http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (5): 759-778, 2006
ISSN 1812-5638

Timing Control During Data Aggregation Operations of Distributed Sensor Networks: Trade-off Between Query Accuracy and Response Delay

Fei Hu and Carter May
Department of Computer Engineering, Sensor Network Research Center,
Rochester Institute of Technology, Rochester, NY, USA

Abstract: Data aggregation in Wireless sensor networks (WSNs) has attracted wide attention since it could reduce the wireless communication data amount and thus save power consumption, which is one of the top concerns in low-power WSN systems. Given a Data Aggregation Tree (DAT) for data query purpose in a WSN and the expected query accuracy, we will show the feasibility to adaptively adjust the waiting period in each DAT level so that the end-to-end query delay could be optimized, i.e., without unnecessary pause in an aggregation node. This study will introduce some important data aggregation parameters (such as desired query accuracy and practical query resolution, etc.) to study the optimal, low-cost timing control scheme. For such an adaptive timing control scheme, we propose to use a Finite State Machine (FSM)-based auto-feedback control algorithm to adjust the waiting period of each DAT node to timely respond to the varying query results performance. For instance, in the beginning of the query task, the FSM algorithm should be able to use the fast speed to increase the waiting period so that more query results can be collected, while the algorithm should use micro-adjustments to slowly adjust the waiting period after the query accuracy has reached a certain level. From quantitative analysis, networking simulations and experimental setup perspectives, we will investigate the efficiency of our FSM-based timing control algorithm in terms of achieving a good trade-off between query accuracy and end-to-end query response delay. Our sensor network simulation results and hardware experiments have verified the validity of our proposed timing control scheme. In addition, we will also compare different FSM algorithms in different WSN data aggregation scenarios.

Key words: Wireless Sensor Networks (WSN), distributed sensor networks, data query, data aggregation, timing control, finite state machine, end-to-end query delay

INTRODUCTION

Advances in electrical engineering and microelectronics have allowed electronic devices to shrink in both size and cost. It has become possible to incorporate environmental sensors into a single device with a microprocessor and memory to interpret the data and wireless transceivers to communicate the data. These sensor nodes have become small and cheap enough that they can be distributed in very large numbers into the area to be monitored and can be considered disposable. Once deployed, these sensor nodes are able to self-organize themselves into a multi-hop wireless network, called Wireless Sensor Network (WSN), which greatly differs from a general Mobile Ad hoc NETwork (MANET) as follows (Akyildiz *et al*., 2002):

- The number of the nodes in a WSN is significantly larger than in a MANET;
- Sensors are usually low-cost devices with severe constraints with respect to energy source, computation capabilities and memory; while actuators generally have relatively higher energy storage, which allows longer wireless transmission distance;
- The sensors are usually stationary or with quite limited mobility;
- Mode of communication in WSNs typically is many-to-one (from sensors to sink), while it is typically peer-to-peer in MANETs;
- Data aggregation may not be important in MANET, however, it is a necessary scheme in data query tasks of large-scale WSNs because it can greatly reduce the communication data amount (Akyildiz *et al*., 2002).

In order to reduce energy consumption (the energy for one bit of wireless communication in typical sensor

Corresponding Author: Fei Hu, Department of Computer Engineering, Rochester Institute of Technology, 83 Lomb Memorial Drive, NY 14623-5603, USA Tel: (585) 475-5139 Fax: (585) 475-5041

networks can be used to execute over 1000 local instructions in each sensor), many sensor network applications attempt to minimize the amount of data transmitted by using some form of data aggregation. A sensor query task that asks for the maximum value sensed is an ideal candidate for aggregation operations because any number of messages can be easily combined into a single value at the aggregator. A query for the average over a field is slightly more complicated in that the number of responses must be communicated to calculate the average, but the final message will still be shorter than multiple individual responses. However, a query for all values sensed would not be a candidate for aggregation because each value must be appended to the message and the final message will be proportional in length to the number of sensors responding.

The focus of this study is to determine appropriate timing control scheme in data aggregation operations, which is a largely unexplored issue in the WSN Middleware layer. In a large scale WSN, there may be a significant delay between the time when an event is sensed (or a query is answered) and the time when the data reaches the sink. For better query accuracy, each aggregating sensor should be able to transmit all data received from all of its children sensors at one time. However, this could lead to an even larger delay between when the data originates and when it is finally reported, as each aggregating node may have to pause to wait for query results from more children sensors (to achieve higher query accuracy) before sending a report to its parent node. Depending on the priority of query accuracy vs. maximum latency, it may make more sense for an aggregator to wait for all, some, or only the first of its children to respond. As an example, Fig. 1 shows a network with a potential aggregation timing problem. In the case in which only leaf nodes respond to a query, if Node B wishes to aggregate responses from its children, it will be faced with a decision of whether to wait for the slower response from Node C or to transmit the response from Node D first and send C's response later. Here we may assume that each transmission over a wireless link takes exactly the same amount of time. If this is not the case, the timing control problem becomes more complicated and less deterministic.

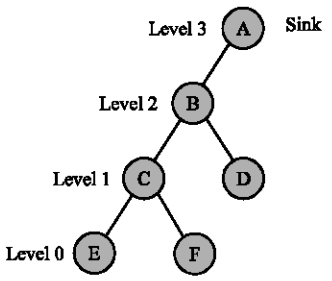

Fig. 1: Potential timing control problem in a data aggregation tree

Based on the timing models described by Solis and Obraczka (2003), existing periodic data aggregation protocols can be classified into three categories, namely: periodic simple, periodic per-hop and periodic per-hop adjusted. (1) Periodic simple aggregation works by having each node wait a pre-defined period of time, aggregate all data items received and send out a single packet containing the result. (2) Aggregation mechanisms in the periodic per-hop category have nodes send the aggregated packet as soon as they hear from all their children. While excessively late reports are dropped as in periodic simple. (3) Finally, periodic per-hop adjusted schemes use the same basic principle of periodic per-hop but schedule a node's timeout based on its position in the distribution tree (rooted at the information sink and spanning all reporting and intermediate nodes). The aggregation scheme proposed in this study falls within this category and, when compared to other existing periodic per-hop adjusted algorithms, presents benefits such as not requiring clock synchronization among nodes and minimizing the timeout scheduling overhead.

In this study, we attempt to answer the following three questions:

- Given a Data Aggregation Tree (DAT) for data query purpose in a WSN and also given the expected query accuracy, is it possible to adaptively adjust the waiting period in each DAT level so that the end-to-end query delay could be optimized, i.e., without unnecessary pause in an aggregation node?

  In present study, we will introduce some important data aggregation parameters (such as desired query resolution, practical query resolution, etc.) to study the low-cost timing control scheme.
- For such an adaptive timing control scheme, how we could use a Finite State Machine (FSM)-based auto-feedback control algorithm to adjust the waiting period of each DAT node to timely respond to the varying query results performance?

  For instance, in the beginning of the query task, the FSM algorithm should be able to use the fastest speed to increase the waiting period so that more query results can be collected, while the algorithm should use micro-adjustments to slowly adjust the waiting period since the query accuracy has reached a certain level.

- From both quantitative analysis and experimental perspective, what is the efficiency of such a FSM-based timing control algorithm in terms of achieving a good trade-off between query accuracy and end-to-end query response delay?

  In this study, we will use math model, (OPNET, 2006) networking simulations and also WSN hardware experiments to verify the validity of our proposed timing control scheme. We will also compare different FSM algorithms in different data aggregation scenarios.

RELATED WORKS

General WSN issues: A survey of the early work on WSNs is provided in (Akyildiz *et al*., 2002). While wired multi-sensor systems were studied in the 1980's (Koniger, 1989; Kleinschmidt and Mock, 1989), since the mid to late 90's, there has been great interest in WSNs (Bult, 1996; DAPRA, 2002). A number of researchers have presented related visions for WSN, including hardware designs and applications (Warrior, 1997; Asada, 1998; Chandrakasan, 1999). Recently much research has been conducted on coverage and connectivity (Meguerdichian and Koushanfar, 2001; Meguerdichian and Gu, 2001), time synchronization (Romer, 2001; Elson *et al*., 2002), sensor localization (Hightower and Borriello, 2001), MAC (Woo and Culler, 2001) and other important sensor network issues.

WSN data query: The database perspective on sensor networks and the use of data aggregation operators to optimize query performance for such sensor databases have been described in Cougar (Yao and Gehrke, 2002), Acquire (Sadagopan *et al*., 2003) and others (Bonnet *et al*., 2000; 2001; Govindan, 2002; Madden and Szewcyk, 2002; Bonfils and Bonnet, 2003). Data centric storage using geographic hashing is discussed in (Ratnasamy, 2002) and DIFS (Greenstein *et al*., 2003) is a distributed indexing technique for data-centric storage. DIMENSION (Deepak *et al*., 2003) provides hierarchical wavelet-based storage for drill-down queries, while (Goel and Imielinski, 2001) describes the use of MPEG-like video coding techniques to exploit both spatial and temporal correlations for energy savings. IDSQ/CADR (Chu *et al*., 2002) is an active querying technique that attempts to balance information gain with communication costs while querying neighboring sensors and guiding queries through the network. Other active querying techniques include the mobile-agent based Rumor Routing technique (David and Deborah, 2002) and ACQUIRE (Sadagopan *et al*., 2003).

Data aggregation research: Data aggregation is an important operation during Data Query tasks since it can greatly save the sensor energy consumption and thus prolong WSN operation lifetime. Directed Diffusion is a routing algorithm that has been proposed for use in wireless sensor networks (Intanagonwiwat, 2002). Its authors mention that it creates a network topology suitable for data aggregation, but stop short of specifying details for any specific implementation. Others have proposed detailed aggregation schemes intended to be used with Directed Diffusion, including Greedy Aggregation (Solis and Obraczka, 2003) and CLUDDA (Chatterjea and Havinga, 2003). However, none of these address the issue of timing control between different levels of sensors. Most aggregation schemes focus on finding an optimal node at which to aggregate data, simply mentioning that aggregation is performed there and often assuming in their calculations that all responses received are aggregated before being propagated.

