Distributed Hop by Hop Congestion Control in Wireless Sensor Networks

By Vishal K. Singh.

1. Introduction

The aim of the project was to develop a distributed hop by hop congestion control mechanism and incorporate the strategy in CODA[1] to improve the congestion control mechanism. The distributed hop by hop mechanism to adjust source rate is incorporated in CODA.

The algorithm changes the source rate at every node in proportion to change in congestion factor which is calculated based on congestion scenario at this node and cumulative congestion factor received from nodes downstream.

2. Design

The high level design consists of following:

1. Calculation of congestion factor in a distributed way based on the following factors:
    a. Channel Loading Conditions
    b. Queue Size.
    c. Number of Retransmissions.

Channel load and queue size are traditionally used indicators for congestion in sensor networks. A high channel load though indicates activity but does not concludes that the current node is affected because there might be high utilization and the node may be at the edge listening and register high load. Queue size is also a good measure of number of packet rate.

We choose to use the amount of time the MAC of sensor nodes spends in retransmission as more accurate measure of effect of congestion on the node and affects the energy tax and fidelity given in CODA.

2. Adjusting source rate of nodes based on the value of congestion factor calculated locally and based on cumulative congestion factor received

3. Propagates the value of congestion factor further upstream towards the source if it is above a certain threshold.

The design is based on [2] applied to sensor networks.

2.1 Calculation of Congestion factor

Congestion factor at a node is an indicator of congestion seen by the current node which comprises of local congestion and a distributed view of congestion in some or all portion of network downstream. The sensor node should receive congestion factor periodically (where periodicity depends on congestion downstream) and adjust the source rate based on this factor. A higher value of subsequent congestion factor indicates increase in congestion and causes source rate adjustment to deal with it.

The following factors can be used in the calculation of the congestion factor locally.

- Channel Load :

As the CODA paper [1] explains, channel load can be a good indicator of congestion around a node. This value is calculated by sensing the channel` at periodic intervals and looking at how many times it was found to be busy. This value can then be used to calculate the congestion factor.

- Queue Size :

Channel load value, though a good indicator of congestion, can be misleading sometimes. Hence, we also use current Queue size as a metric to calculate the congestion factor. The decision to include Queue occupancy in the calculation was motivated by the results of experiments mentioned in another paper [3]

- Retransmission Counter:

The NACK or ACK based retransmission counter and retransmission time over a unit period of time over which rate regulation happens.

Total congestion factor = Function of (Local congestion factor, Received congestion factor)

This equation is yet to be established though we implemented giving them 3/4 and ¼ weights and performance was found to be improved. The optimal value has to be established mathematically as well as experimentally. This Total congestion factor is propagated upstream, which indicates the weights given to local and received determine the rate dependency proportion on local congestion or congestion downstream.

2.2. Rate adjustment based on received Congestion Factor

Rate adjustment limits the rate at which packets are sent based on the how it sees the congestion in entire network. This also decides how much energy is expended in the network. CODA makes a node sleep for random time when a suppress message is received. In our approach we adjust the rate instead of making the node sleep.

Based on the difference between the current congestion factor and received congestion factor, the node adjusts its rate using either an AIMD strategy or a rate control formula as in [2].

3. Changes from CODA strategy and Implementation Details

- No sleep when suppress message is received
- No Closed loop control

Code level Changes (Functions)

- sendSuppressMsg :
  This function was modified to incorporate the congestion factor to be sent as an indication of the congestion state downstream.
- suppressMsgRcvd :
- calcCongestionFactor :
  This function calculates the congestion factor based on the factors above and is called from 2 places
  1. channel sampling code
  2. DataMsgRcvd
- sendNakMsg
- rcvdNakMsg
- Channelsense.result :
-

The values of congestion factor received is updated and used to adjust the rate. It is also used to make the decision about further propagation of suppress messages.

