Hossain and Khan *EURASIP Journal on Wireless Communications and Networking* (2016) 2016:195
DOI 10.1186/s13638-016-0698-x

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

Efficient consensus algorithm for the accurate faulty node tracking with faster convergence rate in a distributed sensor network

Rajkin Hossain* and Muhidul Islam Khan

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 node 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.

Keywords: Wireless sensor networks, Consensus algorithm, Distributed systems, Convergence rate, Faulty node tracking, Binary consensus, Average consensus

1 Introduction

The miniaturization of electronic devices provide the impact on the development of wireless sensor network (WSN). WSNs consist of so many tiny sensor nodes capable of sensing, transmitting and receiving signals. Distributed sensor nodes are deployed in an ad hoc manner and responsible for various applications such as monitoring, surveillance, security and healthcare load balancing [1]. Sensor nodes communicate with each other in a multihop fashion. Nodes need to perform some tasks at each time step and report a central base station. For large distributed sensor networks, it is difficult to deploy so many central base stations in the surveillance area. But with the central base stations, task management becomes centralized. The task management technique by distributed sensor nodes cooperatively have been studied in recent years. It is not feasible to take decision by one independent node. Consistent decision making using cooperation among sensor nodes by some information exchange mechanism is suitable for a distributed system. To reach a decision collectively using cooperation is called to reach consensus [2].

To reach consensus in a quicktime is a challenging issue in a distributed sensor network. Consensus algorithms help to reach on a particular decision in a way that the globally optimal decision is reached in a distributed way. Binary consensus and average consensus are two widely used consensus algorithms. These are two distributed iterative information exchange algorithms.

Basically consensus algorithms are used in control area. Recently the consensus algorithm is also used in distributed network. This helps to achieve global optimal decision in a distributed fashion without any central controller. There are some works to reach consensus using either binary consensus or using average consensus algorithm. To our best knowledge, there is no work to

*Correspondence: rajkin69@gmail.com
Department of Computer Science and Engineering, BRAC University, Dhaka, Bangladesh

evaluate these two consensus algorithms for convergence rate comparison.

Consensus algorithm 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 [3]. Recently, the most attractive consensus algorithm is the gossip algorithm, 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.

We apply binary and average consensus algorithms in a distributed sensor network. No study has analyzed the convergence rate for an irregular network topology with the faulty nodes. Hence, in this paper, we investigate the proposed algorithm in irregular network topology, such as C-shaped, I-shaped, H-shaped, D-shaped, and O-shaped topology with faulty nodes. An irregular sensor model is introduced to evaluate the robustness of the algorithm. We apply segment tree data structures to reach consensus decisions at faster time. In addition we also investigate the proposed algorithm in regular network topology by creating random networks.

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

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 explains our network model. In Section 4 we present background and our proposed method. Section 5 is our proposed algorithms, and Section 6 experimental results and evaluation for distributed sensor networks. Section 7 concludes this paper with a brief summary.

2 Related works

The widely used consensus algorithm to reach a decision in a distributed environment is gossip algorithm [4]. In gossip algorithm, pair of nodes are chosen randomly to exchange information and update their values. It is easy to implement. But, the random information exchange creates more overhead in the network and it takes more time to reach consensus. Generally, those algorithm works for various irregular topologies like ring topology. Sometimes, randomized consensus algorithm can run faster but there is guarantee. This algorithm terminates after a particular amount of time even if the consensus is reached or not. Time limitation leads to lower efficiency and higher sensitivity to disturbances. The average consensus algorithm [5] helps to reach the consensus by updating their local values using average values of their neighbor's values. Another widely used consensus algorithm is binary majority consensus [6]. Connectivity between nodes also affect the consensus value for the gossip algorithm. Besides few randomized consensus algorithms [7], [8] also exist. This consensus algorithm does not provide guaranteed consensus for the additive noise in the network. Few papers used also gossip algorithm but the iteration number is fixed [9]. Its an overview of the use of consensus algorithm in cooperative control and those consensus algorithms are also single and double integrator dynamical systems. Those algorithms run for rigid body attitude dynamics. Few papers focus on average consensus algorithm on asymmetric interaction mechanisms, with time-varying weights on each edge, it is possible to provide a substantial increase of convergence rate with respect to the symmetric time-invariant case [10]. There are some research works which are focusing on energy efficiency and so they used cluster-based distributed consensus algorithm in forms of both fixed linear iteration and randomized gossip. Some rare works found like working on autonomous vehicle [11]. As distributed algorithms are typically iterative and suffer from time and energy consumption so they come up with this idea. As standard gossip algorithms can lead to a significant waste of energy by repeatedly recirculating redundant information they propose and analyze an alternative gossiping scheme that exploits geographic information. By utilizing geographic routing combined with a simple resampling method, they demonstrate substantial gains over previously proposed gossip protocols [12]. Few paper works study distributed broadcasting algorithms for exchanging information and computing in an arbitrarily connected network of nodes. Specifically, they study a broadcasting-based gossiping algorithm to compute the (possibly weighted) average of the initial measurements of the nodes at every node in the network. They showed that the broadcast gossip algorithms converge almost surely to a consensus [13]. Some paper works on greedy based like that paper presents greedy gossip with eavesdropping (GGE), a new average consensus algorithm for wireless sensor network applications [14]. Some paper works interested in the problem of computing the average consensus in a distributed fashion on random geometric graphs they describe a new algorithm called Multi-scale Gossip which

employs a hierarchical decomposition of the graph to partition the computation into tractable sub-problems [15]. Some paper works on superposition gossiping for average consensus in cluster wireless sensor networks where the nodes in each cluster exploit the natural superposition property of wireless multiple-access channels to significantly decrease local averaging times [16]. A paper proposed an on-demand distributed clustering algorithm for multi-hop packet radio networks. They try to keep the number of nodes in a cluster around a pre-defined threshold to facilitate the optimal operation of the medium access control (MAC) protocol [17]. Few papers consider the problem of organizing a set of mobile, radio-equipped nodes into a connected network. They require that a reliable structure be acquired and maintained in the face of arbitrary topological changes due to node motion and/or failure [18].

3 Network model

The considered network model is undirected. The network model may have some faulty nodes but we do not have any idea about that at first. We consider a clustered network with a cluster head for each cluster. Cluster heads can communicate with all the nodes in their respective clusters. Cluster head of a particular cluster can communicate with the cluster heads in the other clusters. Cluster heads also work as normal nodes with an additional characteristic is that they cannot be faulty which is our assumption. We assume the connection between the cluster heads form a complete graph and the graph is completely connected.

Each node of the sensor network hold three classes of values which are energy, binary states, and average states. As we are working on both binary and average consensus, we need to use it. When a node's energy becomes zero or less, it becomes a faulty node and cannot communicate throughout the network.

Figure 1a defines a sample of network model where blue bold circles define cluster heads; we may see each sensor node's neighbor, cluster area, and their network area. Figure 1b defines a sample network model but here we just show for single cluster area where a cluster head exist which sensor id is 5.

In Fig. 1b, edge defines physical distances and double circles define cluster heads. In Fig. 1a, we can see that one cluster area nodes has not any communication link to other cluster area sensor nodes.

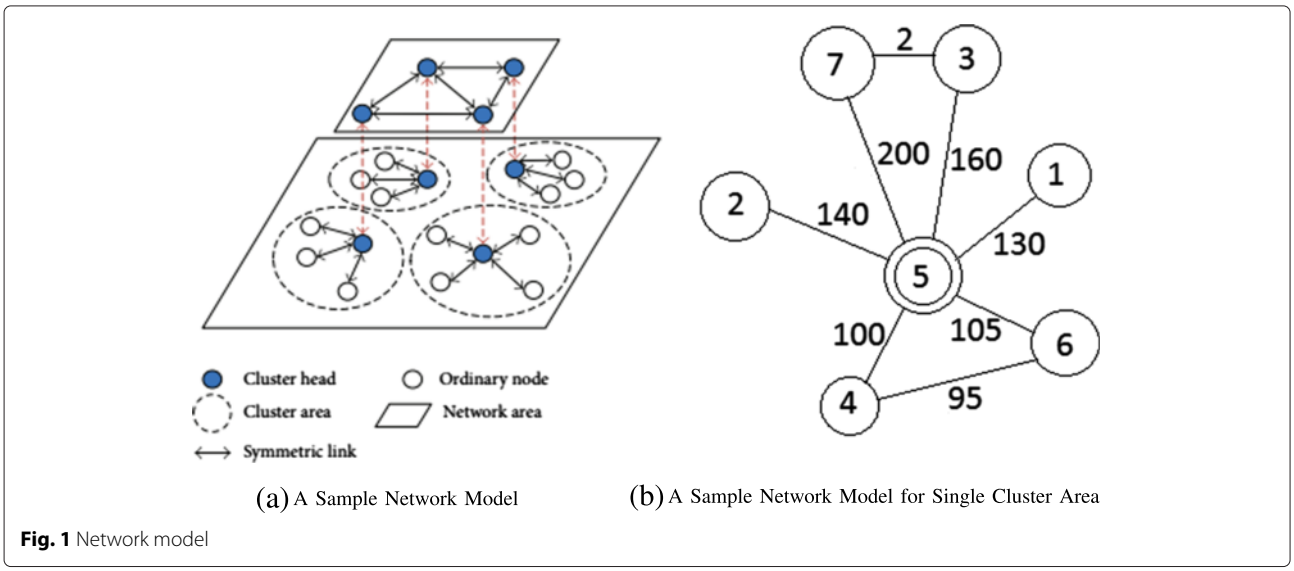
The reason is one's cluster area sensor nodes do not fall under other cluster area sensor nodes generally. But, there can be a communication link possible if their transition radius crosses other sensor nodes.

