Routing of mobile agents in a single channel Wireless Sensor Network

Manik Raina, Subhas Kumar Ghosh, Ranjeet Kumar Patro
Honeywell Technology Solutions Laboratory
151/1 Doraisanipalya, Bannergatta Road
Bangalore, India - 560068
Email: {manik.raina, subhas.kumar, ranjeet.patro}@honeywell.com

***Abstract*— This paper proposes fault tolerant algorithms for routing mobile agents through a homogenous single channel Wireless Sensor Network. The routing algorithms have a property that every sensor in the Wireless sensor network is covered by the itinerant mobile agents. The various algorithms exploit local knowledge alone to make all decisions and do not need any global properties of the Wireless Sensor Network. We present a case for why these algorithms are required. This is followed by theoretical analysis of the problem in the randomly deployed Wireless Sensor Network where we prove the correctness of our algorithms and justify our intuitive insights by mathematical proofs. This is followed by a comparative simulation where our algorithms are simulated and compared on various parameters like energy expenditure, robustness, network lifetime and latency.**

I. Introduction

Ad hoc Wireless Sensor Networks (WSN) are gaining prominence in areas of sensing, detection and tracking. Applications of this technology range from monitoring and control in military, ecological, enviromental and domestic systems. WSNs are also used in automating buildings, universities factories etc. The composition and deployment scenarios of this technology are varied. Each node in this network consists of sensors which communicate with one another via a wireless channel.

The Wireless Sensor Network is also often quoted as one of the most constrained computational enviroment where sensor memory, power and computational capabilities are all limited. This enhances the appeal of this technology as it makes sensing devices low end in cost terms. This lowering in cost makes the sensors prone to failure and energy depletion, a fact often resolved by resorting to dense deployment of sensors. This fact must be kept in mind during algorithm development for WSNs.

In many usage scenarios sensors are randomly deployed, sometimes scattered by aeroplanes in hostile terrain and hence no assumptions can be made about the topology of the network, which the sensors in the WSN have formed post deployment. Algorithms which route data and code through such a network must provably work over any arbitrary deployment. The sensors are also prone to transient changes.

We present algorithms for grid and random Wireless Sensor Networks which count the number of nodes in a network at a given time. This problem is simple to formulate and can provide useful information about the network's state. We use *agents*, which can move around the network hopping from sensor to sensor. The agents carry their own data and state.

We consider the random deployment of sensors over a region and provide a theoretical basis for the algorithms we propose over the random graphs, performing mathematical analysis of the those algorithms and their basis.

The Itinerary of our algorithms is decided online and is not predecided.

A. Why a mobile agent based approach?

Mobile agents carry code which enables them to take dynamic decisions which may not be possible in other solutions. For example, as shall be shortly stated, we give examples of network wide reprogramming, network calibration and parameterization as some of the problems which shall be solved with our solution. For example, in reprogramming the sensor network, the mobile agents can reduce the time needed for reprogramming and the bandwidth used (leading to savings in network lifetime) by carrying extra information which helps it decide which nodes need to be reprogrammed. Some nodes can be skipped for reprogramming as a result. Now consider calibration, some calibration schemes require complex techniques which may involve more than one sensor at a time in the calibration process. Thid may need logic (code) to be carried around to the nodes to determine which sensors need to be involved in the calibration.

Qi et al [8] discuss the mobile agent paradigm in wireless sensors and some issues.

B. Some applications of our solution

There are several problems which can be easily solved using our algorithms directly, especially problems which need *every* sensor to be reached and some data gathered from each. For example

- Propogation of model parameters to every sensor node. Elnahrawy and Nath [2] describes a Bayesian approach for cleaning and querying noisy sensors where noisy observations at each sensor are turned into probabilistic uncertainity models of the readings by a *cleaning module* which needs a noise model and prior knowledge. Algorithms in our paper can be utilized to update prior knowledge at each sensor node when the prior knowledge

models are dynamic. This updation model consists of updating sensor nodes about the parameters of the prior knowledge which have changed. For example, changes in parameters of the prior knowledge like mean μ and standard deviation σ can be sent to every sensor.

- Global calibration of communication and sensing parameters. Certain sensor parameters depend on the ambient enviroment and physical parameters pertaining to outdoor propogation models of RF are critical for radio communication and sensing which are determined by experimentation and need to be propogated to every sensor in the network. As an example, the average large scale path loss for an arbitary T-R seperation can be expressed as

$$\overline{PL}(Db) = \overline{PL}(d_0) + 10nlog\frac{d}{d_0}$$

The path loss exponent varies for different eviroments as shown [3] below

Environment	Path loss exponent, n
Free space	2
Urban area cellular radio	2.7 to 3.5
Shadowed urban cellular radio	3 to 5
In building line-of-sight	1.6 to 1.8
Obstructed in building	4 to 6
Obstructed in factories	2 to 3

TABLE I
PATH LOSS EXPONENTS FOR DIFFERENT ENVIROMENTS

Since sensor software at the time of manufacture may not be certain about the deployment, the parameter n can be reparameterized at each sensor for enhancing the accuracy of the signal model.
For acoustic sensors there is a similar drift in the veolocity of sound due to temperature and humidity variations. The drift can be represented as $c_{air} = 331.5 + 0.6\theta$ where θ is the temperature in celsius. For acoustic sensors, any drift in the speed of sound will cause serious errors in measurements.
Similarly, the PCS extended Hata model [3] (an outdoor RF propogation model) also needs to be parameterized depending on where the deployment occurs. The model is

$$L_{50} = 46.3 + 33.9log\ f_c - 13.82log\ h_{te} - a(h_{re}) + (44.9 - 6.55log\ h_{te})log\ d + C_M$$

The parameter C_M is 0Db for medium sized cities and suburban areas and 3Db for metropolitan areas. Again, this parameter needs to be communicated to each sensor depending on the eventual deployment since the deployment scenario may not be known at manufacture when the software is burned into the sensors.

