Node Coverage Algorithms in Wireless Sensor Networks Using Mobile Agents

RAINA Manik KUMAR Subhas PATRO Ranjeet
(*Honeywell Technologies, Bangalore* 560076, *India*)
(E-mail: manik.raina@gmail.com, subhas.kumar@honeywell.com, ranjeet.patro@honeywell.com)

Abstract This paper proposes fault tolerant algorithms for routing mobile agents in a single channel wireless sensor network which cover every node in the network. These algorithms use local knowledge (assume no knowledge of global deployment or topology, *etc*). We propose the algorithms and show mathematical analysis to support our claims. The paper ends with simulation studies and discussion of results.

Key words Wireless sensor networks, mobile agents

1 Introduction

Ad hoc wireless sensor networks (WSN) are gaining prominence in areas of sensing, event detection and target tracking. Applications of this technology range from monitoring and control in military, ecological, environmental and domestic systems. WSNs are also used in automating buildings, universities factories *etc*. The Wireless sensor networks are one of the most constrained computational environments with restricted battery power which penalizes excessive network transmissions and CPU utilization. Algorithms for such networks must be computationally low cost, less prone to sensor failure and highly energy efficient. Further, they must not depend on any network topological information for efficient or correct operation as random deployment means such information is very expensive to gather. We consider the problem of designing algorithms which result in some computation occurring on all nodes of a sensor network. We propose three algorithms which meet some of these objectives.

1.1 Why a mobile agent based approach?

Mobile agents carry code and data which enables them to take dynamic decisions which may not be possible in other solutions. For example, while calibrating a sensor network, several schemes require more than one sensor to be involved in the process. This may need logic (code) to be carried around to the nodes to determine which sensors need to be involved in the calibration. Further, the agents running our algorithms return to the node (which originated the process) with the status of their activities which could contain valuable information they have gleaned from the sensor network.

1.2 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

1) Propagation of model parameters to every sensor node for systems which rely on models whose parameters frequently change with time or environmental conditions. For example, Elnahrawy and Nath[2] describe an approach which needs a noise model and prior knowledge. Our solution could update sensors with this information.

2) Global calibration of communication and sensing parameters of the sensors themselves. For example, the average large scale path loss for an arbitrary RF T-R separation and the speed of sound (for acoustic sensors).

3) Software redeployment: The mobile agents help identify if some sensors are running faulty or obsolete software and prompt them to update their versions from a base station or another sensor node in the sensor network. Even though Reijers and Langendoen[1] describe a scheme for distributing software updates wirelessly in a sensor network, their paper does not discuss a practical scheme which ensures that these updates actually reach every sensor in the network.

2 Related works

Marwaha *et al*[2] describe a hybrid algorithm which reduces route discovery and end-to-end latency in mobile ad-hoc networks. 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

Received December 16, 2005; in revised form March 15, 2006

what others are doing. In our scenario, we ensure our "labeling" scheme 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. Wu *et al*[3] 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 existence of global information and a hierarchical model of the wireless sensor network. This is different from our view where we do not put any such topological restrictions.

3 Problem formulation

3.1 Assumptions

We assume that each sensor contains a node identifier which could perhaps be pre-distributed. The node identifier for the n′th sensor can be represented as ID_n. The edge set $E(G)$ which represents the communication links in the wireless sensor network G forms according to the spatial constraint $\forall u, v \in V(G) \left(e_{uv} \in E(G) \equiv \left\|\vec{r_u} - \vec{r_v}\right\| \leqslant r\right)$, where r is the communication radius of the sensors in question.

3.2 Agent

An agent A_i is defined as an entity consisting of data, code and state. We formally define $jump(A_i, x, y)$ as the agent′s move from sensor x to sensor y.

3.2 Itinerary

An itinerary U_i of agent A_i is defined such that agent A_i takes subset of G while performing it's task. An *itinerary history* of agent A_i is defined as H_{A_i}. When an agent jumps from x to y, it stores the ID of y on the itinerary history (which we also call the stack) and performs an operation $g()$, which is a function evaluated at each sensor which could include gathering data, setting some flags, triggering operations in every sensor, *etc.* If y already exists on the itinerary history, it does nothing. Stack operations like push and pop are used on the itinerary history.