We provide both regular and irregular networks. Both network models are exactly the same as this network model, but we did not mark the networks cluster head or cluster area.

4 Background and proposed methods

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. The node of a segment tree takes a given priority- based value from a specific range. 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 two children which are left and right. The left child's id is TreeNode ID * 2



(a) A Sample Network Model (b) A Sample Network Model for Single Cluster Area

Fig. 1 Network model

and the right child's id is TreeNode ID * 2+1. As segment tree generally works in recursive process so every node except leaves go to it is both left and right children recursively. The base case of recursive process is stopped when calling reaches leaf nodes. Generally a segment tree has three methods which are build, query, and update. The depth of a segment tree is log (number of nodes). The general three methods of a segment tree described below. To describe the below part, we considered the segment tree of Fig. 2 and set priority as taking Max Value and N defines number of nodes.

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 Fig. 2. 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 is actually top-down processing. So from top node 1 to leave nodes 4, 5, 6, 7, it goes when its reached leaves node it saves the corresponding array value. For example leave node 7 which take the maximum value of ranges [4-4] saves array value of four 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 two 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 be got from backtracking. So, from its left and right child value, it takes the maximum among them as both left and right child value are computed when we ready to save values 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 is how segment tree build process works. The complexity of build process is O(nlog(n)) as we updated every leaf nodes individually.

Query The query process is just like searching from it. For example, what is the max value from array index 2 to 4? The process is as follows: we start from root node if we see that a node range 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's left child take ranges [1–2] which set under query range but not totally so we again divide 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 −9 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 the max value between 7 and −9 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 get to that leaf node by recursively and update it from there. For example, let us update three 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

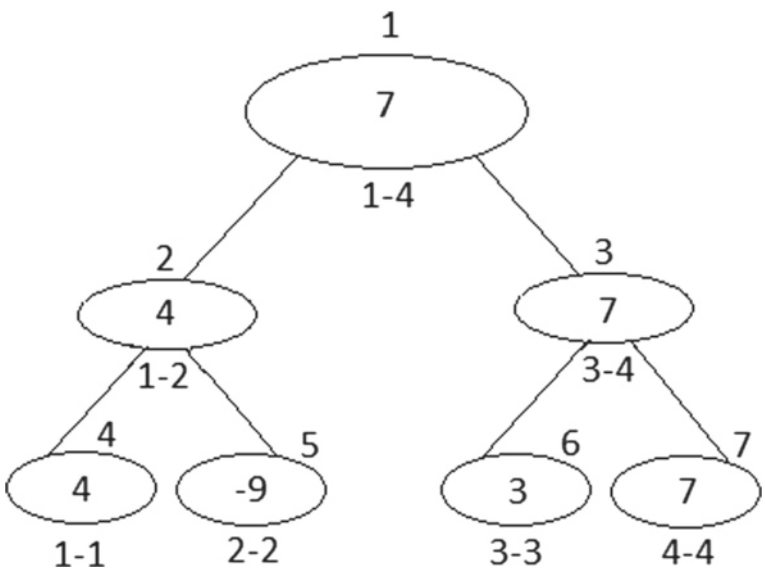
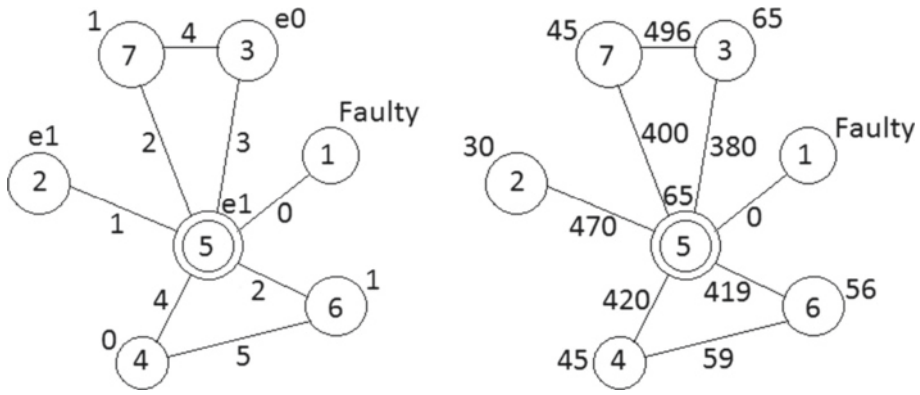


Fig. 2 A segment tree



(a) Graph for Binary Consensus (b) Graph for Average Consensus

Fig. 3 Two graphs

reach tree node id 3. Now, the max value will be 10 instead of 7. At last, we will reach the root node max value will be 10 instead of 7. That is how segment tree update 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, it is not needed. So, we just skipped it. Time complexity is O(log(n)). In our problem for update process, we implemented faster bottom-up process instead of recursive top-down process.

4.2 Creating two weighted graphs

In our main graph, edge has 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 node's, unique id, node id of two end pointer. In Fig. 1b, edge represents distance. So, we should make more two weighted graphs, but here, edge represents not only physical distances but also above described characteristics. Here, actually Fig. 3a, b are same graphs. To make it clear, we have drawn two graphs to view the edges for binary and average consensus individually. We collected the edge information from Table 1, sensor node information, and link information from Fig. 1b. In both of the figures, cluster head is defined as a 5 sensor node id by drawing a double circle.

Table 1 Table representation for both graphs

Sensor 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

Binary consensus When we try to reach a consensus decision in a distributed network, every node of the network can hold initially one of the two values, 0 and 1. 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 five states which can be described informally as:

- 0 - The node believes the majority opinion is most likely false.
- 1 - The node believes the majority opinion is most likely true.
- e0 - The node believes the majority opinion might be false.
- e1 - The node believes the majority opinion might be true.
- F - The node is a faulty node. Convergence occurs when all nodes have states 0,e0 or 1,e1.

Updating protocol for binary consensus:

For updating, we have to follow updating protocols [3]. The protocols are given below with an example:

1) (0,1) → (e1,e0)
2) (e0,1) → (1,e1)
3) (0,e1) → (e0,0)
4) (e0,e1) → (e1,e0)
5) (0,e0) → (e0,0)
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

Table 2 Table representation binary consensus edge priority

Binary protocol ID	Protocol priority or consensus weight
1	5
2, 3	4
4	3
5, 6	2
7	1
8	0

We introduce a new protocol here which is (s,F)→(s,F) as mentioned at number 8 above. We must give priority to these protocols. We assign priority based on decreasing order serial from the updating protocols. We propose serial no.1 as highest priority. Table 2 represents total priority table for binary consensus.

Let us describe the situation in Fig. 4. First node B and C run updating protocol. The states will become e0 and e1. Then, A and B run updating protocol; their states will become e1 and 1. Now, a consensus is reached because all the states are either e1 or 1 . Actually in Fig. 4, we did not consider any cluster head or faulty node we just tried to show how updating protocols are used for binary consensus.

Average consensus For average consensus, every state initially takes some value and here energy plays very important role. While pair wise updating in average consensus the new states value for both pair is its average value. Besides, we propose its weight which is below:

A = Energy(u) - distance and B = Energy(v) - distance
weight(u,v) = A + B if A and B >= 0
otherwise weight(u,v) = 0

Updating protocol for average consensus: there is only one updating protocol for average consensus which is simple its average by consideration of number of iterations which is given below:

$$x_i(k) = x_j(k) = [x_i(k-1) + x_j(k-1)]/2 \quad (1)$$

4.3 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 5 is the edge array of our main graph, and Table 3 defines our edge graph. Algorithm 1 is based on creating Edge Array and Algorithm 2 is based on creating edge graph.

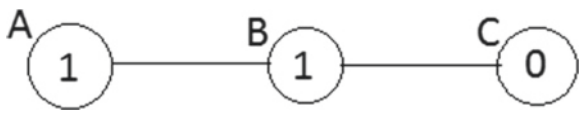


Fig. 4 Understanding binary consensus

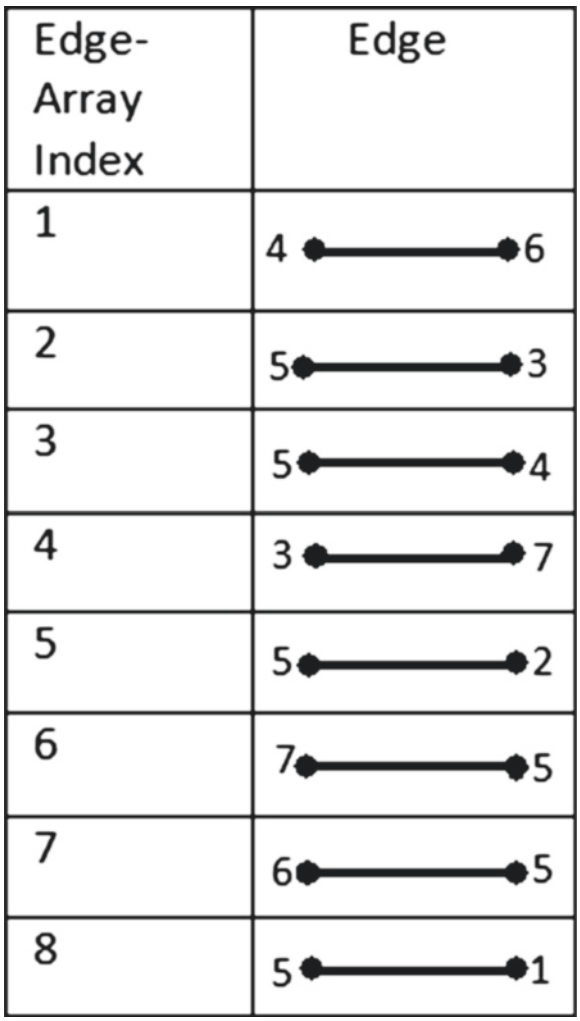


Fig. 5 Edge array

5 Proposed algorithms

Here, we applied our proposed algorithm for consensus in two different independent segment trees. We first build

