Energy Efficiency and Traffic Optimization through Data Aggregation in Wireless Sensor Networks

Claudio Barone[1] Fabrizio Ciarlo[2] Sebastiano Testa[3]
[1,2,3]Department of Computer Engineering

Abstract— In this paper we analyze and compare two algorithms for query based converge cast in WSNs and show that it's possible to achieve better performances in terms of energy consumption and total number of packet transmitted by means of data aggregation. We also show the impact of data aggregation on the total latency.
Key words: WSN, Traffic Optimization

I. INTRODUCTION

A sensor network generally consists of one or more sinks which expect to receive specific data streams by expressing interests or queries, or when some events occur. The sensors in the network act as sources of such data streams and push the data to the appropriate sink following multi-hop paths.

This communication pattern is known as convergecast because the traffic flows from the whole network to a single node, travelling backwards the broadcast tree.

Because of the cheap hardware, the small batteries and the requirement of unattended operation in remote or even potentially hostile locations, sensor networks are extremely energy-limited. So the classical routing algorithms, which are not energy-aware, are not well suited for such networks.

However since various sensor nodes often detect common phenomena, there is likely to be some redundancy in the data that the source nodes communicate to a sink. In-network filtering and processing techniques like data aggregation can help to conserve the scarce energy resources.

In this paper we study the energy savings and the traffic reduction at the expense of latency that can be obtained by introducing data aggregation in the described scenario. We also show how such performances are affected by factors such as the radio range of the nodes, the number of nodes, the maximum out-degree of the nodes in the broadcast tree.

II. RELATED WORKS

Data aggregation in WSNs has been a largely studied topic. In [1] the authors classify aggregation protocols in WSN as tree-based or cluster-based.

Tree-based protocols perform flat routing, i.e. all nodes in the network, except the sink(s) or base station(s), have the same capabilities and equal roles in performing the routing task. Every node is connected to the sink by a multi-hop path and the more a node is near the sink the more paths pass through it. So the routing structure is a tree with the sink as the root, source nodes as the leaves, and intermediate nodes which are allowed to perform data aggregation in order to reduce the number of transmission and save energy. SPIN [5] and Directed Diffusion [6] are the major protocols in this category.

In cluster-based protocols sensor nodes are grouped into cluster based on their proximity. Each cluster has a cluster-head which aggregates data from the members of the cluster and transmit data to the sink. The most famous of such protocol is LEACH [2].

Tree-based and cluster-based protocols are structure-based protocols: efficient aggregation is obtained through an explicit maintenance of a structure. The other extreme is to use opportunistic aggregation where packets are aggregated only if they happen to meet at a node at the same time. In such case there would not be overhead for structure construction. This is the idea behind the structure-free aggregation protocol proposed in [1].

The algorithms we propose in this paper are tree-based.

III. ALGORITHMS

We developed two different algorithms. Both of them follow the same common elements: there is a single network flow that consists of a single data sink attempting to gather information from a number of data sources; such data sources are the nodes which sense the environment. The network flow consists of two phases:

A. Phase 1(Broadcast):

The sink sends a query for data; the query diffusion produces a broadcast tree in the network.

B. Phase 2(Convergecast):

The sensor nodes having the appropriate data (a random generated number) respond with a reply packet which follows a multi-hop path towards the sink. Note that only a subset of the nodes (the leaf nodes of the tree) is the "sensor nodes".

C. Algorithm 1: Simple Relaying Convergecast

```
Algorithm1 RConvergecast
upon event <INIT> do
        nseq := 0;
upon event <RECEIVE | [QUERY, seq] from father> do
        if (seq > nseq) then
                nseq := seq;
                trigger <BCAST | [QUERY,seq]>
upon event <RECEIVE | [REPLY, src, seq] > do
        if (seq == nseq) then
                trigger <SEND | [REPLY, src, seq] to father>
```

In this algorithm intermediate nodes just relay packets from their children nodes to their parent node.

