Consensus Algorithm in Distributed Network

By

Rajkin Hossain (ID: 12101043)

Under the Supervision

Of

Dr.Md.Muhidul Islam Khan

Department of Computer Science & Engineering

School of Engineering & Computer Science

BRAC University

Consensus Algorithm in Distributed Network

Thesis submitted in partial fulfillment of the requirement for the degree of

Bachelor of Science

In

Computer Science and Engineering

Under the Supervision

Of

Dr.Md.Muhidul Islam Khan

By

Rajkin Hossain (ID: 12101043)

April 2016

DECLARATION

We do hereby declare that the thesis title “Consensus Algorithms in a

Distributed Network” is submitted to the Department of Computer Science and Engineering of BRAC University in partial fulfillment of the completion of Bachelors of Science in Computer Science and Engineering. We hereby declare that this thesis is based on results obtained from our own work. Due acknowledgement has been made in the text to all other material used. This thesis, neither in whole nor in part has been previously submitted to any University or Institute for the award of any degree or diploma. The materials of work found by other researchers and sources are properly acknowledge and mentioned by reference.

Dated: 21 April 2016

Signature of Supervisor	Signature of Author
M. Islam	Rajkin
______________________	________________
Dr.Md.Muhidul Islam Khan	Rajkin Hossain(12101043)

Assistant Professor

Department of Computer Science and Engineering

BRAC University

Dhaka, Bangladesh

Final Reading Approval

Thesis Title:
Consensus Algorithms in a Distributed Network.

Date of Submission: 21th April, 2016

The final report is satisfactory and it's all materials are also acceptable and ready for the submission to the Department of Computer Science and Engineering, BRAC University.

Signature of Supervisor

M. Islam

____

Dr. Md. Muhidul Islam Khan

Assistant Professor

Department of Computer Science and Engineering

BRAC University

Dhaka,Bangladesh

CONTENTS

ABSTRACT

One of the challenging issues in a distributed computing system is to reach on a decision with the presence of so many faulty nodes. These faulty nodes may update the wrong information, provide misleading results and may be nodes with the depleted battery power. Consensus algorithms help to reach on a decision even with the faulty nodes. Every correct node decides some values by a consensus algorithm. If all correct nodes propose the same value then all the nodes decide on that. Every correct nodes must agree on the same value. Faulty nodes do not reach on the decision that correct nodes agreed on. Binary consensus algorithm and average consensus algorithm are the most widely used consensus algorithm in a distributed system. We apply binary consensus and average consensus algorithm in a distributed sensor network with the presence of some faulty nodes. We evaluate these algorithms for better convergence rate and error rate.

Index Terms: Wireless sensor networks; Consensus algorithm, Distributed systems, Convergence Rate, Faulty Node Tracking, Binary consensus, Average consensus.

CHAPTER 1

1.1 Introduction

The advancement of radio equipped modules and miniaturization of electronic components motivate the development of wireless sensor network (WSN) in which numerous distributed sensor nodes are usually deployed to perform a wide variety of applications, such as monitoring, surveillance, security, health care, and load balancing [1,2]. Nodes are usually deployed randomly in an ad hoc manner, and for certain tasks, the detection values at different nodes are conditionally independent. Conventionally, tasks are executed in a centralized manner that is straightforward to implement. However, it is not scalable for an increasing number of nodes and sometimes it is expensive and impossible to deploy and maintain such a central controller [3]. Thus, the management technique and distributed decision-making algorithm that organize these multiple distributed agents to carry out a task cooperatively have been extensively studied in recent years. Individual detection by one node in the distributed dynamic WSN system is not sufficient to perform decision making without knowledge of the global network. A consistent decision must be reached among these geographically dispersed sensor nodes through some type of information exchange mechanism. This decision, based on common interests, is referred to as reaching consensus using the detection values of the sensor nodes.

Although the consensus algorithm has been thoroughly studied in the control area, it is of vital important in the distributed sensor network. It is acted as a way to achieve globally optimal decision in a totally decentralized way, without sending all the sensors data to a fusion center [4]. Recently, the most attractive consensus algorithm is the gossip algorithm [5], where pairs of nodes are selected at random to exchange and update their values. Compared with routing algorithm, it is robust and easily implemented. It is not necessary to put much effort on route discovery and route maintenance, and it is a distributed iterative information exchange scheme. However, random information exchange between neighbors also leads to overhead and increases the time to reach consensus in the network. In addition, the connectivity of the network affects the accuracy of the final consensus value.

Lots of research has worked on improving the convergence rate of gossip algorithm. Geographic gossip, which combined the gossip algorithm with geographic routing, was recently proposed [7]. This algorithm increases the diversity of the pairwise gossip operation by randomly choosing pairwise gossip nodes within the entire network rather than selecting them from adjacent nodes. The improved approach of geographic gossip was path averaging [8], where the average was performed at each node along the route between the exchanging pair nodes. However, in these two mechanisms, the probability of packet loss increased when sending messages along longer routes. Additionally, as the distance between pairwise nodes that are exchanging information increases, extra energy is consumed to set up and maintain the two-way route between them. The broadcast gossip algorithm which takes advantage of the broadcast characteristic of the wireless medium was proposed [9]. This scheme enables all the neighbors of the wake-up node to listen to the data transmission and perform updates. The other approach that makes use of the broadcast characteristic of wireless medium was the eavesdropping gossip [10], where each node can overhear the data broadcasted by its neighbors and the exchange pair of one node is optimally based on all the data that it received. Subsequently, cluster-based gossip algorithm has been proposed [3, 11, 12]. In [3], each node had a timer which was decremented by 1 at each time and the cluster head was chosen as one node's timer expires. This cluster formation method is simple and easy to implement and to distribute. However, it is impractical in reality because some of the nodes may not have chance to join a cluster. In [11], an algorithm that combines cluster and geographic routing was proposed for a large-scale sensor network. The network is firstly divided into grid clusters, and then the standard gossip algorithm is executed to reach a local consensus in each cluster area. A representative node is subsequently chosen in each cluster, and pairwise gossip is executed among these representative nodes via multihop routing until the consensus goal is reached. However, in the cluster stage, the nodes are divided into groups according to their locations. Thus, an imbalance in node numbers in different cluster slows the convergence rate for reaching consensus. In [12], the authors analyzed the data transmission scheme in the cluster, based wireless network, but they did not mention how to form a cluster, and if the cluster head is collapses, the whole network can no longer reach consensus.

Referring to cluster mechanisms, different cluster algorithms are proposed. For example, the Lowest Identifier (LID) [13] which chooses the node with the lowest ID as a cluster head is a simple clustering method. Its cluster formation method is similar to [3] and its cluster head chosen method is similar to [11]. Highest-Connectivity Degree Algorithm (HCDA) [14] is a connectivity-based cluster formation algorithm which is based on the neighbor number of a node. In HCDA, there is no restriction on the number of nodes in a cluster. When the number of nodes in one cluster is too large, the burden of the cluster head becomes too heavy which may lead to communication bottleneck. The lifetime of the whole network is short because of the imbalance of the network load. There are also some other algorithms, such as distributed clustering algorithm [15], distributed mobility adaptive clustering [16], and weighted clustering algorithm [17] which introduce weight on the selection of a cluster head. All of these above algorithms have not considered the impacts of the number of nodes in one cluster on the network capacity and throughput. Therefore, in this paper, we introduce a throughput/capacity aware cluster mechanism for the gossip algorithm and evaluate its convergence performance.

The contributions of this paper are the following.

(i) We applied our own Binary Consensus Algorithm.
(ii) We applied our own Average Consensus Algorithm.
(iii) An irregular sensor model is introduced to evaluate the robustness of the algorithm.
(iv) We involved segment tree data structures to reach consensus decisions in the fasten time.
(v) We used random graphs and plot out for both binary and average consensus for nodes vs iterations, nodes vs faulty nodes, nodes vs error-rate. AS binary consensus has no error rate so we ignored error-rate for binary consensus.
(vi) We used various topologies like: C,D,O,I,H

Actually we are applying our own binary consensus and average consensus algorithm in a distributed network for reaching a consensus decision. Our algorithm runs for huge network. Initially we don't have any idea of the whole network besides there can be many faulty nodes. Our mail goal is to reach consensus decision as fast as possible but in few cases may be random choose will run faster but we are ignoring random choices because random choices might get huge amount time to reach a consensus decisions. So we can claim that the probability of our algorithm run faster than random choice based algorithm like gossip algorithm.

1.2 Related Works

In a centralized networked system, where there are central base stations to process the works for sensor nodes, it is easy to reach on a decision after gathering all the information from the nodes [6]. But in a distributed network with a huge number of sensor nodes it is difficult to process the information without the central base station. Sensor nodes work cooperatively to reach a particular decision. Consensus algorithms help to achieve that. The widely used consensus algorithm to reach a decision in a distributed environment is gossip algorithm [7]. In gossip algorithm, pair of nodes are chosen randomly to exchange information and update their values. Comparing with other routing algorithms, it does not need any route discovery and route maintenance. It is easy to implement. But the random information exchange creates more overhead in the network and it takes more time to reach consensus. Connectivity between nodes also affect the consensus value for the gossip algorithm.