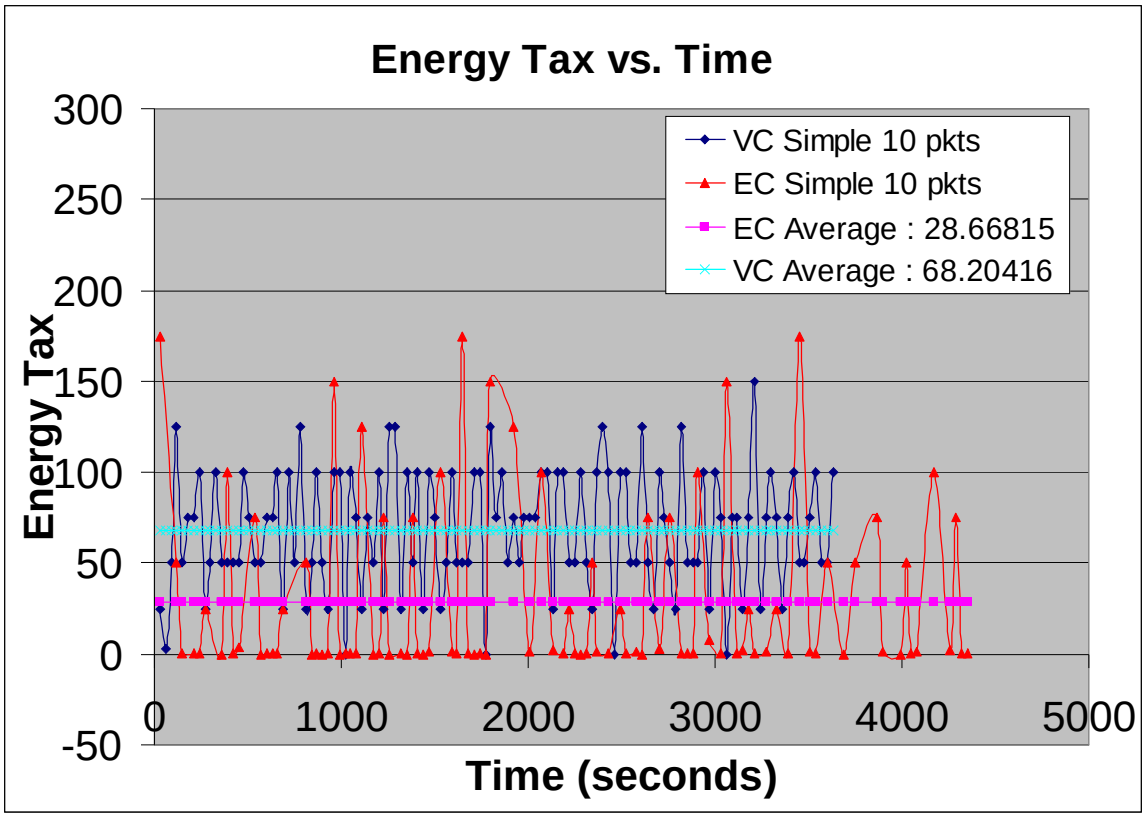
4. Experiments

We did many experiments to compare the performance of vanilla coda and our coda for different source rates, different radio models and different topologies. Our varied our source rates from 5, 7, 10, 13, 15, 17, to 20. We also experimented with different radio models such as simple, lossy generated using lossy builder generator, and lossy one-hop models generated manually. We also changed the topology of the nodes to compare how the systems responded to increasing the number of nodes and sources and congestion localities in the network.

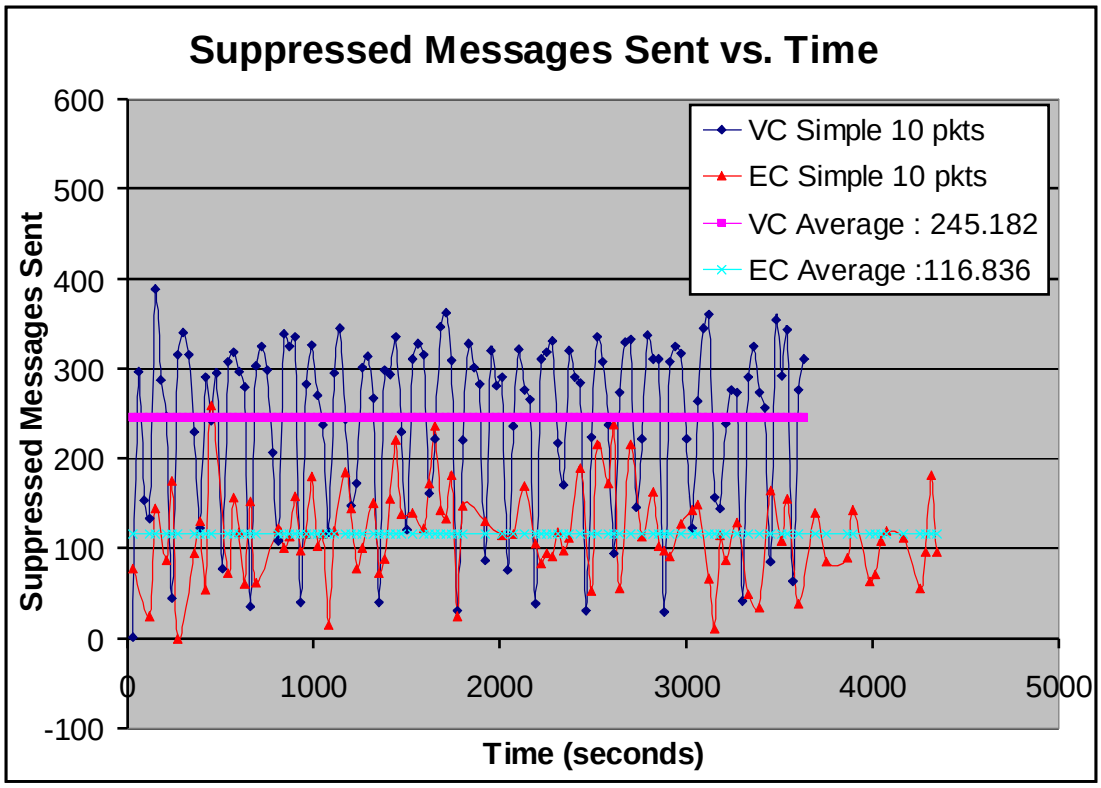
5. Results and Analysis

As we expected, we got good results and significant improvements over CODA. The results along with the graph. The following are comparison of coda(VC) with our coda(EC) for simple model and lossy models.

