Distributed Information Extraction in Wireless Sensor Networks using Multiple Software Agents with Dynamic Itineraries

Govind P. Gupta[1], Manoj Misra[1] and Kumkum Garg[2]
[1] Department of Computer Science & Engineering, Indian Institute of Technology
Roorkee-247667, India
[e-mail: {gpg09dec, manojfec}@iitr.ernet.in]
[2] Faculty of Engineering, Manipal University
Jaipur-302026, India
[e-mail: kgargfec@gmail.com]
*Corresponding author: Govind Gupta

Received May 21, 2013; revised December 1, 2013; accepted December 21, 2013; published January 29, 2014

Abstract

Wireless sensor networks are generally deployed for specific applications to accomplish certain objectives over a period of time. To fulfill these objectives, it is crucial that the sensor network continues to function for a long time, even if some of its nodes become faulty. Energy efficiency and fault tolerance are undoubtedly the most crucial requirements for the design of an information extraction protocol for any sensor network application. However, most existing software agent based information extraction protocols are incapable of satisfying these requirements because of static agent itineraries and large agent sizes. This paper proposes an Information Extraction protocol based on Multiple software Agents with Dynamic Itineraries (IEMADI), where multiple software agents are dispatched in parallel to perform tasks based on the query assigned to them. IEMADI decides the itinerary for an agent dynamically at each hop using local information. Through mathematical analysis and simulation, we compare the performance of IEMADI with a well known static itinerary based protocol with respect to energy consumption and response time. The results show that IEMADI provides better performance than the static itinerary based protocols.

Keywords: Wireless sensor network, software agent, dynamic itinerary, agent migration, information extraction

http://dx.doi.org/10.3837/tiis.2014.01.008

1. Introduction

A Wireless Sensor Network (WSN) is a distributed network, which consists of a large number of battery-powered sensor nodes and one or more sink nodes [1]. Sensor nodes are small electronic devices that have limited communication and computation capabilities. They are deployed for monitoring physical phenomenon such as temperature, light, humidity, vibration, sound and so on [1]. Generally, WSNs are deployed for specific applications, to accomplish certain objectives over a period of time. For this, it is crucial that the WSN continues to function over a period of time, even if some of its nodes become faulty. Energy efficiency and fault tolerance are undoubtedly the most crucial requirements for the design of an information extraction protocol for any WSN application. The design of a mobile software agent [2] based information extraction protocol should be carefully designed to optimize parameters like itinerary length, agent size and fault tolerant migration.

In recent years, the mobile agent computing paradigm has been introduced in WSNs for various purposes such as data aggregation and collection, topology discovery, network diagnostics and health monitoring, application reprogramming, etc. Successful functioning of these operations depends on the itinerary of the mobile software agent. A mobile software agent [2] is a software entity that can access the sensor nodes one by one, perform assigned task and fetch the results back to the sink node. When software agents are employed for information extraction task in the WSNs, the selection of the agent's itineraries is extremely vital because it significantly affects the overall energy consumption, latency and information extraction cost. Thus a scheme that plans optimal length itineraries with minimum energy consumption, response time and low complexity for the nodes is required.

Most of the proposed itinerary planning algorithms [3-10] for agent based information extraction use a centralized algorithm that is executed at the sink node and that computes itineraries for the agents prior their migration. This is known as a static itinerary. The most notable issues associated with these algorithms are size of the agent packet and disruption in agent migration if some nodes fail along the itinerary. In the static itinerary based approach, each agent needs to carry a pre-computed itinerary which grows as network size increases, thereby also increasing agent size. Since static itineraries are computed using a centralized algorithm that requires updated global network topology information at the sink node and it does not offer quick response to possible topology changes resulting from node failure.

In order to solve the above issues, we propose a distributed algorithm for the Information Extraction protocol based on Multiple software Agents with Dynamic Itineraries, called IEMADI, where multiple agents are deployed in parallel to perform tasks assigned to them. In IEMADI, an agent computes its itinerary dynamically at each hop using local information and offers efficient and fault tolerant agent migration within the network. IEMADI is suitable for both periodic as well as query based information extraction and is resilient to sensor node failures.

In this paper we evaluate the performance of IEMADI through mathematical analysis and extensive simulation experiments and compare it with a well-known static itinerary based protocol TBID [8] and a distributed information extraction protocol, called Directed Diffusion(DD) [22]. Simulation results show that IEMADI performs notably better than TBID and DD in terms of average energy consumption, response time, network lifetime, and success rate of agents' round trips, in the presence of node failures.

The remainder of this paper is organized as follows. In Section 2, we briefly review work related to the research presented herein. Section 3 presents the network model and the assumptions made. We describe the proposed protocol in Section 4. Section 5 gives the mathematical analysis. In Section 6, we present the performance evaluation of the proposed protocol. Finally, Section 7 concludes the paper and presents ideas for future work.

2. Related Work

In this section, we review existing mobile agent itinerary design algorithms for WSNs. WSNs usually have lower communication bandwidth than wired networks, due to which, sensory data traffic may exceed network capacity, resulting in collisions and energy wastage. To solve the problem of large sensory data traffic, Qi et al. [3] proposed a mobile agent based distributed sensor network (MADSN) for energy efficient data aggregation. The authors proved through mathematical and simulation studies that by sending mobile agents for data aggregation to the sensor nodes, a large amount of redundant data may be filtered at the sensor nodes, resulting in saving network bandwidth and reduced network latency.

In [4], Qi et al. proposed two heuristic algorithms, Local Closest first (LCF) and Global Closest First (GCF) for itinerary design of an agent performing data aggregation. In LCF, each mobile agent originates its itinerary from the sink and chooses a sensor node with the shortest distance to its present location as the next-hop node for data aggregation. In GCF, each agent chooses a sensor node with the shortest distance to the center of the sensing field as the next-hop node.

In [5], a genetic algorithm based approach is proposed for itinerary design of an agent. This algorithm derives a lower cost itinerary than LCF and GCF algorithms, but takes more time for itinerary calculation, which cannot be tolerated for time-sensitive applications. The algorithms proposed in [4] and [5] use only a single mobile agent deployed from the sink that successively visits all sensor nodes. The main drawback of these algorithms is that they are not scalable, i.e. their performance goes down as the network size increases. This is due to fact that the size of the mobile agent increases as it visits more and more sensor nodes, resulting in increase in overall energy consumption and the agent’s round trip time.

To overcome the drawback of single agent based itinerary design algorithms, Gavalas et al. [6] proposed a heuristic algorithm, called Near-optimal itinerary design (NOID). NOID calculates an appropriate number of agents that minimize overall communication cost and derives near optimal itineraries for each of them. NOID outperforms single agent based protocols proposed in [4] and [5], both in terms of data fusion cost and the overall response time. The main drawback of NOID is that it is not suitable for highly dynamic networks [6].

Cai et al. [7] proposed a genetic algorithm based Muti-Agent itinerary planning (GA-MIP) approach to address the drawback of the single agent based approach. The main drawback of this algorithm is that it requires a number of evolutionary iterations to determine efficient itineraries. This approach is time expensive and is not suitable for time-critical applications.

In [8], Konstantopoulos et al. proposed a tree based itinerary design (TBID) algorithm, which improves upon NOID. TBID uses a greedy like approach for building a number of trees and determines the itineraries of the agents using post order tree traversal with possible shortcutting approach. The main drawback of this algorithm is that it is not suitable for dynamic network where network topology frequently changes due to channel fading or node failures.

In [9], a clone based itinerary design (CBID) algorithm is proposed where sensor nodes are organized in a spanning tree rooted at the sink node. The sink node dispatches multiple agents, one for each branch of the tree, for data aggregation tasks. When an agent visits a node with two or more child nodes, it makes clones of itself, one for each child and sends it to its children. When all cloned agents return to the location of the master agent, they hand over their accumulated aggregated data to it. The main drawback of this algorithm is that an agent packet needs to carry additional information with it about when and where to clone, resulting in poor scalability.