Another widely used consensus algorithm is binary majority consensus [18]. Binary majority consensus algorithms generally agrees on binary values selected from the range defined as {0, 1} or {1, 1}. The agreed state should be the value that the majority has been agreed. This algorithm terminates after a particular amount of time even the consensus is reached or not. Time limitation leads to lower efficiency and higher sensitivity to disturbances. The average consensus algorithm [8] helps to reach the consensus by updating their local values using average values of their neighbors' values. This consensus algorithm does not provide guaranteed consensus for the additive noise in the network. We apply binary consensus and average consensus in a distributed sensor network. We compare their convergence rate to find out the faulty nodes in a network. We use advanced data structure for making the consensus algorithms faster.

1.3 Network Model

The given network model is undirected and may be disconnected excluding cluster heads. The model might have many faulty nodes but we don't know which nodes are faulty initially and every connected component of that model had cluster head. Neighbors of each node define as under a certain radius the existing nodes. Cluster head also work as normal nodes but additional characteristics of them are they can't be faulty and every connected components cluster head make a complete graph which means they can communicate to each other's which also means including cluster head the whole network is connected. Figure 1 defines a sample of network model. Figure 2 is the graph representation of this network model. In both figures cluster head is 5 which has enough energy and can communicate to any other node under its radios also can communicate to other cluster heads. Figure 3 is a sample overview what we described here. Each node of the sensor network hold 3 values.

1) Energy

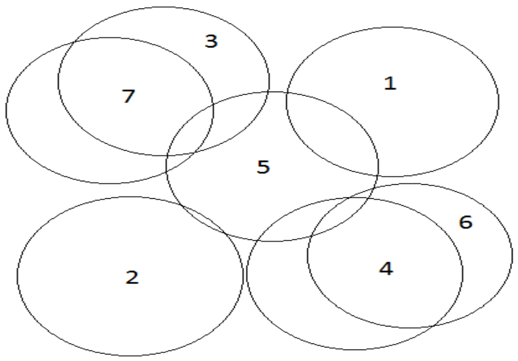


Figure 1: Network Model

2) Binary States

3) Average States

As we are working on both binary and average consensus we need to use it. When a node energy become zero or less than it become a faulty node and can not communicate throughout the network. But if a node need a few energy to communicate we assume it can take energy from its cluster head but just can only one time. For the next time its denoted as faulty node. In out proposed algorithms we actually work on edges. We greedily chooses edges

and update their states individually for both binary and average states. Their updating rules are also different. So by updating their states we reach a consensus decision.

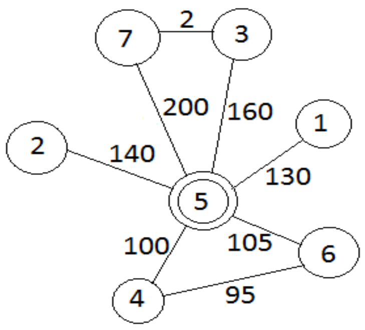


Figure 2: Network Model
in Graph Representation

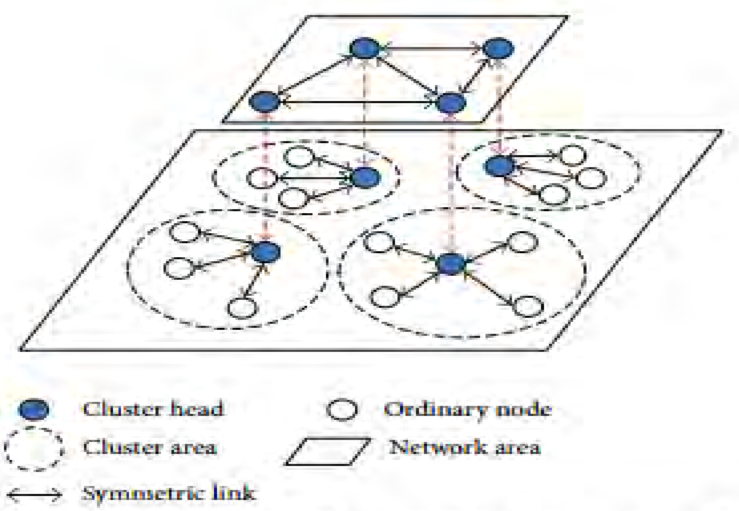


Figure 3: Model of the two-tier system model

CHAPTER 2

2.1 Binary Consensus with Updating Protocols

When we try to reach a consensus decision based on a network. Every nodes of the network can hold initially one of the two values: zero and one. When two nodes communicate and run the updating protocols, they compare their current state and then each assume a new state based on what they have seen. When binary consensus algorithm is running a node may be in one of the four states which can be described informally as:

1] 0 - The node believes the majority opinion is most likely false.

2] 1 - The node believes the majority opinion is most likely true.

3] e0 - The node believes the majority opinion might be false.

4] e1 - The node believes the majority opinion might be true.

Convergence occurs when all nodes have states ∈ $\{0,\mathrm{e}0\}$ or $\{1,\mathrm{e}1\}$

We know for binary consensus every node will have value either 1 or 0. So if we want to reach consensus we have to update through node to node communication. For updating we have to follow updating protocols. The protocols are given below with an example:

Those Protocols are from [18].

1] (0,e0) -> (e0,0)

2] (0,e1) -> (eo,0)

3] (0,1) -> (e1,e0)

4] (e0,e1) -> (e1,e0)

5] (e0,1) -> (1,e1)

6] (e1,1) -> (1,e1)

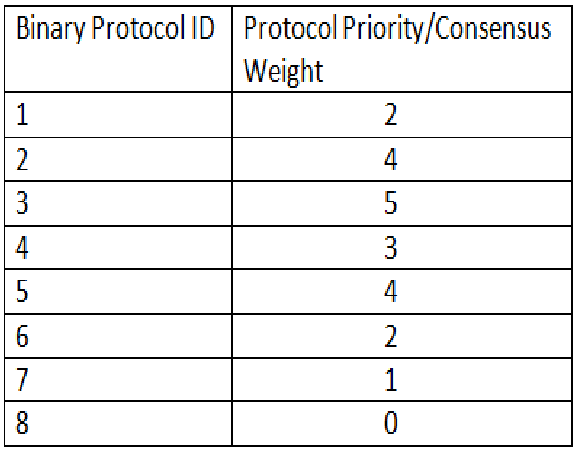
7] (s,s) -> (s,s) , for s = 0,1,e0,e1

8] (s,F) -> (s,F) here F indicates faulty node

We must give priority to these protocols. Priority based on highest number. Figure 4 describes Protocols information as a table based. Let's describe this situation in Figure 5. First B and C stats run updating protocol the states will become e0 and e1. Then A and B starts run updating protocol; their states will become e1 and 1. Now consensus has reached because all the states $\in \{1,e1\}$.

Actually in Figure 5 we didn't consider any cluster head or faulty node we just tried to show how updating protocols are used for binary consensus.

Note: Protocol number 8 is proposed by ourselves. F only defines faulty, it can have above 4 states but when any node is define faulty it will must follow protocol number 8.



Binary Protocol ID	Protocol Priority/Consensus Weight
1	2
2	4
3	5
4	3
5	4
6	2
7	1
8	0

Figure 4: Binary Protocol Informations.

2.2 Average Consensus with Updating Protocols

Assume that n static sensor nodes are independently deployed in a unit square area and that the network topology is represented as $G = (V, r, E)$, where $V = \{1, 2, 3, . . . , n\}$ represents the set of nodes and r is the connectivity radius. A pair of nodes $(i, j)$is connected and can directly

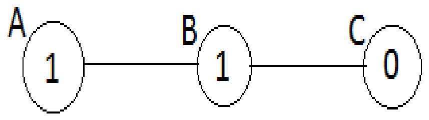


Figure 5: Sensor nodes

communicate with each other if their Euclidean distance is smaller than r. The edge set is saved in E and the set of node's neighbors in one hop is denoted by $(i) = \{j \in V; (i, j) \in E\}$. The degree of this node, which is equal to its number of neighbors, can be defined as $di = |(i)|$ [20]. Each node i in the network has an initial value (0), representing an observation of some type. The initial value vector of all these nodes can be defined as (0) = [x1(0), x2(0), . . . , xn(0)]. In this paper, we deal with the average consensus which means that the consensus equilibrium value is equal to the average value of the initial value held by each node. It has been reported that the average consensus is reached for the case in which the communication topology is fixed and connected [21]. A connected network is one in which a path exists between every pair of nodes [20]. The average of these values is x = (1/n) ∑n i=1 (0) [21]. At kth iteration, each node i maintains an estimation (k) that is generally different from that of other nodes. A vector (k) = [x1(k), x2(k), . . . , xn(k)] is used to define the values of all the nodes. Suppose the network is connected and the communication relationship is symmetric; that is, node i and node j can receive the information from each other correctly based on the wireless link between them for a given time slot. The ultimate goal of consensus is to drive the estimated vector value (k) infinitely close to the average vector X = [x, x, . . . , x]with a minimal amount of information exchange. To match the distributed nature of WSN, an asynchronous time model is adopted by the average consensus algorithm to trigger the node wake up and execute the average consensus algorithm. The clock in each node is assumed to have a tick rate based on the Poisson process. During the average algorithm updating works according to the following equation:

$$xi\ (k) = xj\ (k) = [xi\ (k-1) + xj\ (k-1)]\ /2 \qquad .... \qquad (1)$$

The metric proposed in [20] is used to evaluate the convergence rate of reaching consensus. This metric defines the normalization of difference between consensus value and the real average value.

