An efficient data aggregation algorithm for cluster-based sensor network

Mohammad Mostafizur Rahman Mozumdar, Nan Guofang, Francesco Gregoretti, Luciano Lavagno
Department of Electronics, Politecnico di Torino, Italy
email: {mohammad.mozumdar, guofang.nan, gregoretti,lavagno}@polito.it
Laura Vanzago
STMicroelectronics, Milano, Italy
email: laura.vanzago@st.com

***Abstract*— Data aggregation in wireless sensor networks eliminates redundancy to improve bandwidth utilization and energy-efficiency of sensor nodes. One node, called the cluster leader, collects data from surrounding nodes and then sends the summarized information to upstream nodes. In this paper, we propose an algorithm to select a cluster leader that will perform data aggregation in a partially connected sensor network. The algorithm reduces the traffic flow inside the network by adaptively selecting the shortest route for packet routing to the cluster leader. We also describe a simulation framework for functional analysis of WSN applications taking our proposed algorithm as an example.**

***Index Terms*— data aggregation, cluster leader, simulation framework**

I. Introduction

In the last decade, the landscape of wireless sensor network (WSN) applications has been extending rapidly in many fields such as factory and building automation, environmental monitoring, security systems and in a wide variety of commercial and military areas. Advancements in microelectro-mechanical systems and wireless communication have motivated the development of small and low power sensors and radio equipped modules which are now replacing traditional wired sensor systems. These tiny modules usually called "motes" can communicate with each other by radio and act like as neurons to collect information from the environment. However, designing applications for WSN is quite challenging because of energy constraint and also for resource limitation on the mote. Because of the requirement of unattended operations in remote or even potentially hostile locations, sensor networks are extremely energy-limited. In [1], authors argued that about 70% of energy consumption inside WSNs is due to the data transmission. Thus aggregation and routing of data inside WSNs need to be dealt with very efficiently to save precious energy stored on a sensor node.

In a densely covered sensor network, local variation of the measured data among nodes is often the same, thus transmitting it individually would lead to a significant waste of energy. Using a clustering topology ([1], [2], [3]) is a popular approach by which the sensor network is partitioned into several clusters to collect data. In clustered topologies, one or more nodes collect data from all other nodes (called data aggregation) and then perform some computations (e.g. average, standard deviation, gradient) based on the collected data, prepare a single packet and send it to the network sink. Instead of routing each data packet from the nodes, a single summarized data packet is transmitted from the cluster, thus reducing network traffic load and ultimately saving energy.

We proposed an algorithm [4] for selecting the cluster leader who will be in charge of data aggregation inside a cluster. Most of the cluster leader selection algorithms assume that in a cluster every node pair can hear each other, which may not be the case in practice for many WSN applications (such as hospital and industrial automation, etc.). That's why in our new contributions described in this paper we assume that a cluster is partially connected, hence we have to consider also in-network data routing for aggregation.

Just like other embedded systems, WSN algorithms need to be verified functionally before being implemented on the actual platform. We proposed a simulation framework [5] for functional analysis of WSN application in a multi-node environment. This framework is based on MathWorks [6] tools and it is capable of modeling, simulation and multi-platform automatic code generation of WSN application.

In this paper, we provide a formal mathematical description of the algorithm described in [4] and also implement it with the simulation framework described in [5]. Thus the major contribution of this paper is to illustrate a complete path to design WSN application by providing a novel algorithm and then refining and analyzing its behavior by modeling it in a high level simulation framework.

The rest of the paper is organized as follows: In section 2, we outline existing literature related to our proposed algorithm. In section 3, we present the algorithm. In section 4, we present our simulation framework, whereas in section 5, we model the algorithm in our simulation framework. We show the performance results of our algorithm in section 6 and finally we conclude in section 7 with future directions.

II. Related Work

In clustered environments, there are two main approaches for data aggregation. The first approach is known as the Cluster Head (CH) method. The idea behind the approach is that one node in the cluster will be elected as the CH at the beginning of each aggregation round. The rest of the nodes in the cluster will send data packets to the CH according to the underlying MAC protocol. The CH will collect all the data packets and will forward these packets to the network sink. To ensure fair distribution of the workload, the cluster leader is selected randomly at each round of aggregation. A widely known example of this type of algorithm is LEACH [7], [8], [9], [10]. In general, convergence time and total energy consumption are linear with respect to the number of nodes for this type of algorithm. However, good performance is usually offset by the lack of robustness in handling a scenario such as the death of the CH in the middle of the aggregation process, that stops the aggregation and causes the loss of data accumulated up to that point.