Table 3 Table representation edge graph

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]

Algorithm 1 Creating edge array

```
Model the network graph(G)
Initialize Edge Array Array(EA)
Initialize Edge − ID Array (Edge − ID)
id=1
for Each edge (u, v) ∈ G do
    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 do
    for Each node v ∈ G.adj[ u] do
        EG[ u] .add(Edge − ID[ Hashed_Value(u, v)] )
    end for
end for
```

two segment trees from corresponding two edge arrays, and by the help of edge graph, we track down both tree leaf index id in STreeNode, S2TreeNode from edge arrays. We did it for future bottom up processing and avoid unnecessary recursion. We also provide algorithms for creating edge array and edge graph which we described earlier.

Modeling Edge graph proved great contribution to make fasten the algorithm. Without doing algorithm time complexity, will be much more higher.

In Edge graph, we tracked the each node's next neighbor edges. We can easily walk from node to its perspective neighbor edges which is very important in our algorithm.

Our main algorithm is basically algorithm 6 and 10 which are our main binary consensus and average consensus. Algorithm 3, 4, 5, and 6 are for binary consensus algorithm. In Algorithm 6, we used algorithm 3, 4, and 5.

Algorithm 7, 8, 9, and 10 are for average consensus algorithms. In algorithm 10, we used algorithm 7, 8, and 9.

Creating edge array algorithm is really simple actually, we just take an array class of edges, then put all the edges in the array, but we need to put the edges all characteristics in the array. In addition, we need to track the vice versa Edge-ID to create edge graph in creating edge graph algorithm.

After creating Edge array, we are ready to create Edge Graph because we efficiently tracked down Edge-ID. We also provide creating edge graph algorithm.

To create Edge Graph, we just iterate all the sensor nodes and save their neighbor edges. We get their neighbor edges from Edge-ID which we tracked earlier.

Algorithm 3 Building segment tree (for 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 4 Tracking highest priority node (for binary consensus)

```
HIGHEST(SNode l, SNode r)
Initialize a SNode(Node) which keep a mainEdge, a dummyEdge and weight of them which are same
rBPID = r.mainEdge.BPId
lBPID = l.mainEdge.BPId
if r.Bweight == l.Bweight then
    if rBPID == 1 and lBPID == 6 then
        Node.mainEdge = r.mainEdge
        Node.dummyEdge = l.mainEdge
    else if lBPID == 1 and rBPID == 6 then
        Node.mainEdge = l.mainEdge
        Node.dummyEdge = r.mainEdge
    end if
else if r.Bweight > l.Bweight then
    Node=r
else Node=l
end if
return Node
```

Algorithm 5 Updating segment tree (for binary consensus)

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

There is a trick to use Edge-ID. First, we should traverse each sensor nodes and their neighbor sensor nodes. So, now it is become easier for us to use Edge-ID efficiently to create Edge graph.

Algorithm 6 Binary consensus algorithm

```
Binary Consensus Algorithm ()
BUILD-TREE(1,1,edgeArray_Length) Build Segment Tree for Binary Consensus
Energy of all sensor nodes stores in Energy Array(Energy)
topnode = 1
while True do
    edge = STree[ topnode] .mainEdge
    if STree[ topnode] .dummyEdge == NULL and OneOrSix(STree[ topnode] .BPId) then
        Binary Consensus Reached..break
    else if STree[ topnode] .Bweight == 1 then
        Binary Consensus Reached.. break
    else if edge.Bweight == 0 then
        Binary Consensus Will Never Reached..break
    else
        distance = physicalDistance(edge.u, edge, v)
        EnergyU = Energy[ edge.u] −distance
        EnergyV = Energy[ edge.v] −distance
        if EnergyU<0 or EnergyV<0 then
            Make just STree[STreeNode[edge.id]] as a faulty edge by updating.
            UPDATE(STreeNode[edge.id])
        else
            update both energy array to EnergyU and EnergyV
            Change STreeNode[edge.id] mainEdge by following binary updating protocols.
            UPDATE(STreeNode[edge.id])
            for Each edge e ∈ EG.Adj[ edge.u] do
                treeID = STreeNode[e.id]
                if e.u == edge.u and e.v! = edge.v then
                    STree[e.id].mBU = edge.BU
                    Change STree[treeID] information of edge-weight and protocol id
                    UPDATE(treeID)
                else if e.v == edge.u and e.u! = edge.v then
                    STree[e.id].mBV = edge.BU
                    Change STree[treeID] information of edge-weight and protocol id
                    UPDATE(treeID)
                end if
            end for
            for Each edge e ∈ EG.Adj[ edge.v] do
                treeID = STreeNode[e.id]
                if e.u == edge.v and e.v! = edge.u then
                    STree[e.id].mBU = edge.BV
                    Change STree[treeID] information of edge-weight and protocol id
                    UPDATE(treeID)
                else if e.v == edge.v and e.u != edge.u then
                    STree[e.id].mBV = edge.BV
                    Change STree[treeID] information of edge-weight and protocol id
                    UPDATE(treeID)
                end if
            end for
        end if
    end if
end while
```

Algorithm 7 Building segment tree (for 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 8 Tracking highest priority node (for average consensus)

```
HIGHEST2(S2Node l, S2Node r)
Initialize a S2Node (Node) which keep a treeNode
if (l.AstateU == l.AstateV) then then Node=r
else if (r.AstateU == r.AstateV)or(l.Aweight > r.Aweight) then Node=l
else Node = r
end if
return Node
```

Algorithm 9 Updating segment tree (for average consensus)

```
UPDATE2(node-id,j)
i = node-id
if j == 1 then
    au = S2Tree[ i] .treeEdge.AU
    av = S2Tree[ i] .treeEdge.AV
    S2Tree[ i] .treeEdge.AU = (au + av)/2
    S2Tree[ i] .treeEdge.AV = (au + av)/2
end if
while (i = i >> 1)! = 0 do
    Left = i * 2
    Right = left + 1
    treeLeft = S2Tree[Left]
    treeRight = S2Tree[Right]
    S2Tree[ i] = HIGHEST2(treeLeft, treeRight)
end while
```

5.1 Segment tree for binary consensus

Now, we have to make a segment tree for the whole part of the edge array. Here, every node of the segment tree can take four elements which are main-Edge, dummy-Edge, weight, and size. While considering the left child and right child for the upper nodes, we will choose the

Algorithm 10 Average consensus algorithm

```
Average Consensus Algorithm ()
BUILD-TREE2(1,1,edgeArray_Length)  Build Segment Tree for Average Consensus
Energy of all sensor nodes stores in Energy Array(Energy)
topnode = 1
while True do
    edge = STree[ topnode] .treeEdge
    if edge.AU == edge.AV then
        Highest Average Consensus Reached..break
    else if edge.Aweight == 0 then
        Average Consensus Reached.. break
    else
        distance = physicalDistance(edge.u, edge, v)
        EnergyU = Energy[ edge.u] −distance
        EnergyV = Energy[ edge.v] −distance
        if  EnergyU<0 or EnergyV<0 then
            Make just S2Tree[S2TreeNode[edge.id]] as a faulty edge by updating.
            UPDATE2(S2TreeNode[edge.id],0)
        else
            Update both energy array to EnergyU and EnergyV
            Change S2TreeNode[edge.id] treeEdge by following average updating protocols.
            UPDATE2(S2TreeNode[edge.id],1)
            Change current selected sensor nodes LEFT-SIDED neighbor edges information by the help of Edge graph
            Update each neighbor edge ids tree edge average U,V state to its neighbor edge U state
            UPDATE2(corresponding treeID,0)
            Change current selected sensor nodes RIGHT-SIDED neighbor edges information by the help of Edge graph
            Update each neighbor edge ids tree edge average U,V state to its neighbor edge V state
            UPDATE2(corresponding treeID,0)
        end if
    end if
end while
```

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 edges.

One edge is saved in main-Edge and another edge is saved in dummy-Edge. Besides if all of their weights follow same protocol (1 or 6 not both) then we save any single edge to tree nodes main-Edge and dummy-Edge will be NULL.

We consider every time top node from the segment tree and if the top node has both edges, we take any node from it and change its two endpoints value and update its information according to the updating protocols.

Here, we might have to change many edges information's 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 Fig. 6 (top node id 1). When we will work on edge [4-6], we should make change edges [4-5] and [6-5] information 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 go to that node and update it from there and by 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 third 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 second lowest or first lowest priority edges so we do not 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.