In [10], Chen et al. proposed an itinerary planning algorithm, called Itinerary Energy Minimum for First-source selection (IEMF) which extends the LCF algorithm by using the estimated communication cost. IEMF selects the next-hop node for agent migration by considering minimum energy cost.

Mpitziopoulos et al. [11] proposed a framework for supporting the visually impaired people, called PROTECT that employs autonomous software agents for locating and informing them for potential risks. PROTECT is executed at the sink where it forms a number of itinerary trees and the final itineraries for the agents are derived by tree traversal method proposed in [12].

The algorithms proposed in [4-11] are centralized algorithms executed at the sink node, which generates static itineraries. There are many drawbacks of the centralized algorithm based itinerary design approach. First, it needs to collect periodically the location and residual energy information from all sensor nodes for calculating the updated itineraries, resulting in high communication overhead. Second, the agent may be unable to complete its round trip if some nodes die or become faulty along the itinerary. Third, each agent has to carry its itinerary, which increases its size as network size increases. Consequently, it takes more energy and time to transmit the agent packet. Finally, it does not offer quick response to possible topology changes.

Thus, static itinerary design does not suit highly dynamic sensor networks with large network size and is not efficient for those applications where network topology changes frequently and accurate topology information cannot be collected in advance at the sink node. Further, most of these algorithms fail to consider the requirement of fault tolerance in the context of mobile agent based WSNs.

In [13], Xu et al. proposed a dynamic itinerary design for getting progressive fusion accuracy. In their approach, an agent selects that sensor node from its neighborhood, which has maximum residual energy, consumes minimum energy for its migration and offers more information gain. This approach is designed for target tracking applications.

In [14], Gupta et al. proposed an agent based data dissemination protocol which decides the agent itinerary dynamically at each hop. The main weakness of this work is that the agent does not visit all nodes of the region.

Intanagonwiwat et al. [22] proposed a distributed information extraction protocol, called directed diffusion(DD). In DD, the sink disseminates interest message regarding the information to be extracted and each node records the neighboring node from which the interest message is received. Upon receiving the interest message, each node initiates the gradient setup phase in which it maintains a vector containing the next hope that has to be used to transmit the result of the query back to the sink node. The main drawback of DD is that it does not eliminate redundant data transmissions.

The advantages of the dynamic itinerary based agent migration approach are that it offers quicker response to possible topology changes and uses local information for computation of an agent's itinerary better suited for resource-constrained sensor nodes. In addition, the size of an agent packet is noticeably smaller because it does not carry a pre-computed itinerary; instead, it decides the itinerary on the fly at each hop. Thus, the dynamic itinerary based agent migration approach suits highly dynamic sensor networks. Keeping this in mind, we propose a distributed algorithm for information extraction based on multiple agents with dynamic itineraries that is suitable for periodic as well as query based information extraction and resilient to sensor node failures.

3. Network Model and Assumptions

This work considers a sensor network consisting of N sensor nodes uniformly distributed in a circular monitoring area of radius R, similar to the model presented in [8]. A sink node is placed at the center of monitoring area. We assume each sensor node is static and knows its coordinates (x, y) in a two dimensional plane by means of some localization algorithms [15-17] where no GPS receiver is required. Each node knows the coordinates of the sink as well. Using these coordinates, each node *x* calculates its polar coordinate, denoted by (x.*r*, *x*.*theta*), where x.*r* is the Euclidean distance of node from the sink and x.*theta* is an angle between the polar axis and the line connecting the sink and node *x*. Since sensor nodes are stationary, the setup of their polar coordinates is a one-time task [18]. We assume that node density in the network is sufficiently high to ensure migration of agent packets along the rings [19].

Table 1. Data structures/variables used in processing

Data structures	Meaning
nodeID	Identifier for the node
nodeEnergy	Remaining energy of a node
ringNo	Identifier for the concentric ring
wedgeNo	Identifier for the wedge
polarCoord	Polar coordinate *(r, theta*) of a node
W_i	Wedge number
k	Number of the agent
r_{max}	Maximum transmission range of the node
maxRingNo	Maximum ring rumber
moveFlag	Flag variable, if true means agent moves left to right , otherwise right to left direction within each ring
V_x	Neighbor table for node x
W	Number of wedge in which sensing field is divided.
α	angular width of the wedge
R	radius of monitoring area
ngbPkt	Neighbor discovery packet
AgentPkt	Software agent packet
E_threshold	Threshold node energy
ma_direction	Flag variable which decides direction of agent's movement.

4. Proposed Protocol

In this section, we present a distributed algorithm for the information extraction (IEMADI) protocol using multiple software agents with dynamic itineraries, for query based information extraction applications for WSNs. The operation of IEMADI consists of two phases: (1) initialization and (2) agent migration. This section describes each of these phases in detail. The data structures used in the design are given in **Table 1**.

4.1 Initialization

In the initialization phase, each node sets up its ringNo and wedgeNo using its polar coordinates. The width of the first ring is $w.r_{\max}$, where w is a constant in the range [0.5, 1.0] and $r_{\max}$ is the maximum communication range of any node. The width of all other rings is $r_{\max}/2$. The pseudo code for the setup of ringNo and wedgeNo is given in **Fig. 1**.

Next, each node starts the neighbor discovery process, where it creates and broadcasts an *ngbPkt* packet. The *ngbPkt* packet contains five fields: nodeID, nodeEnergy, ringNo, wedgeNo, and polarCoord, where nodeID is the identifier for the node, nodeEnergy is the remaining energy of the node, ringNo is the ring identifier to which node belongs, wedgeNo is the wedge identifier to which node belongs, and polarCoord is the polar coordinate of the node.

If a node receives an *ngbPkt* packet from its own wedge, it updates its neighbor table with the values in the nodeID, ringNo, wedgeNo, nodeEnergy and polarCoord fields. The structure of neighbor table is shown in **Fig. 2**. If a node x receives an *ngbPkt* packet from another wedge, it sets x.boundaryNode to true and informs its boundary state to its neighbors by broadcasting a beacon packet. At the end of this phase, each node knows its ringNo, wedgeNo and also all its neighbors within its transmission region in its wedge.

Algorithm: Setup for ringNo and wedgeNo for each node

```
Begin
For sensor node x :
Input: polar coordinate(x.r, x.theta)
        ω : a constant in range of [0.5,1.0]
Output: ringNo, wedgeNo, neighbor table
Initially: x.ringNo=0; x.wedgeNo=0; i=2;
1: maxRing = $\lceil 1+(R-w.r_{\max})/(r_{\max}/2) \rceil$;
2: if (x.ringNo== 0 && x.wedgeNo == 0)
3:     x.wedgeNo = $\lceil x.theta/\alpha \rceil$
4:     if ( $x.r \le w.r_{\max}$ )
5:         x.ringNo = 1;
6:     else
7:       while (i < maxRing)
8:         if $((w.r_{\max} + (i-2).\frac{r_{\max}}{2}) < x.r \le (w.r_{\max} + (i-1).\frac{r_{\max}}{2}))$
9:               x.ringNo = i;
10              break;
11:        end if
12:        i++;
13:    end while
14:  end else
```

```
15: end if
End
```

Fig. 1. Algorithm for the setup of ringNo and wedgeNo for node x.

nodeID	nodeEnergy	ringNo	wedgeNo	polarCoord	boundaryNode

Fig. 2. Structure of a neighbor table

Header					
ma_ID (1 byte)	ma_direction (1 bit)	moveFlag (1 bit)	maxRingNo (6 bits)	Query logic	Payload

Fig. 3. Structure of the mobile agent packet

4.2 Agent Migration Process

In this phase, the sink node creates and dispatches software agents to the first ring of each wedge W_i, where i=1, 2… *W*. Here *W* is the number of wedge regions that depends on angular width (α) of the wedge region. The structure of the mobile agent packet is shown in **Fig. 3**. The header of a mobile software agent packet contains four fields: ma_ID, ma_direction, moveflag and maxRingNo, where ma_ID is the identifier for the agent, ma_direction is a flag variable which shows the direction of the agent migration from one ring to another, moveFlag is a flag variable which shows the direction of agent migration within each ring, and maxRingNo is the maximum ring number which is computed by the sink node since it knows the radius of the monitoring area (R) and width of each ring.