The second approach is peer-to-peer or gossip-based algorithms. These algorithms have been proposed as a fault-tolerant approach for the distributed computation of aggregation functions. Let us consider a network of N nodes where each node has some information. Making each node aware of the information stored in every other node is the main goal of the gossip approach [11], [12]. In wired networks, this phenomenon has been widely studied and it has various important applications. Several gossip algorithms have been developed to compute aggregate commutative functions such as max, min and average among N values distributed over N nodes. In general, these algorithms are quite complex but simplified versions are available for simple aggregation functions (computing sum or average of N values) [13]. In gossip based algorithms, the trade-off is robustness versus convergence time and energy consumption. Generally, the time and energy consumption are $O(NlogN)$ for a cluster of N nodes. The root of this inefficiency is due to the point to point communication that does not take inherent advantage of the broadcasting nature of wireless channel. A variant of gossip-based algorithms called DRG has been presented in [14]. It takes the advantage of broadcasting. However, implementing DRG in a clustered environment, where every node of the cluster accesses the same communication channel, would create a substantial number of collisions and jeopardize the advantage of the approach. Furthermore, nodes only store the final result but do not keep track of partial results. So, if an application requires a non linear aggregation function, then the gossip based approach cannot be utilized.

A hybrid approach that combines the robustness of the gossip algorithm with the efficiency of the cluster head algorithm is introduced in [2]. The algorithm is called EERINA. Initially every node in a cluster plays the same role and only at the end of the aggregation phase the cluster leader is selected. It takes advantage of the broadcast medium to minimize the number of transmitted messages. The combination of bandwidth efficiency along with the late selection of the cluster leader ensures a high degree of robustness with respect of node malfunctions, failures or temporary disconnections, with very limited timing and performance overhead. Furthermore, the algorithm is quite scalable and allows network changes (e.g. node deletions and additions) without updating the overall network structure.

The robustness and simplicity of this approach motivated us to look at further details. EERINA assumes that in a WSN cluster every node pair can hear each other, which may not be the case in practice. In this paper, we present a novel algorithm which has all the advantages of EERINA and can perform aggregation by selecting the cluster leader in a distributed *partially connected* sensor network.

III. Algorithm

We considered the scenario where nodes in a cluster are partially connected. That means that a node is connected to only some of the (nearby) nodes of the cluster. An example of simple cluster setup is shown in figure 1.

Figure 1. A simple view of the partially connected WSN cluster setup (arrow indicates connectivity)

The goal is to select a cluster leader among the nodes, who will aggregate the data from the sensor network and will send it to the upper layer of the network.

The algorithm has four major phases:-

- *Initialization Phase*
- *Contention Phase*
- *Exchange Phase*
- *Termination Phase*

In the initialization phase, every node transmits and receives packets randomly. At the end of the initialization phase, nodes move to the contention phase where they compete with each other to become the cluster leader. Every node that has heard from other nodes can become a potential cluster leader and transmits a Contention Packet (CP) to the rest of the network that restricts other nodes from becoming the cluster leader. After receiving the CP, a node recognizes that some other node of the cluster is trying to become the cluster leader and it immediately stops its attempt to send the CP. The CP contains information about the cluster leader and node IDs that have been heard by the cluster leader. A node that receives the CP packet, checks immediately whether it has been heard by the cluster leader or not, by parsing the CP packet. If it has been heard by the cluster leader then it checks whether it has extra information that the cluster leader does not have. If the node does not have extra information, then it will not participate in the exchange phase. In the exchange phase, only some nodes will be active, namely the cluster leaders, the nodes that have not been heard by the cluster leader and the nodes that have been heard by the cluster leader but have extra information. In the exchange phase, these nodes will transmit packets to and receive packets from each other. At the end of the exchange phase, all the nodes will again participate in the contention phase and a potential cluster leader will be elected. The loop of contention-exchange phases continues until the termination condition is met and a cluster leader (who has heard from all nodes) has been selected. The flow of the algorithm is shown in figure 2.