While updating, we must also minimize energy from that edge two pointer node. Besides, binary consensus has zero percent error rate. If segment trees topNode contains faulty nodes of the network so it follows lowest priority-based updating protocols. So, consensus will never occur we can claim that easily.

Besides Binary consensus has zero percent error rate. Figure 6 defines binary consensus algorithm simulation for a single iteration. In algorithms mBX means mainEdges binary state X, BPId means binary protocol ID.

Our binary consensus algorithm focus on how fast we can reach a situation when all the sensor nodes value states are e1 or 1 or all the sensor nodes value e0 or 0.

That is why we gave the priority of each node to node states, and greedily, we try to reach a consensus situation. The lower priority-based edges come late for processing and higher priority edges come earlier for processing.

- Complexity: O(treeNodes * (maxEnergy/minDistance) * treeNodes * log(treeNodes))

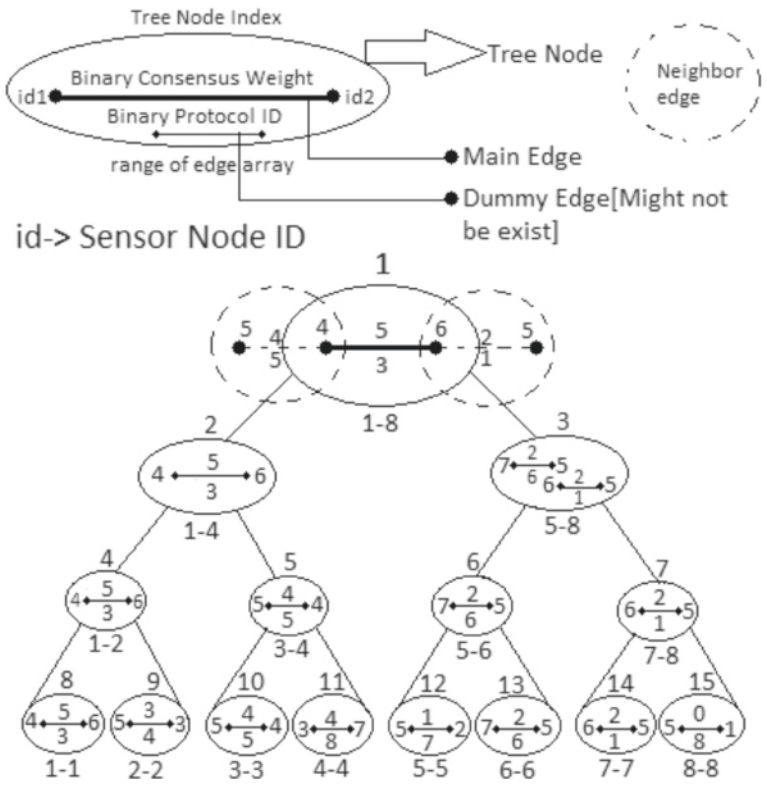


Fig. 6 Segment tree for binary consensus

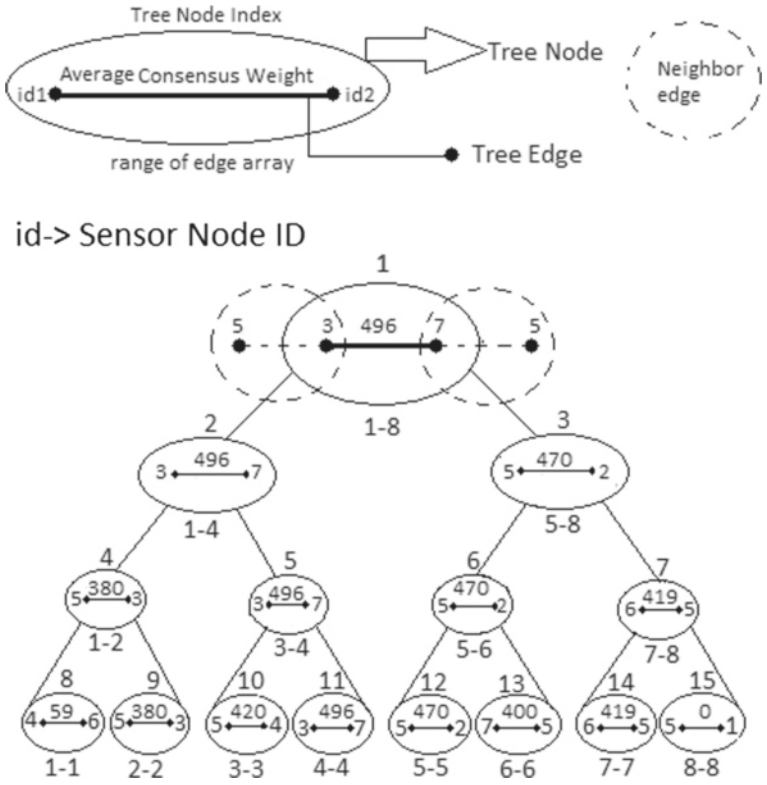


Fig. 7 Segment tree for average consensus

5.2 Segment tree for average consensus

Now, we have to make another segment tree for the edge array. Here, 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 S2TreeNodeArray (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.

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. When Top node Average consensus weight is not zero, we will continue our average consensus algorithm otherwise we should stop. Figure 7 defines average consensus algorithm simulation for a single iteration. Average consensus must have some error rate. The formula for calculating error rate is given below:

$$\frac{||x(k) - \bar{x}||}{||x(0) - \bar{x}||} \tag{2}$$

We will calculate this error rate after every single iterations. If we reached the highest possible average consensus point, we cant reduce error rate more. Few shortcuts tAX means treeEdges average state X.

Our average consensus algorithm focuses on how fast we can reach a situation when all the sensor nodes value states are same and second focus is we should less reduce the energy for the sensor nodes.

That is why we proposed the edge weight for average consensus. The lower priority-based edges comes late for processing and higher priority edges come earlier for processing. For average consensus, lower priority-based edges mean lower edge weight.

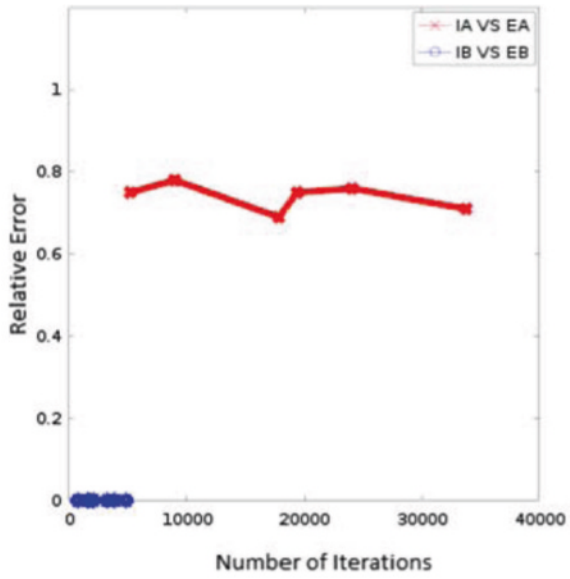
Table 4 Table representation for random topology

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
I(A2)	5202	9605	18937	24847	29498	44647
I(B2)	814	1566	2618	3011	3886	4553
E(A2)	0.73	0.77	0.68	0.67	0.70	0.59
F(B2)	4	5	19	23	17	31
E(B)	0.00	0.00	0.00	0.00	0.00	0.00
F(A)	0	0	0	0	0	0

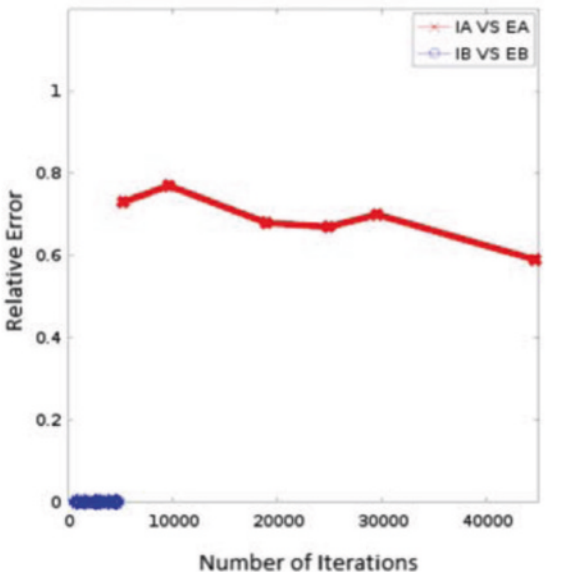
- Complexity: O(treeNodes * (maxEnergy/minDistance) * treeNodes * log(treeNodes))

6 Experimental results and evaluation

We simulate our environment with the distributed sensor nodes using Java programming language. We worked for both random and various topologies. For random



(a) First Time



(b) Second Time

Fig. 8 Iterations vs ErrorRate

topology, we considered random shaped but for shaped topology we considered five shaped network models which are *C*, *D*, *I*, *H*, and *O*. After reaching consensus decision, we get all sensor nodes average values from leaves of the segment tree because leaves of the segment tree stores single array updated values. We did it exactly like how we use build-tree for going leaves of the segment tree. Here, actually iteration means how many times we choose an edge to update their protocol. So, the less number of iterations means the algorithm reaches on consensus faster. We take data thrice for various topologies and twice for random topologies to clearly see the results variations. We described the defination and generation for various selected five topologies.