CHAPTER 3

3.1 Energy and Faulty Nodes

But in wireless sensor networks energy consumption is a big issue. We know that every sensor node have some energy to hold. By processing those energy is reduced and when any sensor node energy become zero then that sensor node become a faulty node. We apply our consensus algorithms on the present of faulty nodes also. When a sensor node becomes a faulty node it can’t participate to reach a consensus decision.

3.2 Definition of Edge and Creating Weighted Graph

In our main graph which edge have many characteristics. That particular edge hold many values which are Binary states, Average states, Binary protocols weight/priority, Average weight, two end pointer energy, distance from peer to peer nodes, unique id, node id of two end pointer. So we should make another weighted from Figure 2 graph but here edge represents not only physical distances but also above described characteristics. Here actually Figure 9 and 12

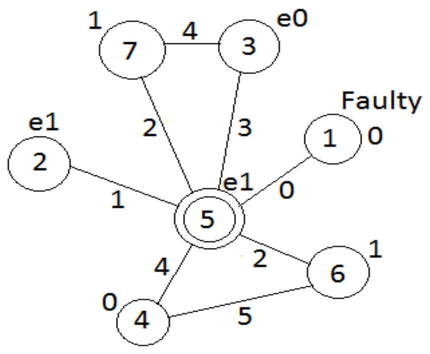


Figure 9: Network Model in Graph Representation for Binary Consensus.

are same graphs. To make it clear we drawn two graph to view the edges for binary and average consensus individually. We collected the edge information from Figure 6 sensor node information and link information from Figure 2.Note: Figure 6 is a sample information.

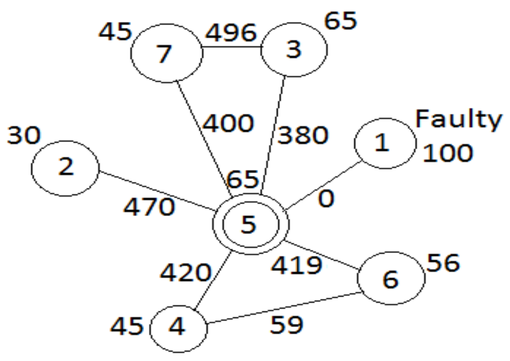


Figure 12:Network Model in Graph Representation for Average Consensus.

Sensor Node Id	Binary States	Average States	Energy
1	Faulty	Faulty	0
2	e1	30	250
3	e0	65	200
4	0	45	120
5	e1	65	500
6	1	56	129
7	1	45	300

Figure 6: Sensor Node Informations

3.3 Creating Edge Array and Edge Graph

Now we have to make an array which stores edges with its information. In edge graph we have neighbor edges [hashed unique id] to every node. Figure 7 and 8 defines edge array and edge graph [table based] of new weighted graph. Algorithm 1 is based on creating Edge graph.

Edge-Array Index	Edge
1	4 •——• 6
2	5•——• 3
3	5•——• 4
4	3 •——• 7
5	5•——• 2
6	7•——• 5
7	6•——• 5
8	5 •——• 1

Figure 7: Edge Array

Sensor Node ID	Neighbor EDGE
1	[1,5]
2	[2,5]
3	[3,5][3,7]
4	[4,5][4,6]
5	[5,1][5,2][5,3][5,4] [5,6][5,7]
6	[6,5][6,4]
7	[7,3][7,5]

Figure 8: Edge Graph in Table

Algorithm 1: Creating Edge Array

```
Model the network graph(G)

Initialize Edge Array(EA)

Initialize Edge-ID Array(Edge-ID)

id = 1

for Each edge u,v ∈ G

    EA[id]=EDGE(u,v,id,BstateU,BstateV,AstateU,AstateV,Bweight,Aweight)

    Edge-ID[Hashed-Value(u,v)] = id

    id = id + 1

end for
```

Algorithm 2: Creating Edge Graph

```
Model the network graph(G)

Initialize edge graph(EG)

for Each node u ∈ G

    for Each node v ∈ G.adj[u]

        EG[u].add(Edge-ID[Hashed-Value(u,v)])

    end for

end for
```

CHAPTER 4

4.1 Segment Tree

In Computer Science, a segment tree is a tree data structure for storing intervals, or segments. It allows querying which of the stored segments contain a given point. It is, in principle, a static structure: that is, its structure cannot be modified once it is built. [Wikipedia]

The node of a segment tree takes a given priority based value from a specific range. It’s almost all methods generally use recursive process. Segment tree has a single root/top node and every node of a segment tree except leaves has 2 children which are left and right. Left child’s id is TreeNodeID*2 and right child’s id is TreeNodeID*2+1. As segment tree generally works in recursive process so every node except leaves go to its both left and right child recursively. The base case of recursive process is stopped when calling reaches leaf nodes. Generally a segment tree has 3 methods which are build, query and update. The depth of a segment tree is log(number of nodes). The general 3 methods of a segment tree described below. To describe below part we considered the segment tree of Figure 10 and set priority as taking MaxValue and n defines number of nodes.

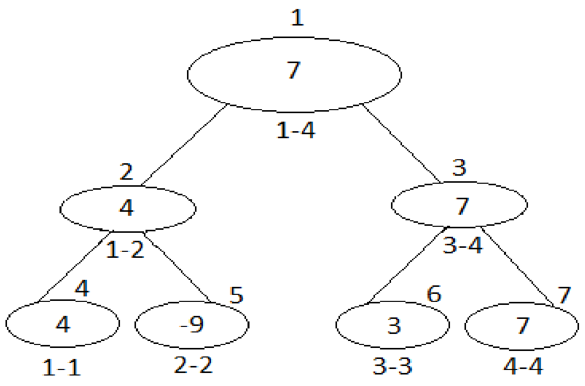


Figure 10: A Segment Tree

- Build: We can build a segment tree from the given set of elements in an array. For example the given array is [4, -9, 3, 7]. The segment tree for that array is given in Figure 10. While building it will start from top node which starts from 1 and it takes the max value of the ranges among 1 to 7. It's actually top-down processing. So from top node 1 to leave nodes 4, 5, 6, 7 it goes when it's reached leaves node it's save the corresponding array value. For example leave node 7 which take the maximum value of ranges [4-4] saves array value of 4 number index which is 7. Then it goes the parent node by backtracking as generally segment tree processes are recursive. Besides segment tree every node parent can be found by divide its id by 2 except the root node as root node has no parent exist. When a node is not a leaf node it considers its both left and right child value which also can got from backtracking. So from its left and rightchild's value it takes the maximum among them as both left and right child value are computed when we ready to save value to that node. For example: Tree node id 1 will save value 7 because its left child has 4 and right child has 7 so it takes the maximum. That's how Segment tree build process works. The complexity of build process is O(nlog(n)) as we updated every leaf nodes individually.

- Query: Its query process is just like searching from it. For example: What is the max value from array index 2 to 4? The process is we start from root node if we see a nodes ranges is set under required query then we will go to its left and right child node. For example: When we are in root node its range is [1 – 4] and required range is [2 – 4] that means required ranges set under root node. We should divide the root node to left and right child. Then we can see that the right child total range sets under query range so instead of divide it by left and right child we should take that node value which is 7. On the other hand root node 1 left child take ranges [1-2] which set under query range but not totally so we again devide it by left and right child which are node id 4[1-1] and 5[2-2] where node id 5 sets under totally to the queried range so we will take its value which is 4 but node id 4[1-1] is out of the queried range so we need to just ignore that node. At last the required answer is max value of 7 and 4 which is 7. Time Complexity is O(log(n)).

- Update: Its update operating is almost like its query operation. After searching the required range its update that leaf node because we assume given range for update operation is like [i-i] so we will got to that leaf node by recursively and update it from there. For example: Lets update 3 number index of that array and change it to 10. So by recursively when we will reach to the leaf node of 6[3-3] we will change it to 10. By backtracking we will reach tree node id 3 now the max value will be 10 instead of 7. At last we will reach to the root node max value will be 10 instead of 7. That's how segment treeupdate process works. Besides if the given range of update operation is [i-j] where i != j then another process can be applied which we call lazy propagation. In our problem its node needed. So we just skipped it. Time complexity is O(log(n)).

Note: In our problem for update process we implemented faster bottom-up process instead of recursive top-down process.

4.2 Edge Energy Weight: (Binary and Average Consensus)

A = Energy[u] – distance

B = Energy[v] - distance

Edge(u,v)weight = A + B if A > 0 and B > 0

Otherwise

Edge(u,v)weight = 0 which means u or v or both don't have enough energy to communicate each other.