We introduced a mechanism of jittering to implement a simple collision avoidance: leaf nodes wait a random time before sending the reply packet to avoid collisions at the receiver (their parent node). In fact, leaf nodes at the same depth in the tree receive the query and are ready to answer nearly at the same time. The random time is value between 0 and D msec.

$$delay = random(0, D)$$

D. Algorithm 2: Data Aggregation Convergecast

```
Algorithm2 DAConvergecast
upon event <INIT> do
        buffer := Ø;
        nseq := 0;
upon event <RECEIVE | [QUERY, seq] from father> do
        if (seq > nseq) then
                nseq := seq;
                startTimer (timeout);
                trigger <BCAST | [QUERY,seq]>;
upon event <RECEIVE | [REPLY, n_agg, id, seq, msg] > do
        if (seq == nseq) then
                buffer := buffer U {(n_agg, id, msg)};
upon event <TIMEOUT> do
        if (buffer ≠ Ø      ) then
                (num_agg, msg) := aggregate (buffer);
                buffer := Ø;
                trigger <SEND | [REPLY, n_agg, myid,
nseq, msg]
                to father>;
upon (|buffer| == MAX_FAN-OUT) do
        (num_agg, msg) := aggregate (buffer);
        buffer := Ø;
        trigger <SEND | [REPLY, n_agg, myid, nseq, msg]
                                to father>;
```

The same considerations about collisions hold in this case. However in this algorithm also the intermediate nodes wait a random time before transmitting their packet.

The aggregate() function is arbitrarily chosen because there are different ways for aggregating data. In some cases it means grouping together all the data received in a compressed message in order to avoid multiple transmissions. However, as we said before, the aggregation has been introduced for exploiting the redundancy of data so the most common form of aggregation allows intermediate nodes to combine different values in a single value. This kind of data aggregation is also called in network filtering or data reduction: the intermediate nodes process and filter the data received before re-transmitting it in order to prevent the sink to process large amount of data and to reduce the number of packets transmitted. In this case the most common aggregation function are min, max, average.

Other forms of data aggregation are duplicates suppression and data fusion, in which row data coming from sources near each other is combined to produce more accurate data.

We chose the max function in our algorithm: each intermediate node in the tree, once received a packet from all its children nodes or once a timeout has expired, computes the maximum value among all the received values (if any) and send it to its father node with a jittering policy.

The timeout is set considering the depth of the node in the tree: the less the depth, the more is the time to wait because a node with low depth is located much more near the sink so it has a lot of successor nodes each taking its time to aggregate data. So each intermediate node, knowing its depth and given the parameters D and MAXDEPTH as the maxim depth of the tree, can compute the timeout in this way:

$$timeout = D * (MAXDEPTH - depth)$$

However this timeout introduces delay so it should be set accurately.

IV. SYSTEM MODEL

A. Tree Construction

We have not investigated about the tree construction and maintenance because it falls outside the aim of our work which is focalized only on data aggregation. Moreover the creation of an optimal data aggregation tree is generally a NP-hard problem [3]. So we hand-constructed the broadcast/convergecast tree in our network, i.e. we fed the simulator directly with a topology in which the broadcast tree is already constructed.

The Python script genTopology.py does in some sense the "dirty work": given the total number of nodes *N*, and a parameter β, it generates a random network topology and constructs the broadcast tree with the sink as the root, and each intermediate node with β as maximum out-degree. The leafs of the tree will be in the simulations the sensor nodes which have data to send to the sink. In addiction the script calculates the number of leaf nodes, the maximum depth of the tree (that will be balanced) and assigns IDs to nodes starting from $ID(sink) = 1$. The source code is available at [7].

B. Energy Model

In our work we used the simple energy model described in [1], where the radio dissipates E_{elec}= 50 nJ/bit to run the transmitter or receiver circuitry and ϵ_{amp}= 100 pJ/bit/m^2 for the transmit amplifier to achieve an acceptable SNR (See Figure 1).