- Software redeployment: The mobile agents help nodes identify if they are running faulty or obsolete software and prompt them to update their versions from a base station or another sensor node in the WSN. Reijers and Langendoen [4] describe a scheme for distributing software updates wirelessly in an ad hoc sensor network. However, their paper does not discuss a scheme or a protocol which ensures that these updates actually reach every sensor in the network. Our algorithms can be used to distribute the updates across the sensor network.
- Certain aggregation scenarios where the solution we describe can be used to collect information and store it for use, perhaps by the base station or even other sensors. A view of this kind of a routing algorithm as a *traversing query* can be proposed.

II. RELATED WORKS

Marwaha *et al* [7] describe a hybrid algorithm which reduces route discovery and end-to-end latency in mobile ad-hoc networks. Though the first algorithm in this paper appears similar to our first algorithm, their algorithms appear to be a product of intuition rather than based on concrete mathematical proofs. Their paper considers algorithms for route updations while our algorithms are for visiting all nodes for the purpose of parameterization, calibration, software redeployment etc. Further, agents in their work act independently which causes many of them to repeat what others are doing. In our scenario, we ensure our "labelling" schemes in multiple agent schemes works such that each sensor is visited by at most one agent with the primary aim of reducing latency. Further, our paper goes on to describe an algorithm which uses both agents and queries. Each of our claims are backed up with mathematical proofs to indicate that our claims hold true regardless of the deployment and the precise conditions under which deviations from optimal conditions may happen.

Wu *et al* [6] describe a genetic algorithm based approach to solving the mobile agent routing problem after proving the optimal routing of agents in a WSN to be NP complete. Their approach, however presupposes a hierarchical model of the Wireless sensor network consisting of powerful processing elements which perform the computationally heavy task of determining the routes and computationally constrained sensors which sense the enviroment. This is different from our homogenous view of the sensor network where the same computationally constrained sensors make decisions about the routing and sense the enviroment as well. Further, their paper assumes that system wide information is present for calculating routes while we route using local information alone.

III. PROBLEM FORMULATION

A. Assumptions

- We assume that each sensor contains a *node identifier* which could perhaps be predistributed. The *node identifier* for the n'th sensor can be represented as ID_n. There must exist a bijection f such that $f : ID_n \rightarrow Z_{n(G)} + 1$, where $Z_{n(G)} + 1$ is the set of integers $\{1, 2, 3, ..., n(G) + 1\}$. It is assumed that each vertex knows the *node identifiers* of it's first hop neighbors.
- Further, the edge set $E(G)$ which represents the communication links in the Wireless Sensor Network G forms according to the spatial constraint $\forall u, v \in V(G)\left(e_{uv} \in E(G) \equiv \left\|\vec{r_u} - \vec{r_v}\right\| \leq r\right)$ where

- $\vec{r_u}$ and $\vec{r_v}$ are the location vectors for the vertices u and v
- r is the transmission radius and is assumed to be the same for all vertices
- $\|.\|$ is the euclidian norm.

B. Graph

A graph $G(V, E)$ is defined as a collection of vertices and edges between the vertices. This graph shall be used to represent sensors and communication links between them. Each sensor shall be represented by a unique $u \in V(G)$. $e_{uv} \in E(G)$ iff a communication link exists between u and v.

C. Agent

An *agent* A_i is defined as an entity consisting of $\left\{D_t^i, S_t^i\right\}$ where D_t^i is the data and S_t^i is the state at time t. Code C is defined as $C : \left\{D_t^i, S_t^i\right\} \rightarrow \left\{D_{t+1}^i, S_{t+1}^i\right\}$.The agent does not carry code, which resides on every sensor $u \in V(G)$. Each agent A_i forms an *association* $\langle A_i, u\rangle$ with vertex $u \in V(G)$ if the agent resides at u. For example, an association $\langle A_i, u\rangle$ implies that agent A_i resides at vertex u. We define $jump(A_i, x, y) : \langle A_i, x\rangle \rightarrow \langle A_i, y\rangle$, for some $x, y \in V(G)$.

D. Itinerary

An *Itinerary* U_i of agent A_i is defined such that

$$\forall x \in V(G)\ ,\ x \in V(U_i) \equiv \exists y, z \in V(G)\ such\ that \\ jump(A_i, y, x) \wedge jump(A_i, x, z) \tag{1}$$

Further,

$$e_{xy} \in E(U_i)\ \equiv\ x, y \in V(U_i) \wedge \\ (jump(A_i, x, y)\ \vee\ jump(A_i, y, x)) \tag{2}$$

Clearly it can be seen that $U_i \subset G$. The itinerary is the subset of G the agent A_i takes while performing it's task.

E. Itinerary history

An *Itinerary history* of agent A_i is defined as H_{A_i} such that $H_{A_i} \subseteq V(G)$. H_{A_i} is contained in D_t^i.

$$jump(A_i, x, y) \rightarrow \begin{cases} for\ y\ \notin H_{A_i} : \\ push(H_{A_i}, f(y)) \\ g() \\ \langle A_i, x\rangle \rightarrow \langle A_i, y\rangle \\ return\ f(y) \\ \\ for\ y \in H_{A_i} : \\ return\ f(x) \end{cases}$$

where $push()$ and $pop()$ are LIFO stack operators. $g()$ is a function evaluated at each vertex the agent A_i visits. This function could do a multitude of tasks including gathering data, setting some flags, triggering operations in every sensor etc.

F. Neighbor set $N(v)$

The Neighbor set of v is defined as follows

$$\forall v, x \in V(G)\ \ (x \in N(v) \equiv e_{xv} \in E(G)) \tag{3}$$

G. Local Knowledge

Local knowledge is defined as the questions which can be answered with the information in $\left(\langle A_i, u\rangle, D_t^i, S_t^i, N(u)\right)$, where u is the vertex at which the agent A_i finds itself.

H. Purpose of the algorithms

Given a randomly deployed Wireless Sensor Network, represented by a fully conneted graph $G(E, V)$, the problem we wish to solve is to be able to visit every sensor in it.

