Efficient Data Aggregation and Transport in Wireless Sensor Networks *

Mario O. Díaz and Kin K. Leung
Electrical & Electronic Engineering Department,
Imperial College, London, UK
ornediaz@gmail.com, kin.leung@imperial.ac.uk

2009
This is an early version. The final version is published in:
Wireless Communications and Mobile Computing
Wiley InterScience (www.interscience.wiley.com)
DOI: 10.1002/wcm.806
http://onlinelibrary.wiley.com/doi/10.1002/wcm.806/abstract

We consider the problem of reporting events using wireless sensor networks. To reduce the data volume generated by each event, the correlated data from the (neighboring) nodes that detect the event must be brought together to be processed and compressed before relaying the result across several hops to the data sink. This process is supported by an aggregation tree, which specifies the flow of information towards the sink. For efficiency, aggregation trees should compress the data close to their sources. We propose two solutions to the event-triggered reporting problem. Firstly, we propose the first protocol to use a staggered schedule in the construction of the aggregation tree. Due to the use of such a schedule, our protocol divides the tree-construction time by roughly the number of hops in the network, and this advantage comes only at the expense of a small degradation of the quality of the obtained aggregation tree. Secondly, we consider a multi-hop cluster-based topology with fixed aggregation points. This topology is appropriate for large networks with unreliable radio links. We approximate the optimal cluster size distribution and evaluate the improvement over a uniform cluster size distribution.

Keywords: wireless sensor networks, event reporting, data aggregation

1 Introduction

Duty-cycling and in-network data aggregation are two very efficient techniques to save power in wireless sensor networks (WSNs). Duty-cycling is the technique of reducing idle listening by turning off the radio modules periodically. Unfortunately, duty-cycling increases the latency experienced by data packets. In-network data aggregation [1] is the process of compressing locally the correlated data of neighboring sensors in order to reduce the amount of data that travels across several hops to the data sink.

A data-aggregation tree can be a very efficient routing structure for data aggregation. In this tree, every node waits for the data from its children nodes, compresses all the received data with its own, and forwards the result to its parent. Most algorithms to construct an aggregation tree involve high overheads because they were designed to report long-lasting data flows that make the setup overhead negligible. In this paper, we strive to minimize the tree construction overhead so that applications involving shorter events can also benefit from data aggregation. Such reporting of short events is motivated by our WINES project [2], on which we investigate the use of WSNs in civil-engineering applications, such as acoustic detection of fractures in bridges. For example, wireless sensors are used to capture the acoustic signatures of cable fractures in suspension-cable bridges. Cable fractures release noise signals for a brief period of time. Prompt reporting of cable fractures is necessary to stop the vehicular traffic over the monitored bridge soon after a severe fracture is detected to have occurred.

Our main contributions in this work are two alternative solutions for event-triggered reporting. The first one is the Fast Aggregation Tree (FAT) protocol, which is a distributed protocol to construct a data aggregation tree quickly in a duty-cycled environment. The speed advantage of FAT over existing protocols grows with the network size. Our second solution for the event-triggered reporting is a fixed multi-hop cluster-based structure where compression of any piece of data can only occur once. This model suits a number of event reporting problems in civil engineering and requires less overhead to adapt to link failure in large networks. We formulate the problem of the optimal cluster size distribution and propose a heuristic approach to solving the problem. We also identify the key variables that affect the improvement over uniform cluster size.

The rest of this paper is organized as follows. In Section 2, we propose the FAT protocol and evaluate the quality of the aggregation tree obtained by the protocol. In Section 3, we evaluate FAT in presence of intermittent links and argue the usefulness of clustering in such conditions. In Section 4, we formulate the optimal cluster-size problem and propose a heuristic to approximate the solution. Finally, Section 5 concludes the paper.

2 Constructing a tree using a staggered schedule

2.1 Power and delay in event-triggered reporting

Consider a multi-hop WSN that monitors an area in which a specific set of events occur on average once every T_e time units. A set of sensor nodes, referred to as *sources*, detect each such event and generate a reasonable volume of data about that event. A special node, referred to as the *sink*, needs the information about every event within a pre-specified maximum tolerable delay, D_l, after the event occurs. The data from the different sources associated with the same event are highly cor-

*This is the pre-peer reviewed version of an article published in Wiley Wireless Communications and Mobile Computing. This work is funded by UK EPSRC Research Grant EP/D076838/1, entitled: "Smart Infrastructure: Wireless Sensor Network System for Condition Assessment and Monitoring of Infrastructure".

related, so these sources should aggregate their data, if possible, before forwarding them across several hops towards the sink. A number of protocols have been proposed for data aggregation [3, 4, 5]. We call a data aggregation protocol *unstructured* if it aggregates data opportunistically, and we call it *structured* if it establishes a routing structure for data aggregation and then uses that structure to transmit and compress the data.

The sensor nodes are constantly sensing the environment, but their radio receivers are duty-cycled to save energy. With period T_c, every node turns on its receiver to check for data packets from its neighbors. This channel-check operation consumes an amount of energy E_c. In addition, the sensor nodes need to verify periodically the wireless links with their neighbors and to share updated topological information with them. This exchange of packets is a maintenance operation that consumes E_m amount of energy, and we assume that it is repeated with period T_m.

When an event occurs, the associated data sources notify their neighbors, thereby starting the construction of a data-aggregation structure, usually a tree, to report the event. The tree construction consumes E_b amount of energy and requires D_b amount of time, excluding the time until the neighbors of the sources check the channel, which can be as high as T_c. The tree construction time, D_b, depends on the duty-cycle period, T_c. After the aggregation structure has been constructed, the data from the sources travels to the sink by following the path indicated by that structure. The data from a source traverses multiple intermediate nodes along that path, and in some of those nodes it is compressed with the data from other sources. Assume that this traversal process consumes E_t amount of energy and requires D_t amount of time.

By considering all factors, the average total power consumption is

$$P_t = \frac{E_c}{T_c} + \frac{E_m}{T_m} + \frac{E_b + E_t}{T_e}. \quad (1)$$

If the wireless links among nodes are stable, frequent maintenance operations are not needed and thus T_m can be set to a large value. In addition, if events rarely occur, T_e is large. Under these two assumptions, (1) suggests that E_c/T_c dominates the power consumption, which can be reduced by increasing the channel-check period, T_c. Note that T_c cannot exceed $D_l - D_b - D_t$ in order to meet the maximum tolerable delay D_l. Therefore, to save power we have to reduce D_b or D_t. However, there is a tradeoff between the two components as follows. Generally, a fast tree construction algorithm (an algorithm with small D_b) generates an inefficient tree. We say that a tree is inefficient if it aggregates the data from the sources far away from them (i.e., close to the sink), because such a tree results in a large D_t. The best tradeoff between D_b and D_t depends on the data volume to be transmitted. For the target applications of the WINES project, we seek a solution for data volumes that are large, but not as large as to make E_b and D_b negligible.