4 Our algorithms

Given a randomly deployed wireless sensor network, represented by a fully connected graph $G(E, V)$, the problem we wish to solve is to be able to visit every sensor in it. This problem is equivalent to determining a Hamiltonian cycle in a graph G is a well known NP complete problem[4]. Given these requirements, we propose three algorithms in our paper. The first algorithm utilizes a single agent to traverse the network and is based on a variant of the depth first search(DFS). While conventional depth first search (DFS) uses coloring, we use a more efficient mechanism utilizing two stacks. This is more energy efficient than conventional color based DFS. The reason is that the single agent would have to reach a sensor to determine if it has been colored already. In our case, that extra jump to a sensor is eliminated. We look at algorithm 1 which uses a single agent. We note that in the algorithm, u always denotes the ID of the sensor on which the agent resides and v the ID of the sensor on which the agent was residing before jumping to u.

Algorithm 1.

```
Visiting vertices in G using a single agent
REQUIRE: $u$ always denotes the ID of the sensor on which the agent
resides and $v$ the ID of the sensor on which the agent was residing
before jumping to $u$
COMMENT- Initialization done only at first node
$agentState = firstVisit$
$Stack = \{\Phi\}$
WHILE $stack$ is nonempty
    IF $agentState$ is $firstVisit$
        COMMENT–Application specific activity at sensor
        $g()$
        IF all neighbors of $u$ already on $stack$
            $agentState = recede$
            $jump(A, u, topOfStack(stack))$
        ELSE
            $findNext(N(u))$
        ENDIF
    ELSE
        COMMENT- We have been to this vertex before
        $pop(stack)$
        $findNext(N(u))$
    ENDIF
ENDWHILE
subroutine: $findNext(list)$
    find $w$ such that $ID_v \leqslant ID_w \leqslant ID_x \forall x \in list) \wedge w \notin stack \wedge w \notin exceptionList$
    IF $w \in stack \wedge ID_u > ID_w$
        $add(w, u)$ in exception list
        repeat previous step of finding $w$
        ENDIF
    IF no such $w$ exists
        STATE $jump(A, u, topOfStack(stack))$
    ENDIF
    $agentState = recede$
    $jump(A, u, w)$
    return to main routine
```

It is noteworthy that when the number of sensors failing increases, a part of the graph gets disconnected and hence there is no way any agent can reach some sensors. That explains why some sensors (which are alive) are not reached by any agent.

4.1 Using more than a single agent

A single agent is far more prone to sensor failures and takes longer amount of time than if we used more than one sensor. We now propose a method using multiple agents $\{A_1, A_2, A_3, \cdots\}$ and prove an improvement in each of the areas enumerated above where a single agent algorithm performs poorly. To ensure integrity, we must ensure that $\forall i, j \in [1, n],\ C_{A_i} \cap C_{A_j} = \{\phi\}$ if $i \neq j$, where C_{A_i} is the set of vertices which agent A_i visits. Based on a system parameter $\lambda \in Z^+$, an agent A traverses a random path P_l in G. $P_l = \{w_0, w_1, \cdots, w_{n-1}\}$ is a path chosen randomly such that it must have at least λ unique vertices and is called the labeling path. The set $L_\lambda = \{l_0, l_1, \cdots, l_{\lambda-1}\}$ is defined as the label set. A assigns labels as follows $\chi : P_l \to L_\lambda$ such that $\chi(w_r) = l_{r \ mod \ \lambda}$. Upon reaching $w_{n-1} \in P_l$, A replicates itself into λ 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.

Let us now turn our attention to the improvement this scheme offers when vertices fail. Since the labeling 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 $|N_i(x)| \geqslant 0$. 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 $C^1_{av} = n(G) \prod\limits_{i \in [0, n-1]} (1 - p_i)$ where $C_{A_1} = V(G)$.

For the general case, $C^\lambda_{av} = \sum\limits_{(0 \leqslant k \leqslant \lambda-1)} |C_{A_{k+1}}| \prod\limits_{(i \ | \ i \ mod \ \lambda \ = \ k)} (1 - p_i)$ where $\bigcup\limits_{(1 \leqslant k \leqslant \lambda)} C_{A_k} = V(G)$.