TAG (Madden and Franklin, 2002), or Tiny AGreggation, is a good example of a periodic per-hop adjusted aggregation mechanism. TAG uses aggregation as queries are processed within the network. Some queries in TAG request reports to be sent from sensors periodically. In this case, TAG intelligently subdivides the data collection epoch into smaller slots. Each slot is the epoch length divided by D, the depth of the tree. Following per-hop adjusted aggregation operation, slots are assigned to nodes in decreasing order, D, D-1, D-2,. .., as the query propagates through the network. This scheme requires knowledge of the network topology and time synchronization between nodes, but allows nodes to power down when not scheduled to transmit or receive. Our proposed aggregation scheme is not affected by the potential sleep schedule and our MAC layer scheme, discussed later, takes full advantage of it.

In addition to TAG (Madden and Franklin, 2002), another well-received aggregation scheme is AIDA (He *et al*., 2004). Though both of these schemes suggest several potentially energy-saving ideas, they focus on disjoint aspects of aggregation. TAG's main focus is on an efficient querying language that is conducive to aggregation, but is mainly an application-level optimization. AIDA, on the other hand, inserts a new layer into the protocol stack that interprets and repackages data near the MAC layer, but does not consider dynamic timing parameters based on application-level requirements. Both of these frameworks are useful for reducing energy consumption in WSNs and are compatible with our timing protocol, but do not adequately address the issue of trade-off between query accuracy and response delay.

In (Abdelzaher *et al*., 2004), the authors demonstrated the application of control theory to resolve fundamental performance trade-offs in sensor networks. Fundamental limits were presented on real-time network capacity. These limits where then used to derive set points of control loops. Two different mechanisms for data aggregation were presented whose combined effect is to maximize information throughput while reducing protocol overhead. Their focus is to utilize WSN capacity limit models and general control theory to achieve a low-complexity data aggregation. While the focus of our study is to control the 'waiting period in each level of the data aggregation tree in order to achieve a good trade-off between query accuracy and end-to-end query response delay.

The concept of cascading timeouts, where nodes would wait for a period of time directly related to their depth in the aggregation tree, was recently proposed in (Solis and Obraczka, 2003). Though this is a potentially useful optimization, its main shortcoming is that it requires significant additional data to be transferred during the setup period. Our proposed timing control scheme (details later) requires minimal overhead, but is flexible enough to allow expansion for later optimizations.

TIMING CONTROL DURING DATA AGGREGATION

Assumption of data query topology

A cluster-tree architecture: Before the description of our timing control in data aggregation, we provide the assumption of the data query network topology. In terms of WSN topologies for the purpose of optimal data aggregation, we propose to make use of both cluster-based and tree-based methods because we have verified that the hybrid tree-cluster architecture can adapt to large-scale routing operations (Fei *et al*., 2006). To form such a topology, first, the sensors should self-organize themselves into different clusters using any clustering algorithm such as our probabilistic approach (Fei *et al*., 2006). There is a cluster-head in each cluster that can aggregate the data from other sensors. Next, a Data Aggregation Tree (DAT) is formed among those cluster-heads with the WSN sink as the tree root. Thus groups of sensor nodes will combine their reports at the lowest level and reports may continue to be aggregated as they pass up the aggregation tree toward the root node. The Cluster-tree is a promising data query topology architecture in terms of overall energy efficiency (Greenstein *et al*., 2003). Since each cluster-head first performs local aggregation in its cluster before forwarding data to the next cluster-head, this study will focus on the data aggregation issue in the entire tree (i.e., the data aggregation issues among cluster-heads instead of inside each cluster).

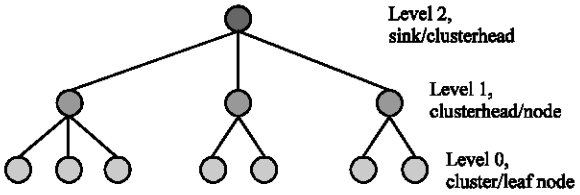

Fig. 2: Data aggregation tree (DAT)

Consider Fig. 2 DAT scenario. In this example, each circle represents a sensor node. Groups of two or three sensor nodes at level i are grouped together under the control of another node at level i + 1. Nodes at level i = 0 are leaves in the tree and function only as sensors. The vast majority of nodes in the network fall into this category. Nodes at level i + 1 and higher, up to the root of the tree, would be considered cluster-heads.

With an increasing cluster-size r (the number of sensors in each cluster), cluster-heads will have to transmit farther to reach each other, but the tree of cluster-heads will be simpler. A simpler tree leads to more energy-efficient topology maintenance at this level. Also with an increasing cluster-size r, maintenance and communication costs for each cluster will be increasing. Thus there is a balance to be struck between cluster size and tree size. The larger the clusters are, the smaller the tree of cluster-heads may be and vice versa. The optimum balance may be calculated offline and goes beyond of the focus of this paper. Please refer to (Heinzelman, 2000) for some quantitative discussions on the determination of the cluster size.

Proposed timing control protocol during data aggregation: As mentioned before, aggregating sensors in the higher levels of DAT have to wait for data from their children to arrive, before they send aggregated data to their parent node. There is a design tradeoff in the maximum amount of time to wait. If an aggregating node waits too long, the results of the query may not be time-relevant. This is especially true in situations in which the data sink may want initial results immediately. On the other hand, if the waiting period is too short, the aggregation node could not collect enough query results from its children nodes and thus the final query accuracy does not meet desired requirement. In this section, we will propose an efficient adaptive control algorithm for the adjustment of waiting period in each DAT level with the objective to guarantee a certain aggregation accuracy and

also to achieve an optimal end-to-end delay. For the convenience of describing our timing control algorithm, we provide the definitions of some variables to be used later:

- N_{opt} - the optimal number of responses the sink desires. This is based on application-level knowledge and should be pre-determined by the system. If the sink receives fewer responses than N_{opt} it will know to increase the waiting period. If the sink receives more than N_{opt} it will know to decrease the waiting period.
- N_{rec} - the number of responses received. Please note that the values of N_{opt} and N_{rec} could be simply the total number of nodes that have responded to the data request or the number of query result packets received at the sink. From the timing control algorithm viewpoint, as long as we use consistent metrics and performance parameters, it should not matter in terms of the efficiency of the algorithm.
- n - the round number for a specific data query task. During a round of data aggregation, the WSN sink (i.e., the root of the DAT) waits for responses to a data request. There could be multiple rounds of data aggregation operations for any data query task. The total number of rounds is application specific and should be determined beforehand by analyzing the network characteristics. The time duration of each round varies depending on the propagation delay of the network, number of collisions, effective throughput and other WSN environment parameters.
- T_n - The sum of the current waiting periods in all DAT levels, i.e., the total timeout period of the current data aggregation round for a specific data query task. The relationship between the entire waiting period T_n (across all DAT levels) and the waiting period in each tree level will be analyzed.
- T_{n+1} - The sum of the waiting periods in all DAT levels to be used for the next data aggregation round for a data query task. The main goal of our timing control scheme is to set up the value of T_{n+1} based on the values of T_n and other parameters related to the query accuracy, such as N_{opt} and N_{rec}.

Our proposed aggregation timing control protocol makes use of a separate 'setup' phase to distribute parameters necessary for aggregation. This is typical of and compatible with, most dynamic WSN routing and MAC protocols. To reduce the protocol overhead of timing control, the setup phase takes some finite amount of time and is followed by a much longer data collection period. The concrete timing durations in each phase can be set up depending on the protocol overhead

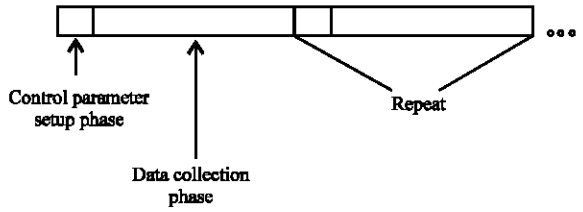

Fig. 3: Setup vs. data collection phases

requirements of WSN applications In this study, we use the ratio 1:10 for the durations of setup phase and data collection phase. These phases are scheduled to occur periodically. Figure 3 shows the phases of timing control protocol.

In the following discussion, we assume that a Data Aggregation Tree (DAT) has already been formed through some current sensor data query schemes such as in (Madden and Szewcyk, 2002; He *et al*., 2004) and we then add our timing control algorithms to set up the waiting periods T_n (n = 1,2,3 …) in each round of data aggregation.

Setup phase

- The WSN Sink (the DAT root) broadcasts the Tree depth request. The request is propagated down through the entire DAT (similar to a simple data query propagation).
- Message reaches the DAT bottom level nodes (i.e., tree leaves) and is then returned back up. Each node returns with its hop count, starting from leaves nodes (at DAT level 0). The parents of these leave nodes realize that they are at level 1 and pass this information back up the tree. This continues until the sink retrieves DAT depth information and is able to construct the entire DAT topology. Each node propagates the 'depth request' to its children and each node returns an answer to its parent. Thus this procedure is accomplished with complexity O (depth).

  In order to further reduce data amount (thus save energy consumption) during this setup phase, some level of aggregation may be performed at this step. Possibilities vary from transmitting information (that completely describes the network) to communicating only the maximum depth of the tree.
- Finally, the sink will calculate and transmit an appropriate value for T_0 based on the depth of the tree, the maximum tolerable latency. In addition, the sink will also propagate the optimal number of responses, N_{opt}, which is related to the concrete query accuracy requirements of a WSN application, to the entire DAT.

After performing this exchange, each aggregating node, including the sink, should have all knowledge necessary for it to calculate an appropriate timeout period. Depending on the level of aggregation performed in step 2, nodes may be aware of how balanced or unbalanced the tree is, or parent nodes may be aware of the entire aggregation tree below their positions.

Data collection phase

- The sink transmits a query message with an updated timeout period T_{n+1}. This operation does not require any time synchronization because the sensors and aggregators just maintain the previous value T_n until notified for change. Aggregators that receive the query with updated timeout period will reduce it uniformly before propagating the query to its children nodes.
- Each aggregator replies with the number of replies received (average, per query) and its depth level in the tree. Note that an aggregator that is the parent of another aggregating node will sum its total number received (N_{rec}) with its children's report.