2.2 Related work

DMAC [6] is a MAC protocol for duty-cycled environments that organizes the nodes in tiers. A node's tier is the length of its shortest path to the sink, as shown in Figure 1 shows. When considering Tier i, we refer to Tier $i+1$ as the *previous tier* and to Tier $i-1$ as the *next tier*. All the nodes are time synchronized, and each node has a schedule indicating when to listen and when to transmit. Figure 2 illustrates the way in which the schedules of nodes in different tiers are staggered. This arrangement greatly reduces the latency towards the sink. However, DMAC's timing policy does not enable data aggregation directly.

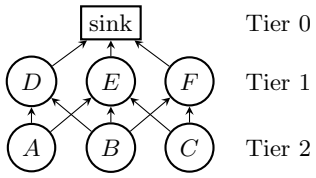

Figure 1: Classification of the nodes in a WSN in tiers, based on the smallest number of hops in which they can reach the sink.

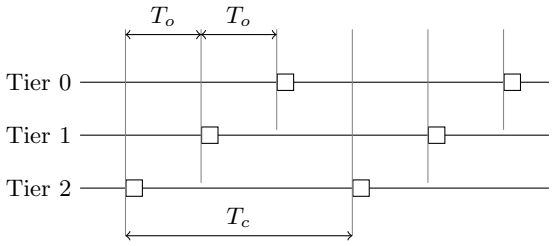

Figure 2: A receive schedule for a tier-based topology that provides low latency towards towards the sink. The squares represent the times in which the nodes turn on their radio receivers. If a node in Tier k generates a packet at a random time, the delay until that packet reaches the sink is at most approximately $T_c + kT_o$, provided that there are no collisions.

DB-MAC [3] and DAA+RW [4] are two unstructured protocols, and as such their D_b is zero. In DB-MAC, each node forwards its data to its neighbor that, according to the information that it gathered by overhearing its neighbors' transmissions, holds the greatest numbers of packets. In DAA+RW, several nodes contend to relay a packet, and nodes with more packets enjoy better chances of winning the contention. In both DB-MAC and DAA+RW, the nodes hold their packets for random periods to increase the probability of data aggregation. This technique is insufficient to guarantee aggregation and introduces delays and overhearing in each packet transmission. Therefore, for sources generating over a dozen packets, DB-MAC and DAA+RW result in high D_t and E_t. Another protocol, ToD [7], combines DAA+RW with a fixed aggregation structure. ToD outperforms DAA+RW in big networks as it guarantees aggregation after a number of hops, but it is not efficient for big data volumes in those few hops because it aggregates data opportunistically.

The optimal data aggregation structure is the Steiner Tree in graphs, which is NP-hard. Oceanus [5] is one of several centralized heuristics that compute at the sink good approximations of the optimal tree, and thus the tree it constructs yields a small D_t. In these centralized heuristics, before the sink computes this tree, the sink needs to receive the identity of all the sources, and after the sink has computed the tree, the information about the tree has to travel to all the nodes involved. Therefore, there are two data flows in opposite directions. The authors of Oceanus do not suggest how to transmit these two data flows and no MAC protocol has been proposed specifically for this purpose. Using an existing MAC, transmitting these two data flows would incur in a high D_b. For example, with a globally synchronous wakeup such as T-MAC [8] the transmission of each of these two flows would require an amount of time KT_c, which is likely to be unacceptably large because $T_c \gg T_0$, as shown in Figure 2. We believe the best existing MAC for this application would be DMAC, on top of which the transmission of the first and the second data flows would last for KT_o

and $K(T_c - T_o)$ time units, respectively. In some application, this value of D_b is still too high, and it would be interesting to reduce D_b further without sacrificing D_t. Such a reduction in D_b would allow to increase T_c, thereby saving power, and still satisfy the maximum tolerable delay D_l.

2.3 The Fast Aggregation Tree (FAT)

We propose the FAT protocol to construct an aggregation tree after each event. FAT is a distributed structured aggregation protocol, and its goal is to provide D_t comparable to that of centralized heuristics but with a much shorter tree construction time, D_b.

FAT assumes that all the nodes are globally time synchronized, which can be achieved by either hardware or software [9]. FAT uses DMAC's tiered architecture, which is depicted in Figure 1. We will use the topology in that figure to explain the FAT protocol with examples. When considering a node, we refer to its neighbors in the next tier as its *potential parents*, because only they may become that node's parents in the tree that FAT constructs. Similarly, we refer to a node's neighbors in the previous tier as its *potential children*. In the topology of our example, D and E are A's potential parents, and D, E and F are B's potential parents. In the initialization of our protocol, each node executes the distributed Bellman-Ford algorithm to determine its tier and its potential parents.

2.3.1 Normal operation

Under the assumption of rare events, most of the time there is nothing to transmit, and this is the *normal operation* of the FAT protocol. However, to keep latency low, every node senses the radio channel periodically with period T_c to check whether it may need to participate in a tree construction process. If during this channel-check operation it senses the channel idle, it turns its transceiver off until its next channel-check operation in order to save power. All the nodes in the same tier check the channel simultaneously, and, T_o units of time later, the nodes in the next tier do the same, as shown in Figure 2.

As an example of FAT's normal operation, if node D senses the channel idle during its periodic channel-check operation, D assumes that none of its potential children, A and B, are engaged in a tree construction process that requires D's collaboration, so D turns off its transceiver until its next scheduled channel-check operation. If, on the other hand, D senses the channel busy, it is probably because A or B deliberately transmitted a signal called *activation tone* at exactly the time when D and the other nodes in Tier 1 were scheduled to check the channel. The activation tones do not bear any information and the sources of these tones only transmit them in order to make their potential parents sense the channel busy. If a node performing a channel check senses the channel busy, we say that it becomes an *offerer*.