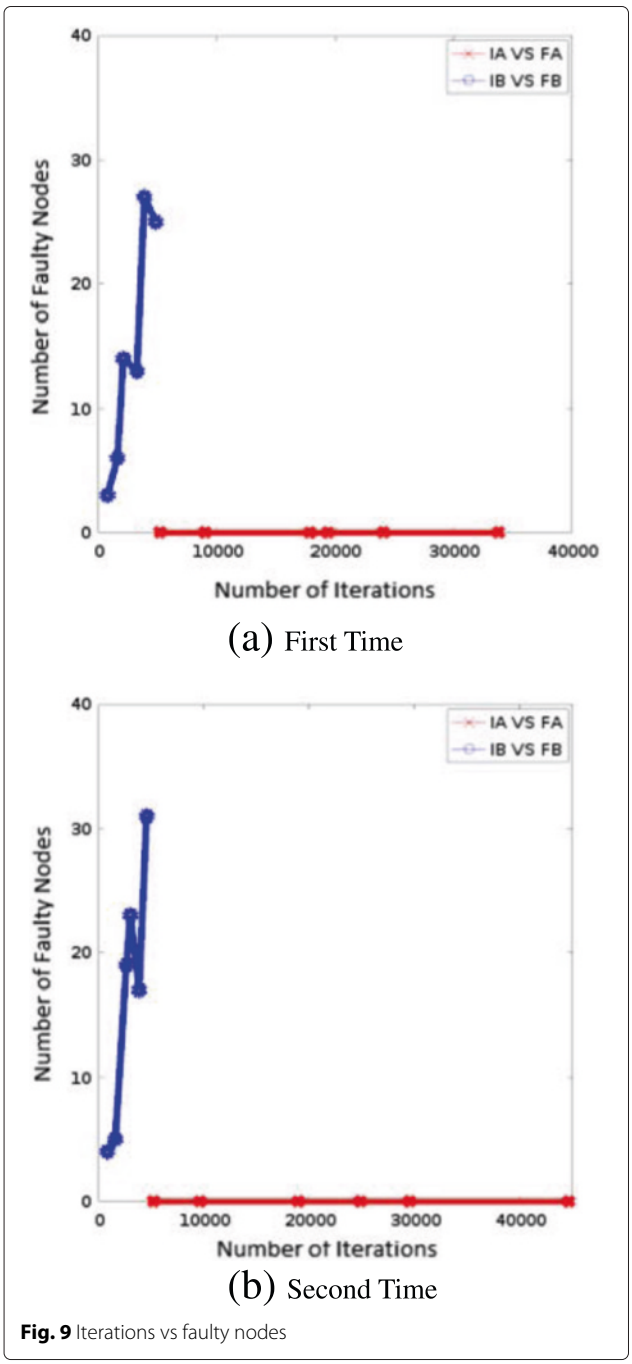


Fig. 9 Iterations vs faulty nodes

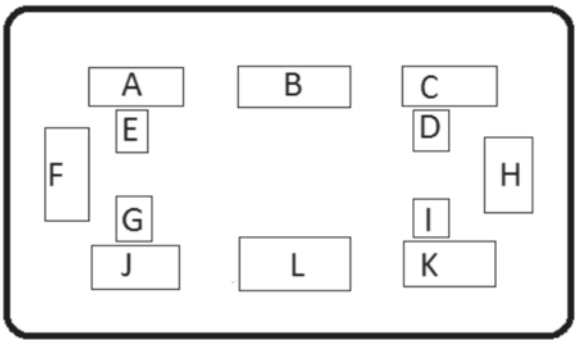


Fig. 10 Graphical figure to generate irregular topologies

6.1 Random topology

For random topology, we considered 300-1800 sensor nodes. Their graphical shaped are randomly chosen. Both times, we plot and draw table the graphs based on (i) Iterations vs Error-Rate and (ii) Iterations vs Faulty-Nodes. Table 4 represents two times iterations results. Here, for example I(A1) means in first testing out of twice iteration number in average consensus, I(B1) for iteration number for binary consensus. E is for error rate, F for faulty node.

Iterations vs error rate If we see in Fig. 8a, b, Iterations vs Error rate graph we can see that for binary consensus relative error rate become zero for any number of iterations which we already described in our algorithm analysis but Error Rate for average consensus in various over iterations but generally maximize iterations minimizes error rate like Fig. 8b.

In irregular topology, we may see that after consensus, reaching point for average consensus error rate became stable and for binary consensus error rate is zero as usual.

Iterations vs faulty nodes Lets see Fig. 9a, b, Iterations vs Faulty Nodes graph for average consensus number of faulty nodes is always zero the actual reason is if we see our average consensus algorithm the edge taking priority includes energy left in a sensor node. So, we gave lowest priority to those nodes. While running the algorithm, they

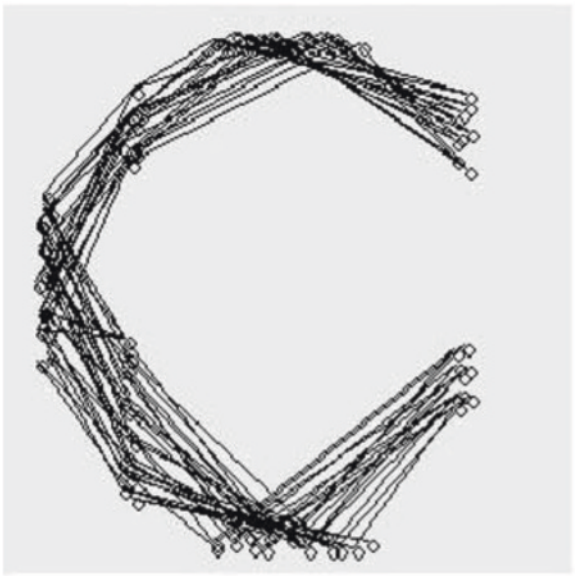
Table 5 Table representation for C topology

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

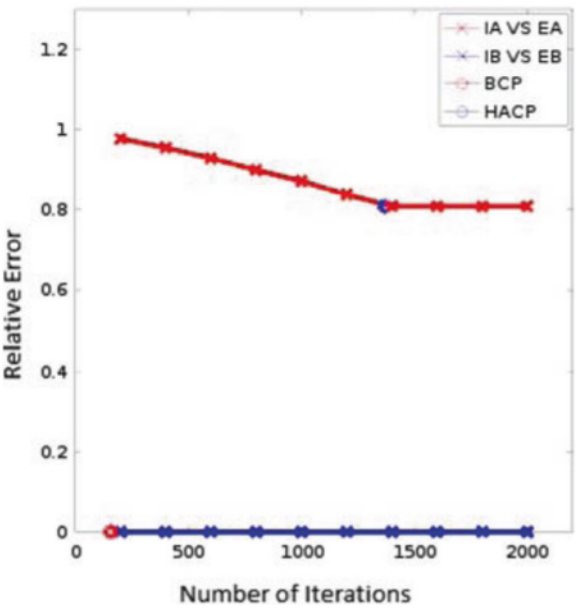
do not come generally to reach consensus for a fixed iteration. But in rare cases for average consensus, there might be some faulty nodes on the other hand for binary consensus; we did not give edge taking priority based on energy in a sensor nodes, so normally, we get some faulty nodes.

6.2 Various topology

We have implemented our algorithm in C [90 sensor nodes],O [130 sensor nodes],I [80 sensor nodes], H[80 sensor nodes] and D[80 sensor nodes]. We represented three times their iteration vs Error-Rate plots and tables with their accurate graphical representations. We also provided their definitions and generations with the use of Fig. 10. We showed here [A-L] 12 regions with boundary boxes. We can set any number of sensor nodes in the box and connect some regions to some regions to graphically represent our desired shaped topology. In each plot,

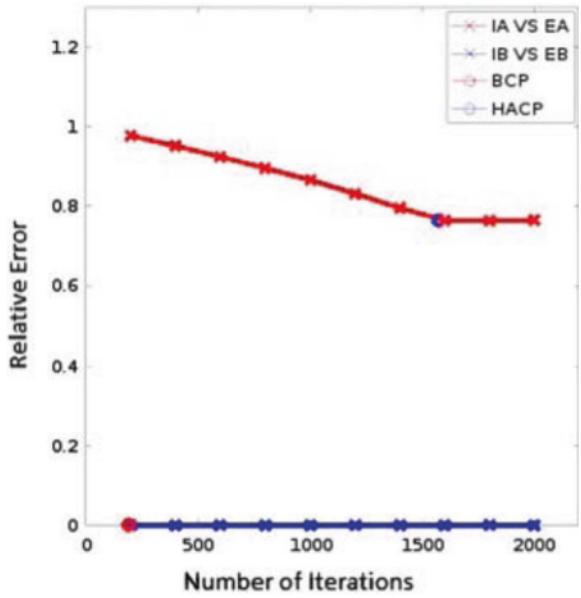


(a) C Topology

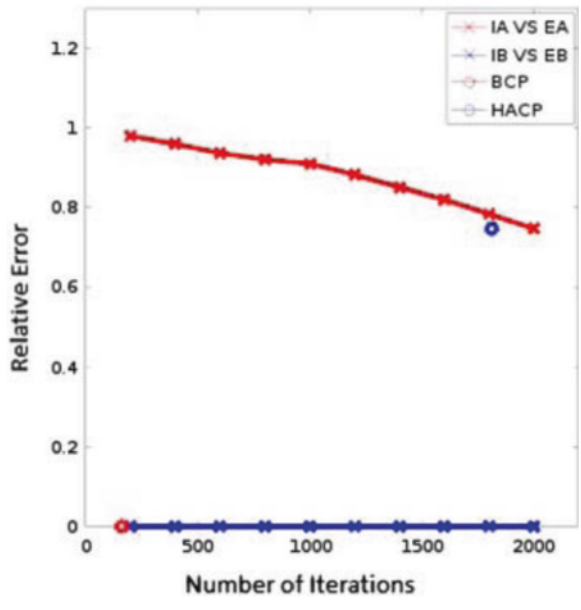


(b) First Time

Fig. 11 Topology and Iterations vs ErrorRate



(a) Second Time



(b) Third Time

Fig. 12 Iterations vs ErrorRate

Table 6 Table representation for D topology

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

figures HACP mean highest possible average consensus reaching point. We may see that after that, line error rate for average consensus is unchanged. BCP means binary consensus reaching point which is always zero, as we claimed error rate for binary consensus is always zero.