  In order to get the most accurate view of the event data, aggregators may also report the number (N_{late}) and timing information (T_{late}) of late reports. This would allow the sink, if powerful enough, to choose an appropriate aggregation period even more intelligently. For example, if the sink does not receive a satisfactory number of responses, it may calculate the benefit of increasing the aggregation period. Disregarding network fluctuations, this can be done by simply counting the number of nodes whose additional latenesses were reported to be less than the minimum increment of the aggregation period.
- The sink will send out an updated T_{n+1} to all aggregators. The updating algorithm is based on a Finite State Machine (FSM) scheme.

Finite State Machine (FSM) in Timing Control Algorithm: The trade-off between the query accuracy and the end-to-end query delay is achieved by the adaptive adjustment of the aggregation period, which is calculated according to a Finite State Machine (FSM) scheme. We have developed several FSMs for query performance evaluation. The FSM aims to maintain the number of messages received as close as possible to the optimal number (determined by the application). In addition, from control timing viewpoint, our FSM is desired to reach this point as quickly as possible. Before the description of our FSM schemes, we list some variables as follows:

N_{opt} Optimal number of responses; determined by the application.
N_{rec} Number of responses received; tallied by the aggregators and reported to the sink
T Maximum aggregation period, distributed by the sink
L^+ Maximum latency (application level)
T_i Maximum aggregation period at DAT level i.
n The round; T is calculated once for each round, which is assumed to be long enough to create a heuristic without becoming stale.
T_{opt} Aggregation period that satisfies $N_{rec} = N_{opt}$
c A parameter chosen by the application in the sink that relates N_{opt} to T_{opt}
N_{late} Number of late packets received by aggregators
T_{late} Time units while waiting for late packets
D Depth of the tree
K Level of node i in the tree.
Δ Difference between T at each level of the tree

If D is known, the sink can derive a maximum T_i for each level in the tree, based on the maximum latency (L^+) of the application:

$$T_i = L^+ - K \cdot \Delta \quad (1)$$

Since the nodes know their own level in the tree, they can individually calculate their aggregation periods, using the above formula (assuming Δ is available). Our timing control protocol is capable of distributing this delta as part of the control message. Though there may be some advantage using a variable Δ, there is a tradeoff in the amount of control data transmitted. This study assumes a constant Δ.

Figure 4 is the simplest FSM (we call it Version 1). At each evaluation of the aggregation period, the optimal number of messages received (N_{opt}) is compared to the actual number of messages received (N_{rec}). If there were too few messages received, the aggregation period is increased by one atomic unit. If more messages were received than were needed, the aggregation period is linearly decreased in a similar manner. The amount of increase and decrease can also be varied depending on the application and network characteristics.

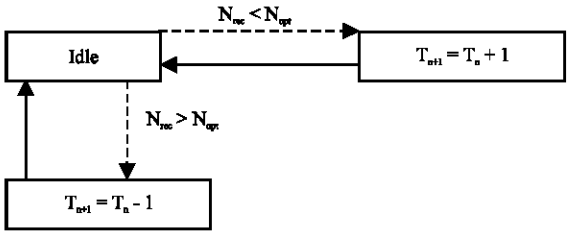

Fig. 4: Finite state machine (Version 1)

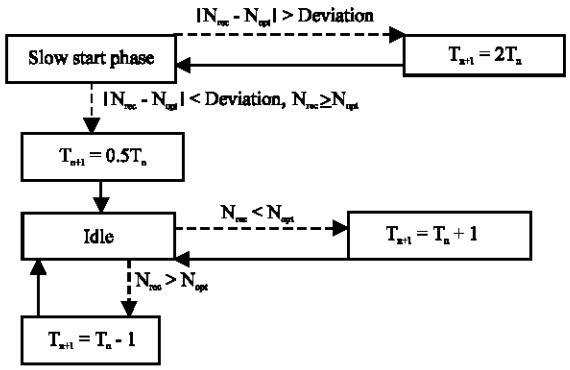

Fig. 5: Finite state machine (Version 2)

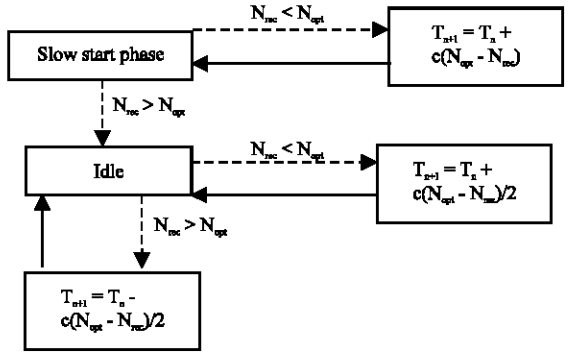

Fig. 6: Finite state machine (Version 3)

To make our timing control scheme better adaptive to the aggregation performance, we introduce the concept of deviation: if the difference between the practical number of received responses (N_{rec}) and the desired optimal number of responses (N_{opt}) is larger than a deviation, we need to change the aggregation period in a faster speed instead of the slow addition/subtraction operations as used in FSM Version 1. As shown in Fig. 5, The FSM Version 2 aims to reduce reaction time, but fluctuates less once a good operating point has been reached. Some parameters, such as the acceptable deviation and the multiplicand, can be selected beforehand based on the empirical query performance results in the specific WSN applications.

The FSM version 2 may not be able to capture the following practical data query situation: in the beginning of the timing control for each query task, we need to quickly increase (maybe at an exponential speed) the aggregation period since we want to largely shorten the waiting delay when there are not many responses from lower tree level sensors. Later on, we may increase the aggregation period at a slower and slower speed or may even decrease the aggregation period. In other words, the increasing/decreasing speed of aggregation period depends on the deviation between N_{rec} and N_{opt} In our FSM Version 3 (Fig. 6), the aggregation period starts out being increased exponentially, then is cut in half after reaching the optimal number of responses and increases or decreases linearly after this point. Note this FSM is similar to the TCP congestion window-size control scheme. In Fig. 6, c is a constant parameter chosen by the application in the sink that relates N_{opt} to T_{opt}.

Control protocol overhead: A data query/collection round may include multiple requests for similar or dissimilar data, but uses the same parameters for timing control for the duration of each aggregation round. At the end of a round, the performance experienced is evaluated and parameters are chosen for the next round.

As mentioned, the aggregators, or clusterheads are likely ordinary sensor nodes, tasked with additional responsibilities. For this reason, the Master (or clusterhead) must be rotated periodically to provide a reasonably even load on each sensor node (Heinzelman, 2000; Smaragdakis and Matta, 2004). This may be done after each round, or after a specific number of rounds. This leads to another tradeoff in energy efficiency. If the network is reconfigured too frequently, it will lead to unnecessary protocol overhead in the form of control messages and calculations. At the other extreme, if the network topology is chosen once and remains static, then a single node will bear responsibility for an unfair amount of time and will likely die before its cluster members. This also decreases the overall usefulness of the network, since when this clusterhead dies, it may take irretrievable data with it.

It is common practice to use a setup phase where we communicate all the necessary control packets for the creation of the network DAT (Heinzelman, 2000; Sichitiu, 2004). At this point we could conceivably inform the aggregating nodes of their depth in the tree. This is the only parameter necessary for a basic aggregation timing control scheme. Another option is to embed certain control information within each data packet transferred. This provides more timely feedback on the state of the system, but adds overhead in the form of additional data for each transmission.

The only additional information that may need to be passed back up the tree during the data collection phase is some optional information such as the number of messages missed by an aggregator due to a short aggregation period, or perhaps just whether or not any messages were received late from the last query. This requires only a few additional bits as part of the data message packets and the rest of the protocol does not depend on its use.

Aggregation Types Chosen by the WSN Sink: In most aggregation schemes, late reports are often discarded (Madden and Szewcyk, 2002; He *et al.*, 2004). If late data is useless to the application it makes sense not to transmit it (or not to forward it all the way up the tree). However, we feel that it is a decision that should be made by the applications. In some situations (such as real-time applications) there may be a hard deadline. In other cases (such as soil condition monitoring in agricultural applications), the application may be willing to accept a longer delay in exchange for a longer network lifetime. We therefore propose support for varying levels of aggregation priority as follows:

- No aggregation
- Some aggregation, send initial results, aggregate and any later results separately
- Some aggregation, send initial results and aggregate, but disregard late results
- Only send aggregated results

The sink will specify one of these flags when submitting a query to the network. These lead to another tradeoff in performance. By supporting this flag we incur a slight increase in the amount of control traffic, but provide much more flexibility to the application running in the sensor network.

Earlier we mentioned the possibility of using aggregation during the setup phase. This could allow the network designer to trade off energy savings for a more complete description of the network, if such is necessary at that point in the network setup. The sink first transmits a depth request to the network. When the nodes reply to this query, we have the following options:

- Full aggregation may be performed. The sink and the intermediate aggregators receive only the maximum depth of the nodes underneath them and each node will only pass a single message back upstream. How balanced or unbalanced the tree is will not be easily deduced by these aggregated reports. This piece of information may be irrelevant to the aggregation function anyway.
- Per-level aggregation may be performed. For example, if a node had three children that were each at level 0, it would report simply that it was at level 1, itself. This requires a single message with the minimum payload. If a node had one child node at level 1 and one child that was a leaf node, it would report both of these levels to its parent, doubling the payload of this node's response. This situation applies to node B in Fig. 1.

This would more completely describe the network, but require more data to be transferred than if using full aggregation. Communication costs would be higher for more unbalanced trees.

- No aggregation may be performed. This will completely describe the network topology to the sink, but require much more communication. This option would likely only be used as a backstop to the existing network topology creation, since such complete knowledge may not be necessary for data aggregation.

QUANTITATIVE ANALYSIS

Height-based differential analysis: In a DAT structure there could exist a time difference between different DAT levels when data is received by a parent node from its children nodes. For example, in Fig. 1, if nodes E, F and D all sense an event, node B will likely receive the report from node D before it receives the forwarded reports from node C. An Event Triggered Scheme (ETS) has been introduced in (Yuan *et al.*, 2003). In this model, each node has an internal timer that, when exhausted, will cause received data to be aggregated and forwarded to its parent node. Having a time-out is necessary to assure that aggregating nodes do not wait forever when messages from their children are dropped due to communication errors or sensor deaths (due to out-of-energy or physical damage). In their scheme, the timer is started when the node either senses an event or receives a report from one of its direct children. Our scheme specifies that a node's internal timer should be started when a query is received from a parent node. The main difference here is that we assume a mainly query-triggered system (i.e., data pull instead of data push). In both cases, only loose time synchronization is required as there is minimal transmission latency between one-hop neighbors.