2.3.2 Response to events

When an event occurs, some nodes detect it and thus become data sources. All the sources independently initiate the construction of a data aggregation tree specifically suited to report the new event. The sources initiate this tree construction by becoming contenders. A *contender* is any node seeking to find a next hop towards the sink. This next hop will be the contender's parent node in the data aggregation tree, so we refer to this node as a *parent*. When a contender obtains a parent, that parent becomes a contender, and so on, except if that parent happens to be the sink. Eventually, this recursive algorithm obtains an aggregation tree containing a path from every data source to the sink.

Let us explain how the nodes respond to an event with an example. Suppose that an event occurs and only nodes A and B become its associated data sources. Because A and B are sources, they become contenders. As such, A and B transmit activation tones at the scheduled channel-check time of the nodes in their next tier, which is Tier 1. Their neighbors in Tier 1, which are D, E and F, detect these tones, and thus become offerers. Immediately after the channel check, the offerers start a period of duration T_o, referred to as *contention period*, during which they keep their receivers active. This period is the only opportunity of the contenders, A and B, to obtain a parent. If an offerer has not obtained any children at the end of the contention period, it can sleep until the next scheduled channel check. Otherwise the offerer becomes a contender and the process repeats itself until the sink.

2.3.3 Contention for a parent

During the contention period, each contender contends against the other contenders to transmit a Parent Request (PR) packet to its *preferred parent*, which is the node among its potential parents that it prefers to have as a parent in the tree. If a PR packet is successfully received by the offerer that was intended as its recipient, that offerer accepts the request to become a parent by replying with a Parent Confirmation (PC) packet. Initially, each contender chooses its preferred parent randomly. However, it may change this choice later, but only until it receives PR packet. However, until it receives a PR addressed to itself, it may change its initial choice if it overhears some packets indicating tyhat a different parent would lead to a better aggregation tree.

We explain why a contender may change its preferred parent with an example. Suppose that nodes A and B initially choose D and F, respectively, as their preferred parents. Suppose that A manages to transmit a PR packet to its preferred parent, D, before B transmits a PR packet to its preferred parent, F. If D receives A's packet correctly, D replies to A with a PR packet. Suppose that A and B receive this PR packet. Node A has just obtained a parent, so it does not need to take any further action. Node B, on the other hand, has just learned that D is A's parent, and this new piece of information makes B reconsider its preferred parent. Node B estimates that D would be the best parent, because D could compress the data from A and B together, so B sets D as its preferred parent. After this, B contends to transmit a PR packet to D, and when this packet reaches D, D confirms that it will be B's parent by replying with a PR packet.

2.3.4 The random nature of FAT

In the previous example, the two sources, A and B, obtained the same parent, D. This result is very desirable because the data of the two sources will be compressed close to them, which implies that the constructed tree is efficient. However, several random factors intervene in FAT, so FAT could have constructed a less efficient tree. For example, if B had transmitted a PR packet before A did, B's initial preferred parent, F, would have replied to B's PR packet with a PC packet, and thus B would have obtained F as a parent. Node A is too far away to receive the PR packet from F, so A would have maintained its original preferred parent, D. As a result, eventually A would have obtained D as parent, resulting in a less efficient tree.

During a contention period, some of the PR packets sent by a contender may not be received by its recipient, so every contender sends PR packets until it receives a PC packet in response. PC packets may also be lost, and this may cause two offerers to obtain the same node as a child, which is a prob-

lem that can be handled after the tree construction. Another potential problem is a permanent failure of the wireless link between a contender and its initial preferred parent. In order to find a parent in spite of this situation, a contender changes its preferred parent if it repeatedly fails to receive a reply in response to a PR packet.

The contention period duration, T_o, is a fixed parameter for the entire network. If T_o is too long, the offerers wait unnecessarily long for PRs and waste energy. If T_o is too short, some contenders may not have time to find a parent. A contender also fails to find a parent if all the links to its potential parents fail. In any of the two cases, we say the *normal construction* of the tree has failed. We provide the following extension to the FAT to handle this eventuality through an *emergency construction* of the tree, which is triggered by *emergency tones*. Every offerer checks for emergency tones at the end of the contention period. All the unsuccessful contenders and the offerers that detected an emergency tone help propagate the emergency tones. They do so by sending activation and emergency tones when the nodes in the previous and next tier are checking for them. The goal is to propagate the emergency tones to all the nodes in the network. Every node that received an emergency tone remains active for enough time to execute a reactive routing protocol such as Ad-hoc On-Demand Distance Vector (AODV). This process ensures that all the data sources find a path to the gateway if it exists.

2.4 Analysis of the Tree Construction Delay D_b

The duration of FAT's normal construction is at most $D_b = KT_o$, where K is the total number of tiers in the WSN. As we mentioned before, $D_b = KT_c$ for centralized protocols such as Oceanus [5]. Considering $T_o \ll T_c$, this means that FAT constructs the aggregation tree much faster than the centralized protocols, especially for large networks. This is the main advantage of FAT, the proposed algorithm. A faster construction allows to reduce T_c and still meet the deadline D_l, thus saving power by reducing the number of channel check operations. The emergency construction of FAT is much longer and energy-consuming than the normal construction of FAT. It consists of the propagation time of the emergency tones and the execution time of AODV. The first part can consist of several multiples of $(T_c - T_o)$, depending on the topology and on which links failed, as can be derived from Figure 2.

2.5 Simulation of the aggregation Tree Quality

Having seen FAT's advantage of requiring lower D_b than the centralized protocols, now we investigate the cost of this advantage in terms of an increased D_t. A better aggregation tree compresses the event information closer to the data sources, thereby reducing the traversal time D_t. A custom simulator has been developed to evaluate D_t for the trees obtained by FAT and the following algorithms:

- *Shortest Path Tree (SPT).* This algorithm is oblivious of the set of sources and simply uses the hop count as a metric when constructing the tree.
- *Dijkstra1.* This algorithm enhances Dijkstra's algorithm to promote aggregation. It still uses the number of hops as the cost metric, but when choosing the next node to add to the tree, it prefers source nodes to non-source nodes if their hop number is the same. In addition, when choosing the next hop for a new addition to the tree, it prefers nodes with data if their hop distance is the same.
- *Centralized1.* This algorithm has several steps. First, it constructs a tree rooted in node N_e using the Dijkstra1 algorithm, where N_e is the node that lies the closest to the event. Then, this algorithm removes all the nodes that are neither sources nor in the path from a source to node N_e, and it links N_e with the sink through the shortest path. This algorithm is optimistic because in practice N_e may be unknown.