IV. Theoretical results

In this section we discuss the underlying graph theoretic principles which have resulted in our algorithms.

A. Impossibility of determining existence of Hamiltonian cycles using local knowledge alone

Determining if a Hamiltonian cycle exists in a graph G is a well known NP complete problem [1]. In this section we show the difficulty of determining if a Hamiltonian path exists in a graph with local knowledge. We shall enumerate the various necessary and sufficient conditions for that purpose.

1) Necessary conditions
   a) Every vertex of a Hamiltonian graph has degree ≥ 2.
   b) If G has a Hamiltonian cycle, for each nonempty $S \subseteq G$, $G - S$ has at the most $|S|$ components.

   To determine the first condition, we would have to check each node which has degree less than two. For this we would have to visit each vertex v or any node in $N(v$ which knows that v has degree less than two. Hence, information of every node is necessary to determine if the necessary conditions are met. For the second condition, the task of determining if a subset of G forms a component even though achievable in polynomial time, this condition needs to be evaluated for every subset of G and for a graph G, the subset count of $V(G)$ increases as $2^{|V(G)|}$.
2) Sufficient conditions
   a) If G is a graph and $|V(G)| > 3$, G is hamiltonian if $\delta(G) \geq n(G)/2$.
   b) In a simple graph G, $\forall u, v \in V(G)$, if $d(u) + d(v) \geq n(G)$, G is Hamiltonian.
   c) A simple n vertex graph G is Hamiltonian iff the closure of G is Hamiltonian.
   d) Let the vertex degrees of a simple graph G be $d_1 \leq d_2 \leq d_3 ... \leq d_n$. If $i < n/2$ implies $d_i > i$ or $d_{n-i} \geq n - 1$, G is Hamiltonian.

   Clearly, the first, second and fourth conditions need the degrees of all nodes of G, which cannot be determined unless the agent travels to every vertex. The third condition needs information abouth the closure of G which needs global information. Hence, information is needed from every vertex to prove if a Hamiltonian path exists. Since the task of proving the existence of such a path does not identify the order of the cyclic sequence of

edges $\in E(G)$ forms the hamiltonian path, a heuristic based mechanism which visits every vertex to count it is not such a bad idea as long as we can bound it's performance. We shall later present mechanisms which loosen the need to visit every vertex by the agent for counting.

B. A quick introduction to our goals

Considering the above, the algorithms we devise a set of algorithms which use agents to visit every vertex in a graph. The algorithms

- Must reach every node in the graph G.
- Must exploit local information available at that vertex. Later we will analyse algorithms which depend on local information beyond what is available at the vertex in question, but still does not depend on any global parameters. Also, after initially concentrating on algorithms which use a single agent, we will move onto algorithms which use more than one agent and then to algorithms which exploit more than just the first hop connectivity information.
- Must succeed in doing so irrespective of the topology of the graph.

C. Theoretical results and proofs when using a single agent

In this subsection, we shall present the proof of correctness of the algorithm to visit all vertices using a single agent as given in algorithm 1. This algorithm uses the neighbor list ($N(u)$) of a vertex u to determine where to go next. The agent utilizes a stack and agent state which it carries with it as it moves from agent to agent. At any time, the agent is in one of the states $\{recede, firstVisit\}$. The agent is in state $firstVisit$ when it visits a node for the first time and in state $recede$ otherwise.

To prove that the algorithm visits all vertices in G, we need to prove that when agent A travels along it's itinerary U_A ($U_A \subset G$), it possess the following qualities

1) Each cycle in G is a path in U_A.
2) $\forall x \in V(G) \implies x \in U_A$.
3) Each vertex in U_A executes $g()$ exactly once.

The third condition is necessary in applications where the operation is to be performed exactly once for every vertex.

Before we can begin the proofs we need a few definitions.

DEFINITION IV.1. *Let P be a uv path in G. P is a set of vertices $\{u, x_1, x_2, x_3..., v\}$. We define fP as $\{f(u), f(x_1), f(x_2), f(x_3)..., f(v)\}$.*

DEFINITION IV.2. *Let P_i and P_j be two (possibly non-disjoint) uv paths in G defined as $\{u, x_1, x_2, x_3..., v\}$ and $\{u, y_1, y_2, y_3..., v\}$.*

$$DIFF(P_i, P_j) \implies \begin{cases} 0 \ when \forall i, f(x_i) = f(y_i) \\ f(x_j) - f(y_j) \ when \\ \exists z | f(x_z) \neq f(y_z) \ and \ j \ is \\ the \ least \ element \ of \ \{z\} \end{cases}$$

Algorithm 1 Visiting vertices in G using a single agent

```
agentState = firstVisit {We follow a convention that u always denotes the ID of the sensor on which the agent resides at this point and v the ID of the sensor on which the agent was residing before jumping to this sensor}
while stack ≠ {Φ} do
  if agentState == firstVisit then
    g() {Application specific activity at sensor}
    list = N(u) {visiting this vertex for the first time}
    if {list − u} == {Φ} then
      agentState = recede
      jump(A, u, topOfStack(stack))
    else
      find w such that(f(w) ≤ f(x)∀x ∈ list) ∧ (w ∉ stack)
      if w ∈ stack ∧ f(u) > f(w) then
        add(w, u)in exception list
        repeat previous step of finding w
      end if
      if no such w exists then
        agentState = recede
        jump(A, u, topOfStack(stack))
      end if
      agentState = recede
      jump(A, u, w)
    end if
  else
    pop(stack) {We have been to this vertex before}
    list = N(u)
    find w such that f(v) ≤ f(w) ≤ f(x)(∀x ∈ list) ∧ w ∉ stack ∧ w ∉ exceptionList
    if w ∈ stack ∧ f(u) > f(w) then
      add(w, u)in exception list
      repeat previous step of finding w
    end if
    if no such w exists then
      jump(A, u, topOfStack(stack))
    end if
    jump(A, u, w)
  end if
end while
```

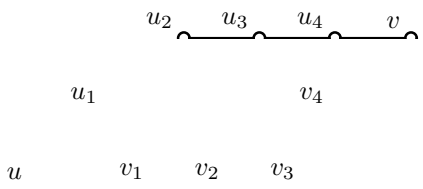
THEOREM IV.3. (Cycle decomposition) *Every cycle in G is a path in U_A.*

Proof. Consider the agent A which attempts to visit the vertices in G. Let the procedure begins at $u \in V(G)$. i.e $\nexists x \in V(G)$ such that $jump(A, x, u)$.