It is necessary for nodes closer to the root of the tree to wait longer than those near the leaves. For example, consider a maximally unbalanced tree, i.e., a chain topology. If there were two leaf nodes as shown in Fig. 7

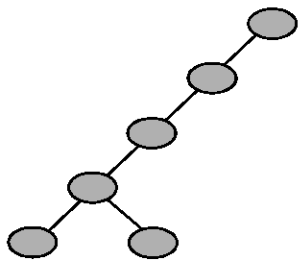

Fig. 7: Chain plus child near leaf

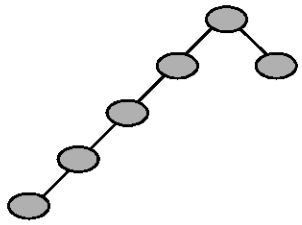

Fig. 8: Chain plus child near sink

and both of them detected an event, there would be no timing problem since the event was detected by leaves at the same tree level. If, however, the network were organized as in Fig. 8 and both nodes again detected an event, timing would be a major issue. In fact, the differential between the levels of the leaf nodes would dictate the timing skew. The parent node that is located above both of these nodes would need to account for this skew, Δ, as follows:

$$\Delta = \text{shd} \bullet (l_1 - l_2) \tag{2}$$

where shd is the Single Hop Delay, or the time, including aggregation and transmission time, that a message takes to be transmitted from a node to its parent, l_1 is the level of the higher node that aggregates data and l_2 is the level of the lower node whose data is to be aggregated. Δ is the additional time that the aggregating node must wait to aggregate data.

Assuming the network topology is communicated to aggregating nodes during the *setup* phase, the above formula indicates to the aggregator how long it may have to wait before delivering data to its parent. If a maximum tolerable latency accompanies each data query command, then the aggregator will be able to decide dynamically whether it should wait to aggregate all data, or forward whatever data it has after the appropriate timeout period. This becomes even more useful when combined with the different types of aggregation. In this case, the sink may allow or disallow additional aggregation data at the cost of query latency.

A Multi-Level Fusion Synchronization (MFS) Protocol is suggested in (Yuan *et al*., 2003). This accounts for the aforementioned aggregation timing lag by making a node broadcast a START message when it transmits data that needs to be aggregated. Nodes that overhear this message are to start their own timers, taking into account their depth in the tree. This assumes that all nodes will hear this START message, especially the aggregator, which will then be able to compute the expected arrival time for this message. Other nodes along the path will be able to calculate the amount of time allotted for them to send their relevant data to be aggregated without it being considered tardy.

On the other hand, there is, however, the chance that nodes along the chain may not hear this START command. In the worst case, the latency will reach the upper bound as follows:

$$L = \sum_{j=0}^{D-1} \text{Max} - j \cdot \Delta = D \cdot \text{Max} - \frac{(D-1) \cdot D}{2} \cdot \Delta \tag{3}$$

If individual aggregators are expected to calculate their aggregation periods, then there still must be some sort of learning phase: either D is given or it must be obtained empirically. We may require that each node should send out hop count requests. Alternatively, this may be performed when data queries are sent out. We choose to use this latter method, as it keeps nodes updated in a more timely manner, but only on demand (when a query is actually in flight instead of requiring an additional periodic control broadcast).

The necessity of a timer based on this START message was proposed in (Yuan *et al*., 2003) mainly because the authors were considering data-push. It is efficient to just start the timer when we receive a response, since a node's depth in the tree is already known at that time.

Communication overhead estimation: In a WSN, communication is the biggest drain on battery power (Akyildiz *et al*., 2002). Accordingly, an estimation of the energy-efficiency of an entire network hinges mainly on the amount of communication that it performs. This includes the overhead of: (1) WSN configuration information (determined by protocol overhead) and (2) the data actually transmitted. It can be helped by minor improvements such as eliminating a bit or two from unnecessary protocol fields. As an example of protocol overhead improvement, packet field length is mainly protocol-independent. A protocol can be optimized to transmit less possible number of bits that still allow effective communication. However, the amount of data actually transmitted is out of our control. If r bits of data need to be sent, disregarding aggregation, compression and similar optimizations, then r bits must be sent. Regardless of how much energy we would like to save, a network that does not communicate interesting data is useless. This leaves only the number and size of control packets to estimate. We aim to quantify the complexity of communicating necessary control information to an entire network.

Assume the depth of the DAT is D. The messages must be retransmitted by every parent node in the tree. Thus the number of transmissions needed to forward this request is the number of nodes within D-1 hops, which we

refer to as f (D-1), (f is a function describing the relationship between the sensor amount and tree levels). Leaf nodes do not need to retransmit the update request.

Most aggregation (and routing) schemes require some sort of setup phase (Intanagonwiwat, 2002; Solis and Obraczka, 2003; Chatterjea and Havinga, 2003). This period is used to inform the nodes some initial parameters. Our schemes require very little additional overhead during this setup phase. The only additional parameter that must be transmitted is the depth of each node in the aggregation tree. This may be discovered with the equivalent of a single data query and a single response from each of the nodes in the network. Based on our protocol (see Section A), the setup phase is necessary, at the very least, to distribute the depth of the tree, D. Thus the complexity of this communication depends mainly on the depth of the tree. Each node will be required to transmit only two messages: one to downstream and one to upstream (If we use full aggregation here, disregarding latency, each node need only return one message). Note that in a full tree where each parent has c children, a tree with only depth D can encompass n nodes according to the following formula:

$$n = c^{D+1} - 1 \tag{4}$$

This could easily be piggybacked on the network topology management packets, or could be performed as a separate transmission, basically as a specialized WSN query. Considering that the clusterheads are able to organized into a tree, such protocol overhead is minuscule compared to the rest of the setup (Intanagonwiwat, 2002). A simple binary tree could contain 50 clusterheads with a depth of only 5 (Solis and Obraczka, 2003). The overhead to create the tree in the first place would be on the order of $O(n^2)$ for a minimum spanning tree, or O(diameter) for a Breadth First Search Tree (Chatterjea and Havinga, 2003). Our setup overhead is only in the order of O(depth).

If c is the number of children that each parent node has, then there are c^D leaf nodes. Here we assume a full tree and it is actually an upper bound on the number of leaves. There will be n nodes in the entire tree, where n is defined as follows:

$$n = \sum_{d=0}^{D} c^d \tag{5}$$

Aggregation amount estimation: We now analyze a network organized into a tree topology in which each aggregating node has exactly c children. Our analysis here assumes that only leaf nodes generate data and that intermediate nodes simply forward their reports (if intermediate nodes also generate data, then aggregation can save even more energy). At each level of the tree, the number of messages that must be retransmitted by level l+1 will be reduced according to the following formula:

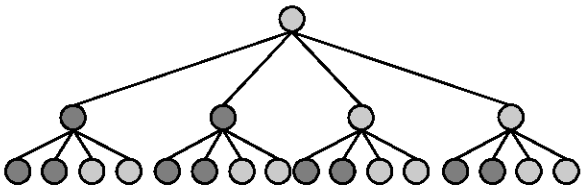

Fig. 9: Energy savings vs. Percentage of aggregation

$$m_{l+1} = m_l\left(c - \lceil pc \rceil + 1\right) \tag{6}$$

where: m_l is the number of messages transmitted by nodes at level l and
p is the probability that messages are aggregated.

Therefore, the total number of messages propagated through the network, M, is

$$M = \sum_{l=0}^{D-1} m_l\left(c - \lceil pc \rceil + 1\right) \tag{7}$$

Figure 9 shows a network where c is 4, p is 0.5 (50%) and n is 21. Dark nodes represent nodes whose data can be aggregated. Lighter nodes may not meet timing requirements for aggregation.

Suppose we do not use aggregation. If each of the leaf nodes sensed an event, there would be 32 messages transmitted. Each leaf node transmits a report. And each of these responses is retransmitted by the intermediate nodes. If messages from darker nodes are aggregated, there will only be 28 messages transmitted. Note that the leftmost two leaf nodes each report individually but their parent need only transmit 3 reports to the root node.

SIMULATION RESULTS

Simulation setup: We have used (OPNET, 2006) to verify the efficiency of our proposed timing control schemes. The reason of choosing OPNET as the sensor network simulation tool is due to its bug-free networking modules and many built-in wireless communication components such as IEEE 802.15-based MAC layer, AODV-based MANET routing protocol, flexible antenna parameter setup and so on.