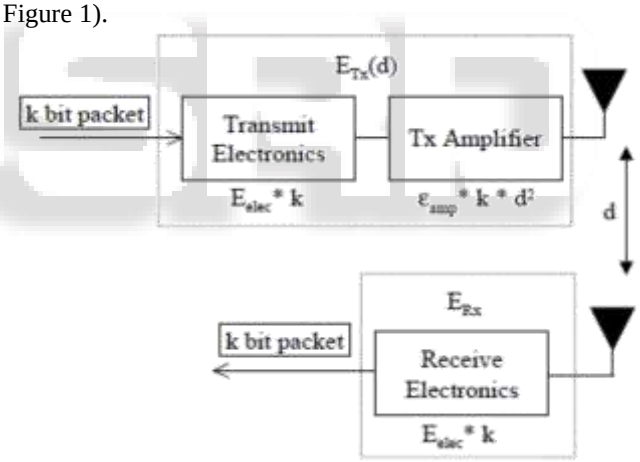

Fig. 1: Radio model

Thus, to transmit a k-bit message at distance d using such radio model, the radio expends:

$$E_{Tx}(k,d) = E_{elec} * k + \epsilon_{amp} * k * d^2$$

And to receive this message, the radio expends:

$$E_{Rx}(k,d) = E_{elec} * k$$

Then we asked to ourselves how to quantify the energy consumption for data aggregation. In [4] authors reported that energy consumption for executing 3000 instructions is equivalent to sending a bit 100 meters by radio, that in our model is exactly 1,05 µJ. Thus CPU computing one instruction spend 350 pJ.

Since our aggregation function computes the maximum, a node that has to aggregate k messages, computes k comparisons and so the cost can be approximated to k times the cost of a single CPU instruction.

$$E_{Agg}(k) = E_{CPU} * k$$

Table 1 sum up the costs of the operations described.

Operation	Energy dissipated
Transmitter/Receiver electronics (E_{elec})	50 nJ/bit
Transmit amplifier (ϵ_{amp})	100 pJ/bit/ m^2
One CPU instruction (E_{CPU})	350 pJ

Table 1: Energy consumption of basic radio/computation operation

C. *Latency Model*

According to our model the total latency of the algorithm is the total time taken for one round of data transmission which is composed of: 1) a broadcast phase and 2) a convergecast phase.

In the broadcast phase latency is the time taken for the query message transmitted by root node (sink) to reach all the nodes in the network. In this phase latency is determined by the latency of the critical path – the path along which time to deliver packets is the longest, so generally the path from the root to the leaf with maximum depth in the tree.

In the convergecast phase latency is defined as time taken from start of transmission of reply messages from leaf nodes until all the data is received by sink. In this case latency depends on the number of parallel transmissions: higher the number, lower is the latency. In fact we will see that reducing the height of the tree by augmenting the maximum out-degree of the nodes causes a decrease of latency because: I) the critical path will be shorter (broadcast phase); II) there will be more parallel transmissions (convergecast phase).

V. SIMULATIONS

We implemented the two algorithms in TinyOS and we used TOSSIM to run the algorithms and test their performance.

The experimental setup is as follows: each simulation is hand-coded in a python script and performs 10 runs of the algorithm. The output results are averaged. In each simulation we measured performances varying some parameters and fixed other parameters. The parameters are: number of nodes *N*, radio range *R*, maximum delay for jittering *D*, maximum fan-out β of the broadcast tree. The metrics are: success rate, total number of transmissions, total energy consumption, latency.

The source code to run the simulations is available at [7].

VI. EXPERIMENTAL RESULTS

We first measured the success rate, i.e. the ratio of the reply packets which are delivered at the sink. Figure 2 shows the success rate for the first algorithm varying the number of nodes (from 25 to 200) in the network, for different values of the parameters *D* (1, 5, 10, 20 sec), and having fixed *R* (20 m). Figure 3 shows the success rate for the second algorithm with the same parameters.