At vertex v, the Itinerary history of A shall contain $H_A = \{f(u), f(u_1), f(u_2)..., f(v)\}$. The agent has traversed the path $P(uu_1u_2u_3u_4v)$. $\forall x \in P \implies x \in U_A$. Let $P' =$

$\{uv_1v_2v_3v_4v\}$ be another uv path in G. We will prove that only one of P or $P^{'}$ exists in U_A using contradiction.

For a path P to exist in U_A, $\forall p \in \{all\ uv\ paths\ in\ G\}$, $DIFF(P,p) \leq 0$. The path P ($uu_1u_2u_3u_4v$)will exist in U iff $\forall uv$ paths P_i in G, $DIFF(P,P_i) \leq 0$. Assume $P^{'}$ also $\in U_A$.



Without loss of generality let us assume that $f(u_1) \leq f(v_1)$. Since

$$f(u_1) \leq f(v_1) \implies jump(A,u,u_1) \tag{4}$$

and

$$jump(A,u,u_1) \implies push(H_A, f(u_1)) \wedge \{\langle A,u\rangle \rightarrow \langle A,u_1\rangle\} \tag{5}$$

Hence based on (5), it can be concluded that A will be at U_1 and $H_A = \{f(u), f(u_1)\}$. Further, from (5) it can be inferred that along path P, $jump(A,u_i,u_{i+1})$ and hence H_A at v is $\{f(u), f(u_1), f(u_2), .., f(v).\}$. At v_1, H_A would be $\{u, u_1, u_2, u_3, u_4, v, v_4, v_3, v_2, v_1\}$. At v, $N(v) = \{f(u), f(v_2)\}$ and a $jump$ to u is forbidden by the algorithm. Since $\forall x \in N(v),\ x \in H_A$, $P^{'} \notin U_A$. □

THEOREM IV.4. (Vertex equivalence) *$V(G)$ and $V(U_A)$ are identical.*

Proof. Assume the agent A begins the procedure at a vertex $u \in G$, (i.e $\nexists r \in V(G)$ such that $jump(A,r,u)$) $u \in U_A$. Since G is connected, $\exists$ a uv path in G. There exists a many-to-one mapping $\mu : \{all\ uv\ paths\ in\ G\} \rightarrow P$, where P is the only uv path in U_A. Since G is connected, $\forall u, v \in V(G)$, there is at least one uv path in G but there is a unique uv path $\mu(all\ uv\ paths\ in\ G)$ in U_A. Since this argument can be extended $\forall u, v \in V(G)$,

$$\forall x \in V(G) \implies x \in U_A. \tag{6}$$

□

THEOREM IV.5. (Uniqueness of enumeration) *At each vertex in U_A, agent A executes $g()$ exactly once.*

Proof. Since $jump(A,u,v) \implies g()$ if and only if $f(v) \notin H_A$ the agent A executes $g()$ at a vertex when it is visiting a vertex for the first time only. Subsequent visits to the same vertex do not execute $g()$. □

We propose and prove a bound on the number of times a vertex v is present in the agent A_i's itinerary U_{A_i}.

THEOREM IV.6. (The Counting cut vertex theorem) *Let any vertex v of graph G be a cut vertex such that $G - \{v\}$ has k components. The the agent shall traverse vertex v at exactly k times.*

Proof. Let vertex v forms is a cut vertex (articulation) of G. By definition, $G - \{v\}$ is disconnected. Let $S = \{C_1, C_2, ...C_k\}$ be the components of $G - \{v\}$. If $v \in U_{A_i}$, then by definition $\exists x, y \in V(G)$ such that $jump(A_i, x, v) \wedge jump(A_i, v, y)$.

If we have to determine the smallest $Y \subset G$ such that $Y = \{v_1, v_2, v_3, ...v_n\}$ and $\forall z \in Y \implies z \in U_{A_i}$ subject to the condition that $\forall j \in [1,k], \exists v_r \in Y$ such that $v_r \in C_j$, then $|Y| \geq k$.

We shall prove this theorem by Contradiction.

Assume $|Y| < k$. Then it is not possible to have a bijective mapping $\Psi : Y \rightarrow S$ since that would require $|Y| = |S|$. Hence $|Y| \geq |S|$.

Now consider $Y^{'} = \{u_1, u_2, ...u_k\}$ such that $Y^{'} \subset V(G)$ and $|Y^{'}| = k$. If $\forall j \in [1,k],\ u_j \in C_j$ and $\forall i \in [1, k-1], jump(A_i, u_j, u_{j+1}) \wedge jump(A_i, u_k, u_1)$. This represents the smallest Y which satisfy all conditions.

It is now clear that vertex v is visited by agent A *at least* k times. Suppose agent A is in component C_j. We define $N_{C_j}(v)$ as

$$x \in N_{C_j}(v) \implies x \in N(v) \wedge x \in C_j$$

Or, $N_{C_j}(v)$ constitutes the first hop neighbors of v which belong to C_j. Assuming A is at some $x \in N_{C_j}(v)$, since $f(v) \in H_A$, by reasoning similar to the one presented in theorem IV.3, A can only return to v from one element in $N_{C_j}(v)$, which proves that vertex v can be reached exactly k times. □

D. Using more than a single agent

In this section we consider the theoretical motivation to pursue the problem at hand with more than one agent. A single agent based algorithm, though is proven to work correctly, does have some shortcomings which can be enumerated as follows.