We implement our data aggregation schemes above the WSN Routing layer since the aggregation function needs to interpret data queries, data responses and sensor information from the application layer. The

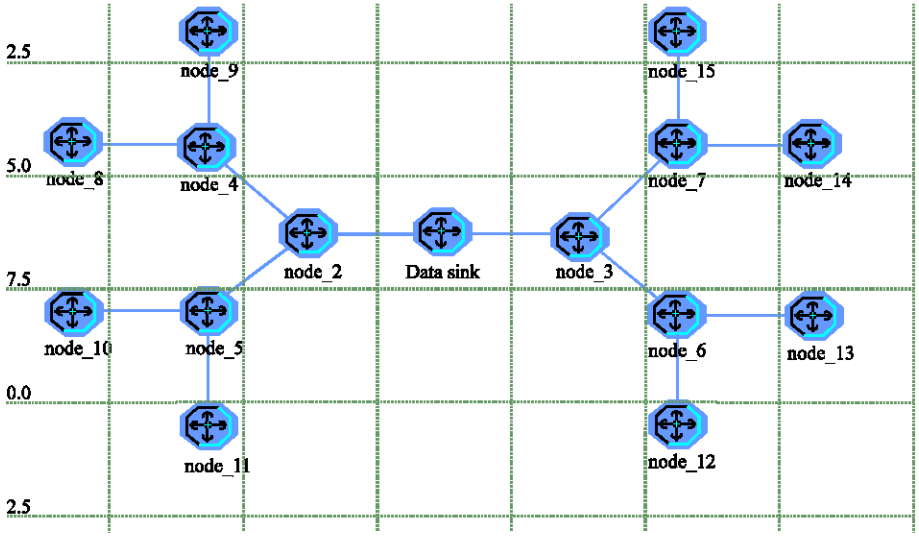

Fig. 10: Simulated topology

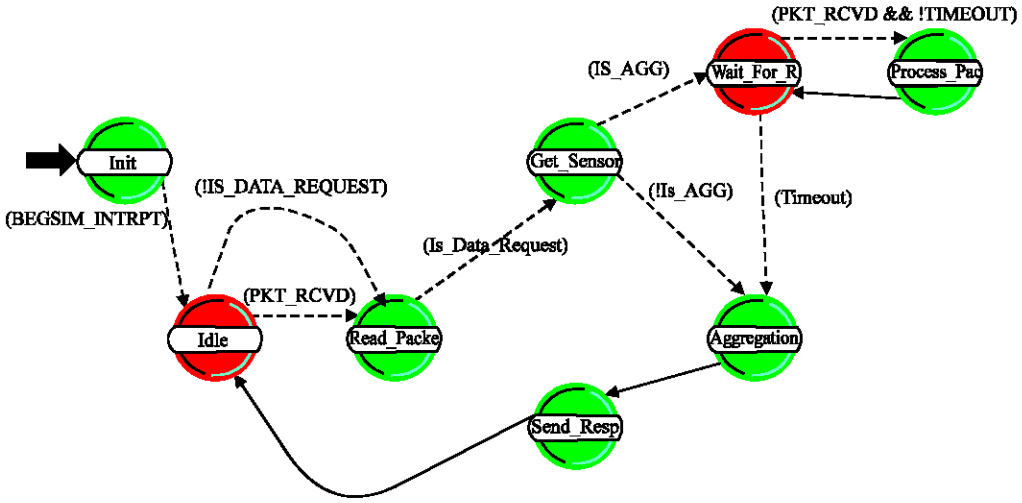

Fig. 11: Sensor node OPNET module

aggregation layer simply needs to know whether the packet received is a data response or data request. It is not necessary to analyze the detailed routing information as long as it can be determined whether the message is directed upstream or downstream in the DAT. If it is a data request, the aggregation function will get the sensor information from the application layer and wait for a certain amount of time to receive data responses. After the time has expired, it will forward the aggregated response to the routing layer, which will function as normal.

We have created a WSN with a tree structure that consisted of 14 sensor nodes and 1 sink that is the root of the tree. Figure 10 shows the spatial layout of the tree used for our simulations.

Figure 11 shows the OPNET module for the operation control of each sensor node. The sensor node starts out in an idle state. In the idle state, it waits for a packet to be received. When a packet is received, it moves to the Read Packet state where it determines if the packet is a data request. If the packet is a data request, it moves to the Get Sensor Data state. If it is not a data request, it simply discards the packet and returns to the idle state. If the timing synchronization using late packets were being used, the Read Packet state is the state where it would forward out any late packet information.

After the node retrieves its sensor information, which will most likely be passed down by the application layer, if the node is not to perform aggregation then it goes directly to the aggregation state. If the node is to

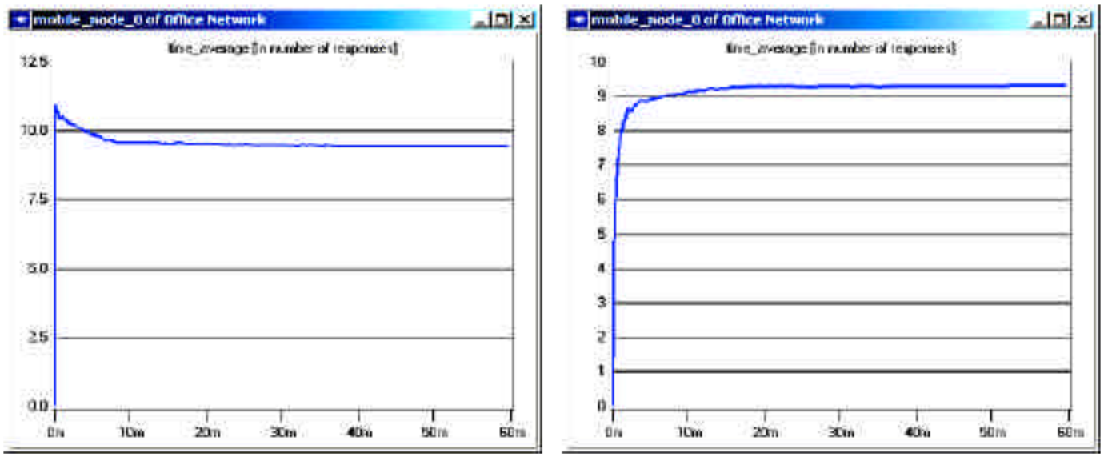

Fig. 12: (a) Number of responses reduced dynamically (b) number of responses increased dynamically

aggregate the data, it moves to the wait for responses state. In this state it waits for responses until the timeout period as specified in the data request packet. If a packet is received, the node transitions to the Process Packet state where it reads the packet, matches the ID and adds the information to a buffer if the packet is a data response to the current data request. After it processes the packet, it transitions back to the wait for responses state.

When the timeout occurs, the node moves to the Aggregation state where the aggregation function is performed on the data. It should also be noted that an optimization to the Wait for Responses state can be added if the number of children of the node is known. If the node receives responses from all of its children, it can immediately transition to the aggregation state. After the data is aggregated, the node transitions to the Send Response packet, where the packet is forwarded to the routing layer. After the node sends the response, it returns to the idle state.

The WSN sink is responsible for initiating data aggregation. The application issues a request via the sink for data from the sensor network along with the desired number of responses and the type of aggregation to be performed. From this information, the sink creates data request packets that encapsulate the query and aggregation parameters. The sink then forwards the data request packets to its child nodes and in turn the request is forwarded throughout the entire tree.

When generating data requests, the sink is assumed to know the optimal number of packets to receive, which is referred to as N_{opt}. Based on our timing control algorithm, the sink determines the appropriate timeout period for the current round of aggregation.

After transmitting an initial data query, the sink waits for responses from the sensor field. This number is then used to determine the adjusted timeout period for the next round of aggregation. This includes comparing the actual number of responses received from the network, N_{rec}, to the optimal number of responses that the application expected to receive, N_{opt}. The optimal number of responses is application dependent and is preset in the sink. If the number of responses received is greater than the desired number, the control algorithm will reduce the timeout value, allowing less time for responses to be collected. If the number of responses received is less than desired, the control algorithm will increase the timeout period, which will proportionally allow more time at each level of the tree.

To analyze the performance of the system, the sink issues data requests, which the nodes respond to. The data that the sink collects consists of the number of data requests sent, the number of responses received, the optimal number of responses, the number of packets received, the number of packets sent and the timeout period.

We use time averaged plots to show number of responses collected in the sink. Using a time average allowed us to account for the outliers caused by MAC-layer transmission collision problems. Figure 12 (a) shows the results of a simulation scenario in which the initial timeout value is large. More results are received than necessary ($N_{rec}>N_{opt}$), so the timeout value is shortened and the number of responses is reduced. Next, a simulation scenario is constructed in which the initial timeout value is too small to allow the optimal number of responses to be received. The results from this simulation (Fig. 12b) show that the number of responses increases until it reaches a steady state value near the optimal number of responses. The timeout value also increases until an optimal value is reached and then levels off.

```
if (n_opt > n_responses) {
    timeout_period++;
} else if (n_opt < n_responses) {
    timeout_period--;}
```

Fig. 13: Timing control algorithm for scenario (a)

```
if(slow_start) {
// Slow start phase
if(abs(n_responses-n_opt) > DEVIATION) {
    timeout_period += timeout_period;
} else {
    timeout_period = 0.5*timeout_period;
    slow_start = 0;
}
} else if(n_responses != n_opt) {
// Not in slow start phase
// increase or decrease proportionally
timeout_period += (n_opt-n_responses)/(abs(n_opt-
n_responses));
}
```

Fig. 14: Timing control algorithm for scenario (b)

Simulation scenarios: Based on our above OPNET simulation models, we will study and compare the following three different WSN data aggregation scenarios:

Scenario (a): In this scenario, we adopt the FSM Version 1 algorithm. If not enough responses are received by the sink the aggregation timeout period will be increased by a single time unit, The Pseudo-codes are shown as follows (Fig. 13):

Scenario (b): In this scenario, the timing control algorithm considers the variation between (1) the optimal number of responses the sink expected to receive and (2) the actual number received. We also introduce the idea of the acceptable deviation (Fig. 14). This can prevent the timing period from being changed dramatically unless there is a large discrepancy between these two parameters.

In Scenario (b), another notable improvement compared to Scenario (a) is the addition of a slow-start phase. From the initial query until the end of this slow-start phase the timeout period increases exponentially. When the difference between the optimal number of responses and the number received drops below the predefined acceptable deviation, the timeout period is reduced by a small amount and slow-start phase is ended. From that point on, the timeout period is increased or decreased in direct proportion to the difference between N_{opt} and N_{rec}.

```
if(slow_start) {
    // Slow start phase
    timeout_period += increase_factor*(n_opt-
    n_responses);
    if(n_responses > n_opt) {
        slow_start = 0;
    }
} else if(n_responses <= n_opt) {
    // Not in slow start phase
    // increase or decrease at a slower rate
    timeout_period += 0.5*increase_factor*(n_opt-
    n_responses);
} else if(n_responses > n_opt) {
    timeout_period += 2*increase_factor*(n_opt-
    n_responses);
}
```

Fig. 15: Timing control algorithm for scenario (c)

Scenario (C): In this scenario, the timing control algorithm uses an increase_factor as described in the following equation:

$$T_{n+1} = T_n + c(N_{opt} - N_{rec}) \quad (8)$$

This is intended to account for the mathematical relationship between the timeout period and the number of responses FSM Version 3. The idea of a slow-start phase used in Scenario (b) is maintained in Scenario (c), but the DEVIATION parameter is eliminated. It is expected to fluctuate slightly above or below the optimal timeout period but to be more responsive to a small difference from the optimal number of responses.