4.3 Build Segment Tree for the Edge Array (Binary Consensus)

Now we have to make a segment tree for the whole part of the edge array. Here in every node of the segment tree can take 4 elements which are main-Edge, dummy-Edge, weight and size. Here dummy edge used just for make sure this node has more than single edge and size will be at range 1 to 2 if a node of a segment tree has null in dummy edge then size will be 1 otherwise 2. To build a segment tree as it goes from top to bottom so when we reach the leaf nodes of the segment tree we mapped the leaf node number to its corresponding edge to STreeNode-Array[have to do it for future purposes] and while considering the left child and right child for the upper nodes in segment tree we will choose the highest priority based edge as main-Edge and dummy-Edge will be NULL but here if highest priority based edge exist more than one and all of them follow protocol id [1 and 6 both] then we take two edge one edge saved in main-Edge another edge saved in dummy-Edge. Besides if all of their weights follow same protocol [1 or 6 not both] then we will save any single edge to tree nodes main-Edge and dummy-Edge will be NULL.

Figure 11 is the build segment tree for the Figure 7 edge array and Algorithm 4 is based on Build segment tree for the edge array for Binary Consensus. Figure 9 graph is actually for Binary Consensus.

Note: Here we take highest priority edge based on binary updating protocols.

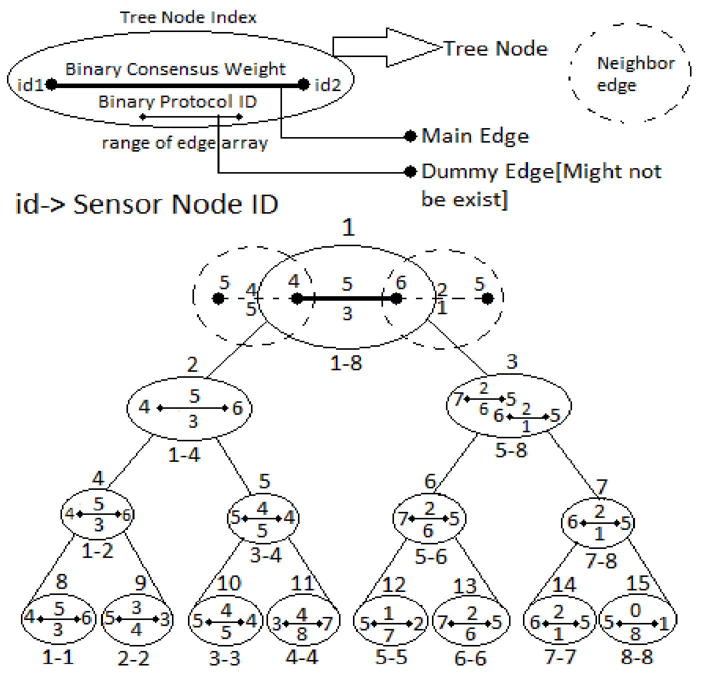


Figure 11: Segment Tree for Binary Consensus

4.4 Binary Consensus Algorithm

This algorithm runs for edges and implemented in Segment tree data structure. For every time we will consider the top node from the segment tree and as initially we have highest priority edges based on binary updating protocols in the top node. If the top node has many edges we will take any node from it and change it two endpoints value and update its priority according to the updating protocols. Here we might have to change many edges priority and their end points value according to updating protocols. Because if the edges end points connected to another edges then we have to change their information also. For example see the Figure 11[top node id 1]. When we will work on edge [4-6] we should make change edge [4-5] and [6-5] information's also. While updating we have to make change their information inside the segment tree by the help of Edge graph and as we track down in STreeNode-array the leaf nodes index of the segment tree so we can directly go to that node and update it from there and by doing bottom up processing we make changes until we reach to the root. While doing bottom up processing we will do the same thing what we did for building the segment tree. When we can see that there is only one edge [mainEdge] in the top node of the segment tree and its priority is 3rd lowest priority which is 3[updating protocol number 1 and 6] so that means the other nodes in the segment tree are as same as top node or other nodes contain faulty nodes or having 2nd lowest or 1st lowest priority edges so we don't need to consider those nodes in this time. But if segment trees topNode contains faulty nodes of the network then as it follows lowest priority based updating protocols so consensus will never occur we can claim that easily. Figure 6 is the Segment tree for Binary Consensus Algorithm.

Note: While updating we must also minimizes energy from that edge two pointer node. Besides Binary consensus has zero percent error rate.

Algorithm 3: Tracking Highest Priority Node (Binary Consensus)

```
HIGHEST(SNode l, SNode r)

Initialize a SNode (Node) which keep a mainEdge, dummyEdge and weight of them which is same.

if(r.Bweight == l.Bweight) then

    if(r and l main.BPId 1 or 6 and different than each other) then

        Node.mainEdge = HighEnergyRemain(r,l)
        Node.dummyEdge = LowEnergyRemain(r,l)
    end if
else if(r.Bweight>l.Bweight) then
        Node = r
else
        Node = l
end if
return Node
```

Algorithm 4: Building Segment Tree (Binary Consensus)

```
Initialize Segment Tree(STree)
BUILD-TREE(node-id,i,j)
if(i == j) then
     STree[i].mainEdge = E[i]
     STree[i].dummyEdge = NULL
     STreeNode[i] = node-id
end if
Left = node * 2
Right = Left + 1
Mid = (i+j)/2
BUILD-TREE (Left,i,Mid)
BUILD-TREE (Right,Mid+1,j)
STree[node-id] = HIGHEST(STree[Left],STree[Right])
```

Algorithm 5: Updating Segment Tree (Binary Consensus)

```
UPDATE(node-id)
i = node-id
while((i = i >> 1) != 0)
    Left = i * 2
    Right = left + 1
    STree[i] = HIGHEST(STree[Left],STree[Right])
end while
```

Algorithm 6: Binary Consensus Algorithm

```
END (id)
if(id == 1 or id == 6) return true
  return false
end if
Binary Consensus Algorithm ()
Energy of all sensor nodes stores in Energy Array(Energy)
BUILD-TREE(1,1,edgeSize)
topnode = 1
  while(true)
    edge = STree[topnode].mainEdge
    if(STree[topnode].dummyEdge == NULL and END(STree[topnode].BPId)) then
```

```
      Binary Consensus Reached….. break
    else if(edge.Bweight == 1) then
        Binary Consensus Reached….. break
    else-if(edge.Bweight == 0) then
        Binary Consensus Will Never Reach…. break
    else
        EnergyU =   Energy[edge.u]-distance(edge.u,edge.v)
        EnergyV =   Energy[edge.v]-distance(edge.u,edge.v)
        if(EnergyU<0 or EnergyV<0) then
            Make STree[STreeNode[edge.id]] //as a faulty edge.
            UPDATE(STreeNode[edge.id]) // follow binary updating protocol.
        else
            Energy[edge.u] = EnergyU
            Energy[edge.v] = EnergyV
            UPDATE(STreeNode[edge.id]) // follow binary updating protocol.
      for Each edge e ∈ EG.Adj[edge.u]
          if(e.u == edge.u and e.v != edge.v) then
               STree[e.id].mainEdge.BstateU = edge.BstateU
               //Change STree[STreeNode[e.id]] information of edge-weight and protocol id
               UPDATE(STreeNode[e.id])
          else if(e.v == edge.u and e.u != edge.v) then
                STree[e.id].mainEdge.BstateV = edge.BstateU
                //Change STree[STreeNode[e.id]] information of edge-weight and protocol id
                UPDATE(STreeNode[e.id])
            end if
        end for
        for Each edge e ∈ EG.Adj[edge.v]
            if(e.u == edge.v and e.v != edge.u) then
                STree[e.id].mainEdge.BstateU = edge.BstateV
                //Change STree[STreeNode[e.id]] information of edge-weight and protocol id
                UPDATE(STreeNode[e.id])
            else if(e.v == edge.v and e.u != edge.u) then
                 STree[e.id].mainEdge.BstateV = edge.BstateV
                 //Change STree[STreeNode[e.id]] information of edge-weight and protocol id
                 UPDATE(STreeNode[e.id])
            end if
        end for
    end if
end while
```

4.5 Build Segment Tree for the Edge Array: (Average Consensus)

Now we have to make another segment tree for the edge array. Here in every node of the segment tree can take only single elements which we name treeEdge. To build a segment tree as it goes from top to bottom so when we reach the leaf nodes of the segment tree we mapped the leaf node number to its corresponding edge to STree2Node-Array[have to do it for future purposes] and while considering the left child and right child for the upper nodes in segment tree we will choose the highest weight based edge for average consensus. For average consensus we define edge for the below formula:

Aweight = absolute((State[u]+State[v]/2)-x(bar))

Figure 13 is the build segment tree for the Figure 7 edge array and Algorithm 8 is based on Build segment tree for the edge array for Average Consensus. Figure 12 graph is actually for Average Consensus.