The agent migration process is divided into two sub-phases. In the first sub-phase, an agent migrates through boundary nodes with minimum theta value from the inner to the outer ring. In the second phase, data aggregation starts from the outer ring and the agent migrates within the wedge in a spiral order from the outer to inner ring towards the sink, as shown in **Fig. 4**. In the initial traversal from sink to a node in outermost ring, an agent does not gather data. It starts gathering data from the node of the outer ring. The advantages of this approach is that it avoid unnecessary carry of data from the nodes during initial traversal from sink to a node in the outermost ring, as a result itinerary cost of an agent decreases.The flowcharts for agent migration process is given in **Fig. 5**.

During agent migration, if next hop in the ring does not meet threshold requirements, an agent selects an alternate node of the adjacent ring to complete the visit of the remaining nodes of the ring. If there is no alternate node available with required threshold, an agent simply jumps to the inner ring to traverse their nodes. If there is no neighbor node available in its neighbor table which satisfies the threshold condition, agent will not be move further and information extraction process is abandoned.

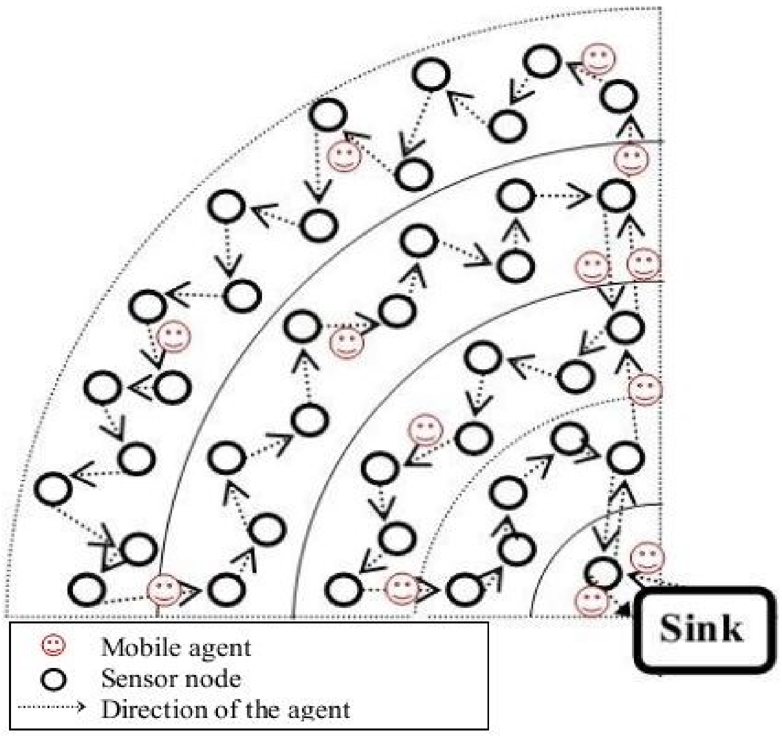


Fig. 4. Agent migration within wedge

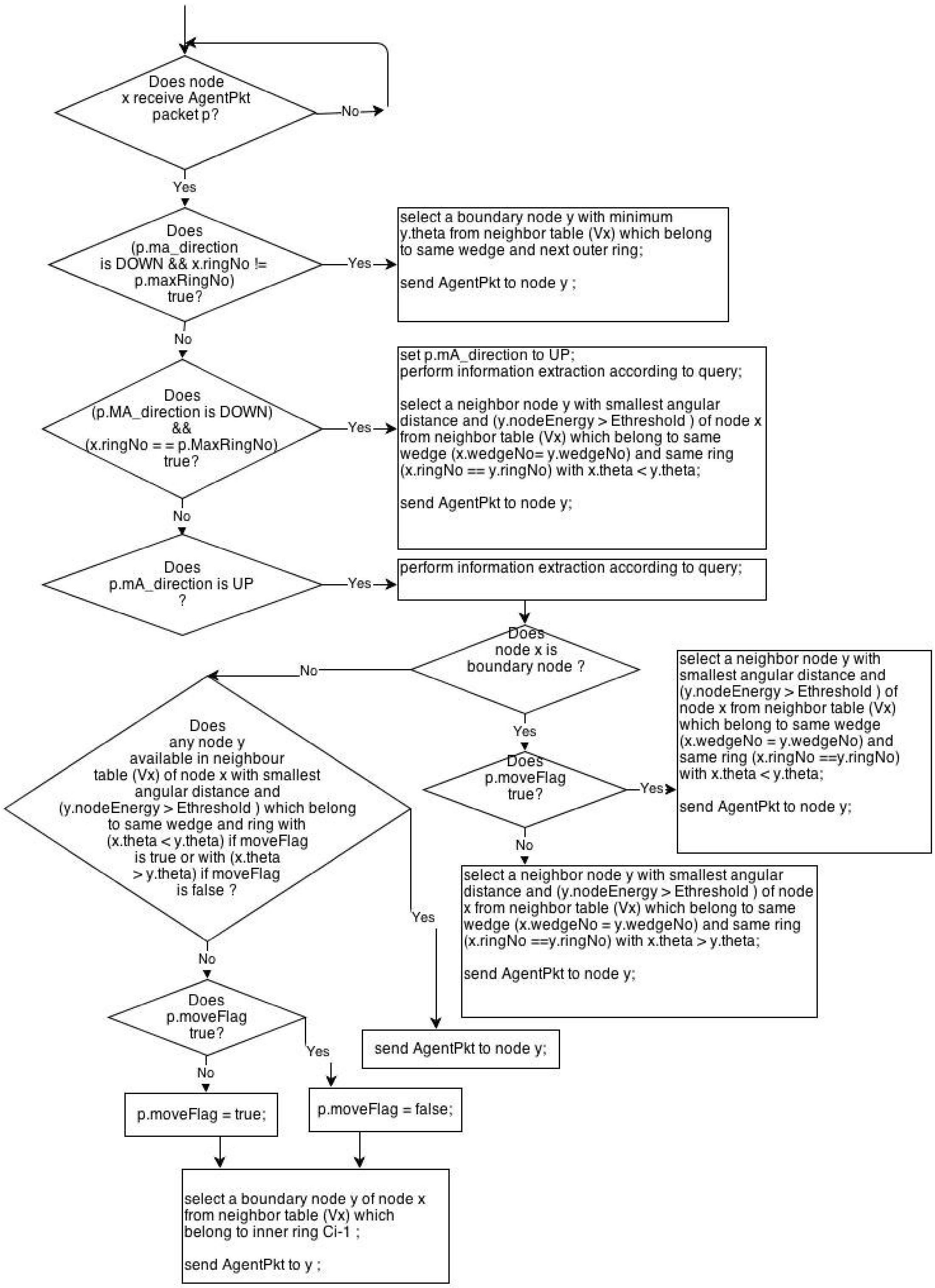

Does node x receive AgentPkt packet p?
No
Yes
Does (p.ma_direction is DOWN && x.ringNo != p.maxRingNo) true?
Yes
select a boundary node y with minimum y.theta from neighbor table (Vx) which belong to same wedge and next outer ring;
send AgentPkt to node y ;
No
Does (p.MA_direction is DOWN) && (x.ringNo = = p.MaxRingNo) true?
Yes
set p.mA_direction to UP;
perform information extraction according to query;
select a neighbor node y with smallest angular distance and (y.nodeEnergy > Ethreshold) of node x from neighbor table (Vx) which belong to same wedge (x.wedgeNo= y.wedgeNo) and same ring (x.ringNo == y.ringNo) with x.theta < y.theta;
send AgentPkt to node y;
No
Does p.mA_direction is UP ?
Yes
perform information extraction according to query;
Does node x is boundary node ?
No
Yes
Does p.moveFlag true?
Yes
select a neighbor node y with smallest angular distance and (y.nodeEnergy > Ethreshold) of node x from neighbor table (Vx) which belong to same wedge (x.wedgeNo = y.wedgeNo) and same ring (x.ringNo ==y.ringNo) with x.theta < y.theta;
send AgentPkt to node y;
No
select a neighbor node y with smallest angular distance and (y.nodeEnergy > Ethreshold) of node x from neighbor table (Vx) which belong to same wedge (x.wedgeNo = y.wedgeNo) and same ring (x.ringNo ==y.ringNo) with x.theta > y.theta;
send AgentPkt to node y;
Does any node y available in neighbour table (Vx) of node x with smallest angular distance and (y.nodeEnergy > Ethreshold) which belong to same wedge and ring with (x.theta < y.theta) if moveFlag is true or with (x.theta > y.theta) if moveFlag is false ?
Yes
send AgentPkt to node y;
No
Does p.moveFlag true?
Yes
No
p.moveFlag = true;
p.moveFlag = false;
select a boundary node y of node x from neighbor table (Vx) which belong to inner ring Ci-1 ;
send AgentPkt to y ;