Figure 2. Flow of the algorithm

A. Initialization Phase

In the initialization phase, each node can be in one of three states: transmit, receive and sleep. Initially, every node computes the next time for transmitting and receiving by exponential-randomization. Then it goes to sleep until any one of these two timers expires.

If the transmitting timer expires, the node at first senses the medium for a certain amount of time and if the medium is free, then it broadcasts the packet to the medium. After transmitting the packet or if the node senses that the medium is busy, then the node computes the next time for transmitting the packet and goes back to sleep. If the receiving timer expires, the node turns on the radio to receive for a certain amount of time. In this state, the node collects packets from the other surrounding nodes of the cluster. Each node maintains a *HeardFromTable* (HFT) which contains information about the nodes from which the node has heard directly (shown in table 1). Every node spends a pre-specified amount of time in the initialization phase, and then it moves to the contention phase.

Let *G* = (*V*, HFT_v) represents a partially connected sensor cluster of *N* nodes, where *V* = $\{ v_1, v_2, v_3, .., v_n \}$ is the set of all sensor nodes in the cluster and HFT_v is the set of all *HFTs* that are stored in *N* nodes (one for each node).

TABLE I.
HFT OF NODE 2

{ Heard From Node, RT(Routed Through) }
{ 1 , {} }
{ 3 , {} }
{ 4 , {} }
......

$$HFT_v = \{HFT_{v_1}, HFT_{v_2}, HFT_{v_3}, .., HFT_{v_n}\}$$

$$\text{HFT}_{v_i} = \{\ \{\ \text{v}_j, RT\}, ..\} \quad v_i, v_j \in V,\ v_i \text{ hears from } v_j \text{ by RT}$$

$$RT = \begin{cases} \text{V}_k & \text{V}_k \subset V, \text{where } V_k \text{ is the set of those intermediate nodes by which } v_i \text{ can hear from } v_j. \\ \{\} & \text{empty, when } v_i \text{ and } v_j \text{ are directly connected to each other.} \end{cases}$$

B. Contention Phase

After termination of the initialization phase, each node enters into the contention phase where the main goal is to elect the cluster leader based on the number of nodes heard in the previous phase. Every node in the cluster participates in the contention phase. At the beginning of this phase, each node activates radio reception and sets a *Back-off Timer* (BT) which is proportional to $N_h - n_i$ (Where N_h is a constant higher than the number of nodes (N) in the cluster, n_i is the number of nodes from which node *i* heard packets, calculated from the *HFT*). So, the *BT* of node v_i is set as

$$BT(v_i) \propto (N_h - n_i)$$
$$\text{where } n_i = \|HFT_{v_i}\|$$

Therefore, the potential cluster leader will be the node $v_{pcl} \in V$ for which

$$BT(v_{pcl}) = min(BT(v_1), BT(v_2), ..BT(v_N))$$

The node whose *BT* expires earlier than the others, becomes a potential cluster leader and transmits a CP. This CP is a special type of packet that contains the ID of the cluster leader and also the node IDs from which the cluster leader has heard one or more packets during the last phase. When the surrounding nodes of the cluster leader hear the CP, they immediately stop the *BT*, wait for a small randomized amount of time and re-broadcast the CP. As the cluster is not fully connected, this flooding of CP ensures that each node of the cluster receives the contention packet, although the node may not be in the radio range of the cluster leader.

For example, as node 2 is connected to the maximum number of nodes, it has a higher probability of hearing packets from most nodes. As a result, its *BT* will most likely expire earlier than others. Node 2 then broadcasts

Figure 3. Node 2 transmits the CP packet

the CP packet (shown in figure 3). Nodes 1, 3 and 4 which are still waiting for the expiration of the *BT*, receive the CP packet from node 2. They immediately stop the timer and recognize that node 2 has become the cluster leader. Nodes 1, 3 and 4 re-transmit the CP packet (shown in figure 4) after waiting for a small random time.

Figure 4. Broadcasting of CP packet-1

Now Node 5 receives the CP packet, stops its timer and broadcasts the CP packet (shown in figure 5). In this way, the CP packet has been transmitted to the whole cluster and every node becomes aware of the cluster leader. If a node hears multiple CPs (either from one or from multiple cluster leader candidates), it will ignore CPs after the first one.