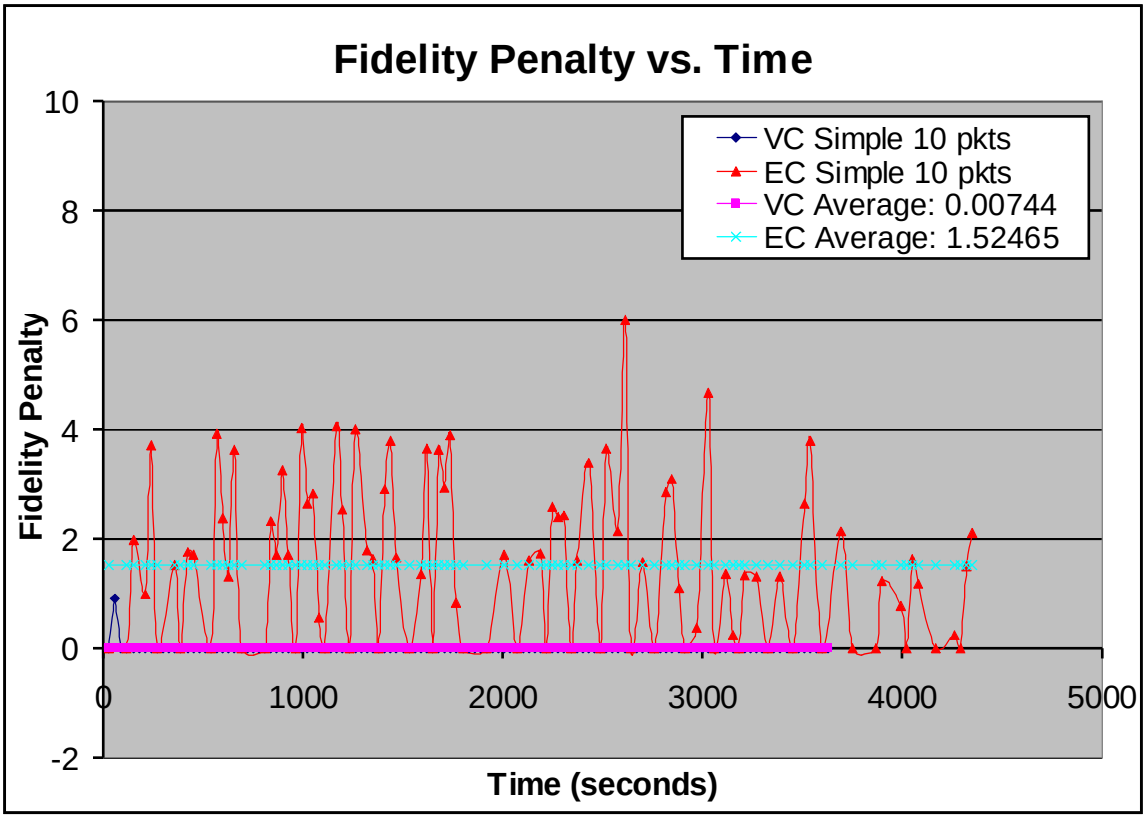
Overall, values for suppression messages sent was counted for all nodes on 30 second intervals. For calculating the average energy tax, the difference was taken between the number of packets sent from the source minus the number of packets received at the sink divided by the total number of packet received at the sink. This number of packets sent and received was calculated over 30 second time intervals and the equation follows that which was used in the CODA paper. For calculating average fidelity penalty for all different models we were using anquation dealing with the total number of packets received at the sink divided by 30 (to compensate for the 30 second time interval). This was also the way fidelity penalty was calculated in the CODA paper.



From the above graph, it can is indicated that on average our enhanced CODA model worked better than the vanilla CODA model. This can be attributed to the fact that the average energy tax for vanilla CODA is 68.2 and for enhanced CODA is 28.68 which is significantly smaller than the original model.