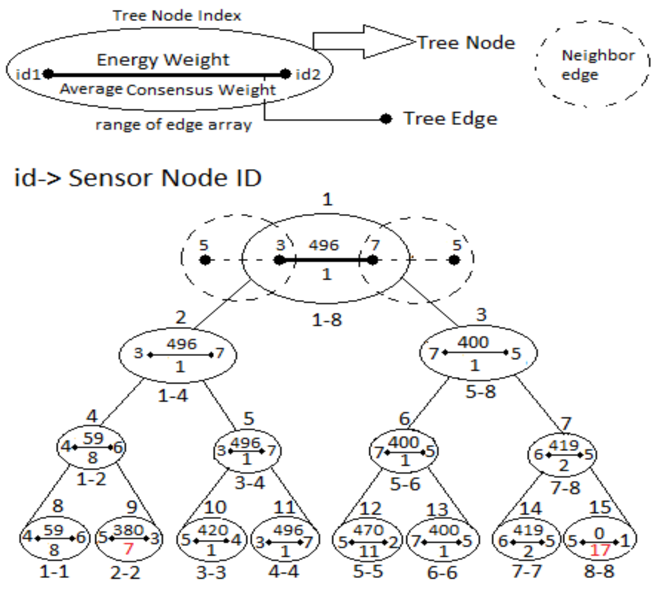


Figure 13: Segment Tree for Average Consensus

4.6 Average Consensus Algorithm

This algorithm runs for edges and implemented in another Segment tree data structure. For every time we will consider the top node from the segment tree and as initially we have highest max weight edges based average consensus weight in the top node. If the top node average consensus weight is zero that means one of two end pointer sensor node or both sensor nodes don't have enough energy to communicate that defines one of them or both are faulty nodes. So if top node average consensus weight is not zero we will continue our average consensus algorithm. Here we might have to change many edges average consensus weight and their end points value according to updating protocols of average consensus. Because if the edges end points connected to another edges then we have to change their information also. For example see the Figure 13[top node id 1]. When we will work on edge [3-7] we should make change edge [3-5] and [7-5] information's also. While updating we have to make change their information inside the segment tree by the help of Edge graph and as we track down in STree2Node-array the leaf nodes index of the segment tree so we can directly go tothat node and update it from there and by doing bottom up processing we make changes until we reach to the root. While doing bottom up processing we will do the same thing what we did for building the segment tree. When we can see that there top node edges contain faulty node then we will stop running our algorithm. Because we are taking maximum edge average consensus weight and faulty node based edges weight is zero so all the others tree nodes in the segment tree contains only faulty sensor nodes. Figure 13 is the Segment tree for Average Consensus Algorithm. Algorithm 10 is based on Average Consensus algorithm.

Note: While updating we must also minimizes energy from that edge two pointer node. There is huge possibility Average Consensus might have error rate. The formula for calculating error rate is given below:

$$\mathrm{err} = \frac{\|x(k) - \overline{x}\|}{\|x(0) - \overline{x}\|}. \quad (8)$$

Algorithm 7: Tracking Highest Priority Node (Average Consensus)

```
HIGHEST2(S2Node l, S2Node r)
Initialize a S2Node (Node) which keep a treeNode
if(l.Eweight == 0) then Node = r
else if(r.Eweight == 0) then Node = l
else if(l.AstateU == l.AstateV) then Node = r
else if((r.AstateU == r.AstateV) or (l.Aweight < r.Aweight)) then Node = l
else if(r.Aweight == l.Aweight) then take one of them which has max Energy Weight
else Node = r
end if
return Node
```

Algorithm 8: Building Segment Tree (Average Consensus)

```
Initialize Segment Tree(S2Tree)
BUILD-TREE2(node-id,i,j)
if(i == j) then
    S2Tree[i].treeEdge = E[i]
    S2TreeNode[i] = node-id
end if
Left = node * 2
Right = Left + 1
Mid = (i+j)/2
BUILD-TREE2(Left,i,Mid)
BUILD-TREE2(Right,Mid+1,j)
S2Tree[node-id] = HIGHEST2(S2Tree[Left],S2Tree[Right])
```

Algorithm 9: Updating Segment Tree (Average Consensus)

```
UPDATE2(node-id)
i = node-id
while ((i = i >> 1) != 0)
    Left = i * 2
    Right = left + 1
    S2Tree[i] = HIGHEST2(S2Tree[Left],S2Tree[Right])
end while
```

Algorithm 10: Average Consensus Algorithm

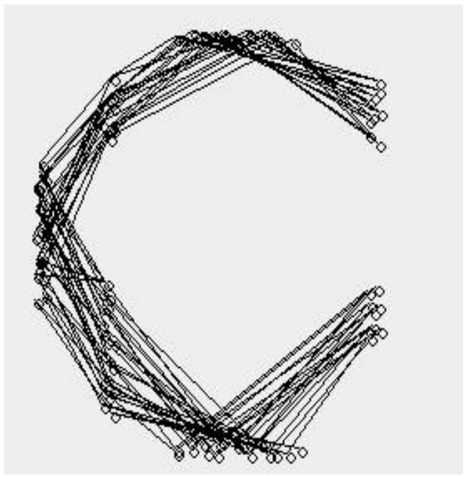
```
Average Consensus Algorithm ()
Energy of all sensor nodes stores in Energy Array(Energy)
BUILD-TREE2(1,1,edgeSize)
topnode = 1
while(true)
    edge = S2Tree[topnode].mainEdge
    if(S2Tree[topnode].treeEdge.AstateU == S2Tree[topnode].treeEdge.AstateV) then
        Highest Possible Average Consensus Occurred… break
    else-if(edge.Aweight == 0) then
        Average Consensus Reached…. break
    else
        EnergyU =   Energy[edge.u]-distance(edge.u,edge.v)
        EnergyV =   Energy[edge.v]-distance(edge.u,edge.v)
        if(EnergyU<0 or EnergyV<0) then
            Make S2Tree[S2TreeNode[edge.id]] //as a faulty edge.
            UPDATE2(S2TreeNode[edge.id]) // do nothing just run update…
        else
            Energy[edge.u] = EnergyU
            Energy[edge.v] = EnergyV
            UPDATE2(S2TreeNode[edge.id]) // follow average updating protocol.
            for Each edge e ∈ EG.Adj[edge.u]
                if(e.u == edge.u and e.v != edge.v) then
                    S2Tree[e.id].treeEdge.BstateU = edge.BstateU
                    //Change S2Tree[S2TreeNode[e.id]] information of edge weight
                    UPDATE2(S2TreeNode[e.id])
                else if(e.v == edge.u and e.u != edge.v) then
                    S2Tree[e.id].treeEdge.BstateV = edge.BstateU
                    //Change S2Tree[S2TreeNode[e.id]] information of edge-weight
                    UPDATE2(S2TreeNode[e.id])
                end if
            end for
            for Each edge e ∈ EG.Adj[edge.v]
                if(e.u == edge.v and e.v != edge.u) then
                    S2Tree[e.id].treeEdge.BstateU = edge.BstateV
                    //Change S2Tree[S2TreeNode[e.id]] information of edge-weight
                    UPDATE2(S2TreeNode[e.id])
                else if(e.v == edge.v and e.u != edge.u) then
                    S2Tree[e.id].treeEdge.BstateV = edge.BstateV
                    //Change S2Tree[S2TreeNode[e.id]] information of edge-weight
                    UPDATE2(S2TreeNode[e.id])
                end if
            end for
      end if
    end while
```

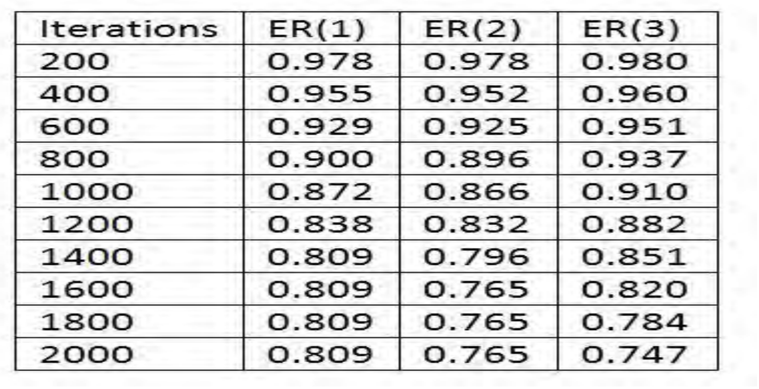
Chapter 5

5. Result and Analysis:

We have implemented our algorithm in C, D, O, I, H topologies and random topologies. Those are given below:

5.1 C Topology





Iterations	ER(1)	ER(2)	ER(3)
200	0.978	0.978	0.980
400	0.955	0.952	0.960
600	0.929	0.925	0.951
800	0.900	0.896	0.937
1000	0.872	0.866	0.910
1200	0.838	0.832	0.882
1400	0.809	0.796	0.851
1600	0.809	0.765	0.820
1800	0.809	0.765	0.784
2000	0.809	0.765	0.747

Table No.1 Iteration vs Relative Error(C topology)

For ER(1,2,3) Minimum Relative Error Rate (0.809,0.765,0.747)

For ER(1) Binary Consensus Reaching Iteration 156 and Average Consensus Reaching Iterations 1368

For ER(2) Binary Consensus Reaching Iteration 191 and Average Consensus Reaching Iterations 1573