Shortcomings of a single agent based algorithm

1) Prone to sensor failure
2) Long latency times

In this section we consider the analysis of those shortcomings and prove that using more than one agents ameliorates the situation. This paper contains simulation to bolster these very theoretical findings. We shall analyze each of the issues in more detail below.

We now consider the problem of sensor failure and it's impact on the results. A single agent A reaches vertices $\in G$ by traversing $U_A \subset G$. The agent A returning with the results along a path P would be unsuccesful if any vertex $\in P$ were to fail.

We now propose a method using multiple agents $\{A_1, A_2, A_3, ...\}$ and prove an improvement in each of the areas enumerated above where a single agent algorithm performs poorly. We illustriate the theoretical analysis assuming the special case that the agents are counting the number of vertices in G though this analysis can be genralized to any problem involving reaching every vertex easily. To ensure integrity of counting, we must ensure that $\forall i, j \in [1,n],\ C_{A_i} \cap C_{A_i} = \{\phi\}$ if $i \neq j$ where C_{A_i} is the set of vertices which agent A_i

counts. The protocol we propose will ensure that. This is ensured by a sensor labelling scheme which ensures only one agent labels every sensor. Based on a system parameter $\lambda \in Z^+$, an agent A traverses a random path P_l in G. $P_l = \{w_0, w_1, ..., w_{n-1}\}$ is a path chosen randomly such that it must have at least λ unique vertices and is called the labelling path. The set $L_\lambda = \{l_0, l_1, ..., l_{\lambda-1}\}$ is defined as the label set. A assigns labels as follows $\chi : P_l \rightarrow L_\lambda$ such that $\chi(w_r) = l_{r \ mod \ \lambda}$. Upon reaching $w_{n-1} \in P_l$, A replicates itself into n agents which traverse backwards along P_l counting the vertices incident upon each $x \in P_l$ using the algorithm similar to the one used by a single agent. This scheme partitions the graph G. The label stored at each vertex can be represented as $\{l_i, t_i, f(u)\}$.

Let us now turn our attention to the improvement this scheme offers when vertices fail. Since the labelling partitions G, each vertex $x \in P_l$ has some neighbors $N_i(x)$ such that $\forall y \in N_i(x) \implies y \notin P_l$. This is called the subtree *incident upon* x such that $|N_i(x)| \geq 0$. Let p_i be the probability of vertex failure when an agent A_i traverses the subtree incident at w_i. p_i depends on how many nodes are in $N_i(x)$ and the time A_i spends in $N_i(x)$ but we are not interested in those questions. Our objective is to show that with an increase in λ, the expectation of the count increases. Suppose $\lambda = 1$ and we use one label, which is identical to a single agent based count. In that case, the expectation of the count is

$$C_{av}^1 = n(G) \prod_{i \in [0, n-1]} (1 - p_i)$$

$$C_{A_1} = V(G)$$

For $\lambda = 2$, two agents A_1 and A_2 count G.

$$C_{av}^2 = |C_{A_1}| \prod_{i|imod2=0} (1 - p_i) + |C_{A_2}| \prod_{i|imod2=1} (1 - p_i)$$

$$C_{A_1} \cup C_{A_2} = V(G)$$

For the general case

$$C_{av}^\lambda = \sum_{(0 \leq k \leq \lambda - 1)} |C_{A_{k+1}}| \prod_{(i \ | \ i \ mod \ \lambda \ = \ k)} (1 - p_i) \qquad (7)$$

$$\bigcup_{(1 \leq k \leq \lambda)} C_{A_k} = V(G)$$

For a equally partitioned graph, $\forall i \in [0, \lambda - 1], \ p_i = p$. In this case,

$$C_{av}^\lambda = \sum_{(0 \leq k \leq \lambda - 1)} |C_{A_{k+1}}| \prod_{(i \ | \ i \ mod \ \lambda \ = \ k)} (1 - p)$$

$$\implies C_{av}^\lambda = (1 - p)^{\frac{\lambda}{n}} \sum_{(0 \leq k \leq \lambda - 1)} |C_{A_{k+1}}|$$

$$\implies C_{av}^\lambda = (1 - p)^{\frac{\lambda}{n}} n(G)$$

Clearly, since $p \leq 1$, C_{av}^λ becomes smaller as $\frac{\lambda}{n} \rightarrow 1$.

$$C_{min}^\lambda = (1 - p) n(G)$$

Since it is not possible to equally partition the graph using our algorithms and graph partitioning is a known NP hard problem, we need to analyze the same assertion for unequally partitioned graphs. For such graphs, $\exists i, j \in [0, \lambda - 1], \ p_i \neq p_j$. let $p_{min} = min\{p_1, p_2, p_3, ..., p_n\}$ and $p_{max} = max\{p_1, p_2, p_3, ..., p_n\}$. Then from (7) the following can be inferred

$$n(G) \prod_{0 \leq i \leq n} (1 - p_{max}) \leq C_{av}^\lambda \leq n(G) \prod_{0 \leq i \leq n} (1 - p_{min})$$

or

$$n(G)(1 - p_{max})^{\frac{\lambda}{n}} \leq C_{av}^\lambda \leq n(G)(1 - p_{min})^{\frac{\lambda}{n}} \qquad (8)$$

Since $P_{max} << 1$, $(1 - p_{max})^{\frac{\lambda}{n}} \approx 1 - p_{max}(\frac{\lambda}{n})$ and hence

$$n(G)(1 - p_{max}\frac{\lambda}{n}) \leq C_{av}^\lambda \leq n(G)(1 - p_{min}\frac{\lambda}{n})$$

Clearly, as λ increases, both the lower and upper bound increase, increasing the value of C_{av}^λ. It can be shown that

$$C_{av}^\lambda \geq C_m(1 - p_{min})(1 - p_{min} - \delta)^{\frac{n}{\lambda} - 1} + (1 - p_{min} - \delta)^{\frac{n}{\lambda}} \sum_{k \neq m} C_k$$

where $\delta = p_{max} - p_{min}$. Here too, we see C_{av}^λ becoming larger as $\lambda \rightarrow n$. To verify this result, this scenario is a strong case for simulation.