C topology We created C topology by connecting sensor nodes in Fig. 10 (i) from D to B (ii), from B to E (iii), from E to F (iv), from F to G (v), from G to L and (vi), and from L

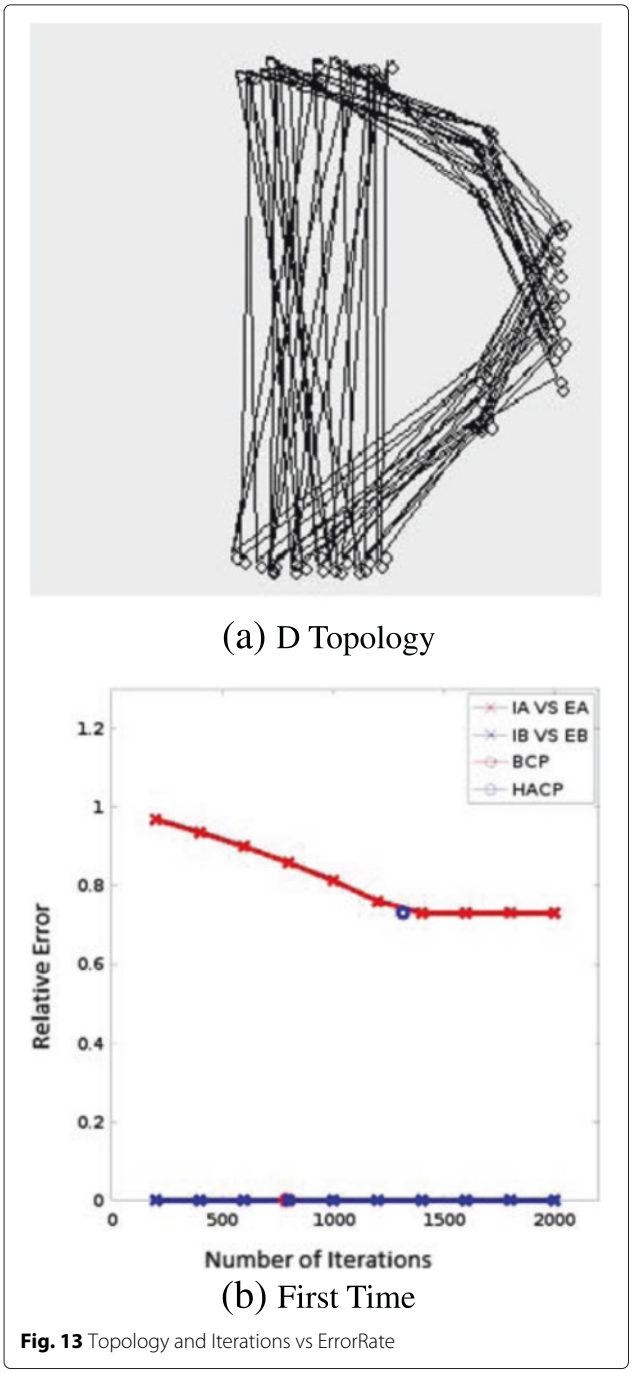


Fig. 13 Topology and Iterations vs ErrorRate

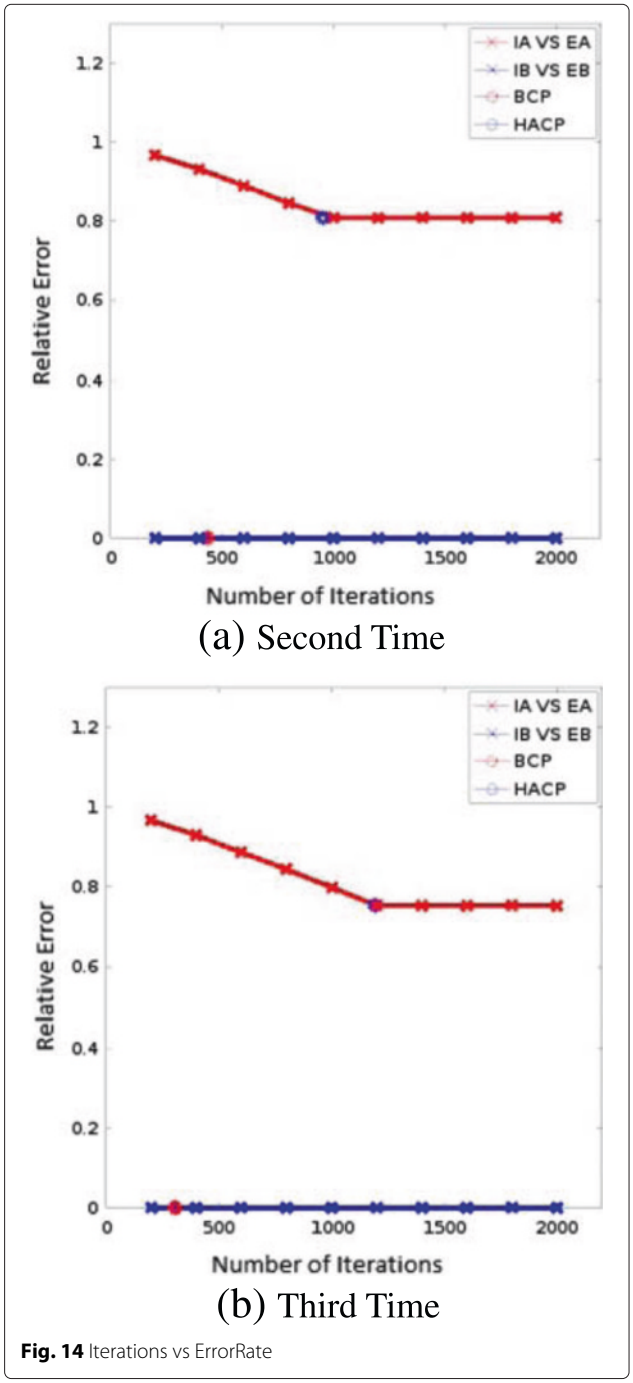


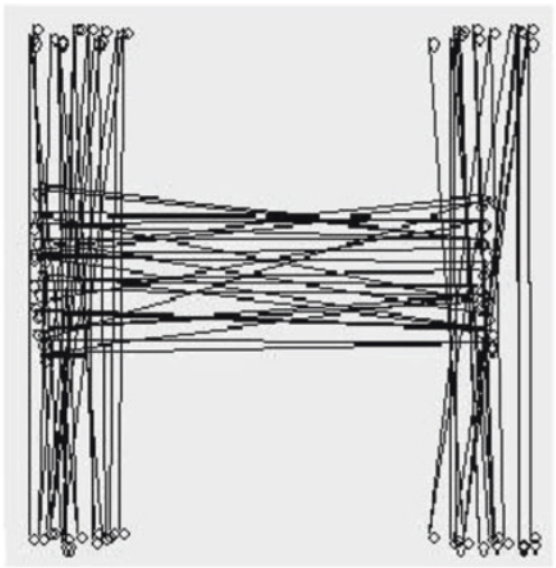
Fig. 14 Iterations vs ErrorRate

Table 7 Table representation for H topology

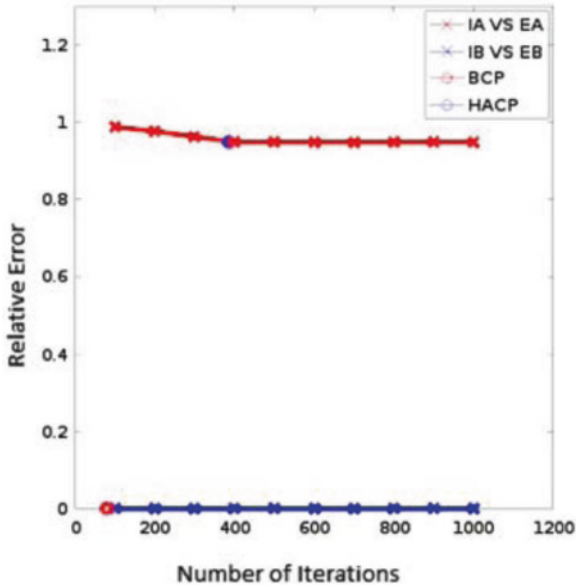
Iterations	*200*	400	600	800	1000	1200	1400	1600	1800	2000
ER(1)	0.988	0.977	0.963	0.949	0.949	0.949	0.949	0.949	0.949	0.949
ER(2)	0.989	0.978	0.964	0.949	0.934	0.932	0.932	0.932	0.932	0.932
ER(3)	0.987	0.975	0.960	0.943	0.942	0.942	0.942	0.942	0.942	0.942

to I. In total, we used 90 sensor nodes to generate C topology. Table 5 represents C topology. Figures 11b and 12a, b represents plots for C topology, and Fig. 11a is actual C topology we examined.