In order to compare fairly D_t for all these trees, we use the same MAC protocol for all of them. We choose CSMA although other protocols may be more efficient. We also simulate the DMAC protocol as benchmark of the achievable performance without data aggregation. DMAC provides very short delay in duty-cycled environments and involves no tree construction overhead.

We simulate 50 nodes randomly deployed over a 200 m by 200 m rectangle. The transmission and interference ranges are 60 m and 150 m, respectively. The sink is at one corner of the rectangle. The event to be reported occurs in the diagonally opposite corner and the $s = 12$ nodes closest to the event location detect the event in some way and thus become data sources. Each source node generates ten packets. The transmission time of a packet containing the result of compressing the data from n sources is

$$T_n = T_1(1 + \alpha(n-1)) \tag{2}$$

where T_1 is the transmission time of a packet containing the data of one source without aggregation with any other data, which is assumed to be 8 ms, and α is a parameter between 0 and 1. For $\alpha = 0$, we can compress the data from any number of nodes into one packet as short as one containing data from only one node. For $\alpha = 1$, which is the other extreme value of α, no compression is feasible.

Figure 3 shows the average of the results of simulating FAT for 400 random deployments. It reveals that the relative performance of the different algorithms in terms of D_t varies greatly with α. The results may also vary for other aggregation models. For $\alpha = 0$, Centralized1 is a reasonable approximation of the optimal solution, which is the Steiner tree. FAT performs about 7 % worse than Centralized1, but FAT offers a similar improvement over SPT.

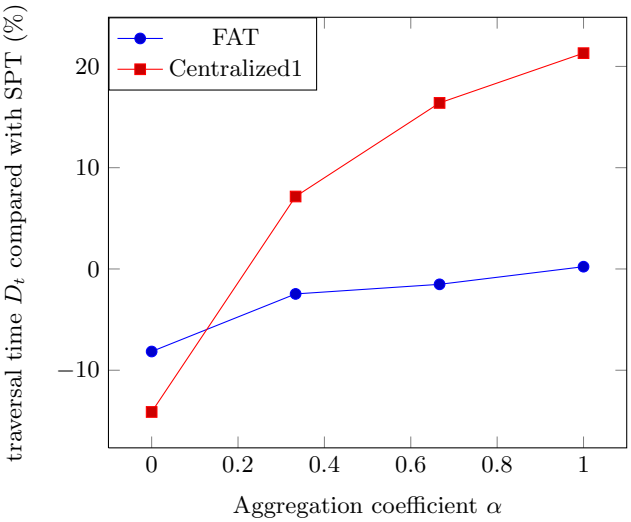

Figure 3: Relative difference in D_t of FAT and Centralized1 when compared to SPT. The smaller the value, the better the tree is. Centralized1 is a good solution when little aggregation is feasible ($\alpha \approx 0$), but performs poorly as more data can be aggregated. For $\alpha = 1$, SPT is the optimal tree.

We can see that FAT offers lower D_t than the SPT. It is important to note that this performance improvement is achieved without significantly increasing D_b. This may seem counterintuitive because one may think that D_b should be zero for the SPT because the SPT is a fixed structure for all events and thus there is no need to construct a new tree for each event. However, it is false that D_b is zero for the SPT. When an event occurs, a new set of nodes become data sources. The ancestors of the new sources do not need to be chosen because they are fixed, but every ancestor needs to know exactly which of its children have packets so that it knows when it does not need to wait for any more packets so that it can transmit the result of compressing those packets to the next hop. We are not aware of any protocol to notify the ancestors of the sources in this way, and we believe that a very efficient way to do so would be to use the same staggered schedule as FAT does, so the resulting protocol would be identical to FAT with the only difference that the recipients of the PR packets are not changed dynamically based on the overhead information. Therefore, if we use this idea with the SPT, the resulting D_b would be very similar to that of FAT.

The Dijkstra1 heuristic approximates the tree that our protocol would obtain if every node, rather than making an initial random choice of the recipient of its PR, made the optimal choice. We do not plot the results of Dijkstra1 because they are very similar to those of the FAT, indicating that the randomness of the initial parent choice hardly degrades the tree quality. For $\alpha = 1$, aggregation is infeasible and the SPT is the optimal tree.

Figure 4 shows that the benefit of good aggregation increases with the number of sources s. It also shows that even for a small number of sources, any aggregation protocol greatly outperforms DMAC.

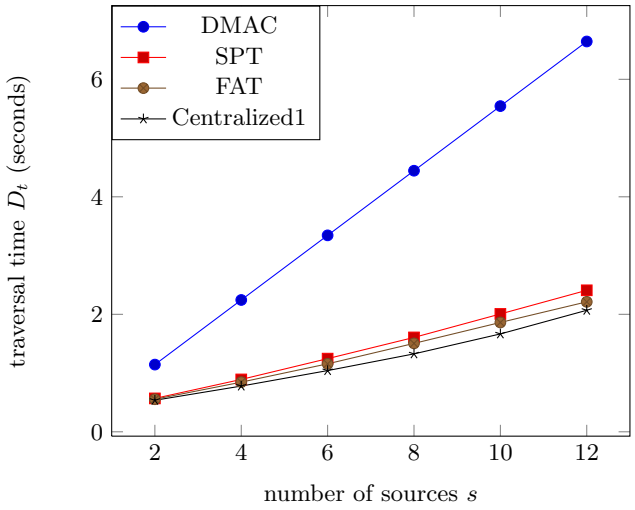

Figure 4: FAT's advantage over both SPT and DMAC increases with the number of sources. We obtained this graph for $\alpha = 0$, which implies maximum aggregation.

3 Clustering for scalability

After a tree construction process has finished, if no path has been constructed from a particular source node to the sink, we call that source node *isolated*. We say that the tree construction *succeeded* if there are no isolated sources. The normal construction of the FAT protocol always succeeds if there is a path from every source to the sink and if the periodic maintenance operations keep every node's topological data up to date. The topological data needed by every node are its tier number and its list of neighbors in the next tier.

3.1 The node-failure problem

Now assume that each node fails with probability f_0, and that the periodic maintenance operations are so infrequent that the node failures remain undetected. FAT succeeds despite these node failures if during its execution every contender can reach at least one of its potential parents. However this is not always the case. Sometimes, when a nodes fails, the nodes in its subtree need to change their tier in order for FAT to succeed.