Figure 5. Broadcasting of CP packet-2

In this phase, every node analyzes the CP packet and decides whether it will participate in the next exchange phase or not. Let I_{CP} be the content of the contention packet and $f_{hn}(HFT_{v_n})$ is a function that returns a set of heard from nodes from HFT_{v_n} (stored in node *n*). For example, $f_{hn}(HFT_{v_2})$ ={1, 3, 4} (HFT_{v_2} is shown in table 1). The content of the I_{CP} packet transmitted from node *N* will be {{N, $f_{hn}(HFT_{v_n})$ },..}. A node v_x[1] will participate in the exchange phase or not as follows:

$$\begin{cases} v_x \in I_{CP} & \begin{cases} f_{hn}(HFT_{v_x}) \subset I_{CP} & \text{not participating} \\ f_{hn}(HFT_{v_x}) not \subset I_{CP} & \text{participating} \end{cases} \\ v_x \notin I_{CP} & \text{participating} \end{cases}$$

We classify four types of nodes participating in the next exchange phase (by using the definitions above).

- *Type 1*: Cluster leader nodes (Always RX mode)
- *Type 2*: Nodes that have not been heard by the cluster leader (TX/RX mode)
- *Type 3*: Nodes that have not been heard by the cluster leader and also have extra information (TX/RX mode)
- *Type 4*: Nodes that have been heard by the cluster leader but have extra information (TX/RX mode)

Figure 6. Scenario of analyzing the CP the packet

To explain the scenario, let us assume that the cluster leader for example Node 2, has heard from nodes 1, 3 and 4 (shown in figure 6). So, it will convey this information in the I_{CP}. When nodes 1 and 3 analyze the I_{CP}, they will find that their packets have already been heard by the cluster leader and also they do not have any extra information of nodes .

$$v_1 \in I_{CP}, \quad f_{hn}(HFT_{v_1}) \subset I_{CP}$$
$$v_3 \in I_{CP}, \quad f_{hn}(HFT_{v_3}) \subset I_{CP}$$

So, nodes 1 and 3 will not participate in the exchange phase. Node 4 knows that the cluster leader has its packet but it also heard from node 5 in the previous phase, which was not heard by the cluster leader (so, it will participate in the exchange phase).

$$v_4 \in I_{CP}, \quad f_{hn}(HFT_{v_4}) not \subset I_{CP}$$

Node 5 will discover that the cluster leader has not heard from it, so it will also participate in the exchange phase.

$$v_5 \notin I_{CP}$$

[1]Here v_x is a node other than the cluster leader. A cluster leader node will always participate in the exchange phase.

C. Exchange Phase

In this phase, the nodes that have not been heard by the cluster leader and/or have extra information will transmit and receive packets more frequently than during the initialization phase[2]. Since in this phase fewer nodes will participate compared to the initialization phase, the increased rate of transmit and receive will help the algorithm to converge more quickly. The cluster leader will be always in listening mode to collect packets from the other participating nodes.

Figure 7. Exchange phase

Continuing with the example of contention phase, when the cluster leader (node 2) receives a packet from node 4, it can find out that node 5 can be reached through node 4 (shown in figure 7). So, it updates the *HFT*, with the information shown in the Table 2.

TABLE II.
HFT OF NODE 2

{ Heard From Node, RT (Routed Through)}
{ 1 , {} }
{ 3 , {} }
{ 4 , {} }
{ 5 , { 4 } }
......

Some nodes may be connected to the cluster leader by multiple hops (shown in figure 8), hence the routed-through (*RT*) data is a list and can contain information about multiple nodes. The nodes that are transmitting extra information to the cluster leader store a local forwarding table. For example in figure 8, node 5 will forward all packets from node 6 and node 4 will forward all packets from node 5 and node 6.

This local forwarding table is synchronized with the cluster leader later in the contention phase by the contention packet. Figure 9 depicts another scenario where there are two different paths between cluster leader node 2 and node 6 (2-4-6, 2-4-5-6). So the *HFT* of the cluster leader might have more than one entry for the same node. In that case, the cluster leader will select the shortest routing path by using following definitions.

[2]In the experiments described below this frequency increases to twice that used in the initialization phase. We are currently analyzing the impact on this parameter on the overall convergence time.