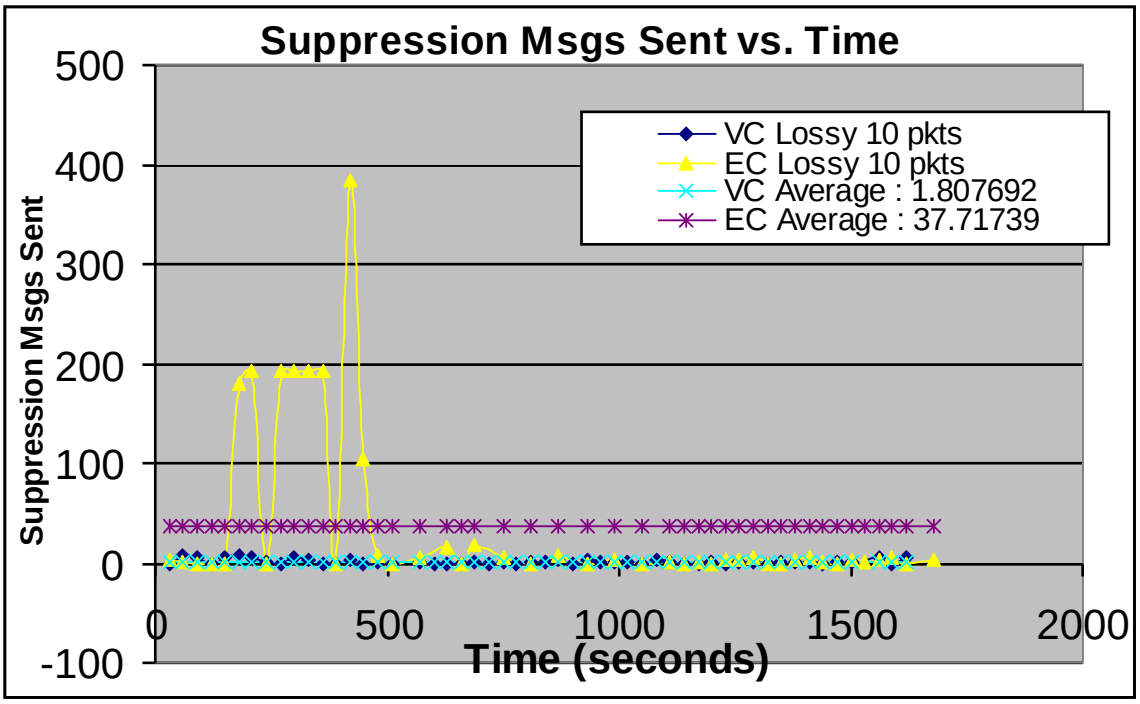


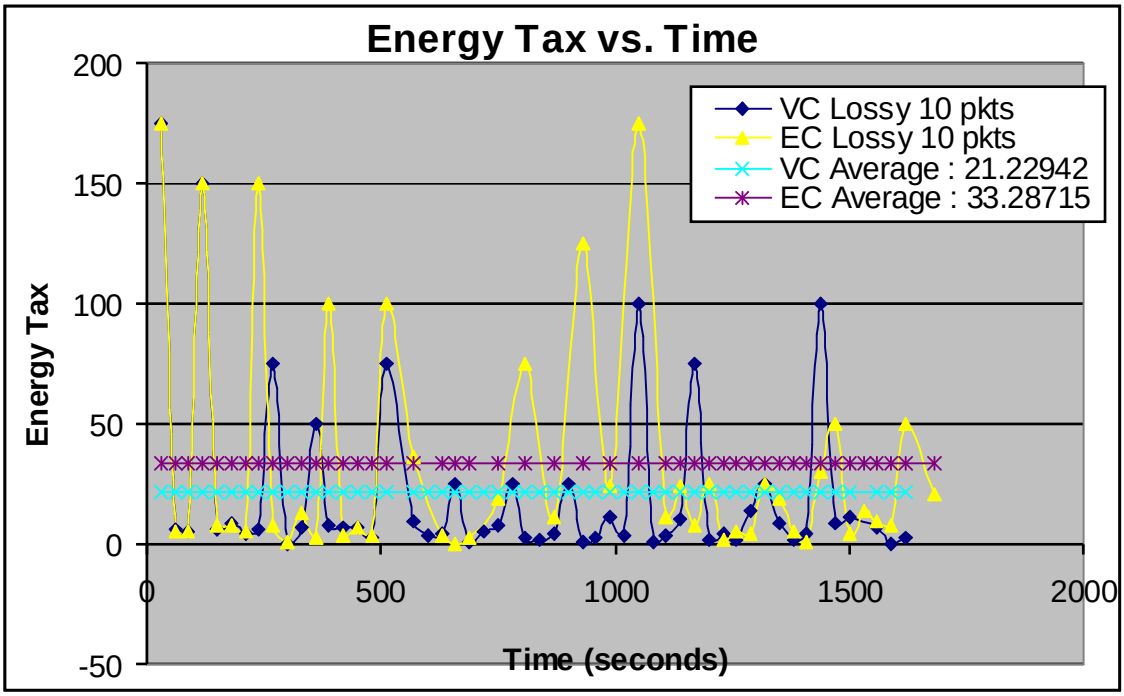
As can be seen from the above graph, vanilla CODA sent out many more suppress messages than did our enhanced CODA version. Varying values of suppress messages sent can be expected as we are propagating it.