Now we turn our attention to latency. We assume that the latency Λ (time taken) for an agent A_i to count the vertices in G is proportional to the number of vertices counted by A_i, C_{A_i}. For $\lambda = 1$, $\Lambda \approx \sum_{(0 \leq k \leq n-1)} C_k$. For the general case, $\Lambda = max\{C_{A_1}, C_{A_2}, C_{A_3}, ..., C_{A_\lambda}\}$. $C_{A_i} = \sum_{k|kmod\lambda=i} C_k$. Clearly, $\sum_{k|kmod\lambda=i} C_k < \sum_{(0 \leq k \leq n-1)} C_k$ for $\lambda > 1$. With this justification, we now present an algorithm which counts the vertices using λ agents.

We present an algorithm 2 which reaches every sensor at and use more than one agents. This algorithm reaches every sensor and performs the sensor specific processing which is application specific. However, there is a slight modification in the semantics of the $jump$ call. In addition to modifying the association $\langle A_i, x \rangle \rightarrow \langle A_i, y \rangle$ it performs an additional operation of labelling the current vertex. This whole operation must be atomic. An agent can jump to a node u such that $u \in P_l \ iff$ the agent state is $recede$.

E. Queries and agents

A query q is defined as a question/response mechanism initiated by the sender of the query S_q to obtain a certain information q_i from the recipient of the query R_q.

$$S_q \xrightarrow{q} R_q$$

$$S_q \xleftarrow{q_i} R_q$$

Further, $q \in Q$, where Q is a predefined master set of possible queries, known to both S_q and R_q. The response q_i is returned to S_q (possibly using none, one or more intermediaries) by R_q in a format the former understands.

When traversing through a WSN, the agent contains code and data, a natural question to ask if the agent needs to visit every sensor. More importantly, given a node u, is it necessary for the agent to visit all nodes in the vicinity of u ? We propose a mechanism of queries of depth d which trigger the per sensor

Algorithm 2 Visiting vertices in G using λ agents

Require: $u \leftarrow ID\ of\ the\ current\ vertex$

$choose P_l \in V(G)\ such\ that\ |P_l| \geq \lambda$
for all $x \in P_l$ **do**
  $label_{current\ node} \leftarrow label$
  $label \leftarrow (label + 1) mod \lambda$
  $jump(A, u, x)$ {The Agent A will find itself at the vertex corresponding to the last entry in P_l}
  $replicate()$ {replicate agent A into λ agents $\{A_1, ..., A_\lambda\}$, from this point on, each agent shall execute seperately. Further, each agent shall have a variable called "label" which describes the label it is associated with. From this point onwards, each agent executes on it's own.}
end for
$\forall A \in \{A_i\},\ agentState_A = firstVisit$
while $stack_{A_i} \neq \{\Phi\}$ **do**
  {each agent will possess it's own stack, all agents execute this procedure independently of one another}
  if $agentState == firstVisit$ **then**
    $g()$ {Application specific activity at sensor}
    $list = N_i(u)$ {determine the subtree incident at u}
    if $\{list - u\} == \{\Phi\}$ **then**
      $agentState_{A_i} = recede$
      $jump(A_i, u, topOfStack(stack_{A_i}))$
    else
      $find\ w\ such\ that (f(w) \leq f(x) \forall x \in list) \wedge (w \notin stack)$
      if $w \in stack \wedge f(u) > f(w)$ **then**
        $add(w, u) in\ exception\ list$
        $repeat\ previous\ step\ of\ finding\ w$
      end if
      if $no\ such\ w\ exists$ **then**
        $agentState_{A_i} = recede$
        $jump(A, u, topOfStack(stack_{A_i}))$
      end if
      $agentState_{A_i} = recede$
      $jump(A_i, u, w)$
    end if
  else
    $pop(stack_{A_i})$ {We have been to this vertex before}
    $list = N_i(u)$
    $find\ w\ such\ that\ f(v) \leq f(w) \leq f(x) (\forall x \in list) \wedge w \notin stack_{A_i} \wedge w \notin exceptionList$
    if $w \in stack_{A_i} \wedge f(u) > f(w)$ **then**
      $add(w, u) in\ exception\ list$
      $repeat\ previous\ step\ of\ finding\ w$
    end if
    if $no\ such\ w\ exists$ **then**
      $jump(A_i, u, topOfStack(stack_{A_i}))$
    end if
    $jump(A_i, u, w)$
  end if
end while

Algorithm 3 jump semantics for λ agents

jump (A_i, u, v)
if $v \notin H_{A_i}$ **then**
  atomic: $\chi(u) = \chi(A_i)$ {Assign the node the label of the visiting agent}
  if $atomic\ operation\ success$ **then**
    $push(H_{A_i}, f(v))$
    $g()$
    $\langle A_i, u \rangle \rightarrow \langle A_i, v \rangle$
  end if
else
  $\langle A_i, u \rangle \rightarrow \langle A_i, v \rangle$
end if

activities which the agent would have performed by queries themselves. Agents will visit selective nodes only. If some nodes need the code, the reply/response mechanism of the queries can take care of that.

Having said that, we propose a mechanism of using queries in our problem. More specifically we propose a method to determine the local topology S_v around a sensor to aid the vertex visiting process, where $S_v \subset G$.