Fig. 5. Flowchart for the agent migration algorithm executed by node x

5. Mathematical Analysis

In this section, we evaluate the performance of our proposed protocol in terms of energy consumption and response time. These two metrics reflect the efficiency of agent based protocols. Assume N nodes are uniformly distributed over a circular sensing field which is divided into W wedge regions, each having an equal number of nodes n.

5.1 Energy Consumption Analysis

We first calculate the total energy consumed (E_{total}) in one round of agent based information extraction. E_{total} depends on the number of agents deployed, size of the agent packet and the number of nodes visited by each agent. In WSNs, information extraction process is performed in rounds. In each round of information extraction, the sink node dispatches a number of software agents where it visits a set of nodes, perform data aggregation and carry relevant information with it and return to the sink. This whole cycle is called one round of information extraction. Let size of agent packet at i^{th} node be $S_{agent_{size_i}}$ Then, E_{total} can be expressed by the following formula:

$$E_{total} = W.\left\{E_{extra} + \sum_{i=1}^{n}((S_{agent_{size_{i-1}}} * E_{Rx}) + E_{DA} + (S_{agent_{size_i}} + E_{Tx}))\right\} \tag{1}$$

Here, $n = N/W$. E_{Rx} and E_{Tx} is energy consumed in receiving and transmitting a byte of message respectively. E_{DA} is the energy consumed in processing at each node. E_{extra} is total energy consumed in forwarding an agent from the sink to the starting sensor node from where agent starts data collection.

Let the size of agent packet when dispatched from the sink node be S_{ini} bytes and on average a d bytes increase in agent size occurs at each sensor node after query based information. The size of agent packet at i^{th} node, $S_{agent_{size_i}}$ is given by:

$$S_{agent_{size_i}} = S_{ini} + i * d \tag{2}$$

For IEMADI protocol, initial size of the agent packet $S_{ini\_forIEMADI}$, is given by:

$$S_{ini\_forIEMADI} = S_{header} + S_{proc\_\text{logic}} + S_{payload} \tag{3}$$

Here S_{header} is size of header. From **Fig. 3**, we can see that header of the agent packet contains four fields and the size of header is 2 bytes. S_{proc_logic} is the size of processing code carried with an agent and $S_{payload}$ is the size of payload, initially value of $S_{payload}$ is zero, when an agent dispatched by the sink. Equation 3 can be rewritten as:

$$S_{ini\_forIEMADI} = 2 + S_{proc\_\text{logic}} \tag{4}$$

Therefore, from equation 2 and 4, for IEMADI, the size of the agent packet at the i^{th} node, $S_{IEMADI\_agent_{size_i}}$ is given by:

$$S_{IEMADI\_agent_{size_i}} = 2 + S_{proc\_\text{logic}} + i * d \tag{5}$$

For IEMADI protocol, extra total energy consumed ($E_{extra\_forIEMADI}$) is equal to total energy dissipation in forwarding an agent from sink to the outer ring of the wedge. $E_{extra\_forIEMADI}$ is given by:

$$E_{extra\_forIEMADI} = \sum_{j=1}^{\max RingNo} S_{ini\_forIEMADI} * (E_{Rx} + E_{Tx})$$

$$= \max RingNo * (2 + S_{proc\_\text{logic}}) * (E_{Rx} + E_{Tx}) \qquad (6)$$

From equation 6, it is evident that value of $E_{extra\_forIEMADI}$ is constant for each wedge. Hence, from equation 1, 5 and 6, total energy consumed in one round of query based information extraction for IEMADI protocol (E_{total_IEMADI}) is given by:

$$E_{total\_IEMADI} = W * \left\{ \begin{array}{l} E_{extra\_forIEMADI} + \\ \sum_{i=1}^{n} ((2 + S_{proc\_\text{logic}} + (i-1) * d) * E_{Rx} + E_{DA} + (2 + S_{proc\_\text{logic}} + i * d) * E_{Tx} \end{array} \right\}$$

(7)

Here $E_{extra\_forIEMADI}$ is constant for all wedges. From equation 7, it is evident that total energy consumed in one round of query based information extraction depends on agent size and number of nodes visited by it. Since wedge angular width $\alpha = (360/W)$ and $W = (N/n)$, we can simplify equation 7 by putting the value of *W, n* and $E_{extra\_forIEMADI}$, as given by:

$$E_{total\_IEMADI} = N * \left\{ \begin{array}{l} \frac{360}{(\alpha * N)} * \{\max RingNo * (2 + S_{proc\_\text{logic}}) * (E_{Rx} + E_{Tx})\} + \\ \left\{ ((2 + S_{proc\_\text{logic}}) + (d/2) * ((\frac{\alpha . * N}{360}) - 3 + \frac{720}{\alpha * N}) * E_{Rx} \right\} \\ + \{E_{DA}\} \\ + \left\{ ((2 + S_{proc\_\text{logic}}) + (d/2) * ((\frac{\alpha * N}{360}) - 1) * E_{Tx} \right\} \end{array} \right\}$$

(8)

The optimal value of α can be obtain by taking differential of equation 8 (E_{total_IEMADI}) with respect to α. We get following expression:

$$\alpha = \frac{360}{N} * \sqrt[2]{\frac{2 * \{\max RingNo * (2 + S_{proc\_\text{logic}})\}}{d}} \qquad (9)$$

For network scenario where N=300 nodes are deployed in a circular monitoring area of radius 100m. The value of S_{proc_logic} and d is 1000 and 100 bytes respectively. If $E_{Rx} = 0.0074$

joules per byte and $E_{Tx} = 0.0074$ joule per byte, we get minimum value of α is 15.19 by putting these values in equation 9. For this network scenario, **Fig. 6** plots the effect of α on the average energy consumption (E_{total_IEMADI}/N) by using the equation 8.

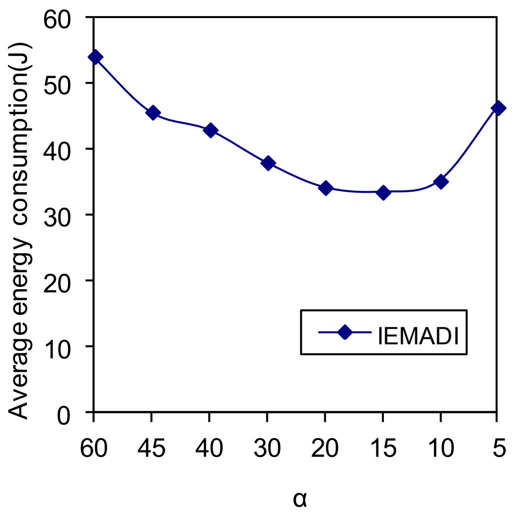


Fig. 6. Average energy consumption vs. α for IEMADI protocol

In a static itinerary based protocol, each agent carries a pre-computed itinerary list. Initial size of the agent packet for static itinerary based protocol such as TBID, $S_{ini\_forTBID}$, is given by:

$$S_{ini\_forTBID} = S_{header} + S_{proc\_\text{logic}} + S_{itinerary\_list} + S_{payload} \qquad (10)$$

Here, value of S_{header} is 1 byte since it contains only one field: agent identifier which occupies one byte. $S_{itinerary\_list}$ is size of itinerary list. Let an agent visit *n* nodes. If the entry for each node in the itinerary list takes *l* bytes, the size of itinerary list ($S_{itinerary\_list}$) will be *n*l*. Initially value of $S_{payload}$ will be zero. Hence equation 10 can be rewritten as:

$$S_{ini\_foTBID} = 1 + S_{proc\_\text{logic}} + (n*l) \qquad (11)$$

The size of the agent packet at the i^{th} node, $S_{TBID\_agent_{size_i}}$ for TBID can be calculated from equation 2 and 10 and is given by:

$$S_{TBID\_agent_{size_i}} = 1 + S_{proc\_\text{logic}} + (n*l) + (i*d) \qquad (12)$$

For TBID protocol, extra total energy consumed ($E_{extra\_forTBID}$) is equal to total energy dissipation in forwarding an agent from sink to the leftmost leaf node of the itinerary tree.