For ER(3) Binary Consensus Reaching Iteration 162 and Average Consensus Reaching Iterations 1811

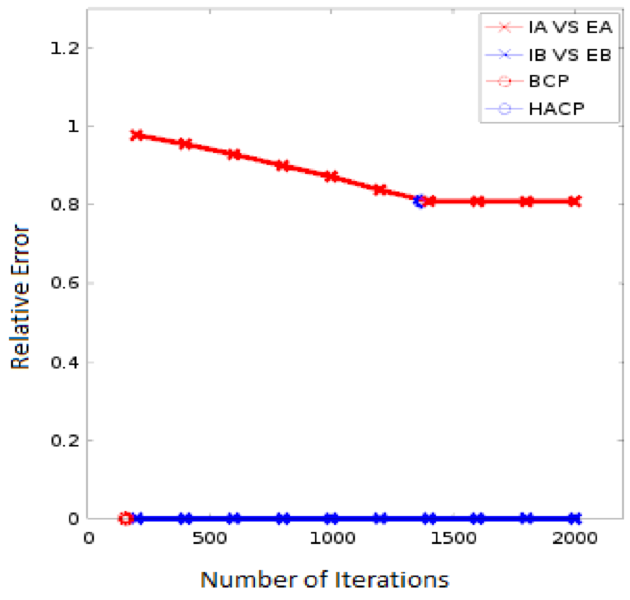
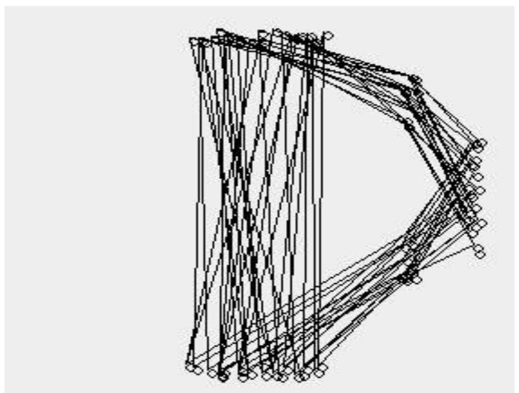
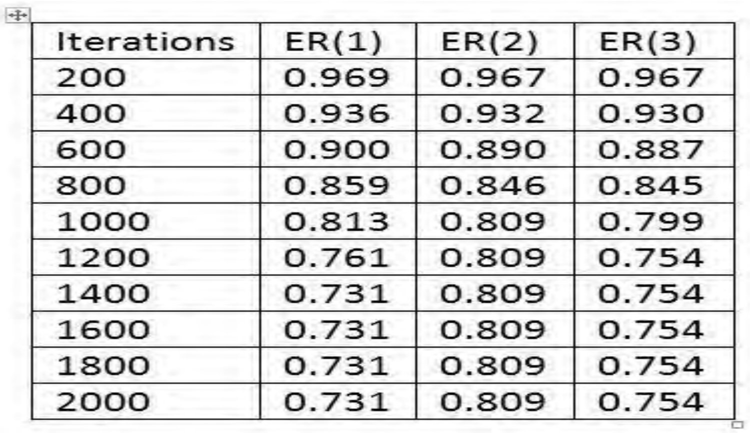


Figure 14: Iteration vs Relative Error(C topology)

5.2 D Topology





Iterations	ER(1)	ER(2)	ER(3)
200	0.969	0.967	0.967
400	0.936	0.932	0.930
600	0.900	0.890	0.887
800	0.859	0.846	0.845
1000	0.813	0.809	0.799
1200	0.761	0.809	0.754
1400	0.731	0.809	0.754
1600	0.731	0.809	0.754
1800	0.731	0.809	0.754
2000	0.731	0.809	0.754

Table 2: Iteration vs Relative Error(C topology)

For ER(1,2,3) Minimum Relative Error Rate (0.731,0.809,0.754)

For ER(1) Binary Consensus Reaching Iteration 788 and Average Consensus Reaching Iterations 1315

For ER(2) Binary Consensus Reaching Iteration 436 and Average Consensus Reaching Iterations 953

For ER(3) Binary Consensus Reaching Iteration 307 and Average Consensus Reaching Iterations 1192

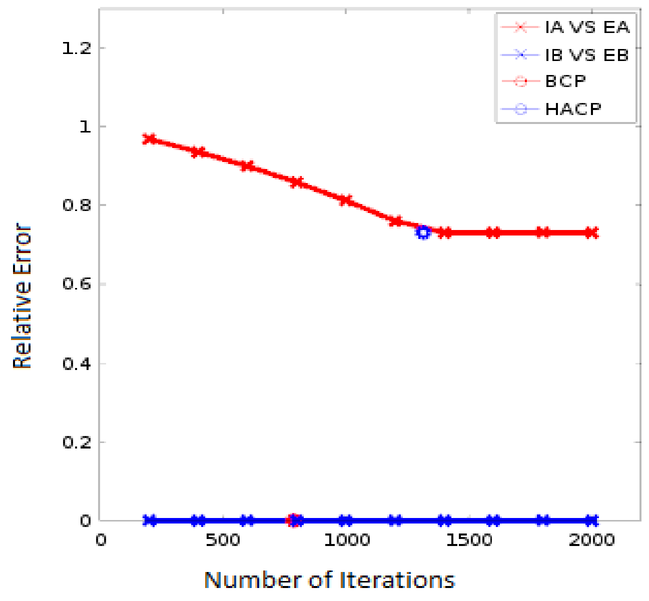
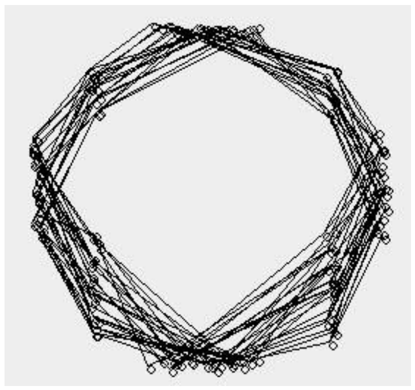


Figure 15: Iteration vs Relative Error(D topology)

5.3 O topology



Iterations	ER(1)	ER(2)	ER(3)
200	0.985	0.985	0.984
400	0.969	0.967	0.968
600	0.951	0.948	0.949
800	0.932	0.929	0.928
1000	0.910	0.908	0.906
1200	0.887	0.885	0.881
1400	0.864	0.860	0.856
1600	0.840	0.837	0.836
1800	0.814	0.827	0.836
2000	0.793	0.827	0.836

Table 3: Iteration vs error rate(O topology)

For ER(1,2,3) Minimum Relative Error Rate (0.793,0.827,0.836)

For ER(1) Binary Consensus Reaching Iteration 243 and Average Consensus Reaching Iterations 1970

For ER(2) Binary Consensus Reaching Iteration 244 and Average Consensus Reaching Iterations 1681

For ER(3) Binary Consensus Reaching Iteration 213 and Average Consensus Reaching Iterations 1546

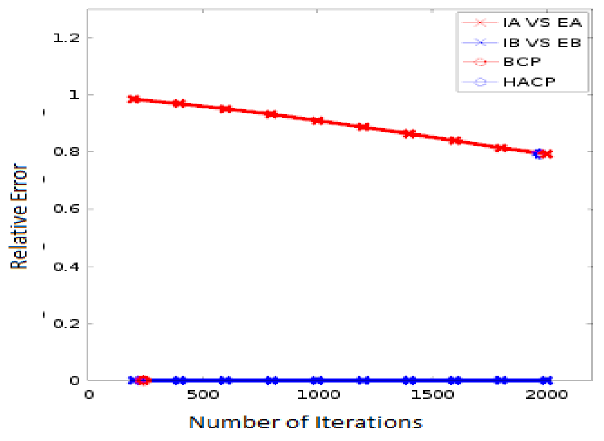
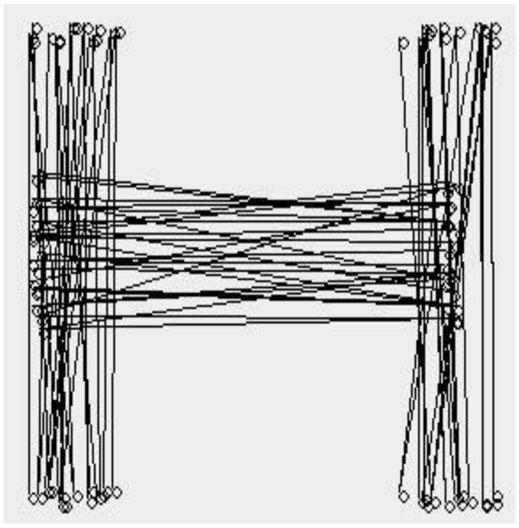
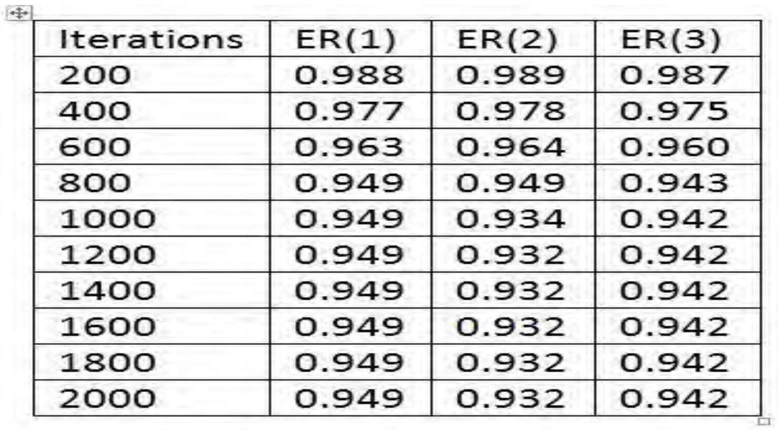