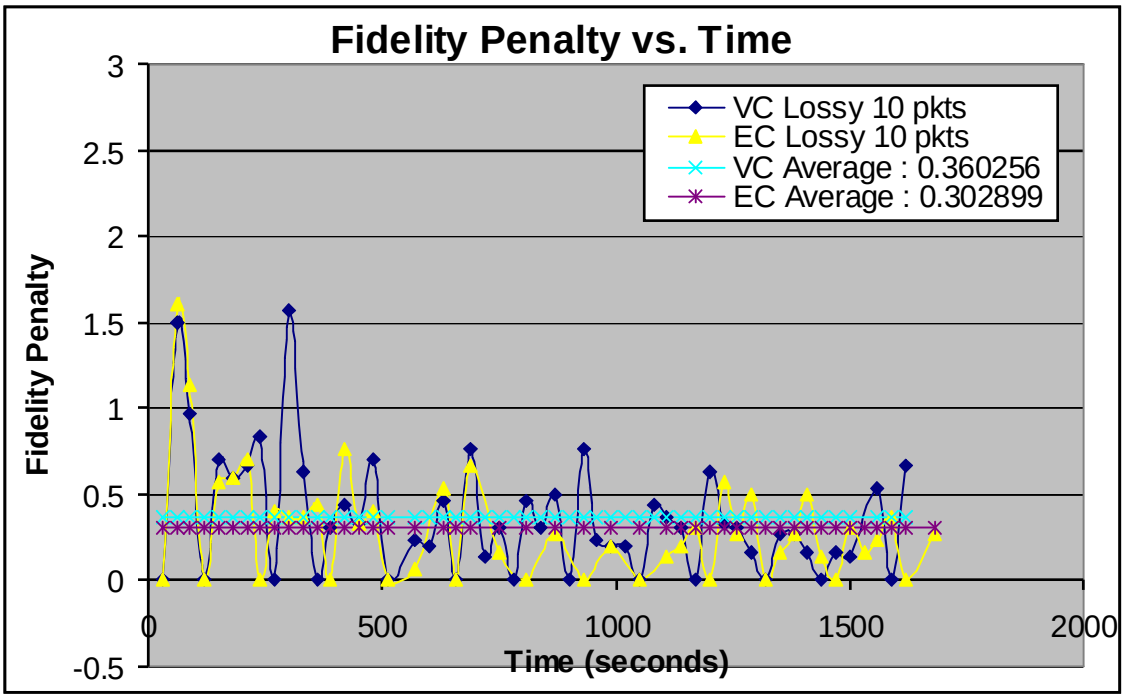


As can be seen from above, the fidelity value again further proves that our model performs better since we obtained an average fidelity penalty of 1.52 whereas vanilla CODA gave an average of .00744.

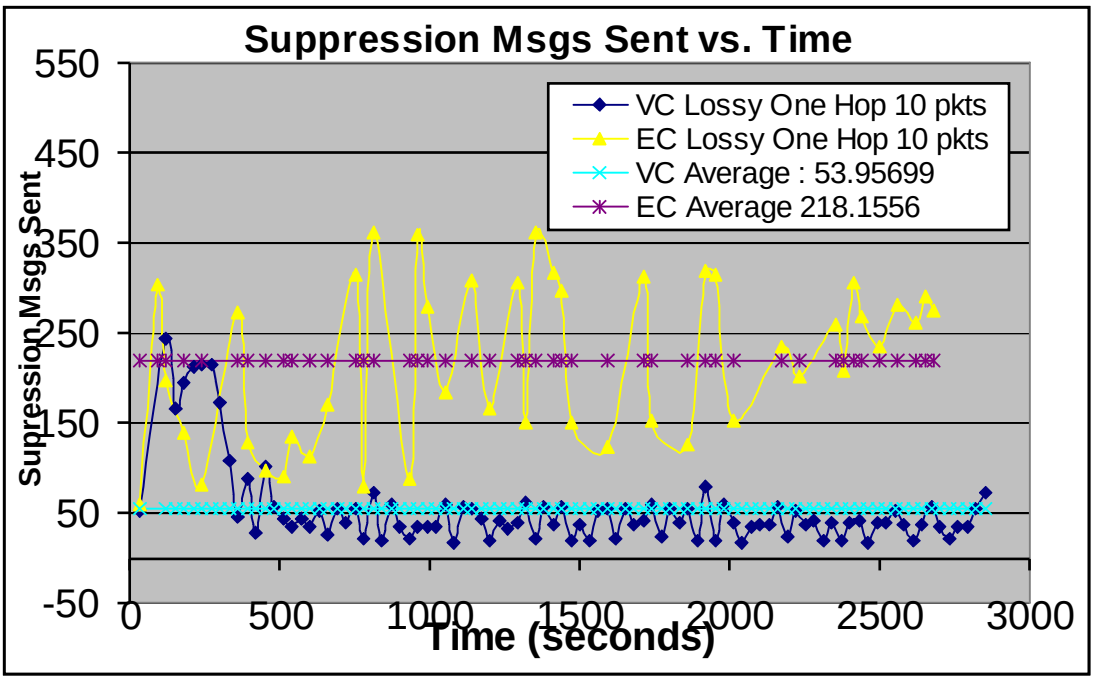
The above three graphs represent the simple models at 1 packet/sec for vanilla CODA and enhanced CODA.