$E_{extra\_forTBID}$ is given by:

$$E_{extra\_forTBID} = \sum_{j=1}^{h} \{S_{ini\_forTBID} * (E_{Rx} + E_{Tx})\}$$
$$= h*(1+S_{proc\_\text{logic}} + (n*l)*(E_{Rx} + E_{Tx}) \quad (13)$$

Here, h is height of itinerary tree. From equation 13, it is evident that value of $E_{extra\_forTBID}$ is constant for each tree. Hence, from equation 1, 12 and 13, total energy consumed in one round of information extraction for TBID protocol (E_{total_TBID}) can be expressed by the following formula:

$$E_{total\_TBID} = W * \left\{ \begin{array}{l} E_{extra\_forTBID} * \\ \sum_{i=1}^{n} \left\{ \begin{array}{l} ((1+S_{proc\_\text{logic}} + (n*l) + (i-1)*d)*E_{Rx} + E_{DA} + \\ (1+S_{proc\_\text{logic}} + (n*l) + i*d)*E_{Tx}) \end{array} \right\} \end{array} \right\} \quad (14)$$

Table 2. Energy consumption analysis

Protocol	Energy Consumption
IEMADI	$W * \left\{ \begin{array}{l} E_{extra\_forIEMADI} + \\ \sum_{i=1}^{n} ((2+S_{proc\_\text{logic}} + (i-1)*d)*E_{Rx} + E_{DA} + (2+S_{proc\_\text{logic}} + i*d)*E_{Tx} \end{array} \right\}$
TBID	$W * \left\{ \begin{array}{l} E_{extra\_forTBID} * \\ \sum_{i=1}^{n} \left\{ \begin{array}{l} ((1+S_{proc\_\text{logic}} + (n*l) + (i-1)*d)*E_{Rx} + E_{DA} + \\ (1+S_{proc\_\text{logic}} + (n*l) + i*d)*E_{Tx}) \end{array} \right\} \end{array} \right\}$

Table 2 shows the mathematical expressions for energy consumption for IEMADI and TBID protocols. It is evident that IEMADI consumes less energy because the agent packet does not carry pre-computed itinerary list, unlike in a static itinerary based protocol such as TBID, where it does. Since size of itinerary list grows as network size increases, higher energy consumption results in case of static itinerary based protocols.

5.2 Response Time Analysis

Response time ($T_{response\_time}$) is the time interval from the moment the first agent is dispatched from the sink node to the time all the agents return back to it. Agent migration time ($T_{mig_{i,i+1}}$) from *i*th node to *(i+1)*th node depends on the communication bandwidth (bytes per sec) and the current size of the agent. If an agent visits *n* nodes in its itinerary, $T_{response\_time}$ is given by:

$$T_{response\_time} = W.T_{inst} + T_{extra} + \sum_{i=1}^{n} (T_{proc} + T_{mig_{i,i+1}} + T_{prop}) \tag{15}$$

Here, T_{inst} is time spent in instantiation of an agent packet at the sink. The value of T_{inst} is constant for each agent. T_{extra} is time spent in forwarding the agent from the sink to the starting sensor node from where the agent starts data collection. The value of T_{extra} is constant for each agent. T_{proc} is the time spent by the agent to complete its assigned task at a node. The value of T_{proc} is constant for each agent. T_{prop} is the agent propagation time and depends on the physical distance travel by it. Let the agent packet be transmitted over a communication channel of bandwidth B bytes per sec. For dynamic itinerary based protocol, agent migration time ($T_{mig_{i,i+1}}$) from *i*th node to *(i+1)*th is given by[20]:

$$T_{mig_{i,i+1}} = (S_{ini} + i * d) / B \tag{16}$$

From equations 15 and 16, response time ($T_{response\_time\_IEMADI}$) for IEMADI is given by:

$$T_{response\_time\_IEMADI} = W.T_{inst} + T_{extra\_forIEMADI} + \sum_{i=1}^{n} (T_{proc} + ((S_{ini\_forIEMADI} + i * d) / B) + T_{prop}) \tag{17}$$

By putting the value of $S_{ini\_forIEMADI}$ from equation 4 into equation 17, equation 17 can be rewritten as:

$$T_{response\_time\_IEMADI} = W.T_{inst} + T_{extra\_forIEMADI} + \sum_{i=1}^{n} (T_{proc} + ((2 + S_{proc\_\mathrm{logic}} + i * d) / B) + T_{prop}) \tag{18}$$

Similarly, from equation 12 and 16, for a static itinerary based protocol, agent migration time ($T_{mig_{i,i+1}}$) from *i*th node to *(i+1)*th is given by:

$$T_{mig_{i,i+1}} = (1 + S_{proc\_\mathrm{logic}} + (n * l) + i * d) / B \tag{19}$$

Therefore, response time ($T_{response\_time\_TBID}$) for a static itinerary based protocol such as TBID is given by:

$$T_{response\_time\_TBID} = W.T_{inst} + T_{extra\_forTBID} + \sum_{i=1}^{n} (T_{proc} + ((1 + S_{proc\_\mathrm{logic}} + (n * l) + i * d) / B) + T_{prop}) \tag{20}$$

Table 3. Response time analysis

Protocol	Response Time
IEMADI	$W.T_{inst} + T_{extra\_forIEMADI} + \sum_{i=1}^{n} (T_{proc} + ((2 + S_{proc\_logic} + i * d) / B) + T_{prop})$
TBID	$W.T_{inst} + T_{extra\_forTBID} + \sum_{i=1}^{n} (T_{proc} + ((1 + S_{proc\_logic} + (n * l) + i * d) / B) + T_{prop})$

Table 3 shows the mathematical expressions for response time for IEMADI and TBID protocols. It is seen that the response time $T_{response\_time}$ is dominated by the agent size and the length of the agent's itinerary.

6. Performance Evaluation

This section presents the performance analysis of the IEMADI under different network scenarios and compares it with TBID [8] and DD[22]. We vary the network size, wedge angle, as well as percentage of faulty nodes and evaluate the average energy consumption, response time, network lifetime and success rate of the agent's round trip under different network scenarios. All the simulation results with 95% confidence interval are shown in this section.