For a equally partitioned graph, $\forall i \in [0, \lambda - 1],\ p_i = p$. In this case, $C^\lambda_{av} = (1-p)^{\frac{\lambda}{n}} n(G)$. Clearly, since $p \leqslant 1$, C^λ_{av} becomes smaller as $\dfrac{\lambda}{n} \to 1$. In the general case $n(G)(1-p_{max})^{\frac{\lambda}{n}} \leqslant C^\lambda_{av} \leqslant n(G)(1-p_{min})^{\frac{\lambda}{n}}$, where $p_{min} = min\{p_1, p_2, p_3, \cdots, p_n\}$ and $p_{max} = max\{p_1, p_2, p_3, \cdots, p_n\}$. Clearly, as λ increases, both the lower and upper bounds increase, increasing the value of C^λ_{av}. Now we turn our attention to latency. We assume that the latency Λ (time taken) for an agent A_i to reach the sensors in G is proportional to the number of vertices reached by A_i, C_{A_i}. For $\lambda = 1$, $\varLambda \approx \sum\limits_{(0 \leqslant k \leqslant n-1)} C_k$. For the general case, $\Lambda = max\{C_{A_1}, C_{A_2}, C_{A_3}, \cdots, C_{A_\lambda}\}$. $C_{A_i} = \sum\limits_{k | kmod\lambda = i} C_k$. Clearly, $\sum\limits_{k | kmod\lambda = i} C_k < \sum\limits_{(0 \leqslant k \leqslant n-1)} C_k$ for $\lambda > 1$. With this justification, we now present an algorithm which reaches the vertices using λ agents.

We present the Algorithm 2 which reaches every sensor and uses more than one agent. This algorithm reaches every sensor and performs the sensor specific processing which is application specific. In the first line of the algorithm, u is assigned the ID of the current vertex where the processing begins. In line 2, P_l is chosen from $V(G)$ such that $|P_l| \geqslant \lambda$. During the *jump* call, the agent A will find itself at the vertex corresponding to the last entry in P_l. This is followed by a call to the *replicate* function which replicates the agent into λ agents $\{A_1, A_2, \cdots, A_\lambda\}$. From that point on, each of the agents acts independently and reaches every sensor not already labelled by another agent or labeled by it′s own color. After the *end for* (end of the for loop), the agents act independently as per Algorithm 1.

Algorithm 2. Visiting vertices in G using λ agents

```
REQUIRE- $u \leftarrow ID\ of\ the\ current\ vertex$
$chooseP_l \in V(G)\ such\ that\ |P_l| \geqslant \lambda$
$\forall x \in P_l$
    $label_{current\ node} \leftarrow label$
    $label \leftarrow (label+1) mod \lambda$
COMMENT- The Agent $A$ will find itself at the vertex corresponding to the
last entry in $P_l$
    $jump(A, u, x)$
    COMMENT- replicate agent $A$ into $\lambda$ agents $\{A_1, \cdots, A_\lambda\}$, from this point on
    each agent shall execute seperately reaching every sensor which is not labelled
    by any other agent or already labelled by it′s own color.
    $replicate()$
endfor
All $\lambda$ agents perform operations independently as in Algorithm 1
```

4.2 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. Queries in sensor networks are cheaper than transmitting agents as they are smaller in size. In this section, we look at queries and agents working together towards achieving a common goal. The queries are cheap because of their small size but their cost increases explode exponentially with query depth (as their replies cannot be aggregated leading to an increase in cost) while an agent is larger than a query but it does not multiply. Intuitively, if we use an agent which travels only to some sensors and sends queries to it′s neighbors within a small depth triggering off the function $g()$ to be evaluated in them, we should have a lower energy consumption. This is because the number of sensors to which the agent has gone is reduced as the agent is larger than the query packet.

Suppose a query is initiated at sensor u; the set of sensors (denoted as $N_d(u)$) which have a path of length d or less from u are said to be within depth d.

