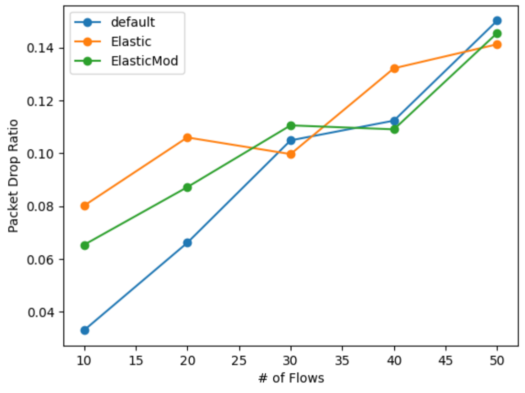
Number of Flows:





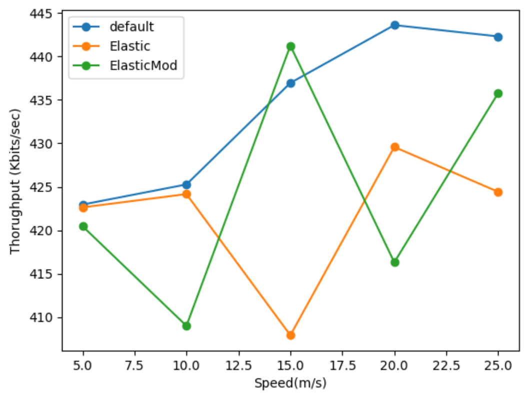


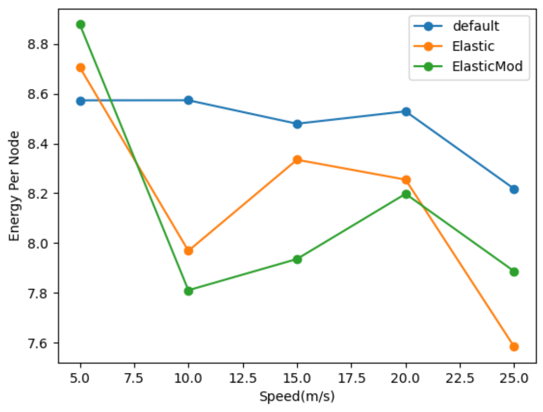


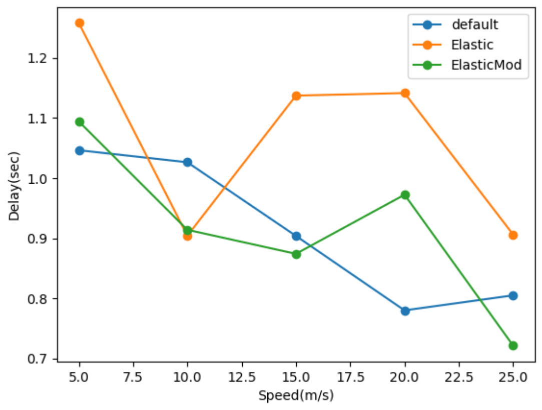


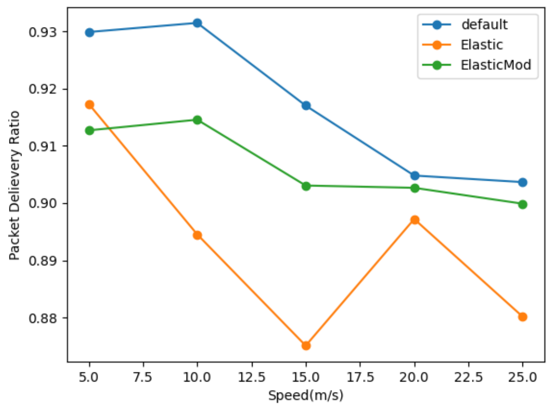
Result: With varying number of flows, it is hard to single out a better CCA. Tahoe is performing better in Energy Consumption. Delay and Drop Ratio by a slight margin.