D topology We created D topology by connecting sensor nodes in Fig. 10 (i) from B to D, (ii) from D to H (iii), from H to I (iv), from I to L, and (v) from L to B . In total, we used 80 sensor nodes to generate D topology. Table 6 represents D topology. Figures 13b, and 14a, b represents plots for D topology and Fig. 13a is actual D topology we examined.

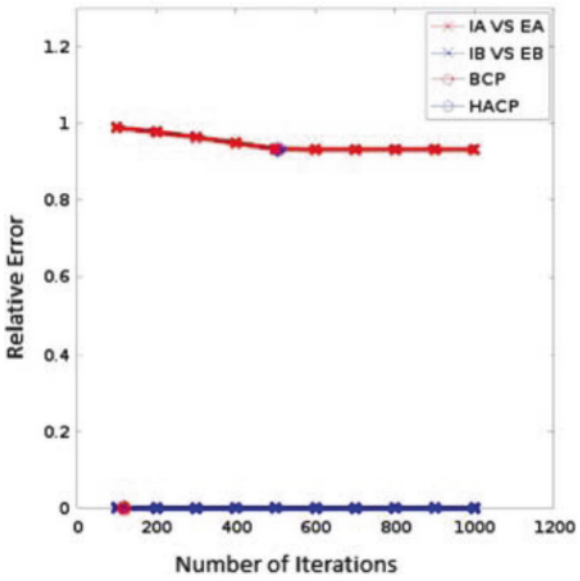


(a) H Topology

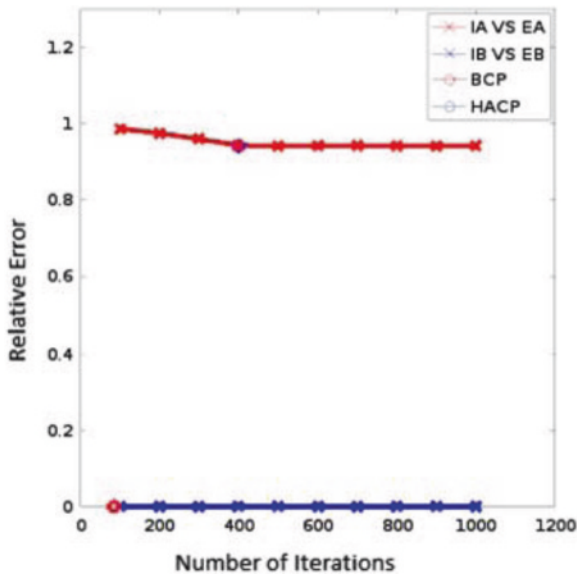


(b) First Time

Fig. 15 Topology and Iterations vs ErrorRate



(a) Second Time



(b) Third Time

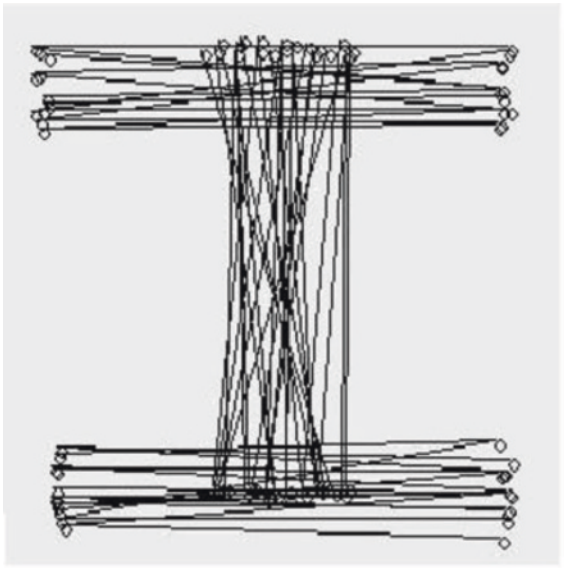
Fig. 16 Iterations vs ErrorRate

Table 8 Table representation for I topology

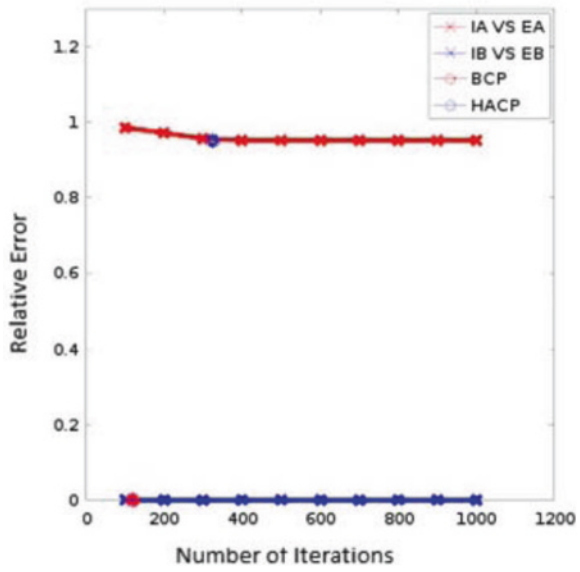
Iterations	*200*	400	600	800	1000	1200	1400	1600	1800	2000
ER(1)	0.986	0.972	0.957	0.952	0.952	0.952	0.952	0.952	0.952	0.952
ER(2)	0.989	0.976	0.962	0.962	0.962	0.962	0.962	0.962	0.962	0.962
ER(3)	0.989	0.975	0.959	0.951	0.951	0.951	0.951	0.951	0.951	0.951

H topology We created H topology by connecting sensor nodes in Fig. 10 (i) from A to J, and (ii) from F to H and (iii) from C to K. In total, we used 80 sensor nodes to generate H topology. Table 7 represents H topology. Figures 15b and 16a, b represents plots for H Topology and Fig. 15a is actual H topology we examined.

I topology We created I topology by connecting sensor nodes in Fig. 10 (i) from B to L , (ii) from A to C, and (iii) from J to K. In total, we used 80 sensor nodes to generate I topology. Table 8 represents I topology. Figures 17b