At a sensor u, the agent issues a query of depth d and all sensors in $N_d(u)$ execute $g()$ and reply to the query with basic topological information. The agent at sensor u determines which sensors in $N_1(N_d(u)) - N_d(u)$ it needs to visit and proceeds towards those sensors. It uses the stack as in the previous algorithms to keep track of where it is going. The subset $N_1(N_d(u)) - N_d(u)$ deserves special mention. This subset is empty if all the sensors at depth d from u have neighbors within $N_d(u)$, in which case queries reach all the sensors and we do not need to dispatch an agent further. All sensors visited by the queries are *marked*, and once *marked*, those sensors do not respond to further queries at later time. Algorithm 3 works on these principles.

The cost of a depth d query can be shown to be bounded as

$$\sum_{0 \leqslant k \leqslant d-1} (q+ku)d_{max}^{k-1} + q^{'}(\frac{d_{max}^{d+1} - (d+1)d_{max}^{d} + 1}{(d_{max}-1)^2})$$

where d_{max} is the highest degree of the sensor graph G, q, $q^{'}$, and k are constants. This expression is clearly exponential in d. For queries to be effective, the depth d must be large enough to avoid the agent visiting many sensors (and saving power) without the queries themselves consuming too much power. We will show in simulation that such an optimal d can be determined.

Algorithm 3.

```
REQUIRE:$u \leftarrow ID\ of\ the\ current\ vertex$
$agentState == firstVisit$ : Initialization done at first vertex
```

```
WHILE 1
    IF $agentState == firstVisit$
    COMMENT - Build local topological information using queries of depth $d$ from vertex $u$
        $N_d(u) \leftarrow Query(u,d)$
        $R_u = N_1(N_d(u)) - N_d(u)$
        IF $R_u == \Phi$
            $agentState = recede$
            $jump(A, u, topOfStack(stack))$
        endif
        $determine\ w \in R_u\ such\ that ID_w \notin stack$
        $jump(A, u, w)$
    ELSE
        $pop(stack)$ : Note- agent state is recede
        $find\ w \in R_u\ such\ that\ w \notin stack$
        IF $\nexists w$
            $jump(A, u, topOfStack(stack)$
        endif
        $agentState = firstVisit$
        $jump(A, u, w)$
    endif
endwhile
```

5 Simulations and results

We considered several simulation scenarios where the sensor network was disconnected. We confirmed that different instances of the agents mentioned in this paper could traverse every connected component of the network. Further, we confirmed that the simulation results were as predicted by our equations.

For illustration, we consider the following scenario (fully connected) of a randomly deployed wireless sensor network consisting of 100 sensors over a 196 square meter region. The communication radius of the sensors was 2 meters.

5.1 Sensor failure and number of agents

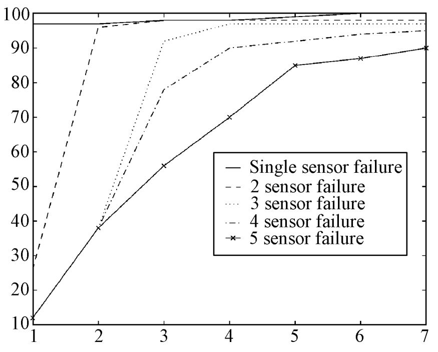

Fig. 1 Number of sensors reached (Y axis) vs number of agents used (X axis) under increasing sensor failures

First, we analyze the effect of random sensor failure on the number of sensors which are reached by our algorithm. We use a different number of agents in our deployment scenario and fail sensors randomly as the agents traverse the network. We determine how many sensors have been reached by the agents which have successfully returned to the originating sensor. As is clearly visible from the results in Fig. 1, if we follow a certain curve (say for example the curve for 5 sensor failures), increasing the number of agents along the X axis has a clear effect on increasing the number of sensors reached in the network. In fact, these curves follow the theoretical prediction that the average number of sensors reached C_{av} varies as $n(G)(1-p_{min}\dfrac{\lambda}{n})$, where n represents the number of agents used. The curve seems to conform to $y = n(G)(1 - \frac{k}{x})$.

It is noteworthy that when the number of sensors failing increases, a part of the graph gets disconnected and hence there is no way any agent can reach some sensors. That explains why some sensors (which are alive) are not reached by any agent.