Figure 8. Multi-hops distance from the cluster leader (node 2 - node 6)

$$\text{Let } HFT_{PCL} = \{\{v_i, RT_i\}, \{v_j, RT_j\}..\}$$
$$\text{if } v_i = v_j$$
$$\begin{cases} \|RT_i\| > \|RT_j\|, & HFT_{PCL} = \{\{v_j, RT_j\}..\} \\ \|RT_i\| <= \|RT_j\|, & HFT_{PCL} = \{\{v_i, RT_i\}..\} \end{cases}$$

Taking the scenario depicted at figure 9, let assume that

$$HFT_2 = \{\{v_i = 6, RT_i = \{4\}\}, \{v_j = 6, RT_j = \{4, 5\}\}..\}$$
$$\text{Hence } v_i = v_j = 6$$
$$\|RT_i\| <= \|RT_j\|, \quad HFT_2 = \{\{v_i = 6, RT_i = \{4\}\}..\}$$

The cluster leader selects the shortest path (2-4-6) and conveys this information in the next contention packet. In the contention packet, the cluster leader will send only the required updated part of the *HFT* to synchronize the local forwarding tables of the nodes. So after receiving the contention packet, node 5 will discard node 6 from its forwarding table. After completion of the exchange phase, all nodes will again participate in the contention phase.

Figure 9. Multiple paths between node 2 and node 6

D. Contention-Exchange alternation

The contention-exchange alternation continues until the termination conditions are met. It may happen that the first contention phase ended with multiple winners, but subsequent exchange-contention phases will reduce the probability of having multiple cluster leaders. The winners of earlier contention phases (potential cluster leaders) will collect new information from the surrounding connected nodes in the exchange phase and in this way they expand their subnetworks. A potential cluster leader that has higher connectivity can collect information from more unheard nodes (in the exchange phase) than other

Figure 10. A simple simulation framework

potential cluster leaders. So in the next contention phase, its *BT* will most likely expire earlier than the last contention phase, hence it can stop its nearest potential cluster leader and then "capture" its subnetwork in the next exchange phase. In this way, the highest potential cluster leader expands its subnetwork and ultimately captures the whole network. The contention-exchange alternation will terminate when a node becomes the cluster leader consecutively twice and there have been no changes in the *HFT* in the last exchange phase. Then the cluster leader node sends the contention packet with the information that a final cluster leader has been elected and this will terminate the contention-exchange alternation loop.

Figure 11. Connectivity matrix for the 16 nodes sensor network

IV. Simulation Framework

Verifying behavior by simulation before implementation is extremely useful to reduce the development time of an embedded application. This is even more true for wireless sensor networks, since their nodes often provide very rudimentary debugging facilities, and sufficiently large networks for realistic analysis may be expensive to deploy. Most of the available sensor nodes on the market (such as MicaZ [15], TelosB [15], Tmote Sky [16]) only provide a few on-board blinking leds as debug aids. This makes code development on the actual platform virtually impossible. To analyze the behavior of the algorithm, we used the simulation framework described in [5]. The simulation framework is developed by using Mathworks [6] tools, namely Simulink [17] and Stateflow [18]. The framework contains two main kinds of blocks such as the *sensor node* and *communication medium*. *Sensor node* blocks are connected to the *communication medium* block, which provides a mechanism for the application developer to define the connectivity between the nodes in sensor network.

The *communication medium* block is implemented in C, so it can be modified to reuse any existing channel and connectivity models. The *sensor node* contains mainly a timer, a random number generator, and a parameterized Stateflow block which actually implements the algorithm

Figure 12. Stateflow modeling of the initialization and exchange phase of the algorithm

running inside each single node (shown in the figure 10). The Stateflow block is a library object and each *sensor node* contains an instance of it. Therefore, every node of the framework is running an independent copy of same algorithm. It is of course also possible to model sensor networks having different algorithms running in different nodes. In that case, one needs to create a small Stateflow library and instantiate objects from it as needed. To model a new sensor network application based on this framework, the application developer only needs to modify the template algorithm implementation (Stateflow-library object) and set the connectivity of the nodes in the *communication medium* block. Then simulation can be started and statistical data can be collected using animated state charts, scopes and displays to perform functional analysis of the algorithm. The algorithm implementation can be refined if the analysis of the results suggest to do so. Eventually the developer will get a refined model which represents the desired behavior.