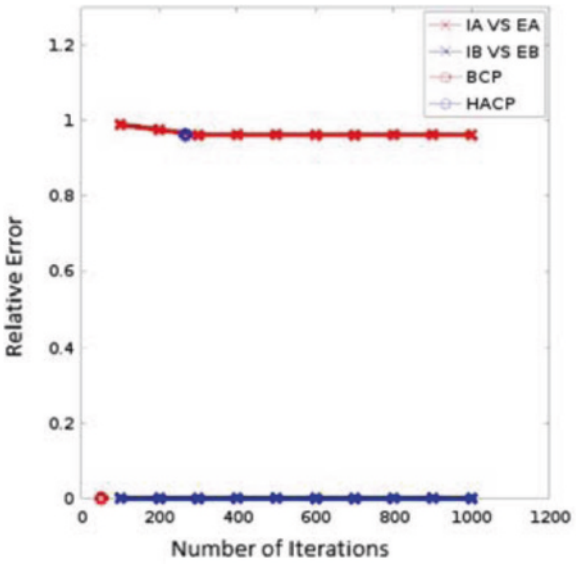


(a) I Topology

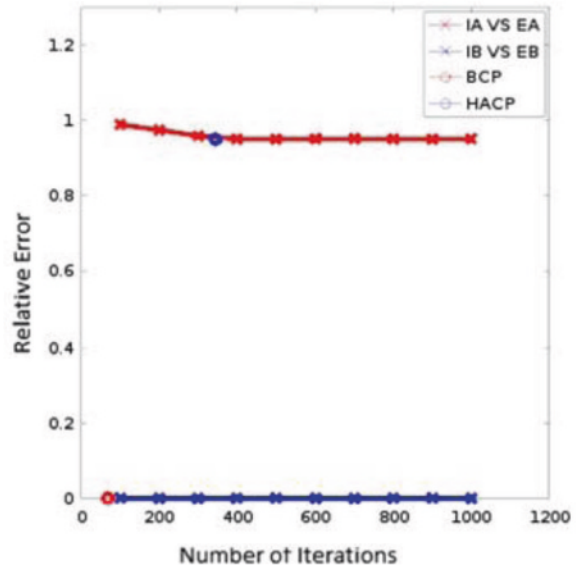


(b) First Time

Fig. 17 Topology and Iterations vs ErrorRate



(a) Second Time



(b) Third Time

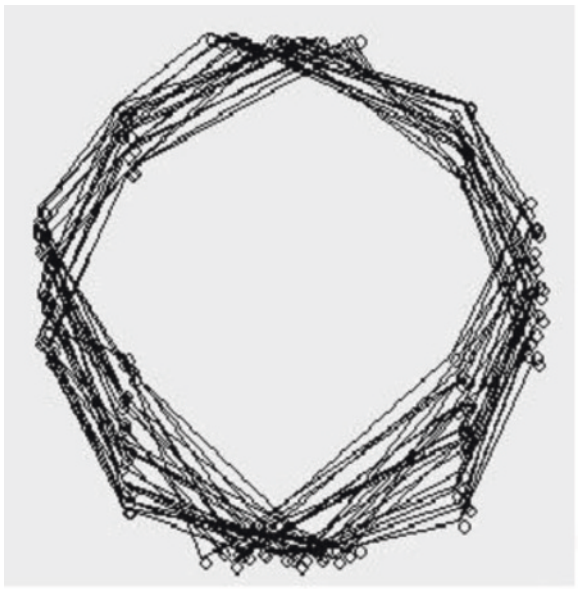
Fig. 18 Iterations vs ErrorRate

Table 9 Table representation for O topology

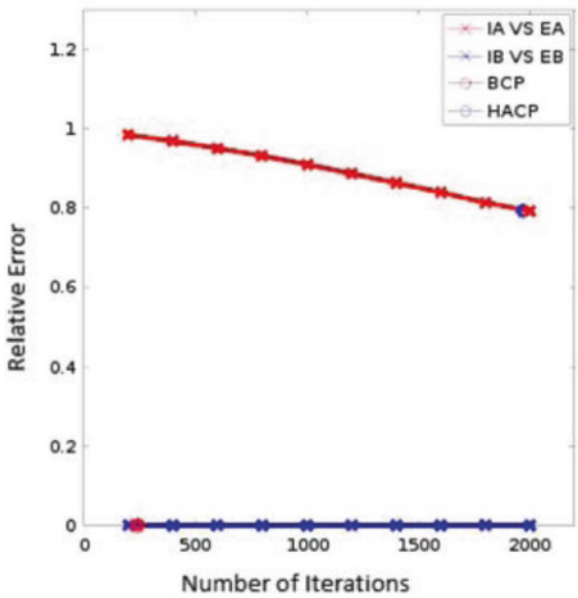
Iterations	*200*	400	600	800	1000	1200	1400	1600	1800	2000
ER(1)	0.985	0.969	0.951	0.932	0.910	0.887	0.864	0.840	0.814	0.793
ER(2)	0.985	0.967	0.948	0.929	0.908	0.885	0.860	0.837	0.827	0.827
ER(3)	0.984	0.968	0.949	0.928	0.906	0.881	0.856	0.836	0.836	0.836

and 18a, 18b represents plots for I topology and Fig. 17a is actual I topology we examined.

O topology We created O topology by connecting sensor nodes in Fig. 10 (i), from B to E , (ii) from E to F, (iii) from F to G, (iv) from G to L, (v) from L to I, (vi) from I to H, (vii) from H to D, and (viii) from D to B. In total, we used 130 sensor nodes to generate O topology. Table 9 represents

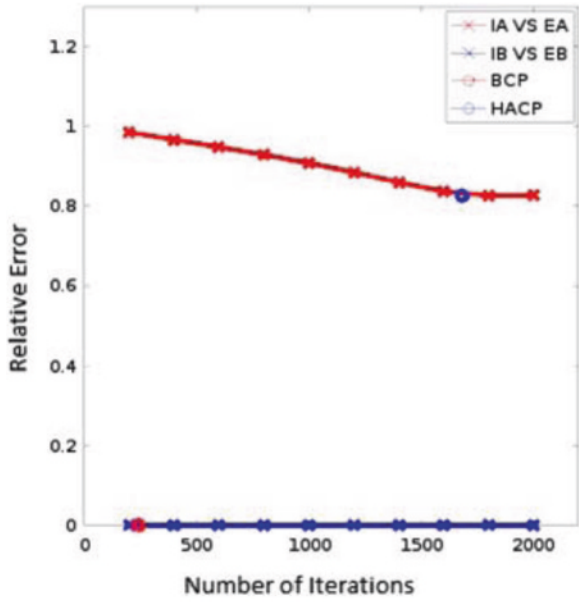


(a) O Topology

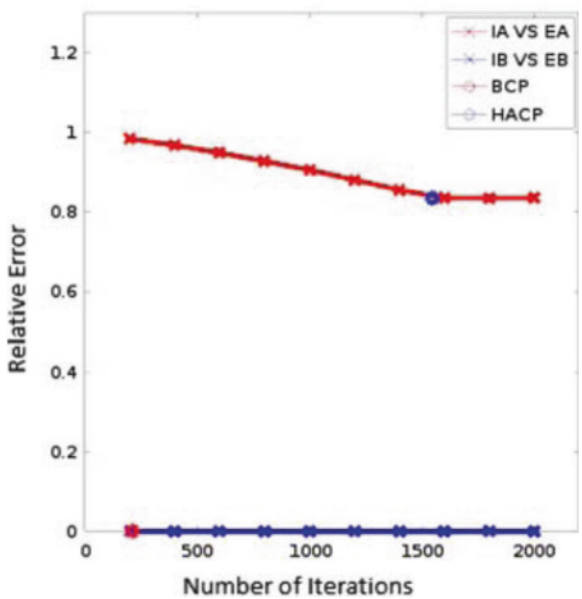


(b) First Time

Fig. 19 Topology and Iterations vs ErrorRate



(a) Second Time



(b) Third Time

Fig. 20 Iterations vs ErrorRate

O topology. Figures 19b and 20a, b represent plots for O Topology and Fig. 19a is actual O topology we examined.

7 Conclusions

In a distributed sensor network, where robustness is an important issue for finding out the faulty nodes efficiently. Consensus algorithms help to reach on a decision by which we are able to track the faulty nodes. Accuracy of knowing the right information about the faulty nodes is very challenging in a distributed environment. We consider different shapes of topology for applying the binary and average consensus algorithm. We also apply binary and average consensus algorithms in a random graph. We observe that average consensus algorithm provides faster convergence by giving less number of iterations. On the other hand, binary consensus is more accurate to find out the faulty nodes. So, using the appropriate consensus algorithm depends on the topology we are using and depends on the application.

In the future, we will consider the robustness issue in details considering some noisy nodes in the environment which provide misleading value. We will choose a discrete event simulator for our work for making the simulation more cognitive.

Competing interests
The authors declare that they have no competing interests.

Received: 7 April 2016 Accepted: 13 August 2016
Published online: 24 August 2016

References

1. IF Akyildiz, W Su, Y Sankarasubramaniam, E Cayirci, Wireless sensor network: a survey. Comput. Netw. **38**, 393–422 (2002)
2. F Benezit, AG Dimakis, P Thiran, M Vetterli, in *Allerton Conference on Communication, Control, and Computing*. Gossip along the way: order-optimal consensus through randomized path averaging (EPFL, Allerton, USA, 2007), pp. 26–28
3. M Draief, M Vojnovic, in *Annual joint conference of the IEEE computer and communications societies (INFOCOM 2010)*. Convergence speed of binary interval consensus (SIAM, San Diego, California, 2010), pp. 15–19
4. ADG Dimakis, AD Sarwate, MJ Wainwright, Geographic gossip: efficient averaging for sensor networks. IEEE Trans. Signal Process. **56**(3), 1205–1216 (2008)
5. Y Li, Z Zhou, T Sato, A cluster-based consensus algorithm in a wireless sensor network. Int. J. Distributed Sensor Netw. Hindawi. **60**(547124), 1–15 (2013)
6. N Al-Nakhala, R Riley, TM Elfouly, in *International Wireless Communications and Mobile Computing Conference (IWCMC)*. Binary consensus in sensor motes (IEEE, Sardinia, 2013), pp. 1337–1342
7. D Culler, D Estrin, M Srivastava, Overview of sensor networks. Computer. **37**(8), 41–49 (2004)
8. A Gogolev, L Marcenaro, Randomized binary consensus with faulty agents. Entropy. **16**, 2820–2838 (2014)
9. WJ Li, HY Dai, Cluster-based distributed consensus. IEEE Trans. Wireless Commun. **8**(1), 28–31 (2009)
10. S Sardellitti, M Giona, S Barbarossa, Fast distributed average consensus algorithms based on advection-diffusion processes. IEEE Trans. Signal Process. **58**(2), 826–842 (2010)
11. W Ren, RW Beard, *Distributed Consensus in MultiVehicle Cooperative Control: Theory and Applications*. (Springer, London, UK, 2010)
12. A Dimakis, AD Sarwate, MJ Wainwright, Geographic gossip: efficient averaging for sensor networks. IEEE Trans. Signal Process. **56**(3), 1205–1216 (2008)
13. TC Aysal, ME Yildiz, AD Sarwate, A Scaglione, Broadcast gossip algorithms for consensus. IEEE Trans. Signal Process. **57**(7), 2748–2761 (2009)
14. D Ustebay, BN Oreshkin, MJ Coates, MG Rabbat, Greedy gossip with eavesdropping. IEEE Trans. Signal Process. **58**(7), 3765–3776 (2010)
15. KI Tsianos, MG Rabbat, in *International Conference on Distributed Computing in Sensor System*. Fast decentralized averaging via multi-scale gossip (ACM, Santa Barbara, Calif, USA, 2010), pp. 21–23
16. M Zheng, M Goldenbaum, S Stanczak, Y Haibin, in *IEEE Wireless Communication and Networking Conference*. Fast average consensus in clustered wireless sensor networks by superposition gossiping (IEEE, Paris, France, 2012), p. 14
17. M Chatterjee, SK Das, D Turgut, Wca: a weighted clustering algorithm for mobile ad hoc networks. J. Cluster Comput. **5**(2), 193–204 (2002)
18. DJ Baker, A Ephremides, The architectural organization of a mobile radio network via a distributed algorithm. IEEE Trans. Commun. **29**(11), 1694–1701 (1981)