We define f_b as the probability of a source node remaining isolated after FAT's normal construction. As an example of the computation of f_b in a simple topology, if every node has n potential parents and the sources are h hops away from the sink, f_b is $1 - (1 - f_0^n)^h$. This example suggests that FAT becomes more robust as the number of hops decreases and the node density increases. We define the node density ρ as the number of nods per unit area. Let r_t be every node's radio transmission range. We define the *normalized node density* as $\bar{\rho} = \pi r_t^2 \rho$, which indicates the expected number of nodes within a node's transmission range.

We use simulations to evaluate f_b in random deployments. For comparison, we also evaluate the probabilities of having isolated source nodes for two other techniques, namely, when using a fixed tree, and after executing FAT's emergency construction, and we represent these probabilities by f_s and f_e, respectively. We deploy the nodes at random locations, but we discard the topologies with over 20 % of disconnected nodes, which rarely occur for $\bar{\rho} \geq 7$. The monitored area is a rectangle of width d and height $4r_t$. The sink and the event center are located at the middle of left and right borders of the rectangle, respectively, a distance d away from each other.

Figure 5 shows the average of simulating 3200 random deployments. We can see that as the width of the monitored rectangle d grows, the fraction of isolated nodes increases, which is logical given that the number of potential failure points increases. We can also see that, as the normalized node density $\bar{\rho}$ increases, f_b decreases, which is what we expect because a higher node density results in every node having more potential parents. The source isolation probability after FAT's normal construction, f_b, is up to 40 % smaller than when using a fixed tree, f_s. However, f_b is unacceptably high even for small, dense networks.

In order for f_b to be sufficiently low, most nodes failures should be detected by the maintenance operations, which can be achieved by setting T_m sufficiently small. Otherwise, a proportion f_b of the sources will initiate the emergency construction, which is slow and costly. We could choose T_m so as to minimize the sum of consumption of the maintenance and reporting operations

$$\frac{E_m}{T_m} + \frac{((1 - f_b)\, E_b + f_b E_e + E_t)}{T_e}, \tag{3}$$

where f_b is a function of T_m. In practice, we may have some constraints. For example, we may have an upper threshold on f_b because we cannot tolerate the delay of the emergency construction. As another example, we may want to have more maintenance operations to record all the link failures in order to estimate f_b, which is useful to measure of the network reliability and to decide T_m.

As the number of hops in the network grows, the cost of keeping the topological information up to date grows. This is because, when a node fails near the gateway, its whole subtree may need to update its tier information. In addition, the cost of the emergency construction, E_e, grows with the number of

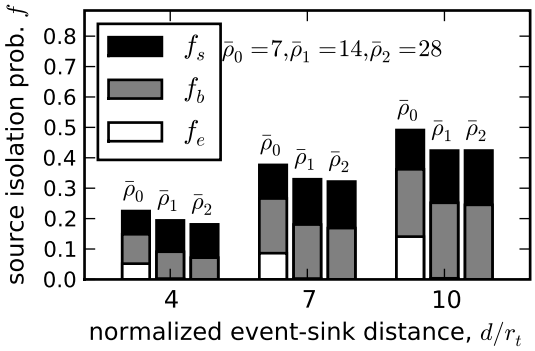

Figure 5: Probability f of a source node becoming isolated when f_0 =5 % of the sensor nodes fail and their neighbors are unaware of this. For three different event-sink distances and for three different node densities $\bar{\rho}$, three different techniques are compared. These techniques are the use of a fixed tree, FAT's normal construction, and FAT's emergency construction, and their respective source isolation probabilities are represented by f_s, f_b and f_e. The values of f should be read from 0 to the upper location of the bar, as opposed to from the difference between the locations of the lower and upper borders of the bar. This representation is possible because $f_s > f_b > f_e$.

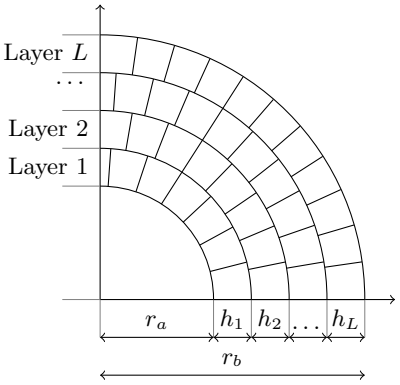

Figure 6: Segmentation of the network in layers and clusters. The sink is at the center and the clusters are approximately squares.

nodes. Therefore, the maintenance operations may represent the major source of energy consumption in large networks.

3.2 Cluster-based topology

We can to enhance the FAT protocol to require less frequent maintenance operations in large networks as follows. We partition the network in clusters, as in Figure 6, and we execute FAT in every cluster, with the cluster head playing the same role as the sink in our initial description of FAT. The cluster heads are the only nodes to belong to two clusters. This is so that the packets from every source can eventually reach the sink. We introduce gaps in the schedules from one cluster to the next one so that not all nodes need to update their tier when a node in another cluster fails. This cluster-based topology reduces the maintenance cost, but unfortunately it aggregates data less efficiently (it yields higher D_t). This is because if an event is detected by nodes in different clusters, their data will travel a longer distance until it is compressed. A cluster-based topology also suffers increased delays (D_b is higher) due to gaps in the schedule.

In cluster-based topologies, the size of the clusters is important. The choice of bigger clusters reduces the probability that an event is detected by multiple clusters, which is beneficial. However, increasing the cluster size also increases the maintenance overhead. Also note that nodes close to the gateway have to relay more packets and thus have less energy than nodes further away from the sink, and thus the optimal cluster size varies with the distance from the sink.

4 One-time aggregation using clusters

The previous section suggests that combining the FAT protocol with a cluster-based topology can potentially reduce the maintenance cost in networks with many tiers and a certain level of link failure probability. The optimal distribution of the cluster sizes as a function of the distance from the sink is difficult to analyze in such a topology. Therefore, here we consider the optimal cluster size problem in a simpler system. This system uses a data compression model that differs from (2) as follows, and it is designed to suit some of our applications in the WINES project [2].

4.1 Unrepeatable compression model

In contrast to the aggregation model in (2), here we assume that aggregation is *unrepeatable*, which means that compressed data cannot be compressed again with other data. We also assume that relaying a packet containing the raw data from one node across one hop costs E_r, and that relaying the result of compressing the data from n nodes across one hop costs E_c, independently of the value of n. We define the compression factor

$$\sigma = \frac{E_r}{E_c}, \tag{4}$$

and we consider values of σ between 4 and 40.