Figure 16: Iteration vs Error Rate(O topology)

5.4 H topology





Iterations	ER(1)	ER(2)	ER(3)
200	0.988	0.989	0.987
400	0.977	0.978	0.975
600	0.963	0.964	0.960
800	0.949	0.949	0.943
1000	0.949	0.934	0.942
1200	0.949	0.932	0.942
1400	0.949	0.932	0.942
1600	0.949	0.932	0.942
1800	0.949	0.932	0.942
2000	0.949	0.932	0.942

Table 4: Iteration vs error rate(H topology)

For ER(1,2,3) Minimum Relative Error Rate (0.949,0.932,0.942)

For ER(1) Binary Consensus Reaching Iteration 79 and Average Consensus Reaching Iterations 387

For ER(2) Binary Consensus Reaching Iteration 119 and Average Consensus Reaching Iterations 507

For ER(3) Binary Consensus Reaching Iteration 84 and Average Consensus Reaching Iterations 401

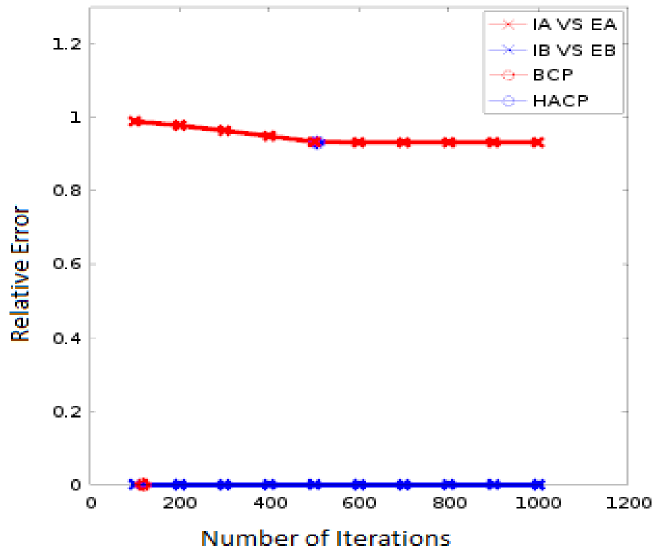
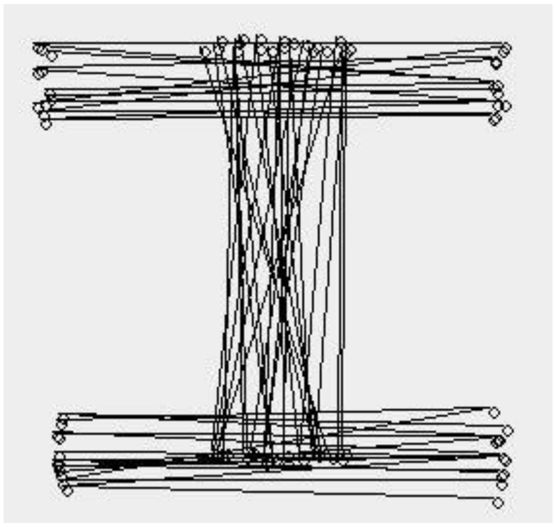
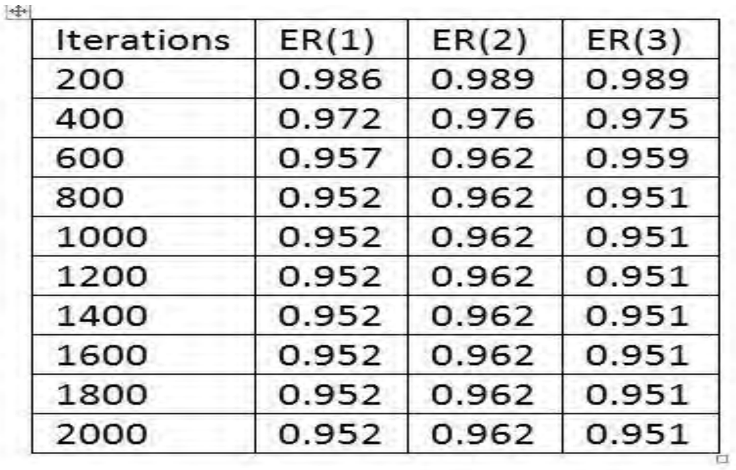


Figure 17: Iteration vs Error Rate (H topology)

5.5 I Topology





Iterations	ER(1)	ER(2)	ER(3)
200	0.986	0.989	0.989
400	0.972	0.976	0.975
600	0.957	0.962	0.959
800	0.952	0.962	0.951
1000	0.952	0.962	0.951
1200	0.952	0.962	0.951
1400	0.952	0.962	0.951
1600	0.952	0.962	0.951
1800	0.952	0.962	0.951
2000	0.952	0.962	0.951

Table 5: Iteration vs error rate(I topology)

For ER(1,2,3) Minimum Relative Error Rate (0.952,0.962,0.951)

For ER(1) Binary Consensus Reaching Iteration 120 and Average Consensus Reaching Iterations 325

For ER(2) Binary Consensus Reaching Iteration 51 and Average Consensus Reaching Iterations 267

For ER(3) Binary Consensus Reaching Iteration 68 and Average Consensus Reaching Iterations 345

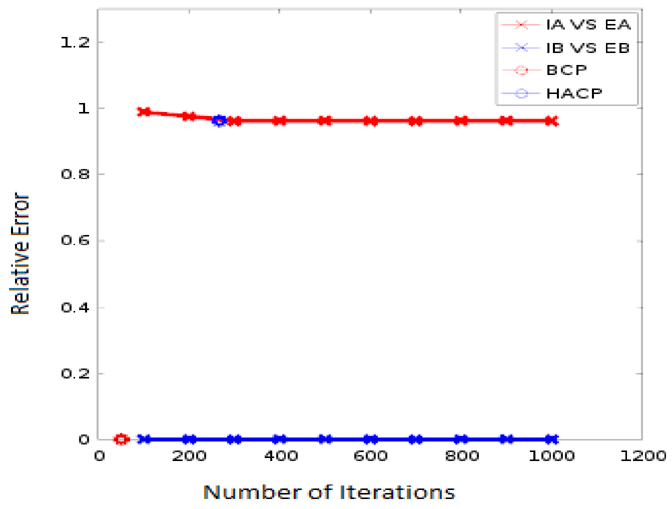


Figure 18: Iteration vs Error Rate (I topology)

Analysis for topologies(C,D,O,H,I)

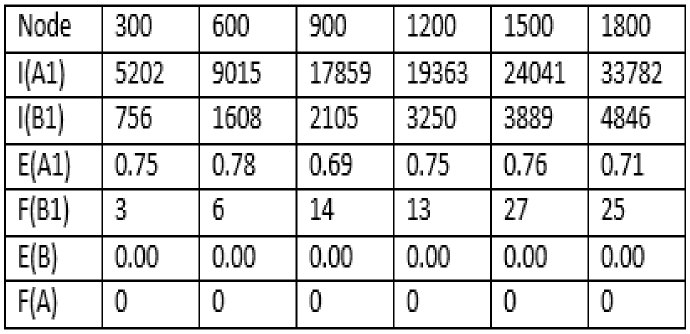
Here we considers 5 shaped network models which are C, D, I, H, O. For all those 5 shaped network models we calculated Relative error based on number of iterations. We took 3 data three times to clearly see the results variations. For C, D, I shaped data we took iterations from 200 to 2000 but for shape H and O we took iterations from 100 to 1000 the reason is we saw that the highest average based consensus reaching time is faster than shaped C,D,I. Here for average consensus we consider only highest average consensus.

From the figure we only consider 3 cases:

1) Number of iterations VS Relative Error
2) Highest Average Consensus reaching point VS Relative Error(Single point)
3) Binary Consensus Reaching point VS Relative Error(Always zero)(Single Point)

Here, IA denotes iterations for average consensus, IB denotes Iterative for binary consensus, EA denotes Relative error for average consensus, EB denotes Relative error for Binary consensus and HACP denotes highest average consensus reaching point. BCP denotes Binary consensus reaching point.