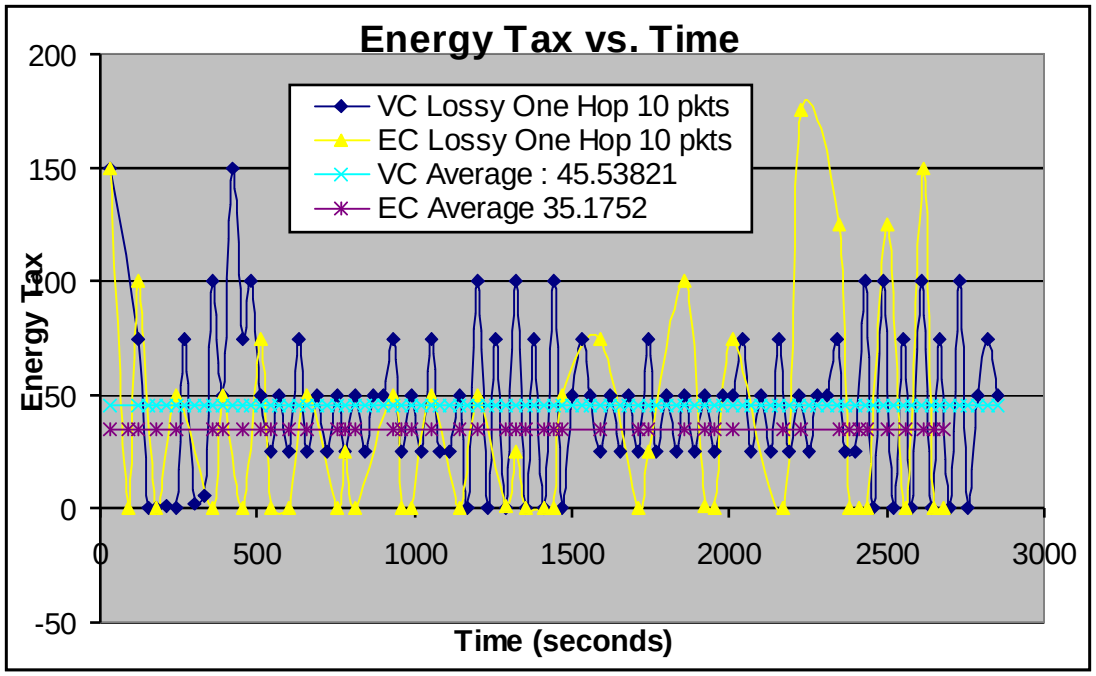

Suppression Msgs Sent vs. Time
500
400
300
200
100
0
-100
0
500
1000
1500
2000
Suppression Msgs Sent
Time (seconds)
VC Lossy 10 pkts
EC Lossy 10 pkts
VC Average : 1.807692
EC Average : 37.71739