Figure 10 shows a simulation setup for 16 nodes using the framework. In the figure, the Stateflow block (*wsn_algorithm*) contains the implementation of the our algorithm.

A. Communication Medium Model

This block contains the medium logic and also models the connectivity between nodes. The logic of the communication medium block is implemented by a C based S-Function, which contains a (parameterized) 16x16 matrix to define the connectivity of the nodes in the sensor network (shown in the figure 11). For example in figure 11, node 1 (row 1) is connected to nodes 3, 10, 12 and 15. Packets are the input and output object of the communication medium block where incoming packets from the nodes will be at first processed by the medium logic and then fed to the appropriate nodes based on the connectivity setup of the sensor network. In this block, we have modeled a simple medium logic which at any point of time computes the input (packet) of a node as the summation of outputs (packets) of nodes connected to it.

B. Node Block

This block contains sixteen nodes as shown in figure 10. The individual node model is fully parameterized and contains mainly a timer, a random number generator and a Stateflow algorithm block. The timer is used for generating time events for the algorithm running inside the Stateflow block *wsn_algorithm*. Incoming and outgoing packets of nodes consist of *data* (contains payload) and *signal* (generates a *PKT* event).

V. Algorithm Modeling

We modeled our proposed algorithm inside the Stateflow *wsn_algorithm* block. Figure 12 shows the implementation of the *initialization* and *exchange* phases of the algorithm. The algorithm starts in the *init* state, in which it computes the next transmit (*tNextTX*) and receive (*tNextRX*) times by calling a Stateflow function (*getRandTimeStamp*). At the next *CLK* event, the application moves to the *Sleep* state from the *Init* state. In the *Sleep* state, the receiving and transmitting timestamps will be decremented by one at every occurrence of the *CLK* event. At the expiration of the transmit timestamp, the algorithm moves to the *Transmit_Mode* state. In this state, it first checks whether the medium it is free or not (by checking the value of *payloadIN*). If the medium is busy, then it just computes the next transmit time and goes back to *sleep* state again. If the medium is free, the algorithm moves to the *Transmit_Packet* state in which it computes the *payloadOUT* and then generates a packet out event (*PKT_EVENT_OUT*). While computing the *payloadOUT*, in the initialization phase, the value of the *extraInfo* is empty, while in the exchange phase it contains the value computed in the *contention* phase. The low byte of *payloadOUT* always contains the ID of the current node. After transmitting the packet, the algorithm moves to the *Sleep* state by computing the transmitting time.

Similarly when the receive timer expires, the algorithm makes a transition to *Receive_Mode* from the *Sleep* state. Note that in the simulation, the algorithm waits for 3 *CLKs* while transmitting a packet and at least 10 *CLKs* in the receiving phase. This is because our algorithm ensures that when a node turns on the radio in receiving mode, it waits for a sufficient amount of time to receive a complete packet. Our algorithm can perform adaptive listening, meaning that if a node starts to receive a packet nearly at the end of the reception interval, it completes the packet reception although the specified receive timer already expired.

In the *Receive_Mode* state, the algorithm waits for the incoming packet event (*PKT_EVENT_IN*). If a packet event occurs, it immediately makes a transition to the *Received_Packet* state. In this state, it waits to receive the complete packet. If another packet event occurs during a packet reception, it moves to the *Receive_Mode* state by increasing the collision counter. If a packet is received successfully then the *HFT* is updated by taking information from the payload of the packet. Here the algorithm also calls a Stateflow function (*checkExtraInfo*) to know the extra information attached in the payload. This segment of the payload is empty in the *initialization* phase but may contain values during the *exchange* phase. When the total waiting time expires in the *Receive_Mode*, the algorithm again moves to the *Sleep* state by computing the next receive time.

In this manner, the algorithm makes transitions between states (*Sleep*, *Transmit_mode*, *Receive_mode*, etc.) until in the *Sleep* state it notices that the timer value exceeds the specified threshold. Then it makes the transition to the *contention* phase (not shown in detail but described in section III). After finishing the computation of the *contention* phase, the algorithm moves to the *init* state to perform the *exchange* phase. This cycle continues until a cluster leader is found in the *contention* phase.