This model is motivated by the problem of reporting fractures in bridges, which belongs to the WINES project [2]. In this problem, when the sensor nodes detect the acoustic signals from the fractures, they generate large data volumes in which the sink is not interested. On the contrary, the sink only needs an event summary containing, for example, the list of nodes that detected the event, the detection time, the probability that the signals stem from a relevant event, and its type. In order to compress the data from multiple sources, the full waveforms of the acoustic data are needed, because the compression involves the cross-correlation of the signals. The summary cannot be cross-correlated with raw data, so compression is unrepeatable. Although from an information theoretic point of view it should be possible to develop algorithms for repeatable aggregation, they are difficult to develop and hard to implement on the simple hardware of the sensor nodes.

4.2 Optimal cluster size distribution problem

We consider a multi-hop WSN with a single sink such as the one in Figure 6. The monitored area is limited by two concentric circles of radii r_a and r_b centered at the sink. We divide the network in L concentric layers, and we define h_i as the difference between the two radii that delimit Layer i. Therefore, we have $\sum_{i=1}^{L} h_i = r_b - r_a$. We divide each Layer i into $\lceil 2\pi r_i/h_i \rceil$clusters, which are approximately rectangles of side h_i. Each cluster has a cluster head, located approximately in the middle of the cluster edge lying the closest to the sink. We assume that compression is unrepeatable and that it may only occur at the cluster heads.

The node density is ρ and the transmission range is r_t. The nodes within $r_a + r_t$ from the sink can reach it directly. This assumption is justifiable because the sink may have a more sensitive receiver or an antenna with a higher gain. We define the normalized network size as

$$\beta = \frac{r_b - r_a}{r_{tx}}. \quad (5)$$

With period T_e, all the nodes inside a randomly located area, referred to as *event area*, become sources. The event area is a square of side s, two of whose sides are approximately parallel to two radii centered at the sink. The raw data from every source travels to the clusterhead of that source, possibly through multiple hops. The clusterhead compresses the data, and the results are forwarded to the sink, also possibly through multiple hops.

We assume that data transmission is the only source of energy consumption. We refer to the traffic of raw data towards the clusterhead as *internal traffic*, and to the traffic of compressed data from the cluster heads to the sink as *external traffic*. We define E_k^i and E_k^e as the expected energy that a node in Layer k spends to forward internal and external traffic, respectively, each time that an event occurs somewhere in the network. We also define the addition of the previous two terms as $E_k^t = E_k^i + E_k^e$, and the external to the total traffic ratio as

$$\varepsilon = \frac{E_k^e}{E_k^t}. \quad (6)$$

Our optimization problem is to find the cluster-size distribution that maximizes the network lifetime. In other words, we seek the optimal values of L and $\{h_k\}_{k=1,\ldots,L}$ that minimize $\max\left(E_k^t\right)$, with $k \in \{1, \ldots, L\}$.

If an event area covers several clusters, the data from different clusters will not be compressed with each other, resulting in a higher E_k^k, which is undesirable. Assume for a moment a uniform cluster-size distribution. If we increase the cluster size, the expected number of clusters affected per event decreases, experimenting a beneficial reduction in E_k^e, but unfortunately E_k^i increases. Therefore, there must be an optimal uniform cluster size that minimizes E_k^t. For this distribution, all the h_k are identical, so all the E_k^i are also identical, but all the E_k^e will not be identical because the nodes closer to the sink have to relay more data. In particular, the most energy consuming layer would be Layer 1. Starting from the uniform distribution, we can extend the lifetime by reducing h_1 and increasing h_L, because we reduce the energy consumption in Layer 1, which is the potential bottleneck. Therefore, a non-uniform cluster-size distribution can provide a longer network lifetime than a uniform one.

4.3 Related work

We only consider here the literature proposing ways to construct a permanent aggregation structure to support the reporting of multiple unpredictable events. In [10], the authors study the use of fixed trees for this problem. However, this approach incurs high maintenance overhead in large networks, as discussed in Section 3. The ToD protocol [7] proposes a hybrid structure consisting of clusters and two spanning trees. By using efficient routing, it guarantees complete aggregation in a few hops when the cluster diameter exceeds the diameter of the event area. Aggregation in ToD can occur in several ways, namely opportunistically within a cluster using DAA+RW, or deterministically in primary and secondary aggregators. To use ToD for unrepeatable compression, we have to select an aggregation point, and we believe this point should be the first secondary aggregator in the path to the sink. The ToD simulations [7] involve only three different cluster sizes, thereby failing to explore the optimal cluster-size problem.

SCT [11] proposes a structure similar to the one in Figure 6, but in which all the clusters have the same size. In addition, the authors of SCT only consider multi-hop clusters very briefly. In the rest of their paper, they choose a cluster size that enables single hop communication with the cluster head. In addition, the authors assume that the sources associated with a given event are uniformly distributed in the entire monitored area, whereas in most problems the sources associated with a given event are located in a reduced area centered at the place where the event occurred.

UCS [12] and EEUC [13] are two techniques to choose clusters of unequal size for data aggregation. They assume unrepeatable compression as we do, but, in contrast to the clusters in our model, their clusters consist of only one hop. They assume that the transmission cost between two nodes grows with d^ϵ, where d is the distance between the nodes and $\epsilon \geq 2$ is a constant. They also consider that all the nodes in the network are sources. Most of the simulations of UCS [12] were obtained for two-layer topologies, and only a few of them for a three-layer topology. The authors chose such a small number of layers because their heuristic is not scalable, as it considers many parameters and uses exhaustive search to find the optimal setting. UCR is simpler, as it increases the cluster size linearly with a single parameter c, so it is easy to approximate the optimal c. However, the authors fail to justify the choice of a linear variation. UCR requires every node to be able to receive messages from the sink, and that the received signal strength depends deterministically with the distance from the sink, which may not be the case for certain operating environments. Furthermore, UCS and UCR do not identify the parameters upon which the improvement of an unequal cluster-size distribution depends.

4.4 Cost analysis

We assume that every node can communicate with its neighbors less than r_t away. If two nodes lie a distance d away from each other, the expected number of hops between them is $\zeta \approx \omega d/r_t$, where ω is a constant that depends on the normalized node density $\bar{\rho}$. For an extremely dense network, the number of hops, ζ, is exactly $\lceil \omega d/r_t \rceil$, with $\omega = 1$. As $\bar{\rho}$ becomes smaller, ω increases. In the following, we use $\zeta \approx \omega d/r_t$, and refer to it as the *hop count approximation*.