Once out of the slow-start phase the timeout period is increased or decreased based on the difference between n_opt and n_responses. However, without the DEVIATION parameter, it is necessary to increase and decrease at different rates. A suitable ratio is determined experimentally. Initially, this algorithm is designed to increase and decrease at the same rate, but it is found that, on average, random wireless transmission errors would cause the timeout period to gradually increase (Fig. 15).

Each of these three scenarios is run under two sets of simulation parameters. The data sink has 14 children and each is programmed to respond to the simulated data query with its sensed data. In the original simulations, the sink requires 10 responses as its optimal number. This allows a longer simulation and shows how each timing algorithm performed during an extended period. As a secondary simulation, the sink requests only 8 responses from all of its children. This represents a query where low

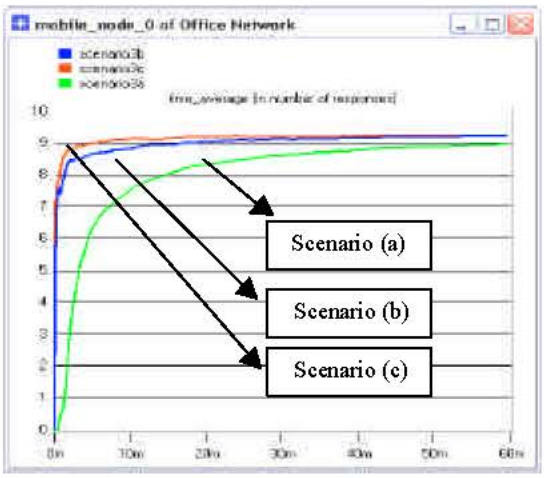

Fig. 16: Number of responses, time average, Nopt = 10

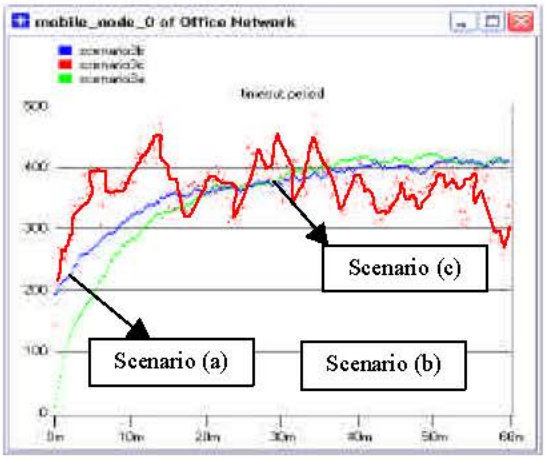

Fig. 17: Timeout period, Nopt = 10

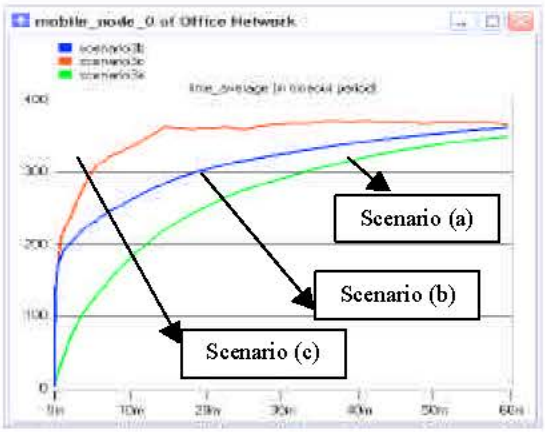

Fig. 18: Timeout period, time average, Nopt = 10

latency is slightly more important and sporadic wireless transmission errors have less effect on the steady-state performance of each algorithm.

Comparison results for the above three data aggregation scenarios: Figure 16 shows the number of responses received by the sink as the simulation progressed. The optimal number of responses for this simulation was 10. The lowest line shows the performance of Scenario (a), the middle line shows that of Scenario (b) and the uppermost line shows the performance of Scenario (c). For clarity, this plot shows the time average of the number of responses received. Due to the realistic performance of the wireless links, there were fluctuations throughout the simulation, which makes plots of the raw data more difficult to interpret but it can be seen that in all cases the number of responses eventually converges toward the goal of 10.

Because the average is taken, the initial reaction time of each algorithm visibly affects the apparent convergence time. As shown in Fig. 16 Scenario (a), the linear FSM reacts the slowest and accordingly takes the longest to converge. Scenario (b) improves the reaction time significantly. It can be seen that the number of responses increases from zero much sooner than for scenario (a). The average number of responses received approaches the desired number most quickly for scenario (c). Since the goal is to provide the desired number of responses to the sink dynamically as quickly as possible, the timing control algorithm used in scenario (c) appears to perform the best out of the three algorithms tested.

In Fig. 17, the trace that originates the lowest and increases slowly is the result of Scenario (a) algorithm. The solid line that appears to begin slightly higher results from Scenario (b) algorithm (Fig. 17). Both of these lines stabilize around 400, or 40 milliseconds. In Fig. 17, the most scattered plot is that of Scenario (c) algorithm. As shown in Fig. 17 Scenario (c), the cost of quickly increasing the number of responses toward the optimal number can be seen in the fact that the plot of the timeout period does not really stabilize as the other two scenarios do. This is expected as the difference between the optimal number of responses and the number received is amplified before affecting the timeout period.

Figure 18 shows that, although Scenario (c) appears not to stabilize, the average timeout period indeed increases and approaches its final value most quickly out of the three scenarios. As evidenced by Fig. 16, the average number of responses also increases most quickly.

In the next simulation, the number of desired responses is lowered to only 8 out of the 14 children. Performance is similar for all three algorithms, but it is

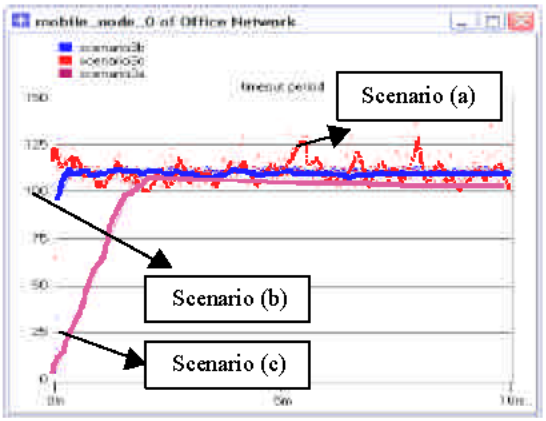

Fig. 19: Timeout period, Nopt = 8

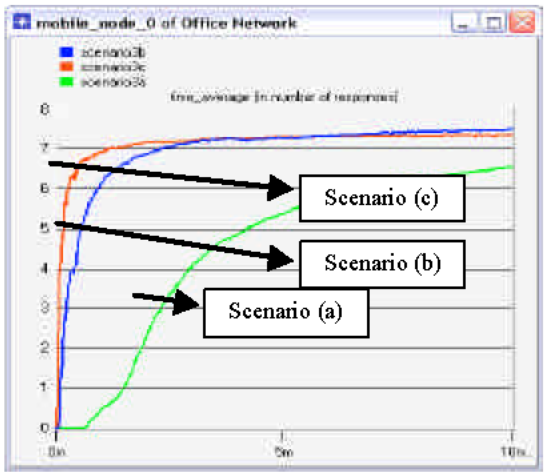

Fig. 20: Number of responses, time average, Nopt = 8

expected that more advantages would be obtained by a simulation setup that could reach a stable operating state more quickly. Under these conditions, the shortcomings of Scenario (a) become more apparent. Though the algorithm is successful in eventually providing the desired number of responses, Scenario (b) and Scenario (c) are much quicker to do so.

Figure 19 shows the how the timeout period is updated by each of the three algorithms. The lowest line shows the performance of Scenario (a) and is clearly linear during its initial increase. The other marked line shows the performance of Scenario (b), which, as expected, reacts more quickly than Scenario (a). The most scattered plot shows the timeout period as controlled by Scenario (c). This line fluctuates less in the initial phase than Scenario (c) line in Fig. 18 because transmission errors have less impact when the sink only needs to receive responses from 8 of its children and this number of messages can be more reliably delivered by the wireless links. This point is illustrated by considering a situation in which responses were required from all 14 children. Even a single transmission error would cause the timeout period to be increased. However, if only 2 responses were required, an average of up to 12 responses could be lost even under stable operating conditions and still satisfy the sink's requirements.

Figure 20 shows the average number of responses received in this simulation case (i.e., Nopt = 8). Similar to the observed timeout periods and results discussed previously (Fig. 19), both Scenario (b) and Scenario (c) provide the desired number of responses with about the same alacrity, with Scenario (c) being slightly quicker. Results from Scenario (a) lag behind these two and its trace is the lowest plot of the three. Although the number of responses in Scenario (a) does not seem to reach an acceptable level within the time range shown in Fig. 20, the number of responses reaches 8 in about 20 simulation time units (but the initial poor results lower the average).

DISCUSSION

- From the above simulation results, it can be seen that the design and implementation of a system involving data aggregation and timing control could be successful. All three algorithms accomplish the goal of dynamically modifying the number of responses received. Of the three, scenario (c) provides the desired number of responses in the timeliest manner under both simulation setups.
- Specifically, it seems that using a DEVIATION parameter will greatly help to stabilize the timeout period by preventing its change unless there is a notable difference between the optimal and the actual number of responses. This is most useful when either sporadic (instead of bursty) wireless transmission errors are expected or the optimal number of responses is very close to the maximum number of children nodes. In both of these cases, some fluctuation in the number of messages received exists even after the optimal timeout period has been reached.
- Increasing or decreasing the timeout period based on the number of responses is effective in its goal of more quickly correcting the number of responses. The two algorithms in which the idea was used, i.e., Scenario (b) and scenario (c), performed better in this task than does scenario (a).

