ALONG & ACROSS ALGORITHM FOR ROUTING EVENTS AND QUERIES IN WIRELESS SENSOR NETWORKS

Tat Wing Chim

Department of Electrical and Electronic Engineering
The University of Hong Kong, Pokfulam Road, Hong Kong
Tel: (852) 2857-8410; Fax: (852) 2559-8738; E-mail: twchim@eee.hku.hk

ABSTRACT

In this paper, we investigate efficient strategies for routing events and queries in a wireless sensor network where energy is a major concern. Our Along & Across algorithm makes use of a hop tree structure. Event attributes are routed along hop levels while queries are routed across hop levels to seek for match. Location information is not assumed. Simulation results show that our algorithm yields much higher hitting probability between event attributes and queries than a previously proposed algorithm, Rumor Routing algorithm in a moderate-traffic environment. As a result, our algorithm consumes up to 72.6% less transmission overhead. As such, our Along & Across algorithm is sound and should be very useful to wireless sensor network developers.

1. INTRODUCTION

A wireless sensor network (WSN) is made up of a large number of densely-distributed tiny nodes with sensing, limited computation and communication powers. Since sensor nodes are of a large number, manual replacement of batteries is not always possible. To elongate the lifetime of a WSN, energy awareness is a critical design issue in the WSN research community. Many routing protocols have been specifically designed for use in different applications of WSNs. A comprehensive survey of these protocols can be found in [1]. This paper focuses in the category of Query-Based Routing.

Many WSN applications require dissemination of sensed data to interested clients. Three major approaches have been proposed so far. The *push* approach requires sensor nodes to broadcast sensed data throughout the network. This approach is inefficient when the number of queries is comparably small. The *pull* approach requires interested clients to broadcast queries throughout the network and sensor nodes transmit data according to the demands. Directed Diffusion protocol [2] is an extension of this approach. The third approach balances these two extremes and tries to eliminate broadcasting of either data or queries. Sensor nodes and interested clients propagate sensed data attributes and queries, respectively, to seek for a match. Rumor Routing algorithm [3] is a representative work under this approach. In Rumor Routing algorithm, a node which witnesses an event probabilistically sends out an agent for advertising the event. The agent propagates using random walk and synchronizes each node's event table on its trail. An interested client also sends out a query using random walk. Upon meeting of event and query at any node, the answer is sent back to the interested client directly. Because of the random nature of random walk, hitting of event and query is not guaranteed. Rumor Routing algorithm assumes a flat random topology and unavailability of node location information. With the relaxed assumptions of a grid or near-grid topology and availability of node location information, Combs, Needles, Haystacks algorithm [4] was proposed. In this algorithm, making use of the grid or near-grid structure, a query is propagated to form a comb structure while an event is propagated to form a needle structure. With proper settings of comb and needle sizes, the hitting probability between query and event can be very high.

In lots of common applications of WSNs, such as providing information about enemies to soldiers on a battle field, sensor nodes are usually randomly dropped onto the field from planes or helicopters. It is almost impossible to obtain a grid or near-grid structure. On the other hand, it is also difficult to obtain node location information in these situations. Global Positioning System (GPS) receivers are too expensive to be installed on sensor nodes. Even GPS receivers are installed, GPS signals may be blocked or jammed. Moreover, localization techniques [5] cannot provide very accurate location information.

In this paper, we follow the assumptions of a flat random topology and node location information being unavailable as made by Rumor Routing algorithm. We also assume sensor nodes are densely distributed. Then we propose the Along & Across algorithm and show that it is far more efficient than Rumor Routing algorithm under identical moderate-traffic scenarios. In brief, we propose the use of a hop tree structure in which every sensor node possesses a hop level. Event attributes are then distributed us-

0-7803-9266-3/05/$20.00

ing an *Along* strategy - along same hop level nodes. Queries are propagated using an *Across* strategy - across hop levels. Cutting of an event level by a query yields a match.

The remaining of this paper is organized as follows: we explain our Along & Across algorithm in detail in Section 2. Next we present our simulation results and evaluate the performance of our algorithm in Section 3. We conclude this paper in Section 4.

2. ALONG & ACROSS ALGORITHM

Our Along & Across algorithm consists of three major parts: building of hop tree, distribution of event attributes and propagation of queries.