First, we define the *depth* (d) of a query q. If $d = 1$, The query reaches all $x \in V(G)$ such that

$$x \in N(S_q)$$

Such a set of vertices is called $N_1(S_q)$. All such nodes individually reply to the query. If $d = 2$, the query reaches all $x \in V(G)$ such that

$$x \in N_1(S_q) \vee \exists y \in V(G) \mid y \in N_1(S_q) \wedge x \in N_1(y)$$

Such a set of vertices is called $N_2(S_q)$. Similarly for $d = 3$, the query reaches all $x \in V(G)$ such that

$$x \in N_1(S_q)\ \vee\ x \in N_2(S_q)\ \vee$$
$$\exists y \in V(G) \mid y \in N_2(S_q) \wedge x \in N_1(y)$$

We use queries when the cost of queries is low compared to the cost of agents being used. We gain some local topological which eliminates the need for the agent to visit every vertex. A hybrid scheme using queries and agents will work when using queries minimizes the total cost associated. We will derive some bounds for cost of queries in later sections. Once local topological knowledge S is built, agents need not visit vertices x such that $N(x) \subset S$. Let S_q be the originator of the queries, let $q \in Q$ be the query request, the total transmission cost of queries is proportional to the size of information in the queries. If d_{max} is the maximum degree of any vertex in a neighborhood S, the total cost of transmission of queries is bounded as

$$Cost_{tx} \leq \sum_{0 \leq k \leq d-1} (q + ku) d_{max}^{k-1}$$

where q is the cost of transmitting the query and u is the additional cost of a node identifier. The reception cost of the queries is bounded as

$$Cost_{rx} \leq q^{'} (\frac{d_{max}^{d+1} - (d+1) d_{max}^{d} + 1}{(d_{max} - 1)^2})$$

Where $q^{'}$ is the cost associated with reply of the query. The cost saved due to queries is $(D_t^i + S_t^i)S^{'}$ where $S^{'} \subset S$ is the set of vertices such that $\forall x \in S^{'}, N_1(x) \subset S$. Hence, for queries to be cost effective

$$\sum_{0 \leq k \leq d-1} (q + ku)d_{max}^{k-1} + q^{'}(\frac{d_{max}^{d+1} - (d+1)d_{max}^{d} + 1}{(d_{max} - 1)^2}) \leq (D_t^i + S_t^i)S^{'}$$

From this expression we can draw some conclusions about when queries are useful. Queries turn expensive in regions where the degree of vertices is high. The query depth must be sufficiently small as the reception cost of cost increases exponentially with query depth d. However, if query depth is not sufficient, we may not gain enough insight into the local topology and the information may not be useful. The tradeoff needs to be studied by simulation. A small value of q and $q^{'}$ will lead to more cost effective queries. Further, queries are useful in large deployments where the itinerary history of the agent is large. All these assertions need to be validated by simulation.

It is worthy of note that in the scenario of queries, the function $g()$ is evaluated at each node either by the visiting agent A or by the query. The receipt of the query executes the function.

We present algorithm 4 for reaching every sensor using queries and agent movement.

Algorithm 4 Algorithm to visit vertices in G using agent A and queries

Require: $u \leftarrow ID\ of\ the\ current\ vertex$
  while 1 **do**
    if $agentState == firstVisit$ **then**
      $S_u \leftarrow Query(u, d)$ {Build local topological information using queries of depth d from vertex u}
      $R_u = V(G) - S_u$
      if $R_u == \Phi$ **then**
        $agentState = recede$
        $jump(A, u, topOfStack(stack))$
      end if
      $determine\ w \in R_u\ such\ that f(w) \leq f(x) \forall x \in R_u$
      $jump(A, u, w)$
    else
      $pop(stack)$ {agent state is recede}
      $find\ w \in R_u\ such\ that\ f(v) \leq f(w) \leq f(x) \forall x \in R_u$
      if $\nexists w$ **then**
        $jump(A, u, topOfStack(stack))$
      end if
      $agentState = firstVisit$
      $jump(A, u, w)$
    end if
  end while

V. Simulation and Results

We simulated our algorithms in Matlab. A Wireless Sensor Network was randomly deployed and our algorithms were run on the simulated network. All the algorithms were verified for correctness, which was the bare minimum expectation, that they indeed reach every sensor. Further, we were looking for proof that our theoretical results were verified in the simulation.

While we simulated our algorithms over various deployment scenarios with varying number of sensors, we present one particular deployment here. We randomly deployed 100 sensors over a 196 square meter region. The communication radius of the sensors was 2 meters. Our first simulation scenario considers sensor failure. We fail sensors at random in increasing numbers. We begin with 1 sensor failure and increase the sensors which fail, we observe that as we increase the number of agents, a greater amount of the sensor network is covered.

The curves do follow a trend similar to what was predicted

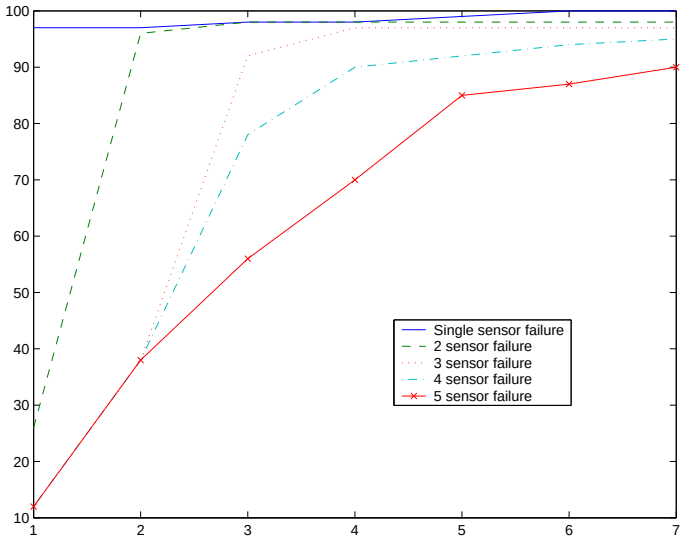


Fig. 1. Sensor failure and number of agents

in the theoretical analysis. The Y axis represents the number of sensors reached and the X axis represents the number of agents used. Next we consider the question of latency, which is the measure of time taken for the agents to cover the sensor network. Assuming a network which is equally and fairly partitioned by the agents (though this may rarely happen in practice), the latency with λ agents should follow a $\frac{C}{\lambda}$ like distribution. Our simulation results show a similar curve to what is theoretically predicted. This can be seen in Fig 2. Now consider the case of agents and queries, we simulate the process of reaching every sensor under three conditions when the ratio of cost of query to a sensor vs the cost of sending an agent is $\frac{1}{1000}$, $\frac{1}{500}$ and $\frac{1}{100}$. The results of the simulation can be seen in Fig 3. The queries grow as an exponent of the degree of the region of the graph and the agent cost is directly proportional to the size of the agent. If queries are not performed to sufficient depth, the agents will be dispatched to too many nodes and hence the full power of the queries will not be exploited. If queries are performed beyond this threshold, the total cost begins to increase since the query cost is exponential and is not sufficient to offset the agent