- Finally, using a multiplying increase_factor also helps scenario (c) react more quickly to receiving a sub-optimal number of responses. Since scenario (c) lacks the DEVIATION parameter, it cannot be concluded that the better performance of this algorithm is due entirely to using an increase_factor. However, by examining Fig. 17 and 19, it can be seen that scenario (b) and scenario (c) vary in their initial reaction times, which helps to significantly raise scenario (c) average timeout period trace. The first few timeout periods are very short and only a small number of responses are received, which causes the difference between the optimal and actual number of responses to exceed the acceptable DEVIATION and thus reduces its importance.
- These above simulation results indicate the importance of the proposed mechanisms for creating an efficient means of dynamically modifying the number of responses received by the WSN sink. A system that specifies an acceptable DEVIATION and uses the variation between the optimal and actual number of responses to update the timeout period, is better than the linear FSM and far better than a static guess of an aggregation timeout period. Using this difference has been shown to be able to intelligently increase or decrease the number of responses received and a multiplying increase_factor shortens the reaction time of the algorithm. When wireless transmission errors occur frequently, the DEVIATION parameter can help to reduce unnecessary changes in the timeout period. A system that incorporates all of these ideas can intelligently and effectively provide the number of responses requested by the sink and the most aggregation and energy savings possible within the amount of time specified.

SENSOR HARDWARE EXPERIMENTS

We have used Crossbow sensor motes (Crossbow, 2006) to build our WSN hardware platform and have carried out some experiments to verify the efficiency of our FSM-based timing control scheme in WSN data aggregation. A WSN node includes two parts (Fig. 21): (1) microprocessor plus radio board (for sensor local processing, TinyOS (Crossbow, 2006) and wireless transmissions). It is also called mote; and (2) Sensor board (for detecting light, temperature, humidity, sound and other types of data). Typically, these sensor motes have extremely low power (a few tens of milliwatts versus tens of watts for a typical laptop computer). When operating at 2% duty cycle (between active and sleep

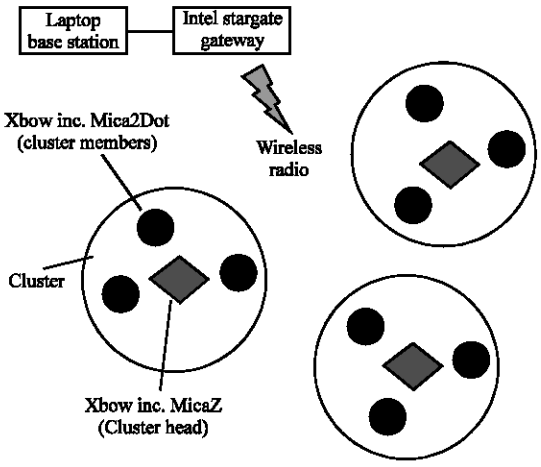

Fig. 21: Hardware setup. Cluster-based topology setup; Note: Each node includes 'sensor board' (to detect data) and 'Mote' (to process data and transmit/receive packets)

modes), we can achieve a lifetime of about 6 months (on a pair of AA batteries). We use MicaZ motes (Crossbow, 2006) that have IEEE 802.15.4 Physical/MAC layers modules for Zigbee-compliance (Zigbee, 2006) purpose. A Crossbow mote has a 4 MHz, 8 bit Atmel microprocessor.

In terms of data query implementation, we use a routing tree that allows a WSN sink (we use a laptop) to disseminate a query and collect query results. The routing tree is formed as follows: The root (i.e., the sink) sends a request. All child nodes that hear this request process it and forward it on to their children and so on, until the entire network has heard the request. Each request contains a hop-count, or level indicating the distance from the broadcaster to the root. To determine their own level, nodes pick a parent node that is (by definition) one level closer to the root than they are. This parent will be responsible for forwarding the node's (and its children's) query results to the sink.

Figure 21 (Left diagram) shows our hardware experimental topology. It includes a laptop (serving as a WSN sink) and a few clusters of nodes. Each cluster has a cluster-head node that is a Crossbow Mica2 sensor and three cluster member sensors (we use Crossbow Mica2Dot). We have built a DAT with 8 clusters. Each DAT level has 1~3 cluster-heads.

To collect the query responses from all sensors, we adopt TinyDB (TinyDB 2006) as the query interface that has a friendly, windows-based user interface in the WSN sink (Fig. 22). For instance, it could use a SQL-like query syntax to collect sound, light and temperature data

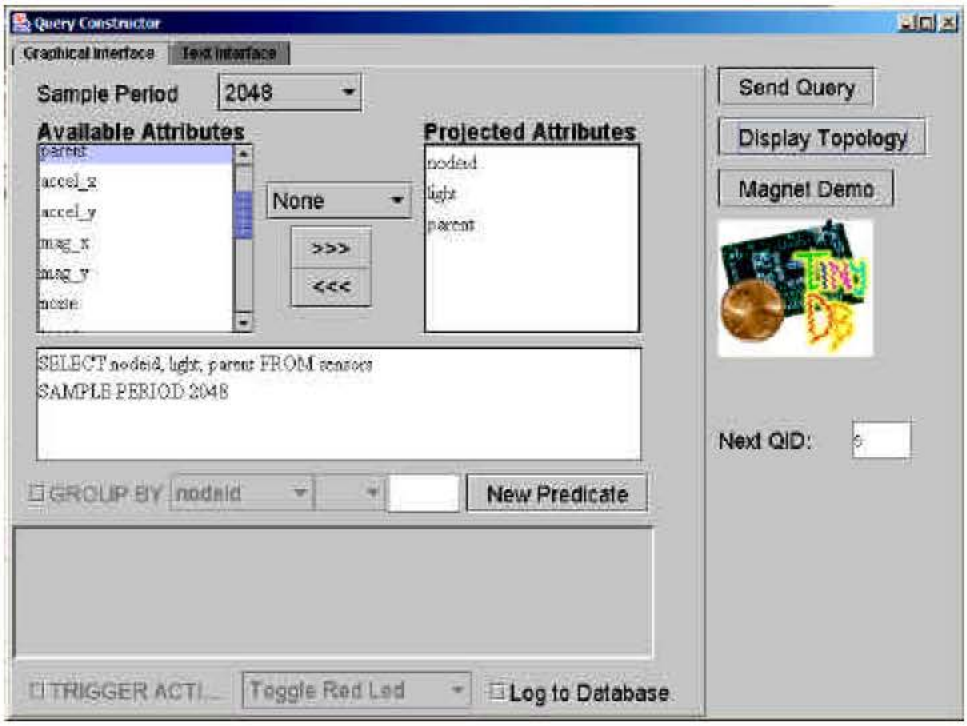

Fig. 22: Sensor data query applications based on Tiny (2006)

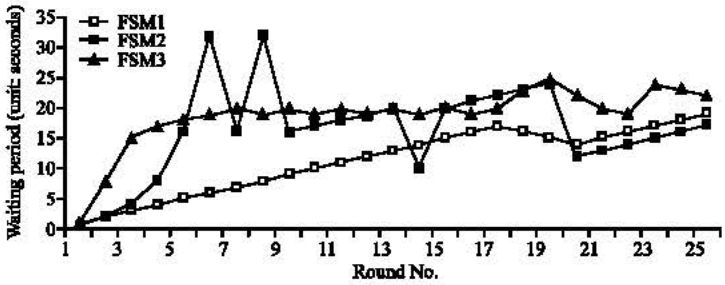

Fig. 23: The adjustment of waiting period in each aggregation round

(Fig. 22). However, we cannot directly use the middleware layer software of TinyDB since it does not any timing control scheme in each tree level. Due to its convenient Java-based software development modules, we have added our FSM-based timing control protocol above its routing layer.

We first investigated the waiting period T_n (n is the data aggregation round No.) adjustment algorithms based on our proposed three different FSM schemes. We can see that the FSM Version 1 (i.e., FSM1 in Fig. 23) takes a long time to reach a stable status in which the system collects near optimal number of query response results. This is because it simply increases/decreases waiting period by one time unit in each aggregation around. Thus its adjustment speed is the slowest among three FSMs. If we use FSM Version 2 (i.e., FSM2 in Fig. 23), the system can quickly increase the waiting period in the beginning of the query operation. Note in the initial phase, the number of collected query responses, N_{rec}, is far less than desired one, N_{opt}, which is assumed to be 18 in our experiments (FSM2 in Fig. 24). However, FSM Version 2 cannot adjust its waiting period at an evenly speed after N_{rec} is close to N_{opt} due to its simple multiplication/division by 2. From both Fig. 23 and 24, we can see that FSM Version 2 causes large fluctuations for both waiting period (Fig. 23) and N_{rec} (Fig. 24).

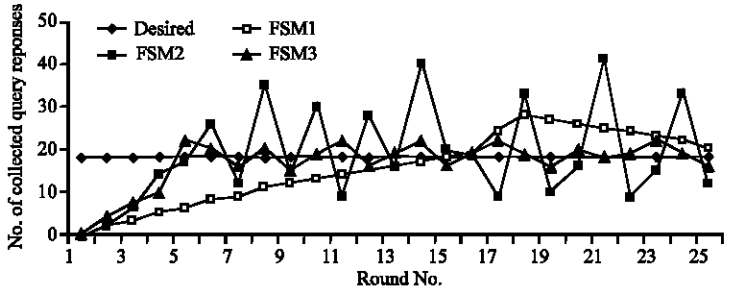

Fig. 24: The No. of collected query responses under three FSMs

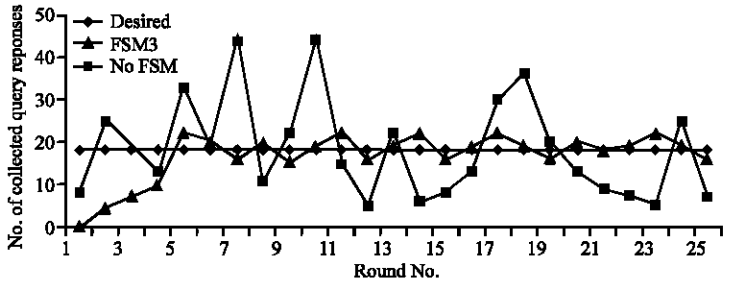

Fig. 25: The No. of collected query responses (1) with our FSM and (2) No FSM

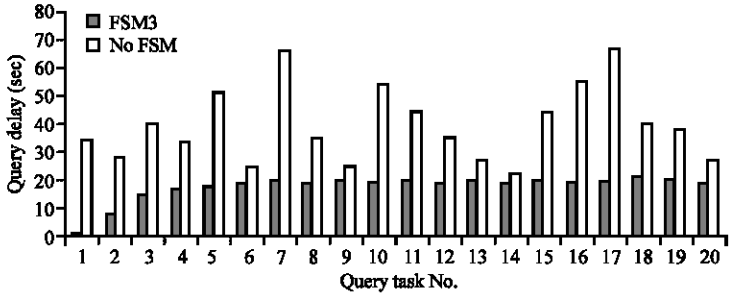

Fig. 26: The query delay: (1) under our FSM and (2) Not using our FSM

FSM Version 3 has good timing control performance because it adopts an auto-feedback control scheme that is similar to Internet TCP congestion control algorithm. In the initial adjustment phase, it behaviors like the TCP slow-start phase, i.e., quickly going close to the N_{opt}. After N_{rec} is close to N_{opt}, it will slowly adjust the waiting period so that the value of N_{rec} does not fluctuate much (FSM3 in both Fig. 23 and 24).