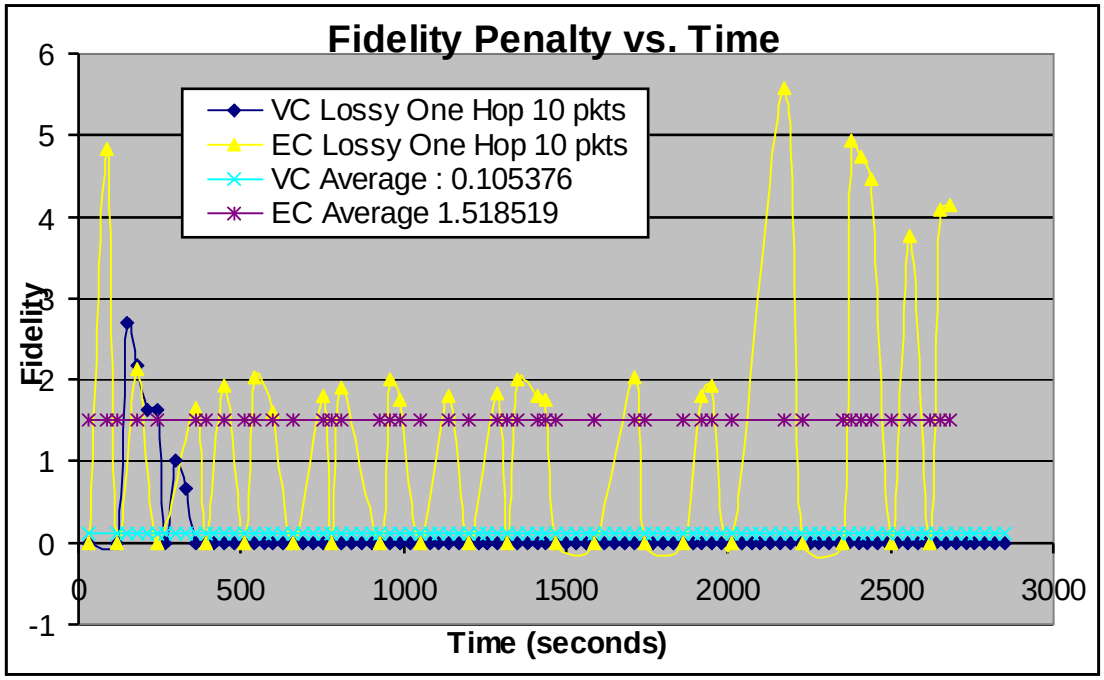

Energy Tax vs. Time
200
150
100
50
0
-50
0
500
1000
1500
2000
Energy Tax
Time (seconds)
VC Lossy 10 pkts
EC Lossy 10 pkts
VC Average : 21.22942
EC Average : 33.28715



The above three graphs represent the lossy models from the lossy model generator at 1 packet/sec for vanilla CODA and enhanced CODA.