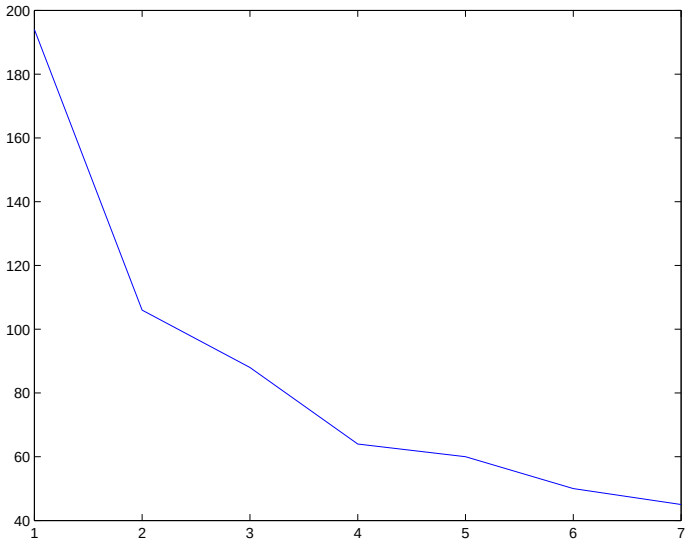


Fig. 2. Latency and number of agents used

cost, no matter how small the query information is. Here too,

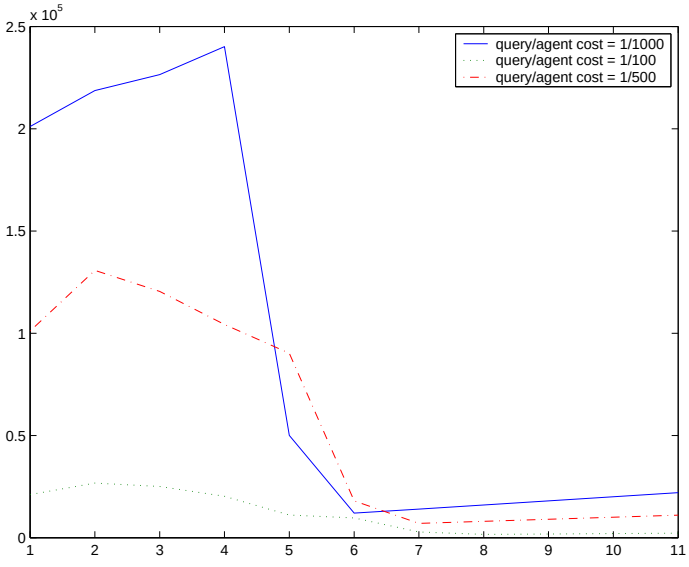


Fig. 3. Agents and queries used together

initially the cost of sending the agent dominates over the cost of queries. hence, the agent costs need to be minimized.

VI. Future work

There is scope for future work in the works presented in this paper. They can be summarized as

- What is the optimal number of agents needed for reaching maximum number of sensors
- What is the optimal query depth for algorithm using queries and agents. The query depth is critical to reduction in power consumption. A clever choice of query depth can increase network lifetime drastically.
- Study of more failure mechanisms other than what have been considered in the paper.
- Theoretical work on the relationship between the query depth d and the query to agent cost ratio.

VII. Conclusion

The need for efficient mechanisms which permit certain logic to reach every sensor to aid calibration, software redeployment etc has motivated us to present the three algorithms along with the theoretical analysis and simulation results, all of which are very encouraging.

We presented a single agent based scheme which was simple yet could be improved by using more agents. While we use theory and simulation to boost our assertions, we present yet another mechanism using queries and agents, creating a very energy efficient mechanism to achieve the same goal. The success of this hybrid approach depends very strongly on the query depth d. More future work would concentrate on the relationship of the parameter d and the parameters of the sensor deployment and agent cost to query cost ratio etc.

Acknowledgment

The authors would like to thank the Ubiquitous computing group in Honeywell Technology solutions lab in Bangalore for valuable feedback in improving the paper.

References

[1] R. M. Karp, *Reducibility among combinatorial problems*, In R. Miller and J. Thatcher, editors, Complexity of Computer Computations, pages 85-103. Plenum Press, 1972

[2] Eiman Elnahrawy and Badri Nath, *Cleaning and Querying Noisy Sensors*, Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications, 2003.

[3] Theodore S. Rappaport, *Wireless Communications, Principles and practice*, Pearson Education Inc.

[4] Niels Reijers, Koen Langendoen, *Efficient Code Distribution in Wireless Sensor Networks*, Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications, 2003.

[5] Wayne Jansen, Tom Karygiannis, *NIST Special Publication 800-19 Mobile Agent Security*.

[6] Qishi Wu, Nageswara S.V. Rao, Jacob Barhen, S. Sitharama Iyengar, Vijay K. Vaishnavi, Hairong Qi, Krishnendu Chakrabarty, *On Computing Mobile Agent Routes for Data Fusion in Distributed Sensor Networks*, IEEE transactions on Knowledge and data engineering, volume 16, No 6, 2004.

[7] Shivananjay Marwaha Chen Khong Tham, Dipti Srinivasan, *Mobile Agents based Routing Protocol for Mobile Ad Hoc Networks*.

[8] Hairong Qi, Yingyue Xu, Xiaoling Wang, Student Member, *Mobile-agent-based Collaborative Signal and Information Processing in Sensor Networks*.