2.1. Building of Hop Tree

The initial step of our Along & Across algorithm is to construct a hop tree which is initiated by a random root node (say the last node being dropped onto a field). This root node generates a *Tree_Build* packet with the format <*SenderID*, *Hop*>, where *SenderID* and *Hop* are initialized to the root's ID and 0 respectively, and broadcasts it to all its neighbors. The tree building process continues by updating and flooding *Tree_Build* packets across the network. When a node receives the first *Tree_Build* packet, it simply stores locally the value of *Hop*, increments it, replaces *SenderID* by its ID and rebroadcasts the packet to all its neighbors. When a node receives the second onwards *Tree_Build* packet, it compares the value of *Hop* with the locally stored one. If the stored value is greater than the received one, the node updates its stored value, increments the received one and rebroadcasts as before. Otherwise, the node does not update its stored value and ignores the packet. As a result, every node in the network is configured with different hop levels. By means of the broadcast nature of wireless channels, all neighbors can overhear the *Tree_Build* packet transmitted by a node. Thus during the tree building process, each node builds up a list of its direct neighbors together with their hop levels. An example of a constructed hop tree is shown in Figure 1 where the number in each node represents the hop level of that node. In this example, all sensor nodes are assumed to have identical transmission range and the transmission range of the root is indicated by dotted line.

2.2. Distrubution of Event Attributes

We first describe how event attributes are distributed to facilitate the query process. Each node keeps an *Event Table* with forwarding directions to all events it knows. When an event source witnesses an event, it includes it into its *Event Table* and forms an *Event_Dist* packet with the format <*EventAttributes*, *EventSrcLevel*> in which *EventSrcLevel* represents the hop level of event source. It then checks its neighbor list to see if there are neighbors having the level *EventSrcLevel*. If yes, it forwards the packet to them. If not, it forwards the packet to neighbors having levels *EventSrcLevel* - 1 or *EventSrcLevel* + 1 equally likely. Any node receiving an *Event_Dist* packet first checks its *Event Table* to see if it has received information about the same event before. If yes, it ignores the packet. Otherwise, it records *EventAttributes* into its *Event Table*, sets the forwarding direction of this event to the sender of the packet and forwards the packet using the same set of rules as the event source. Again, by means of the broadcast nature of wireless channels, all neighbors can overhear the *Event_Dist* packet transmitted by a node. These nodes handle the overheard packet as if it is destined at them except that they will not help to forward it. Figure 1 shows how an event is distributed among nodes with hop level 4 in the tree structure.

2.3. Basic Propagation of Queries

When a node receives a query, it first checks its *Event Table* to see if there is a match. If not, it forms a *Query_Prop* packet with the format <*QueriedEventAttributes*, *HopList*> where *HopList* is used to record hops traversed by the packet. The query source then flips a coin to determine whether it should forward the packet upwards (to upper levels) or downwards (to lower levels) first and sends out the packet. A node receiving a *Query_Prop* packet first updates *HopList* and then checks its *Event Table* to see if there is a match. If yes, it directs the packet to the neighbor leading to the event according to its *Event Table*. If not, it forwards the packet to a neighbor in the opposite direction as the sender. For example, it forwards a packet received from an upper level neighbor to a lower level neighbor. If there is no such neighbor (for example, the node already has the lowest or highest level), it means the query cannot be answered in the selected direction. In this case, the node holding the packet forms a *Query_Failed* packet which is identical to the *Query_Prop* packet and forwards it back to the query source using the reverse path shown in *HopList*. When the query source receives the *Query_Failed* packet, it forms a new *Query_Prop* packet and forwards it to a direction opposite to the one selected before. The above process repeats. If the query source receives *Query_Failed* packets from both upper and lower directions, it means the query cannot be answered. In this case, the query is flooded across the whole sensor network to seek for an answer. Figure 1 shows how a query initiated from a level 2 node is propagated to seek for an answer.

2.4. Advanced Propagation of Queries

When a *Query_Prop* packet is going upwards, an intermediate node may not be able to find an upper level neighbor.