5.6 Random Topology



Node	300	600	900	1200	1500	1800
I(A1)	5202	9015	17859	19363	24041	33782
I(B1)	756	1608	2105	3250	3889	4846
E(A1)	0.75	0.78	0.69	0.75	0.76	0.71
F(B1)	3	6	14	13	27	25
E(B)	0.00	0.00	0.00	0.00	0.00	0.00
F(A)	0	0	0	0	0	0

Table 6: Node vs Iteration vs error rate vs Faulty Node(Random topology)

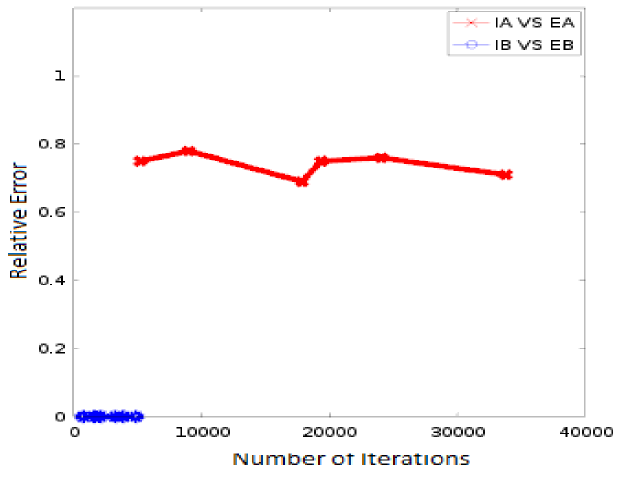


Figure 19: Iteration vs Error Rate

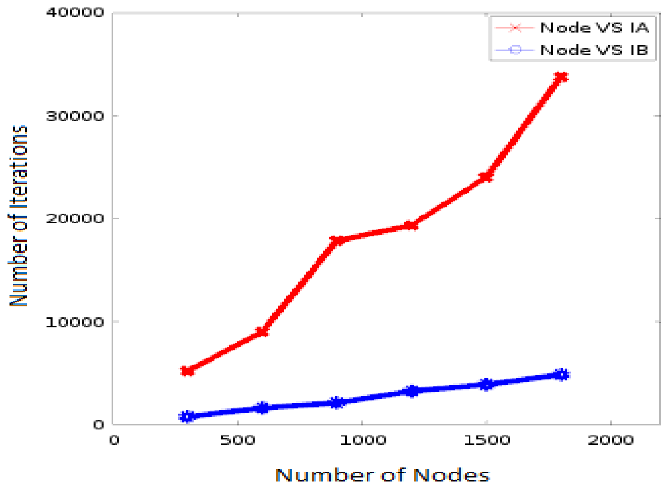


Figure 20: Nodes vs Iteration

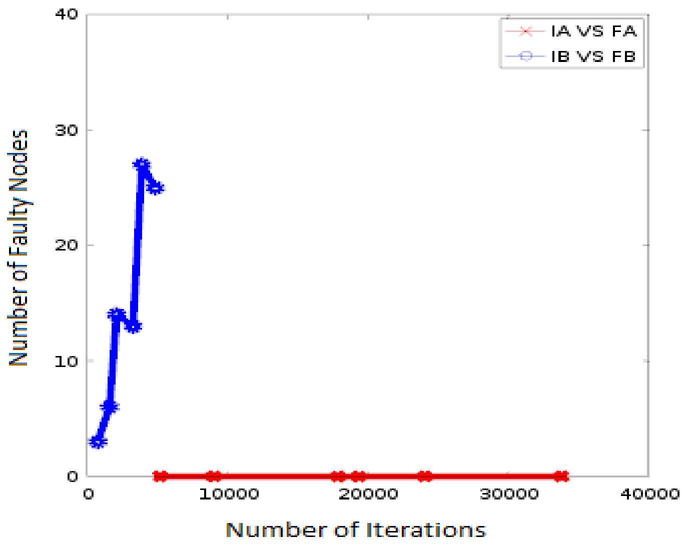


Figure 21: Iteration vs Faulty Nodes

Analysis for Random Topologies

After creating random graph we got those data. Here

I denote to Iteration, E denotes to Error, F denotes to faulty node, A for Average Consensus, B for Binary Consensus. For example, I(B1) means 1st time data of Iteration for Binary Consensus reaching time. Here is very interesting thing we have noticed error rate for binary consensus is always zero which is obvious but the number of faulty node for average consensus is always zero too which is not obvious at all. The reason is actually we proposed average consensus updating protocol which protocol always give priority to the sensor node remaining energy and their distance among them and as we define faulty node when a sensor node has zero energy remain so those edges based on low differences of energy and distance have the lowest priority based edges so from our tree we already reached consensus before handling them. So generally we have zero faulty nodes for average consensus. But in extreme rare cases there might be few faulty nodes for average consensus.

From the diagram we consider 3 cases:

1) No. of Iteration vs Relative Error
2) No. of Iterations vs Number of Faulty Nodes
3) No. of Nodes vs Number of Iterations.

IA means Iteration for Average Consensus, IB means iteration for Binary consensus

EA means Relative Error for Average Consensus, IB means Relative Error for Binary consensus

FA means Number of faulty nodes for Average consensus, FB number of faulty nodes for Binary consensus.

In the case of average consensus we know that there is no exact time for reaching consensus. We can reach at every step at consensus level and of course error rate is different but here we claimed the highest average consensus based on the given network condition in case 3(Number of Nodes VS Number of Iterations). That means minimum error rate we can reach for the given network condition. So here we calculated highest average consensus. Besides, for binary consensus we know that we can only reach consensus level when all non-faulty sensor nodes follow the same protocol. So for binary consensus error rate is zero.

REFERECENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor network: a survey,” Computer Network, vol. 38, pp. 393–422, 2002.
[2] D. Culler, D. Estrin, and M. Srivastava, “Overview of sensor networks,” Computer, vol. 37, no. 8, pp. 41–49, 2004.
[3] W. J. Li and H. Y. Dai, “Cluster-based distributed consensus,” IEEE Transactions on Wireless Communications, vol. 8, no. 1, pp. 28–31, 2009.
[4] S. Sardellitti, M. Giona, and S. Barbarossa, “Fast distributed average consensus algorithms based on advection-diffusion Processes,” IEEE Transactions on Signal Processing, vol. 58, no. 2, pp. 826–842, 2010.
[5] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, and A. Scaglione, “Gossip algorithms for distributed signal processing,” Proceedings of the IEEE, vol. 98, no. 11, pp. 1847–1864, 2010.
[6] W. Ren and R. W. Beard, Distributed Consensus in MultiVehicle Cooperative Control: Theory and Applications, Springer, London, UK, 2010.
[7] A. D. G. Dimakis, A. D. Sarwate, and M. J. Wainwright, “Geographic gossip: efficient averaging for sensor networks,” IEEE Transactions on Signal Processing, vol. 56, no. 3, pp. 1205–1216, 2008.
[8] F. Benezit, A. G. Dimakis, P. Thiran, Vetterli, and M. Gossip, “Along the way: order-optimal consensus through randomized path averaging,” in Proceeding of the Allerton Conference on Communication, Control, and Computing, pp. 26–28, Allerton, Ill, USA, September 2007.
[9] T. C. Aysal, M. E. Yildiz, A. D. Sarwate, and A. Scaglione, “Broadcast gossip algorithms for consensus,” IEEE Transactions on Signal Processing, vol. 57, no. 7, pp. 2748–2761, 2009.
[10] D. Ustebay, B. N. Oreshkin, M. J. Coates, and M. G. Rabbat, ¨ “Greedy gossip with eavesdropping,” IEEE Transactions on Signal Processing, vol. 58, no. 7, pp. 3765–3776, 2010.
[11] K. I. Tsianos and M. G. Rabbat, “Fast decentralized averaging via multi-scale gossip,” in Proceeding of International Conference on Distributed Computing in Sensor System, pp. 21–23, Santa Barbara, Calif, USA, June 2010.
[12] M. Zheng, M. Goldenbaum, S. Stanczak, and Y. Haibin, “Fast average consensus in clustered wireless sensor networks by superposition gossiping,” in Proceedings of the IEEE Wireless Communication and Networking Conference, pp. 1–4, Paris, France, April 2012.
[13] D. J. Baker and A. Ephremides, “The Architectural organization of a mobile radio network via a distributed algorithm,” IEEE Transactions on Communications, vol. 29, no. 11, pp. 1694–1701, 1981.

[14] M. Gerla and J. Tzu-Chieh Tsai, “Multicluster, mobile, multimedia radio network,” Wireless Networks, vol. 1, no. 3, pp. 255–265, 1995.
[15] S. Basagni, “Distributed clustering for Ad Hoc networks,” in In Proceeding of the International Symposium on Parallel Architectures, Algorithms and Networks, pp. 23–25, Perth, Australia, June1999.
[16] S. Basagni, “Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks,” in Proceedings of the 50th IEEE Vehicular Technology Conference (VTC ’99), pp. 19–22, Amsterdam, The Netherland, September 1999.
[17] M. Chatterjee, S. K. Das, and D. Turgut, “WCA: a weighted clustering algorithm for mobile Ad hoc networks,” Journal of Cluster Computing, vol. 5, no. 2, pp. 193–204, 2002.
[18]M. Draief and M. Vojnovic, Convergence speed of Binary interval consensus," in proceedings of annual joint conference of the IEEE computer and communications societies (INFOCOM 2010), San Diego California, March 15-19, 2010