5.2 Time required to reach all nodes and its relation to the number of sensors

Next we consider the question of latency which is the measure of time taken for the agents to cover the sensor network. The Y axis of this curve represents the time taken to cover the network and the X axis represents the number of agents used. 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, though the curve is not exactly a $\frac{C}{x}$ like curve, due to the simplifying assumption that the network is equally partitioned, which did not hold exactly true for our simulation. This can be seen in Fig. 2.

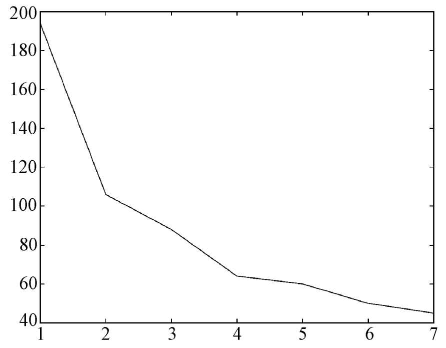

Fig. 2 Time taken in seconds to reach entire sensor network (Y axis) and number of agents used (X axis)

5.3 Hybrid algorithm consisting of agents and queries and energy reduction

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 cost, no matter how small the query information is.

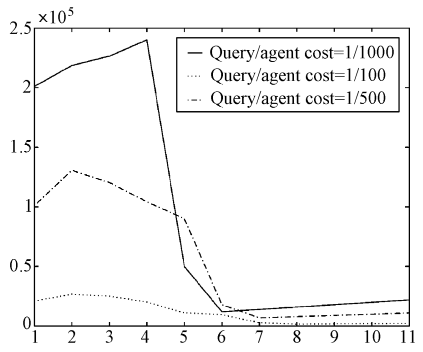

Fig. 3 Energy consumption (Y axis) plotted against query depth (X axis).

Here too, initially the cost of sending the agent dominates over the cost of queries, hence, the agent costs need to be minimized. We notice that there is an increase in cost when we increase query depth initially. The reason for that phenomenon is that the query depth increase incurs a cost penalty without sufficiently mitigating the cost in terms of reducing the number of sensors which the agent visits. As the query depth d reaches its optimal, fewer sensors are visited by the agent. This leads to the steep reduction in the energy requirement curve as shown. This is very promising.

6 Future work

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

1) What is the optimal number of agents needed for reaching maximum number of sensors

2) 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.

3) Study of more failure mechanisms other than what have been considered in the paper.

4) Theoretical work on the relationship between the query depth d and the query to agent cost ratio.

7 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*.

References

1 Reijers N, Langendoen K. Efficient code distribution in wireless sensor networks. In: Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications, San Diego, CA, USA: ACM Press, 2003. 60~67

2 Tham C, Marwaha S, Srinivasan D. Mobile agents based routing protocol for mobile ad hoc networks. In: Proceedings of IEEE Globecorn, Taipei, Taiwan: IEEE Press, 2002. 163~167

3 Wu Q S, Nageswara S V Rao, Jacob Barhen, Sitharama Iyengar S, Vijay K Vaishnavi, Qi H, Krishnendu Chakrabarty. On computing mobile agent routes for data fusion in distributed sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, 2004, **16**(6): 740~753

4 Karp R M. Reducibility among combinatorial problems, In R. Miller and J. Thatcher, eds., Complexity of Computer Computations. New York: Plenum Press, 1972. 85~103

RAINA Manik Works for the coordinated networks group at Honeywell Labs, Bangalore. He received his degree from NIT Calicut, India. His research interests include coding theory, iterative decoding, LDPC codes, WSN and UWB networks, and MAC and physical layer for UWB networks.

KUMAR Subhas Works for the Computational Systems group at Honeywell Labs, Bangalore. His research interests include cryptography, approximation algorithms, randomized algorithms, and computational complexity theory.

PATRO Ranjeet Works for the coordinated networks group at Honeywell Labs, Bangalore. He obtained his master degree from the Indian Institute of Science, Bangalore. His research interests include MAC analysis of 15.4 and 15.3 like networks, optimization, and operations research and its applications to everyday problems.