VI. Results

Figure 13 shows the convergence time with respect to the average connectivity (number of edges incident to a node). The convergence time is high for lower connectivity because a large number of iterations is required to propagate the data over a larger number of hops. It decreases with increasing connectivity up to a certain level (for a 16 node setup, the lowest convergence time is achieved when the connectivity is around 6). After this threshold, the convergence time again rises slowly due to increased collisions. To compare the algorithm with EERINA, figure 14 shows the convergence time of both algorithms with respect to the number of nodes (with average connectivity set to 6). The convergence time of the proposed algorithm is shorter compare to EERINA when the number of nodes increases. In EERINA, all nodes are connected to each other, hence increasing the number of nodes will increase the number of collisions and the convergence time. We also modeled a simple grid-based sensor network (shown in figure 15) to explore the relationship between convergence time and radio transmission power. If we increase radio transmission power, each packet will travel farther, hence the average connectivity among the nodes will increase. Figure 15 shows that increasing the radio power can decrease the convergence time up to a certain level. But after crossing this threshold, if we increase the radio power, the convergence time increases because of higher collisions of packets.

Figure 13. Convergence time (ms) vs. average connectivity between nodes

Figure 14. Convergence time (ms) with respect to increasing number of nodes

VII. Summary and Outlook

We presented an algorithm for selecting the cluster leader in a partially connected sensor network. Although the algorithm can work in a fully connected sensor network, it is best for a partially connected network. The algorithm can find a cluster leader in a robust way by using fewer packets than previous work [2], thus reducing the energy consumption of the sensor network. In this

Figure 15. Convergence time (ms) vs. distance covered by radio

paper we did not address the time synchronization that is needed at the start of the algorithm. One solution could be to broadcast a beacon packet to start the algorithm. We also modeled both EERINA and our proposed algorithm by using the framework described in [5] and show its performance in a variety of cases.

References

[1] H. Çam, S. Özdemir, P. Nair, D. Muthuavinashiappan, and H. O. Sanli, "Energy-efficient secure pattern based data aggregation for wireless sensor networks," *Computer Communications*, vol. 29, no. 4, pp. 446–455, 2006.

[2] L. Necchi, A. Bonivento, L. Lavagno, L. Vanzago, and A. Sangiovanni-Vincentelli, "Eerina: an energy efficient and reliable in-network aggregation for clustered wireless sensor networks," in *Proceedings of the Wireless Communications and Networking Conference*, 2007, pp. 3364–3369.

[3] A. Bonivento, C. Fischione, A. Sangiovanni-Vincentelli, F. Graziosi, and F. Santucci, "Seran: A semi random protocol solution for clustered wireless sensor networks," in *Proceedings of MASS, Washington D.C.*, November 2005.

[4] M. M. R. Mozumdar, F. Gregoretti, L. Lavagno, and L. Vanzago, "An algorithm for selecting the cluster leader in a partially connected sensor network," in *ICSNC '08: Proceedings of the 2008 Third International Conference on Systems and Networks Communications*, Malta, 2008, pp. 133–138.

[5] M. M. R. Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, and S. Olivieri, "A framework for modeling, simulation and automatic code generation of sensor network application," in *Proceedings of the Fifth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, SECON*, June 2008, pp. 515–522.

[6] *The MathWorks - MATLAB and Simulink for Technical Computing*. www.mathworks.com.

[7] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, 2002.

[8] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," *Proceedings of Hawaii International Conference on System Sciences*, vol. 8, p. 8020, 2000.

[9] W. Heinzelman, A. Sinha, A. Wang, and A. Chandrakasan, "Energy-scalable algorithms and protocols for wireless microsensor networks," *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, pp. 3722–3725, 2000.

[10] W. B. Heinzelman, A. P. Ch, IEEE, A. P. Chandrakasan, Member, H. Balakrishnan, , and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, vol. 1, pp. 660–670, 2002.

[11] D. Liu and M. Prabhakaran, "On randomized broadcasting and gossiping in radio networks," in *COCOON '02: Proceedings of the 8th Annual International Conference on Computing and Combinatorics*. London, UK: Springer-Verlag, 2002, pp. 340–349.