Table 4. Simulation parameters

Parameter	Value
Terrain Shape	Circular
Terrain radius	150m
Number of nodes	50-300
Node density	0.0052 nodes/sq. m.
Transmission range	25m
Simulation time	3000 sec
Initial battery power	18720J
Size of query logic (agent's processing code)	1000 bytes
Average byte accumulated by an agent at each node (d)	100 bytes
Agent execution time (T_{proc})	55ms
MAC protocol	T-MAC
Confidence level for results	95%

6.1 Simulation Setup

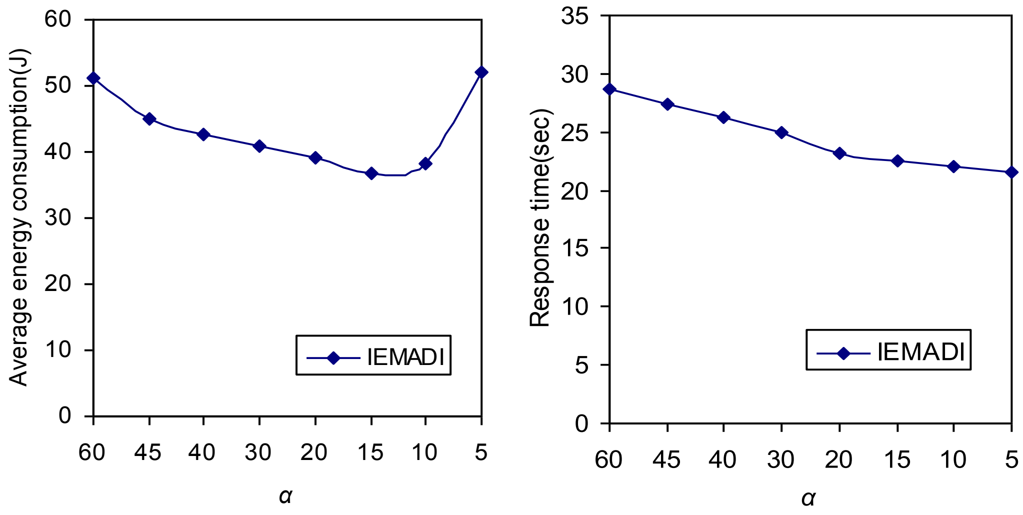
We use a discrete event simulator Castalia3.2 [21] that is based on OMNeT++ platform, for all evaluations. We adopt the network model used in [8] and [18], in which 50-300 nodes are uniformly distributed within a circular sensing field of different radius to maintain the same node density. For simplicity, the sink node is placed at the center of the sensing field. The transmission radius of sensor nodes is 25m. All sensor nodes have the same initial battery power 18720J (equivalent to one 2AA battery). We have used T-MAC protocol at MAC layer to simulate the proposed scheme. In our experiments, T-MAC protocol takes care of disconnection issues arises between the adjacent sensor nodes due to nodes'sleep. The simulation parameters are listed in **Table 4**. We used the following metrics for performance evaluation:

1. **Average Energy consumption (E_{avg}):** The amount of energy consumed by a node in performing the required network operation in each round of data collection.
   E_{avg} = (Sum of energy consumption at each node) / (number of nodes)
2. **Response time($T_{response\_time}$):** The average time interval required to complete one round of data collection.
   $T_{response\_time} = (\sum_{i=1}^{p} \max \{T_{i,j}\}_{for j=1..k}) / p$. Here, $T_{i,j}$ is the time taken by the j^{th} agent of i^{th} round to complete its data aggregation task, k is number of agents and p is the round number.
3. **Network lifetime:** Lifetime of a sensor network basically depends on the energy dissipation of individual sensor nodes. It is defined as time duration until the first sensor node in the network dies due to battery exhaustion. Since agents are used for collecting the aggregated data from the network, the network lifetime is calculated as total number of successful rounds of data collection by the agents.
4. **Success rate of agent's round trip:** The number of agents received by the sink as a percentage of the total number of agents dispatched by it.

6.2 Impact of Angular Width (*α*) Variation

Fig. 7(a) and **(b)** show the effects of the increase in angular width (α) of the wedge on the average energy consumption and response time respectively. For this experiment, we use a network scenario where 300 nodes are uniformly deployed with fixed node density (0.0052 nodes/ sq. m.) and value of the payload (d) is 100 bytes. From the **Fig. 7(a)**, we can observe that average energy consumption decreases as the value of α decreases from 60 to 15, but further increases as the value of α decreases from 15 to 5. Based on this result, we can conclude that when the value of α is below a threshold value, wedge region will become very narrow and an agent travels in line from sink to outer ring and return backs with almost using same path. Due to this reason, an agent visits twice each node of the wedge. As a result each node has to forward the agent packet two times, this increases average energy consumption. For the first case when value of α decreases from 60 to 15, size of wedge region decreases and each agent visits nodes of the wedge except boundary nodes only one time. However, number of nodes of the wedge visited by an agent decreases with decrease in value of α from 60 to 15, as a result average energy consumption decreases due to payload size of the agent packet decreases.

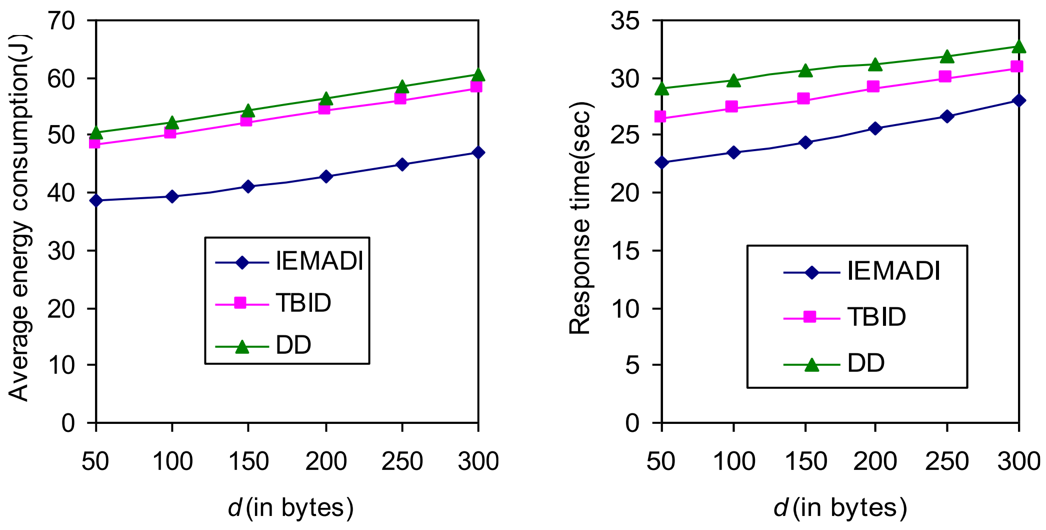
From the **Fig. 7(b)**, we can observe that response time decreases as the value of *α* decreases from 60 to 15, but almost same as the value of *α* decreases from 15 to 5. This is because as the value of *α* decreases, the number of nodes visited by the agents also decreases, resulting in decrease in response time.



(a) Average energy consumption (b) Response time

Fig. 7. Performance analysis of IEMADI with varying value of α

6.3 Impact of payload size (*d*) variation



(a) Average energy consumption (b) Response time

Fig. 8. Performance comparison of IEMADI and TBID with varying value of payload size (*d*)

Fig. 8(a) and **(b)** show the effects of the increase in the value of *d* on the average energy consumption and response time respectively. From the **Fig. 8(a)**, we can observe that as the value of *d* increases, average energy consumption also increases since the agent packet size increases. In the IEMADI, average energy consumption increases slower compare to IEMADI when the value of *d* increases. This is because IEMADI uses dynamic itineraries for the agents in which the agent does not carry itinerary list which makes its size smaller than the agent used in TBID. In addition, IEMADI uses shorter itinerary length for the agent compare to TBID. DD consumes higher energy than IEMADI and TBID. This is due to fact that each node sends their raw data to the sink and intermediate nodes not only transmit its data but also transmit its children's data as well.

From the **Fig. 8(b)**, we can observe that response time grows as the value of *d* increases. In the IEMADI, response time increases slower compare to TBID when the value of *d* increases due to the smaller size of agent packet and itinerary length used in IEMADI compare to TBID. However, DD takes more response time than IEMADI and TBID as the payload increases. The reason for this is same as discussed in above paragraph.