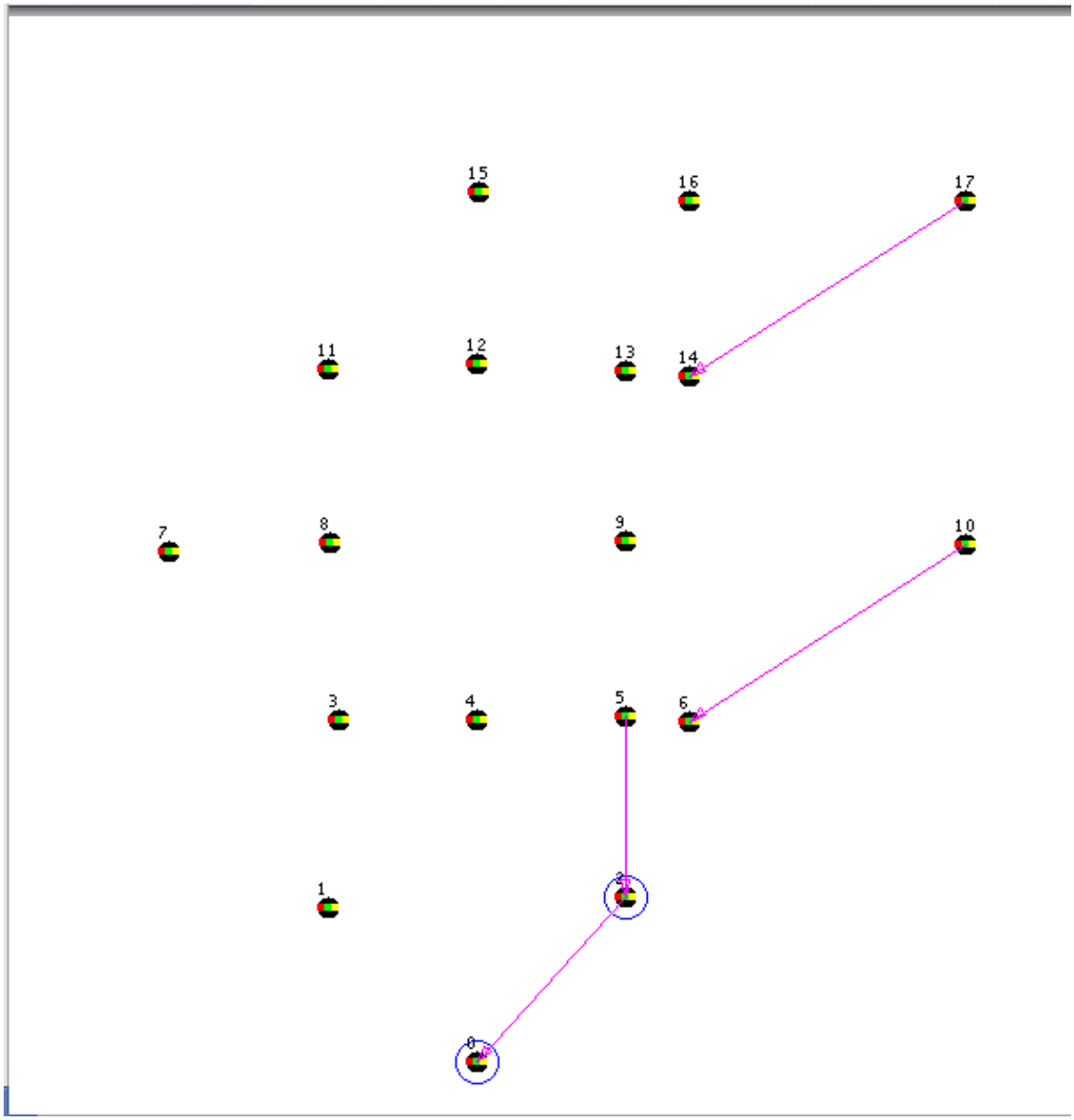




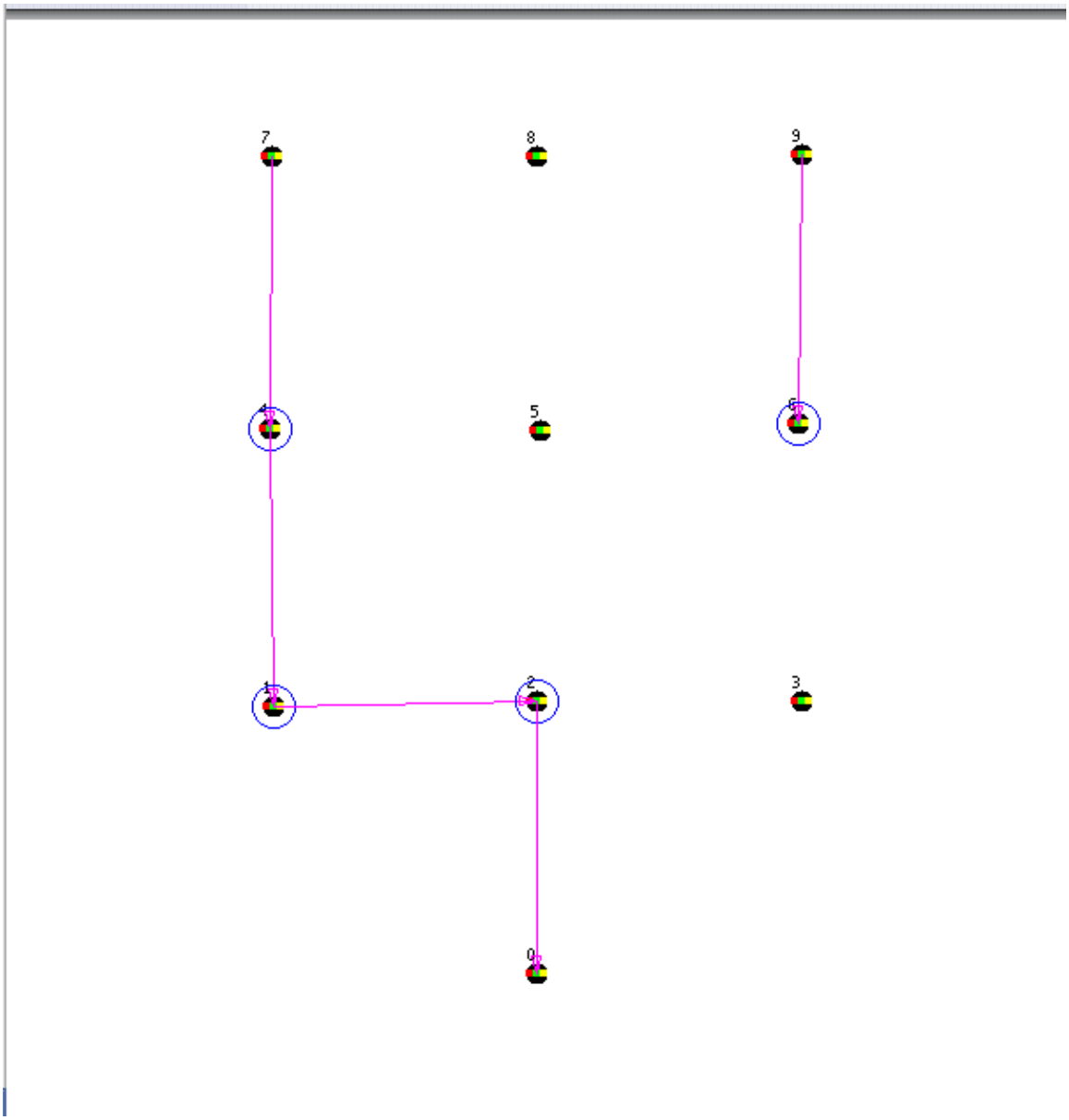
The above three graphs represent the lossy models for the one-hop model at 1 packet/sec for vanilla CODA and enhanced CODA.

6. Conclusion and Future work

The future work is to do it on motes and see the result.



Topology with 18 nodes



Topology with 10 nodes

References:

1) Chieh-Yih Wan, Shane B. Eisenman and Andrew T. Campbell, "CODA: Congestion Detection and Avoidance in Sensor Networks", ACM SenSys 2003, November 2003.
2) "Hop-by-hop Congestion Control over a Wireless Multi-hop Network," Y. Yi and S. Shakkottai. Proceedings of IEEE Infocom, Hong Kong, March, 2004.
3) Bret Hull, Kyle Jamieson, Hari Balakrishnan, "Techniques for Mitigating Congestion in Sensor Networks" ACM SenSys 2004, November 2004.