4.4.1 Internal traffic

The expected distance from a random location in a cluster in Layer k to its cluster head, which lies at the middle of the lower part of the cluster, is

$$h_k^{-2} \int_0^{h_k} \int_0^{h_k} \sqrt{(x - h_k/2)^2 + y^2}\, \mathrm{d}x\, \mathrm{d}y \approx 0.76 h_k. \quad (7)$$

Therefore, the expected number of hops of the internal traffic is $\zeta = 0.76\omega h_i/r_t$. The probability that a node will detect an event when it occurs somewhere in the network is s^2/A, where $A = \pi\left(r_b^2 - r_a^2\right)$ is the monitored area. The number of reporting nodes grows with the number of nodes in the cluster m_k, but this load is also divided among the m_k nodes. Therefore, the expected internal cost per event for a node in Layer k is

$$E_k^i = \frac{0.76\omega s^2}{A} E_r h_k. \quad (8)$$

4.4.2 External traffic

We assume the event area is a square whose side is s and whose orientation matches the one of the clusters. The expected number of events reported in Layer k when an event occurs somewhere in the network is

$$n_k = \frac{q_k \left(h_k + s\right)^2}{A_f} \tag{9}$$

where q_k is the number of clusters in Layer k, and it is given by $q_k = \lceil 2\pi r_k / h_k \rceil$, and $A_f = \pi \left(\left(r_b - s\right)^2 - r_a^2 \right)$ is the area where the lower left of the event may occur.

When an event occurs, the number of compressed packets that travel to the sink is equal to the number of clusters intersecting with the event area. The total number of external packets that traverse Layer k is $\sum_{j=k+1}^{L} n_j$, and that traversal consists of $\omega h_k / r_t$ hops. There are $2\pi r_k h_k \rho$ nodes in Layer k, and we assume that their load is uniformly distributed among them. Therefore, the expected external cost per event for a node in Layer k is

$$E_k^e = E_c \left(\omega \frac{h_k}{r_t} \right) \frac{\sum_{j=k+1}^{L} n_j}{2\pi r_k h_k \rho}. \tag{10}$$

4.4.3 Optimization problem

Considering all factors, the expected energy consumption of a node in Layer k when an event occurs is

$$E_k^t = \frac{E_c \omega}{2\pi r_t r_k \rho} \sum_{j=k+1}^{L} \frac{2\pi r_j \left(h_j + s\right)^2}{h_j A_f} + \frac{0.76 \omega s^2}{A} E_r h_k. \tag{11}$$

We can compute the optimal layer-size distribution by solving the following optimization problem:

$$\begin{aligned} &\text{minimize } \max\left\{E_k^t\right\} \\ &\text{subject to } \sum_{i=1}^{L} h_i = r_b - r_a. \\ &\text{over } h_k \gg r_t \end{aligned} \tag{12}$$

We have imposed the constraint that $h_k \gg r_t$ because otherwise the hop count approximation becomes unreasonable.

The optimization problem (12) has L continuous variables and it is non-convex, so it is difficult to solve. Based on the physical understanding of the problem we propose the following heuristic. We start with all the layers of equal size. In each iteration, we extend by Δ the size of the most energy consuming layer and we shrink the least energy consuming layer by that same amount. The parameter Δ does not need to be very small because in a real deployment we may not have very precise geographic information. This heuristic approximates the optimal distribution of sizes given L. We execute this heuristic for all the possible values of L, which is an integer, and select the L that yields the minimum $\max\left\{E_k^t\right\}$.

4.5 Simulations

We define the *variety improvement* ν as the relative lifetime improvement of our unequal cluster-size heuristic compared to the optimal uniform cluster size. In this section, we use simulations to evaluate how ν depends on three key parameters, namely the normalized network size β, the normalized event size $\gamma = s/r_t$ and the compression factor σ. In our simulations, we fix the node density around $\bar{\rho} = 10$ because with this value, the network becomes connected network with a high probability. In many WSNs, we can control $\bar{\rho}$ by changing the transmission range r_t.

The left-most column of small figures in Figure (7) shows the influence of the network size β. As the network grows, the number of external packets grows, further straining the nodes close to the gateway. To reduce the number of external packets, the optimal average layer width, mean(h_k), grows, but not as much to avoid an increase in the external to total traffic ratio ε. As ε increases, ν decreases.

The middle column of Figure (7) analyzes the effect of changing the event size γ. As the event area grows, more clusters are affected by each event, and the load upon the nodes near the gateway increases. To reduce this load, the optimal average cluster size is increased. Larger clusters lead to greater importance of internal packets, and thus to a reduction in ε and an increase in ν.

The right column of Figure (7) presents the results when we vary the compression factor σ. When more compression is feasible, it is not so important to reduce the number of external packets, so the optimal average layer width decreases. Despite this decrease, the internal traffic becomes the main source of power consumption, so ε decreases. At the same time, the variety improvement ν grows.

In the three columns, we see that when ε decreases, ν increases. This is what we expect. When ε is small, the internal traffic accounts for the greatest share of the energy consumption ($E_k^t \approx E_k^i$). The internal traffic of Layer k, E_k^i, is linearly proportional to the height of Layer k, as given by (8), and thus E_k^t is almost linearly proportional to h_k. Therefore, when ε is small, we can control E_k^t very easily simply by changing h_k. This makes it easier to distribute the power consumption more uniformly between the layers, and for this reason a small ε leads to a big ν.

To summarize, our algorithm achieves up to a 12 % improvement in terms of network lifetime over a uniform cluster size. Such improvement is increased when the WSN consists of few hops, when the events are detected by many sensors, and when data can be compressed significantly. This improvement does not come at the cost of any additional overhead during the network operation, because the layers are set during the network initialization.

5 Conclusions and future work

We have proposed the FAT protocol, which is the first protocol that combines a staggered schedule with data aggregation. By using this schedule, FAT divides the tree construction delay by the number of hops in the network when compared to existing protocols. A faster tree construction allows the nodes to check for packets less frequently, which saves energy. We used simulations to compare the efficiencies of the aggregation trees obtained by the FAT protocol and a centralized heuristic. Our results show that the centralized heuristic was only 7 % better than our protocol, which is a small price to pay for the reduction in the tree construction time. However, a key advantage of FAT is its speed and scalability with the network size.