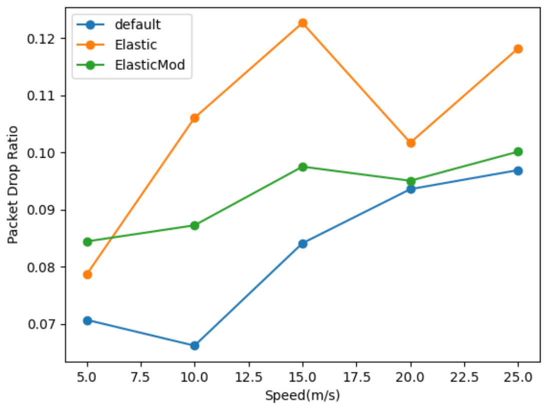
Speed:







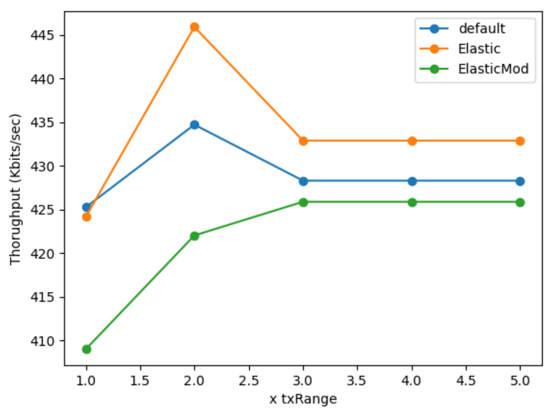


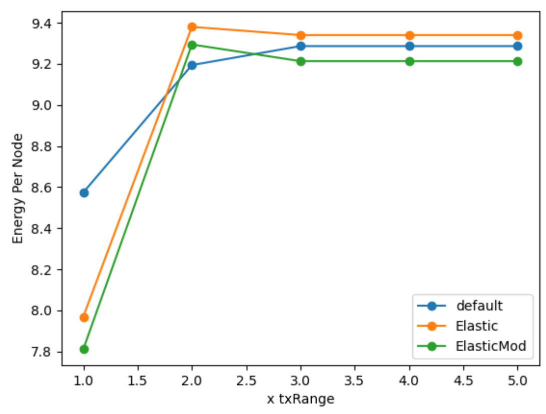


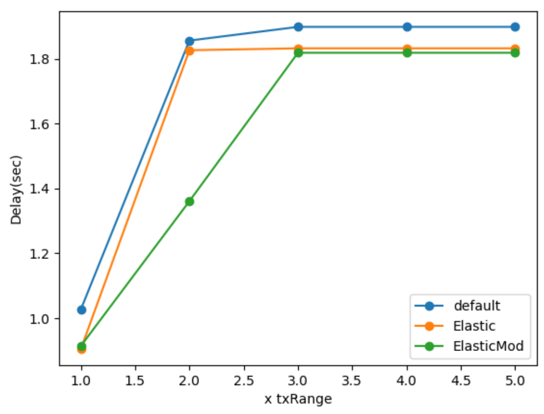
Result:

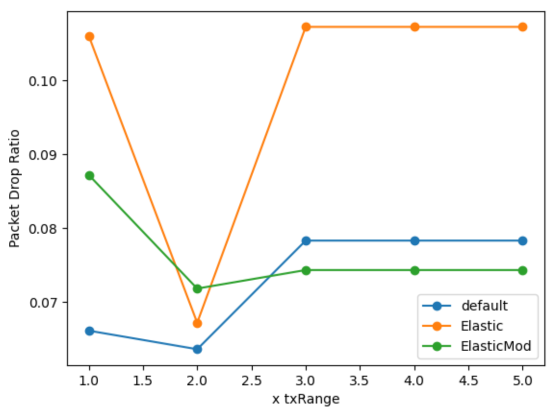
Hard to distinguish between these CCA on throughput vs flows graph. Energy consumption is trending to go low for both Elastic and ElasticMod for higher flows. Elastic Modis better on Delay by slight margin. Drop ratio is significantly better in Elastic TCP.

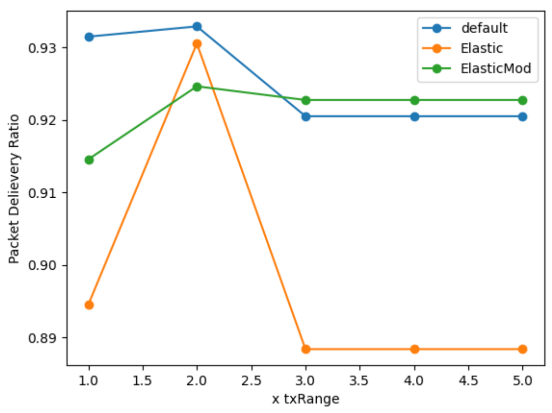
TxRange:









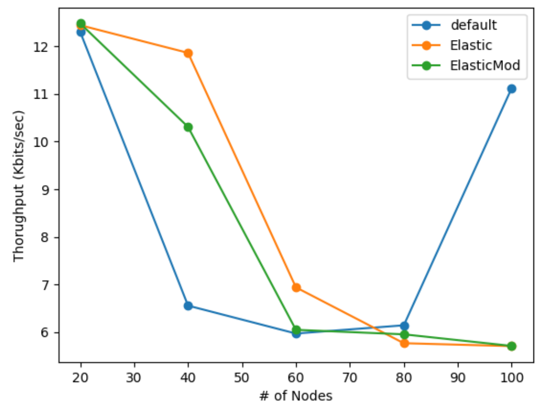


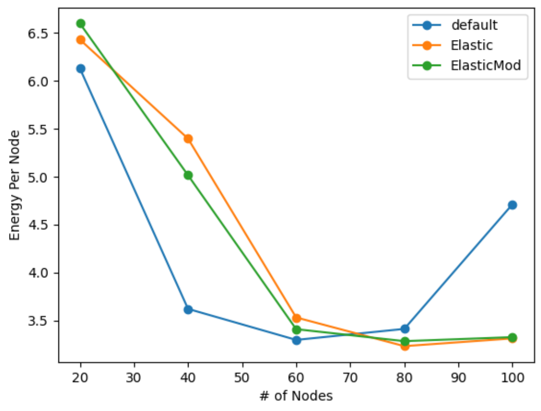
Result:

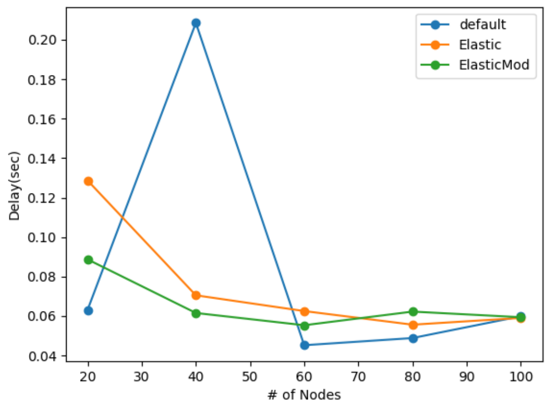
For TxRange to 3*TxRange the performance is varying. But higher TxValues have no impact since the TxRange is getting larger, probably more than the 500m topology used in tcl. Throughput is better in Elastic TCP. Delay and Energy Consumption is better in ElasticMod.

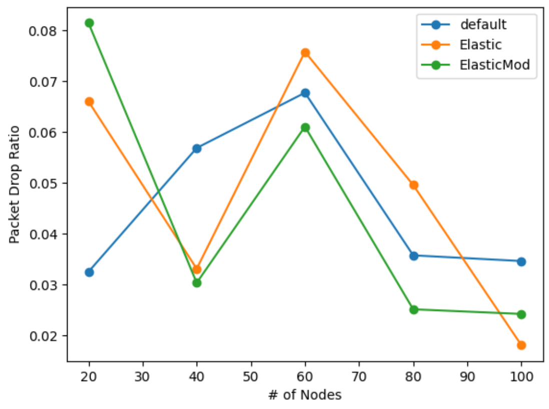
Wireless 802.15.4 (static):

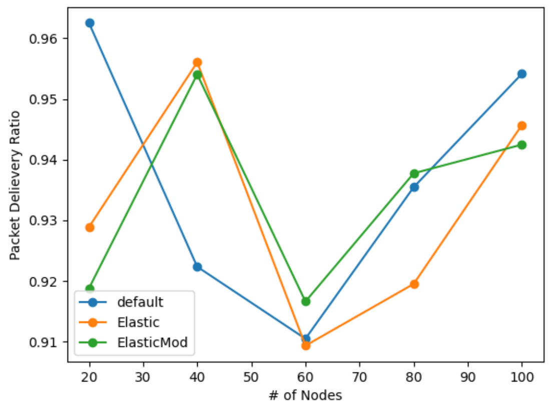
Number of Nodes:







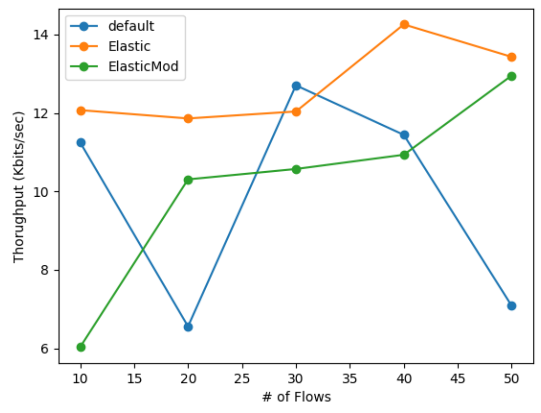


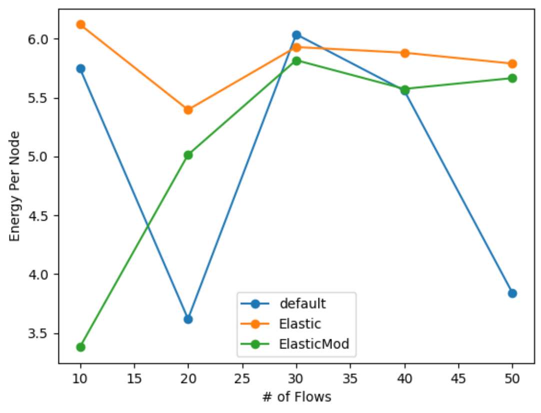


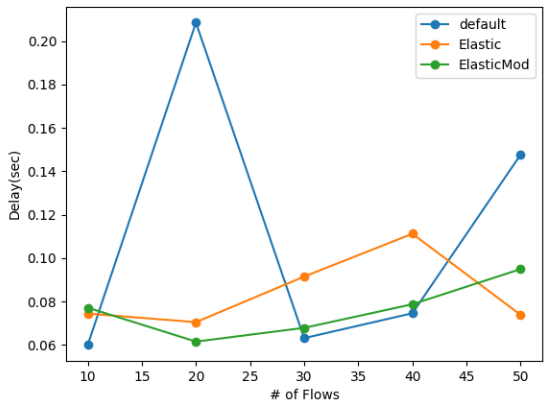
Result:

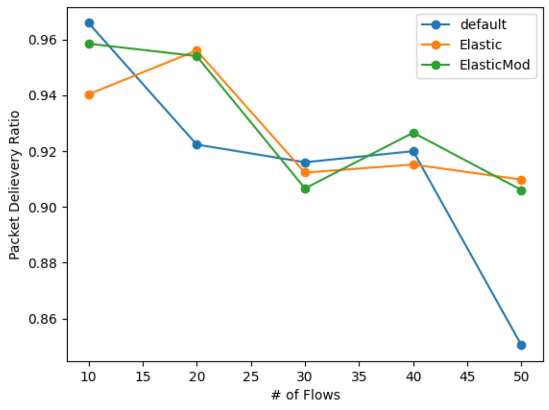
For 802.15.4 with varying number of nodes, it is much hard to predict which one is overall a better CCA. Running multiple times for each scenario with random seed and taking their average should produce a better comparison result.

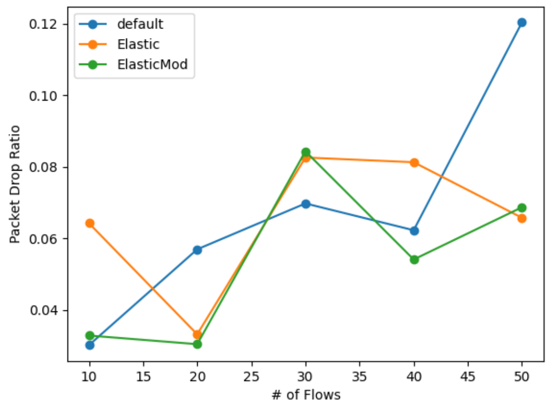
Number of Flows:







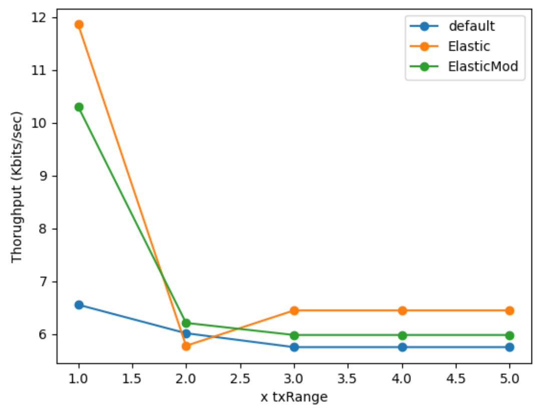


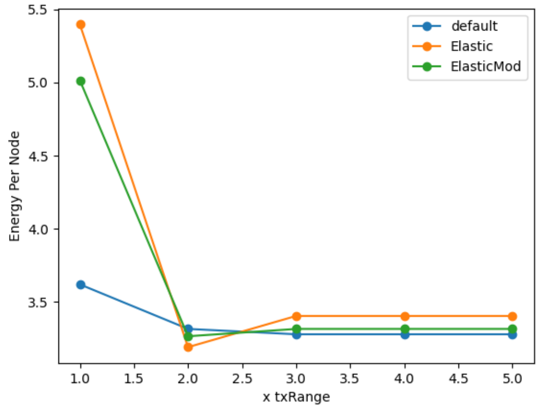


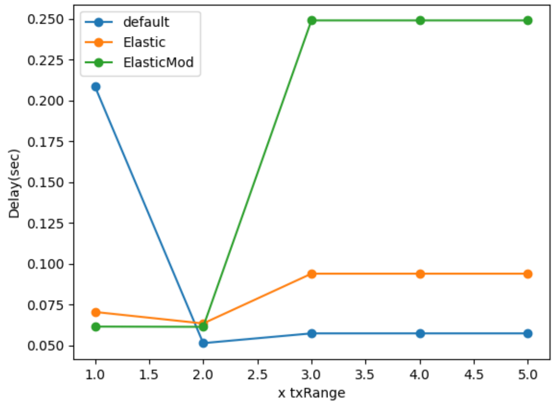
Result:

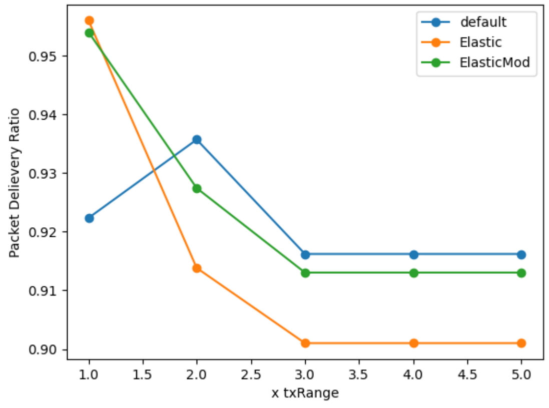
Throughput is better in ElasticTCP, number of flow does not effect its performance significantly. Delivery ratio is better for both version of ElasticTCP. On some cases of flow, Tahoe is behaving worse (Delay with 20 Flows, Delivery Ratio with 50 flows).

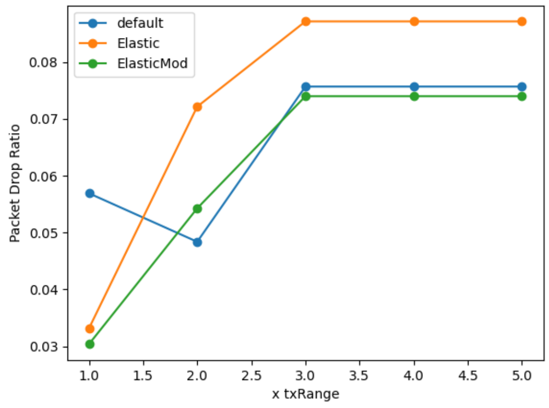
TxRange:











Result:

Similar to our findings in 802.11 scenario, higher values of TxRange has no impact for 500m topology. However, in small ranges, Tahoe is performing worse throughput wise, although its energy and delivery rate is better (since sending less packets). ElasticMod is showing huge delay for higher TxRanges.

Summary Findings:

Elastic TCP was introduced for high BDP (bandwidth delay product) network with wired topology. The scenario provided in the paper is producing a good result for this CCA.

But using it for wireless transmission provided is producing inconsistent result when comparing it with TCP Tahoe.

My thinking behind the modification was to avoid the sharp drop in cwnd when getting 3DupAcks. And also see how changing the **WWF(window correlated weighting function)** changes the performance. But using any factor of root is not advisable. So my option was to either not doing root at all or taking a root of 4. Not doing root was performing worse in Dumbbell Network. So, root of 4 was chosen. But my findings show that square root (as mentioned in paper) is the overall better choice.

Varying TxRange was not obvious in ns2. I found that the range is proportional to a variable Pt^4 in Phy/WirelessPhy class in threshold.cc file; in case of TwoRayGround propagation model.

Packet rate was not possible to vary due to using TCP as a transport layer. Setting the packet size in FTP does not affect the performance in any way.