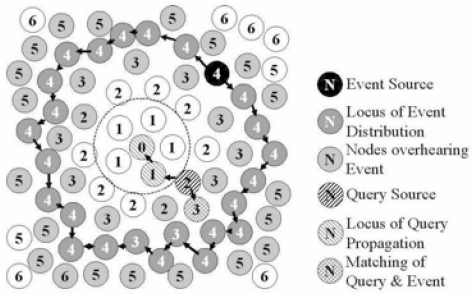

Fig. 1. Sample Hop Tree with Event and Query

Thus in this subsection, we propose an advanced propagation model to improve this situation. This model makes use of a *Query Routing Table (QRT)* to be maintained by all nodes. After the hop tree is built, a node without upper level neighbors forms a *QRT_Build* packet with the form <*HighestHop*> where *HighestHop* is initialized to the hop level of that node. The node then forwards the packet to a lower level neighbor at random. When a node receives a *QRT_Build* packet, it stores the value of *HighestHop* and the corresponding incoming direction into its *QRT* and removes all entries with lower *HighestHop* values. It then continues to forward the packet downwards by randomly picking up a lower level neighbor. As a result, some nodes including the root build up a *QRT* with information about how to reach the top of the tree. When a node receives a query, it first checks its *Event Table* to see if there is a match. If not, it forms a *Query_Prop* packet as before but instead of flipping a coin, it always forwards the packet downwards first. When the packet reaches the root, the root forwards the packet upwards by randomly picking up a neighbor which can lead to the top of the hop tree according to its *QRT*. This advanced query propagation model ensures a *Query_Prop* packet go through all hop levels in the tree. The only drawback here is that the root as well as nodes nearby may need to handle more traffic. Without loss of generosity, since there are more high level nodes than low level ones, the probability that a query is originated by a high level node is much greater than that by a low level node. That is given any query, it is more likely that its originating level is higher than the corresponding event level. Thus a match should occur in the mid-way and the query does not need to be routed to the root.

2.5. Optimization of Return Path

When the event source receives a *Query_Prop* packet, it forms a *Query_Answer* packet and forwards it back to the query source using the reverse path shown in *HopList*. However, this reverse path may be longer than necessary. As such, we propose an optimization method to shorten the reverse path. Instead of sending the *Query_Answer* packet to the immediate next hop on the reverse path, a node receiving this packet sends it to the furthest ahead neighbor. For example, if the reverse path is $5-4-3-2-1-0$ and node 5 finds that both node 3 and node 2 are its neighbors. Then node 5 will direct the packet to node 2 (the furthest ahead neighbor). As a result, the packet travels through 3 hops instead of 6 hops.

3. SIMULATION RESULTS

3.1. Simulation Models

To measure how effective our Along & Across algorithm is, we implemented Rumor Routing algorithm and our algorithms using a simulator written in C++ language. We compared their performance over 10 trials for a dense network setting [5] (500 nodes were uniformly distributed across an area of 50m $\times$ 50m). Similar to [3], a simple radial propagation model was used. That is, each node could reliably send packets to any other nodes within its 5m transmission range. Simulations were conducted with minimum node degree of 6 since network partitioning tends to occur for minimum node degree less than 6 [5]. The resulting average node degree and average maximum node degree over the 10 trials are 16.5 and 27 respectively.

Following the rough proportion of event and query numbers to sensor node number given by [3], we assume there are 15 events and 150 queries. Each event is generated by a random node and all events are assumed to be distinct. Each query is also generated by a random node and is querying one of the 15 events. Two queries can be querying the same event.

Rumor Routing algorithm requires setting query (or agent) and event TTLs. We set these two values to 60 and 40 respectively. We found that with these two values, the number of transmissions used for event distribution (M_e) and query propagation without flooding the network (M_q) in both Rumor Routing and Along & Across algorithms are comparable (as shown in Table 1).

3.2. Simulation Results

Since our Along & Across algorithm is based on a hop tree structure, we make a brief analysis of the hop tree structure here. For each of the 10 random topologies, we built a hop tree rooted at a randomly selected node. On average, 99% of nodes have at least two same hop level neighbors while 82.24% of nodes have at least one upper hop level neighbor. For each event, let's define the percentage of nodes having the same level as the event source and have information about that event be the event spreading factor. Among the 10 topologies and among all the 15 events for each topology, the average event spreading factor is 98.81%. That means whenever an event source sends out an *Event_Dist* packet,

almost all nodes on the same hop level can receive it. As such, our Along strategy for event distribution is very efficient.