Similarly to every tree structure, the FAT has a disadvantage that, if a link becomes unavailable, all the nodes that lie in its sub-tree may need to be reconfigured. Our simulations showed that the FAT method is vulnerable to outdated topological information, so the topology information should be updated periodically. As the network grows, the cost of the maintenance operations grows. However, this cost can be reduced by combining the FAT with a cluster-based topology.

We formulated the optimal-cluster size problem under a data compression model that differs from the one of the FAT protocol

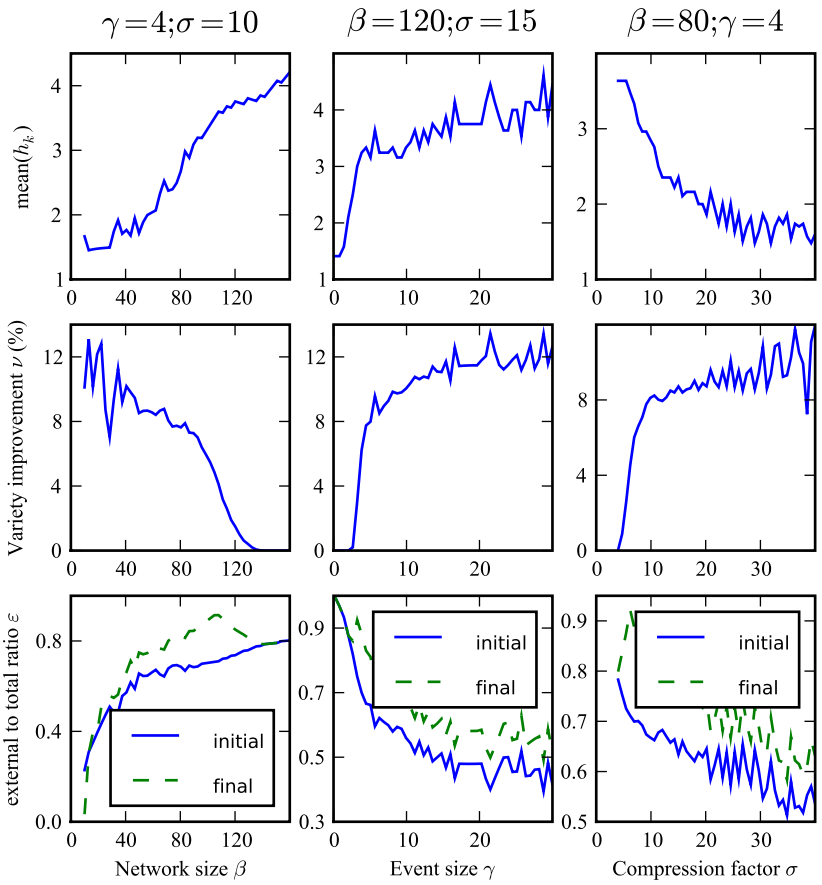

Figure 7: Influence of the network size β, the event size γ and the compression factor σ in our system. In the 3×3 matrix of panels, each column represents the same set of simulations, in which we fix two variables, which are shown at the top of each column, and we change the remaining variable. In each row, we show three different variables. In the third row, we plot ε before and after applying our heuristic.

and from the ones commonly found in the literature, but that serves very well to report the acoustic signals from fractures in bridges and tunnels. Our model assumes significant compression in a single point. We proposed a heuristic to approximate the optimal cluster sizes, which can achieve up to a 12 % increase in lifetime when compared to the uniform cluster-size approaches. We selected three key parameters that allow us to explain the lifetime improvement, and evaluated the influence of those parameters with simulation. In our future work, we shall detail how to reduce the maintenance overhead of FAT. We plan to use exhaustive search methods to show how close our cluster-size heuristic result is from the optimal solution.

References

[1] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Wireless Commun. Mag.*, vol. 14, no. 2, pp. 70–87, 2007.

[2] WINES Consortium, "Wired and Wireless Intelligent Networked Systems (WINES) – Smart Infrastructure Project," *[Online]. Available: www.winesinfrastructure.org*, 2008.

[3] G. di Bacco, T. Melodia, and F. Cuomo, "A MAC protocol for delay-bounded applications in wireless sensor networks," in *Proc. Mediterranean Ad Hoc Networking Conference*, Jun. 2004, pp. 208–220.

[4] K. W. Fan, S. Li, and P. Sinha, "Structure-free data aggregation in sensor networks," *IEEE Trans. Mobile Comput.*, vol. 6, no. 8, pp. 929–942, 2007.

[5] A. F. Harris III, R. Kravets, and I. Gupta, "Building trees based on aggregation efficiency in sensor networks," *Elsevier J. Ad Hoc Networks*, vol. 5, no. 8, pp. 1317–1328, Nov. 2007.

[6] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An adaptive energy-efficient and low-latency MAC for tree-based data gathering in wireless sensor networks," *Wiley J. Wireless Comm. and Mobile Computing*, vol. 7, no. 7, pp. 863–875, 2007.

[7] K. W. Fan, S. Li, and P. Sinha, "Scalable data aggregation for dynamic events in sensor networks," in *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys)*, 2006, pp. 181–194.

[8] K. G. Langendoen, "Medium access control in wireless sensor networks," in *Medium Access Control in Wireless Networks, Volume II: Practice and Standards*, H. Wu and Y. Pan, Eds. Hauppage, New York: Nova Science Publishers, May 2008, pp. 535–560.

[9] R. Mangharam, A. Rowe, and R. Rajkumar, "FireFly: a cross-layer platform for real-time embedded wireless networks," *Springer J. Real-Time Systems*, vol. 37, no. 3, pp. 183–231, Dec. 2007.

[10] L. Cai and D. Corneil, "Tree spanners," *SIAM J. Discrete Mathematics*, vol. 8, pp. 359–359, 1995.

[11] Y. Zhu, R. Vendantham, S.-J. Park, and R. Sivakumar, "A scalable correlation aware aggregation strategy for wireless sensor networks," in *Proceedings of the First Int'l Conf. Wireless Internet*, 2005, pp. 122–129.

[12] S. Soro and W. Heinzelman, "Prolonging the lifetime of wireless sensor networks via unequal clustering," in *Proceedings of the 19th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS)*, 2005.

[13] G. Chen, C. Li, M. Ye, and J. Wu, "An unequal cluster-based routing protocol in wireless sensor networks," *Springer J. Wireless Networks*, vol. 15, no. 2, pp. 193–207, Feb. 2009.