6.4 Impact of Network Size Variation

Fig. 9(a) and **(b)** show the result of the comparison of IEMADI with TBID and DD in terms of average energy consumption with varying number of nodes for d=100 and d=200, respectively. From **Fig. 9**, we observe that the average energy consumption increases with the increase in network size because of increase in agent's itinerary and size, which increases as the number of nodes increases. IEMADI consumes approximately 12% less energy compared to TBID. This is due to the use of dynamic agent migration, where the agents need not carry the pre-computed itinerary. As a result the size of the agent packet becomes smaller and saves communication energy. However, DD consumes more energy than IEMADI and TBID. This is because each node in DD sends their data to the sink through multihop transmission towards the sink and intermediate nodes are also work as relay nodes for forwarding the data packets of its children nodes.

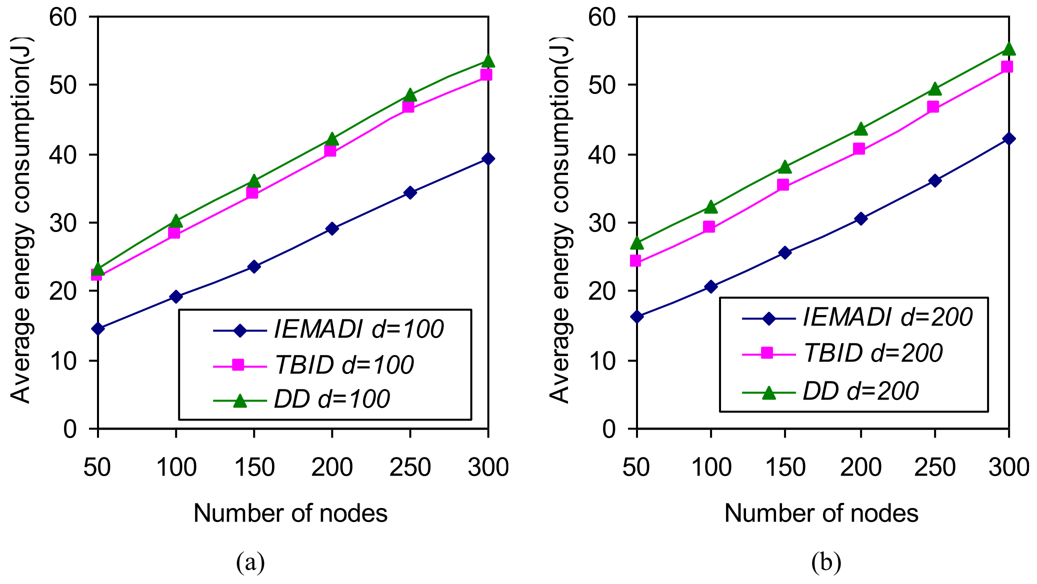


Fig. 9. Performance comparison of IEMADI, TBID, and DD in terms of average energy consumption

Fig. 10 (a) and **(b)** shows the comparison of IEMADI with TBID and DD, in terms of response time with varying number of nodes for *d*=100 and *d*=200, respectively. We observe that the response time increases with increase in network size, because of increase in agent's itinerary and size. Since an agent accumulates more data as the network size increases, it takes more time to transmit the agent. IEMADI takes approximately 2% less time as compared to TBID to complete one round. This is again due to the dynamic agent migration scheme, where the agents need not carry pre-computed itineraries; as a result the size of agent packet becomes smaller and thus reduces response time. Response time for DD is less when number of nodes are less compare to TBID. When number of nodes increases, response time for DD is increases compare to TBID.

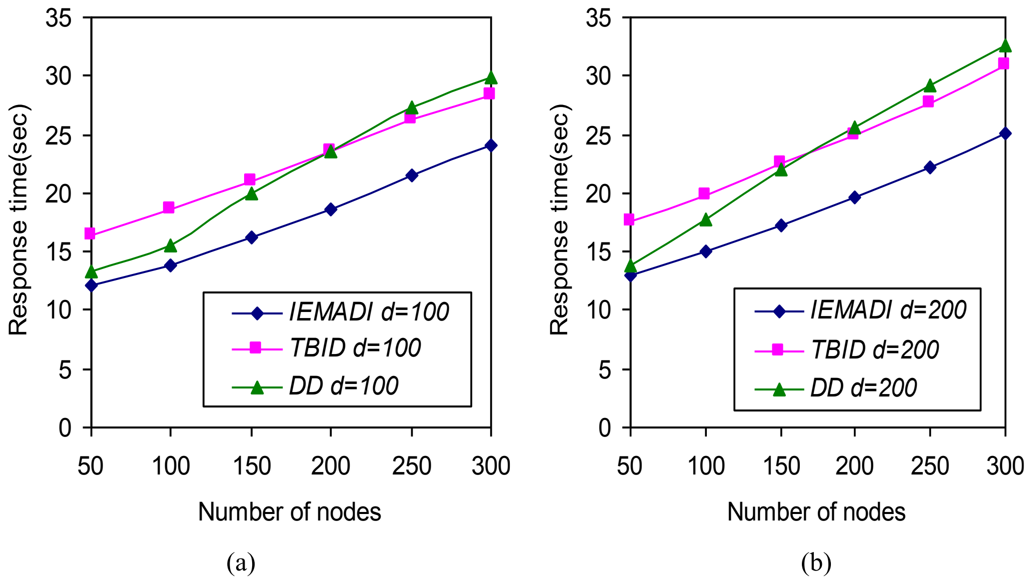


Fig. 10. Performance comparison of IEMADI, TBID, and DD in terms of response time

Fig. 11 gives the comparison of IEMADI with TBID and DD, in terms of network lifetime with varying number of nodes. It is evident that network life time decreases with the increase in network size because of increase in agent's itinerary length and size. This is due to fact that an agent has to visit more nodes as the network size increases. The network life time of IEMADI is approximately 28% higher than TBID. This is because of the calculation of dynamic itinerary at each hop and smaller size of agent. As a result, it takes less energy to transmit an agent and thus enhances network lifetime. However, in TBID, sink needs to collect topology information periodically, resulting in more energy consumption. In addition, the agents carry pre-computed itineraries which increase its size, hence take more energy of the node for agent migration. The network life time of IEMADI is approximately 30% higher than DD.

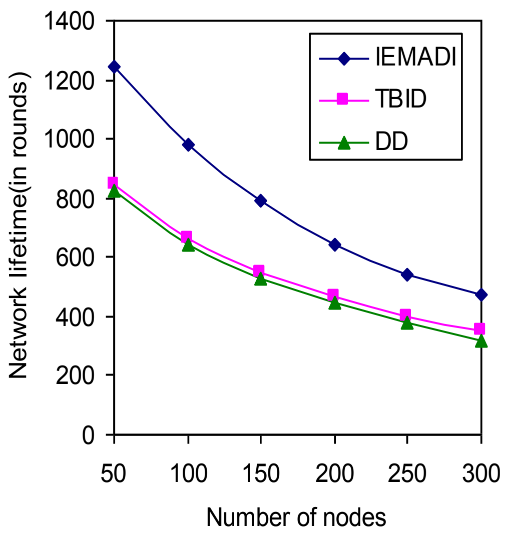


Fig. 11. Network lifetime vs. number of nodes

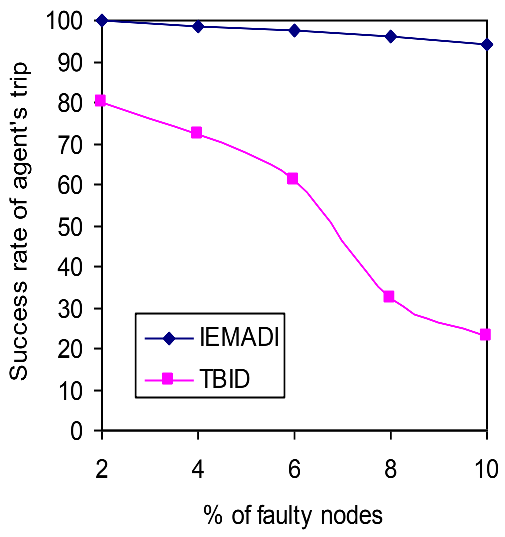


Fig. 12. Success rate of agents' trip in presence of faulty nodes

6.5 Impact of Node Failures

Fig.12 gives a comparison of IEMADI with TBID in terms of success rate of agents' round trip with varying percentage of faulty nodes present in the network. It is seen that success rate in IEMADI is notably higher than in TBID. This is because IEMADI calculates agents' itineraries at each hop dynamically using local information; as a result it bypasses faulty nodes. However, in TBID, an agent carries a pre-computed itinerary and strictly follows the node sequence in it. For this reason, it is unable to complete its round trip, if some nodes fail along the itinerary.