Next, we investigate the efficiency of our timing control scheme in terms of achieving good trade-off between end-to-end query delay and query accuracy. Suppose we do Not adopt the above proposed FSM-based timing control scheme, all DAT levels just simply wait for a constant duration, say, 20 sec, we then study the value of collected number of query responses from sensors. To reflect the actual harsh WSN communication environments, we intentionally put some metal blockages between sensors. Thus the transmission error rate will be higher than the line-of-sight wireless communication case.

As shown in Fig. 25 (no-FSM case), the constant waiting period causes very unpredictable N_{rec} due to the high packet error rate in wireless communication environments and the quickly changing wireless link quality (that can be measured by Signal-to-noise ratio). However, our FSM-based timing control scheme can adaptively adjust the waiting periods based on last value of N_{rec} and does not cause a large fluctuation of the trace (Fig. 25, FSM3 curve).

Next, we change the WSN middleware layer to investigate another no-timing-control case: For each issued SQL query command (Tiny, 2006), we just simply wait until the sink (i.e., the laptop connecting to the WSN) collects enough number of responses, say, N_{opt} and then record the end-to-end query delay (from the DAT leaves sensors to the sink). We have conducted dozens of query task tests and Fig. 26 shows the delay results for 20 query tasks we randomly choose from our tests).

While our FSM-based timing control scheme can achieve a controllable query delay, a query task without timing control scheme can have a highly variable delay that is typically larger than the delay in our scheme.

CONCLUSIONS

Data aggregation should be performed in WSNs when possible because Redundant data transmissions brings high communication overhead that consumes a large amount of energy in WSNs. Current data aggregation schemes either require too much overhead or cause unnecessary waiting delay in some aggregation nodes. In this study, we have analyzed the issue of timing control when aggregating data in Wireless Sensor Networks. We have shown, through mathematical analysis, simulations and experiments, that the aggregation waiting period, which is the most important parameter for deciding the tradeoff between query accuracy and response delay, can be updated adaptively in an attempt to provide an appropriate number of data query responses.

Our future work is to include other WSN factors (such as network density, sensor detection frequency, mobility modes, etc.) in our timing control design so that a better trade-off between query accuracy and end-to-end query delay can be achieved.

REFERENCES

Abdelzaher, T., T. He and J. Stankovic, 2004. Feedback control of data aggregation in sensor networks. Conference on Decision and Control, pp: 311-315.

Akyildiz, I.F., W. Su, Y. Sankarasubramaniam and E. Cayirci, 2002. A survey on sensor networks. IEEE Commun. Mag, 40: 11-20.

Asada, G., 1998. Wireless integrated network sensors: Low power systems on a chip. Proceeding of European Solid State Circuits Conference. The Hague, Netherlands, pp: 197-203.

Bonnet, P., J.E. Gehrke and P. Seshadri, 2000. Querying the physical world. IEEE Personal Communications, 7: 21-30.

Bonnet, P., J. Gehrke and P. Seshadri, 2001. Towards sensor database systems. Mobile Data Manage., pp: 314-320.

Bonfils, B.J. and P. Bonnet, 2003. Adaptive and decentralized operator placement for in-network query processing. Proceeding of International Workshop on Information Processing in Sensor Networks (IPSN), pp: 105-110.

Bult, K., 1996. Low power systems for wireless microsensors. International Symposium on Low Power Electronics and Design, Monterey, California, pp: 123-129.

Chandrakasan, A., 1999. Design considerations for distributed microsensor systems. Proc. IEEE Custom Integrated Circuits Conference (CICC '99), pp: 1020-1023.

Chatterjea, S. and P. Havinga, 2003. CLUDDA-Clustered diffusion with dynamic data aggregation. 8th CaberNet Radicals Workshop. Ajaccio, Corsica, France, pp: 1201-1210.

Chu, M., H. Haussecker and F. Zhao, 2002. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. Intl. J. High Performance Computing Appli., 41: 301-312.

Crossbow, M., 2006: http://www.xbow.com

DARPA Sensor Information Technology (SensIT) Program, Program Manager: Sri Kumar, DARPA/IPTO, 2000-2002. Program description available online at http://www.darpa.mil/ipto /solicitations/CBD 00-25.html.

David, B. and E. Deborah, 2002. Rumor routing algorithm for sensor networks. First Workshop on Sensor Networks and Applications (WSNA), pp: 156-166.

Elson, J., L. Girod and D. Estrin, 2002. Fine-Grained network time synchronization using reference broadcasts. Proc. Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA, pp: 67-75.

Fei, H., Yu Wang and Hongyi Wu, 2006. Mobile telemedicine sensor networks with low-energy data query and network lifetime considerations. IEEE Trans. Mobile Compu., 5: 101-115.

Goel, S. and T. Imielinski, 2001. Prediction-based monitoring in sensor networks: Taking lessons from MPEG. ACM Comp. Commu. Rev., 31: 410-411.

Govindan, R., J. Hellerstein, W. Hong, S. Madden, M. Franklin and S. Shenker, 2002. The sensor network as a database. Technical Report 02-771, Computer Science Department. University of Southern California.

Greenstein, B., E. Deborah, G. Ramesh, R. Sylvia and S. Scott, 2003. DIFS: A distributed index for features in sensor networks. Proc. First IEEE International Workshop on Sensor Network Protocols and Applications (SNPA), Anchorage, Alaska, pp: 201-206.

He, T., B.M. Blum, J.A. Stankovic and T. Abdelzaher, 2004. AIDA: Adaptive application-independent data aggregation in wireless sensor networks. ACM Transactions on Embedded Computing Systems (TECS), 3: 298-305.

Heinzelman, W.R., 2000, Application-specific protocol architectures for wireless networks. M.Sc Thesis, Massachusetts Institute of Technology.

Hightower, J. and G. Borriello, 2001. Location systems for ubiquitous computing. IEEE Computer, 34: 200-210.

Intanagonwiwat, C., 2002. Directed diffusion: An application-specific and data-centric communication paradigm for wireless sensor networks. Ph.D Thesis, University of Southern California.

Kleinschmidt, P. and R. Mock, 1989. Sensor systems in industrial applications. In Proc. CompEuro'89, Hamburg, West Germany, pp: 210-215.

Koniger, M.E., 1989, Sensors for space application. In Proc. CompEuro '89, Hamburg, West Germany, pp: 100-106.

Madden, S., M.J. Franklin, J. Hellerstein and W. Hong, 2002. TAG: A tiny aggregation service for ad-hoc sensor networks. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02), pp: 310-315.

Madden, S., R. Szewcyk, M.J. Franklin and D. Culler, 2002. Supporting aggregate queries over ad-hoc wireless sensor networks. WMCSA, pp: 1501-1505.

Meguerdichian, S., F. Koushanfar, M. Potkonjak and M. Srivastava, 2001. Coverage problems in wireless ad-hoc sensor networks. Proc. IEEE Infocom, 3: 301-312.

Meguerdichian, S., F. Koushanfar, G. Qu and M. Potkonjak, 2001. Exposure in wireless ad hoc sensor networks. Proc. 7th Annual International Conference on Mobile Computing and Networking (MobiCom '01), pp: 1001-1005.

OPNET Networking Simulation Tool, 2006: http://www.opnet.com

Ratnasamy, S., 2002. GHT: A geographic hash table for data-centric storage. Proc. 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '02), pp: 890-898.

Romer, K., 2001. Time synchronization in ad hoc networks. Proc. 2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), pp: 561-567.

Sadagopan, N., B. Krishnamachari and A. Helmy, 2003. The ACQUIRE mechanism for efficient querying in sensor networks. 1st IEEE International Workshop on Sensor Network Protocols and Applications (SNPA'03), pp:12-17.

Sichitiu, M.L., 2004. Cross-layer scheduling for power efficiency in wireless sensor networks. IEEE INFOCOM, pp: 610-622.

Smaragdakis, G., I. Matta and A. Bestavros, 2004. SEP: A stable election protocol for clustered heterogeneous wireless sensor networks. SANPA 04. Second International Workshop on Sensor and Actor Network Protocols and Applications, pp: 121-129.

Solis, I. and K. Obraczka, 2003. In-Network aggregation trade-offs for data collection in wireless sensor networks. INRG Technical Report 102, Available: http://inrg.cse.ucsc.edu/techreports/tr102.pdf

Tiny, D.B., 2006: http://telegraph.cs.berkeley.edu/tinydb/

Warrior, J., 1997. Smart Sensor Networks of the Future. Sensors Magazine, pp: 201-210.

Woo, A. and D. Culler, 2001. A transmission control scheme for media access in sensor networks. Proc. 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking (ACM MobiCom), Rome, Italy, pp: 201-207.

Yao, Y. and J. Gehrke, 2002. The cougar approach to in-network query processing in sensor networks. SIGMOD, pp: 310-326.

Yuan, W., S.V. Krishnamurthy and S.K. Tripathi, 2003. Synchronization of multiple levels of data fusion in wireless sensor networks. In IEEE GLOBECOM, pp: 1121-1127.

Zigbee, 2006: http://www.zigbee.org.