Table 1 summaries the performance comparisons between Rumor Routing (RR) algorithm and our Along & Across (AA) algorithm (with both basic and advanced query propagation stragegy). The items being compared include transmission overhead for event distribution (M_e), transmission overhead for query propagation without flooding (M_q), percentage of answered queries without flooding (A_q), average hop delays experienced by queries (from sending out *Query_Prop* packet to receiving *Query_Answer* packet) (D), total transmission overhead (T) and transmission overhead per node (T_n). Note that transmission overhead includes those for tree building and *QRT* construction where appropriate.

The results show that the random nature of random walk in Rumor Routing algorithm makes the hitting probability low. In particular, when the number of events is not large enough, there is not much agents to help to spread event information out. As a result, only 62.8% of queries can be answered before flooding is applied. On the contrary, our algorithm with both basic and advanced query propagation strategy gives higher hitting probability (80.7% and 99.4% respectively) without flooding the network.

Concerning the average hop delay experienced by queries, both Rumor Routing algorithm and our advanced approach yield comparable performance. With basic query propagation strategy, the queries experience about 16% higher delay on average because of the fact that a query source may pick up a wrong direction initially and some time is wasted in searching the wrong direction. In the future, we will consider some smarter ways such as history information and hop level of query source in making the initial decision of propagation direction.

In the simulations, we assume flooding is used once a query cannot be answered. Therefore, the lower the hitting probability between queries and events, the higher the probability that flooding is used and the higher the total transmission overhead is. This explains the performance of total transmission overhead and transmission overhead per node among the algorithms in the last two rows of the table.

To summarize, with the same number of queries being answered, our Along & Across algorithm with basic query propagation strategy requires 30.7% less transmission overhead than Rumor Routing algorithm. With our advanced query propagation strategy, the saving can be up to 72.6%. Therefore our Along & Across algorithm is sound.

The density and failure rate of sensor nodes may slightly affect the performance of our Along & Across algorithm. However, due to limited space here, we leave the related analysis to our future work.

Algorithms	RR	AA Basic	AA Advanced
M_e	900	1017.3	1017.3
M_q	7173.4	7971.2	7199.2
A_q	62.8%	80.7%	99.4%
D	47.8	55.6	47.9
T	35609.8	24683.6	9765.1
T_n	71.2	49.4	19.5

Table 1. Performance Comparisons between Algorithms

4. CONCLUSION

In this paper, we proposed Along & Across algorithm for efficient routing of events and queries in a WSN where energy is a major concern. Our Along & Across algorithm makes use of a hop tree structure. Event attributes are routed along hop levels while queries are routed across hop levels to seek for match. Just like a previously proposed algorithm, Rumor Routing algorithm, location information is not assumed. Simulation results showed that our algorithm yielded much higher hitting probability between event attributes and queries than Rumor Routing algorithm in a moderate-traffic environment. As a result, our algorithm consumed up to 72.6% less transmission overhead than Rumor Routing algorithm. As such, our Along & Across algorithm is sound and should be very useful to wireless sensor network developers.

5. REFERENCES

[1] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," *IEEE Wireless Communications*, pp. 6 – 28, Dec. 2004.

[2] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed Diffusion for Wireless Sensor Networking," *IEEE/ACM Transactions on Networking*, pp. 2 – 16, Feb. 2003.

[3] D. Braginsky and D. Estrin, "Rumor Routing Algorithm For Sensor Networks," in *Proceedings of the WSNA '02*, Sept. 2002, pp. 22 – 31.

[4] X. Liu, Q. Huang, and Y. Zhang, "Combs, Needles, Haystacks: Balancing Push and Pull for Discovery in Large-Scale Sensor Networks," in *Proceedings of the Sensys '04*, Nov. 2004, pp. 122 – 133.

[5] S. Y. Wong, J. G. Lim, S. V. Rao, and W. K. G. Seah, "Density-Aware Hop-Count Localization (DHL) in Wireless Sensor Networks with Variable Density," in *IEEE Proceedings of the WCNC '05*, Mar. 2005.