7. Conclusion

This paper has proposed an information extraction protocol, based on multiple mobile agents with dynamic itineraries, for reducing the impact of faulty nodes on the agent's migration and enhancing network lifetime. We studied the performance of the proposed protocol IEMADI through mathematical analysis and extensive simulation experiments, and compared it with a static itinerary based protocol, TBID and a distributed information extraction protocol, DD. The mathematical analysis and simulation results confirmed that IEMADI performs notably better in terms of average energy consumption, response time, network lifetime and success rate of agents' round trip, as compared to TBID and DD. In future, we plan to extend this work for mobile wireless sensor networks and study the impact of node mobility on the agent migration.

References

[1] Akyildiz I.F., Su W., Sankarasubramaniam Y., Cayirci E., "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, 2002. Article (CrossRef Link)

[2] Chen, M., Gonzalez, S., Leung, V., "Applications and design issues for mobile agents in wireless sensor networks," *IEEE Wireless Communication*, 14, (6), pp. 20–26, 2007. Article (CrossRef Link)
[3] Qi H., Iyengar S. S., and Chakrabarty K., "Multi-Resolution Data Integration Using Mobile Agents in Distributed Sensor Networks," *IEEE Transactions on Systems, Man, and Cybernetics*, Part C, vol. 31, no. 3, pp. 383–391, Aug. 2001. Article (CrossRef Link)
[4] Qi H. and Wang F., "Optimal Itinerary Analysis for Mobile Agents in Ad Hoc Wireless Sensor Networks," in *Proc. 13th International Conference on Wireless Communications (Wireless'2001)*, vol. 1, Calgary, Canada, pp. 147–153, Jul. 2001.
[5] Wu Q., Rao N., Barhen J., Iyengar S., Vaishnavi V., Qi H., Chakrabarty K.,, "On Computing Mobile Agent Routes for Data Fusion in Distributed Sensor Networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 6, pp. 740–753, Jun. 2004. Article (CrossRef Link)
[6] Gavalas, D., Mpitziopoulos, A., Pantziou, G., Konstantopoulos, C., "An approach for near-optimal distributed data fusion in wireless sensor networks," *Springer Wireless Network*, vol. 16, pp. 1407–1425, 2009. Article (CrossRef Link)
[7] Cai W., Chen M., Hara T., Shu L., Kwon T., "A genetic algorithm approach to multi-agent itinerary planning in wireless sensor networks," *Springer Mobile Network application*, 16 (6), pp. 782–793, 2011. Article (CrossRef Link)
[8] Konstantopoulos, C., Mpitziopoulos, A., Gavalas, D., Pantziou, G., "Effective determination of mobile agent itineraries for data aggregation on sensor networks," *IEEE Transaction on Knowledge and Data Engineering*, vol. 22(12), pp. 1679–1693, 2010. Article (CrossRef Link)
[9] Mpitziopoulos, A., Gavalas, D., Konstantopoulos, C., Pantziou, G., "CBID: a scalable method for distributed data aggregation in WSNs," *Hindawi International Journal of Distributed Sensor Network*, vol. 2010, Article ID 206517, pp.13, 2010. Article (CrossRef Link)
[10] Chen, M., et al., "Itinerary Planning for Energy-efficient Agent Communication in Wireless Sensor Networks," *IEEE Transactions on Vehicular Technology*, pp. 1-1, 2011. Article (CrossRef Link)
[11] Mpitziopoulos A., Konstantopoulos C., Gavalas D., and Pantziou G., "A pervasive assistive environment for visually impaired people using wireless sensor network infrastructure," *Journal of Network and Computer Applications*, vol. 34, pp. 194-206, 2011. Article (CrossRef Link)
[12] Averbakh I. and Berman O., "Sales-delivery man problems on treelike networks," *Networks*, vol. 25, pp. 45-58, 1995. Article (CrossRef Link)
[13] Xu, Y., & Qi, H., "Mobile agent migration modeling and design for target tracking in wireless sensor networks," *Ad Hoc Networks*, vol. 6(1), pp. 1–16, 2008. Article (CrossRef Link)
[14] Gupta G. P., Misra M., and Garg K., "Multiple Mobile Agents based Data Dissemination Protocol for Wireless Sensor Networks," in *Proc. of Springer International Conference on Advances in Computer Science and Information Technology, Networks and Communications*, pp. 334-345, 2012.
[15] Bulusu N., Heidemann J., and Estrin D., "GPS-Less Low Cost Outdoor Localization for Very Small Devices," *IEEE Personal Communication*, vol. 7, no. 5, pp. 28-34, Oct. 2000. Article (CrossRef Link)
[16] Mao G., Fidan B., Anderson B. D., "Wireless sensor network localization techniques," *Computer Networks*, vol. 51(10), pp. 2529-2553, 2007. Article (CrossRef Link)
[17] Wang Y., Wang X.,Wang D., Agrawal D. P., "Range-free localization using expected hop progress in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20(10), pp.1540-1552, 2009. Article (CrossRef Link)
[18] Rachuri K. K., Murthy C.S.R., "On the scalability of expanding ring search for dense wireless sensor networks," *Journal of Parallel and Distributed Computing*, vol. 70(9), pp.917–929, 2010. Article (CrossRef Link)
[19] Intanagonwiwat C., Estrin D., Govindan R., Heidemann J., "Impact of network density on data aggregation in wireless sensor networks," in *Proc. of ICDCS'02, the 22nd International Conference on Distributed Computing Systems*, pp.457, Jul.2002.

[20] Verma V, Joshi R. C., Xie B, Agrawal D. P., "Combating the bloated state problem in mobile agents based network monitoring applications," *Computer Networks*, vol.52(17), pp.3218 – 3228, 2008. Article (CrossRef Link)
[21] Castalia Simulator (March 2012) [online] http://castalia.npc.nicta.com.au/.
[22] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 2-16, 2003. Article (CrossRef Link)

Govind Gupta received the B.E. degree in Computer Science and Engineering in 2000 from CCS Meerut University, Meerut, India, and M.Tech degree in Computer Science and Engineering in 2007 from UP Technical University, Lucknow, India. He is currently a PhD candidate in the Department of Computer Science & Engineering at Indian Institute of Technology, Roorkee, India. His research interests include mobile computing, wireless ad hoc and sensor network, network security, distributed computing and performance evaluation. He is a student member of IEEE.

Manoj Misra received his Bachelor's degree in Electrical Engineering in 1983 from HBTI Kanpur, India and Master's in Computer Science in 1986 from University of Roorkee, India. He received his PhD in Computer Science in 1997 from University of Newcastle, Upon Tyne, UK. He is currently a Professor in the Department of Computer Science and Engineering at Indian Institute of Technology, Roorkee. He has guided several PhD theses, ME/MTech dissertations. His areas of interest include mobile computing, distributed computing, wireless ad hoc and sensor network and performance evaluation. He is a senior member of IEEE.

Kumkum Garg received her Bachelor's in Electronics Engineering from University of Roorkee, India in 1971 and Master's in Computer Engineering in 1977 from University of Roorkee, India. She received her PhD in Computer Engineering in 1984 from University of London, UK. She has been a Professor in the Department of Computer Science and Engineering, Indian Institute of Technology, and Roorkee, India from 1989 to 2010. She has been the Head of Information superhighway Centre from 2005 to 2006 at IIT, Roorkee. Currently, she is Professor of Computing and Dean, Faculty of Engineering at Manipal University Jaipur, India. She has guided several PhD theses, ME/MTech dissertations and completed various projects. Her research interests include mobile computing, mobile agents, network security, trust, wireless ad hoc and sensor network. She is a senior member of the IEEE Computer Society.