[12] J. Luo, P. T. Eugster, J.-P. Hubaux, P. Th, and E. J. pierre Hubaux, "Route driven gossip: Probabilistic reliable multicast in ad hoc networks," in *Proceedings of the INFOCOM*, 2002, pp. 2229–2239.

[13] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 2003, p. 482.

[14] J.-Y. Chen, G. Pandurangan, and D. Xu, "Robust computation of aggregates in wireless sensor networks: distributed randomized algorithms and analysis," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*. Piscataway, NJ, USA: IEEE Press, 2005, p. 46.

[15] *Crossbow Technology : Inertial Systems*. www.xbow.com.

[16] *Sentilla, Pervasive Computing Solutions*. www.sentilla.com.

[17] *Simulink - Simulation and Model-Based Design*. www.mathworks.com/products/simulink/.

[18] *Stateflow - Design and simulate state machines and control logic*. www.mathworks.com/products/stateflow/.

Mohammad Mostafizur Rahman Mozumdar received his Bachelor degree in Computer Science and Engineering from the Bangladesh University of Engineering and Technology in 2002 and Master of Science in computer science from the Technical University Aachen, Germany in 2004. In September 2005, he joined in the computer science department of the American International University Bangladesh as an Assistant Professor. Currently, he is a Ph. D. candidate at the Electronics Department of Politecnico di Torino. His research is mainly in the domain of wireless sensor network applications, including automated code generation techniques to improve platform and operating system independence.

Nan Guofang received the Bachelor degree in Department of Automation from Beijing Institute of Technology in 1998, and M.Sc. degree in School of Automation from Tianjin University in 2002, he received his Ph.D. degree in the Institute of Systems Engineering from Tianjin University in 2004. He has been with the Tianjin University in China since 2006 as an associate professor, and he is now working as postdoctoral research associate at Department of Electronics in Polytechnic Institute of Turin. His research interests include sensor networks, distributed database and related topics.

Francesco Gregoretti graduated in 1975 from Politecnico di Torino, Italy where is now a Professor in Microelectronics. From 1976 to 1977 he was an Assistant Professor at the Swiss Federal Institute of Technology in Lausanne (Switzerland) and from 1983 to 1985 Visiting Scientist at the Department of Computer Science of Carnegie Mellon University, Pittsburgh (USA). His main research interest have been in digital electronics, VLSI circuits, massively parallel multi-microprocessor systems for VLSI CAD tools and in image processing architectures. More recently his research has been focused to co-design methodologies for complex electronic systems, to methodologies for reduction of electromagnetic emissions and power consumption of processing architectures by the use of asynchronous methodologies.

Luciano Lavagno graduated magna cum laude in Electrical Engineering from Politecnico di Torino (Italy) in 1983 and received his Ph.D. in Electrical Engineering and Computer Science from the University of California at Berkeley in 1992. From 1984 to 1988 he was with CSELT Laboratories (Torino, Italy), where he was work package leader in the ESPRIT 802 CVS project. In 1988 he joined the Department of EECS at UC Berkeley, where he worked on logic synthesis and testing of synchronous and asynchronous circuits. Between 1993 and 2000 he has been the architect of the POLIS project. He then participated in the architecting and development of the VCC system-level design tool from Cadence Design Systems and was involved in the ESPRIT 25443 COSY project, which applied VCC to designs from Philips and Infineon. He has served on the technical committees of several international conferences, workshops and symposia in his field (technical Program Chair of the Design Automation Conference in 2002 and 2003). He is currently an Associate Professor at Politecnico di Torino, Italy. His research interests include embedded system design, with a special focus on wireless sensor networks, and asynchronous circuit design and testing.

Laura Vanzago holds a degree in Physics from the University of Milano, Italy. She joined STMicroelectronics in 1994 and she had been working for seven years in the Central R&D Division in several projects related to CAD development and design methodologies for IC Design. In 2001 and 2002 she was a Visiting Industrial Fellow at the University of California at Berkeley, working on System Level Design Methodologies for HW/SW platforms in the field of wireless communications. Since 2003 to 2008 she leaded several research initiatives related to the design of platforms for Wireless Sensor Networks in the Advanced System Technology Division of STMicroelectronics. Since 2008 she is responsible of the wireless sensor products program in the Subsystem Product Groups Division of STMicroelectronics in Agrate Brianza, Italy.