For the first algorithm the delivery rate decreases with the dimension of the network. That's because a larger number of nodes produces a larger number of collisions. With *D* =10 sec the algorithm performs well also in a network of 200 nodes (82% of success rate) so we used *D* =10 as reference parameter for the following simulations.

The second algorithm performs well even in large networks (success rate of 87% in the worst case). We will see that the parameter D is crucial in the total latency.

Secondly we investigated the total traffic generated by the two algorithms. Figure 4 compares the total number of transmissions varying the number of nodes (from 25 to 200) in the network and varying β (from 4 to 10). R and D are fixed respectively at 20 m and 10 sec. It's clear that the second algorithm generates much less traffic due to data aggregation. We can see that for the first algorithm the traffic decreases with the increasing maximum fan-out: that's because each intermediate node has more children and so

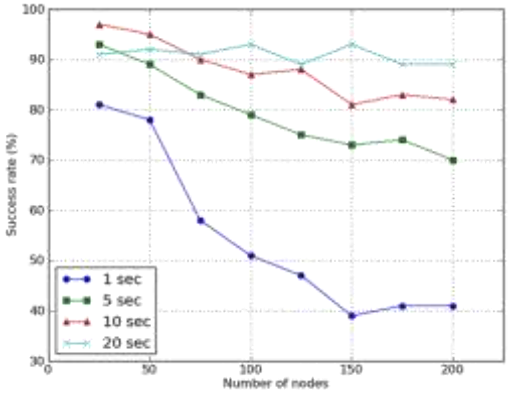

Fig. 2: Success rate for algorithm1. Radio range and max fan-out fixed respectively at 20 m and 4.

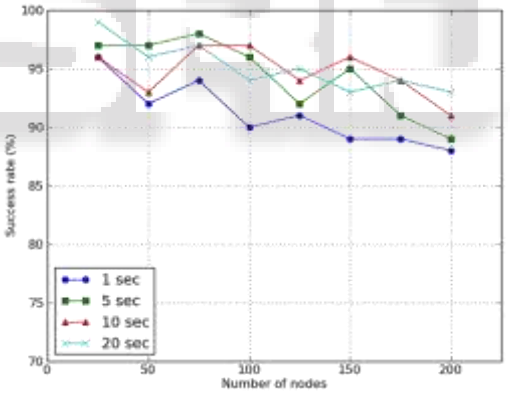

Fig. 3: Success rate for algorithm2. Radio range and max fan-out fixed respectively at 20 m and 4.

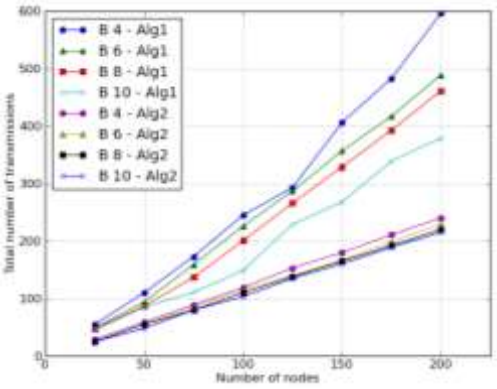

Fig. 4: Total number of transmissions. Radio range and max-delay are fixed respectively at 20 m and 10 sec.

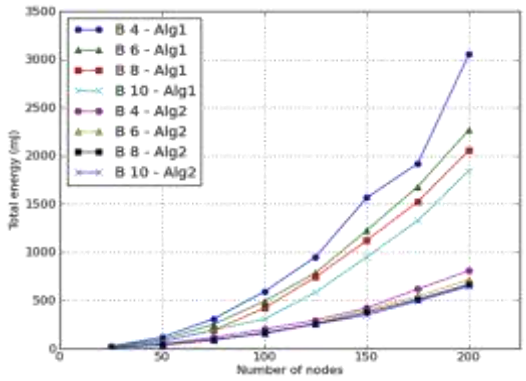

Fig. 5: Total energy consumption in mJ. Radio range and max-delay are fixed respectively at 20 m and 10 sec.

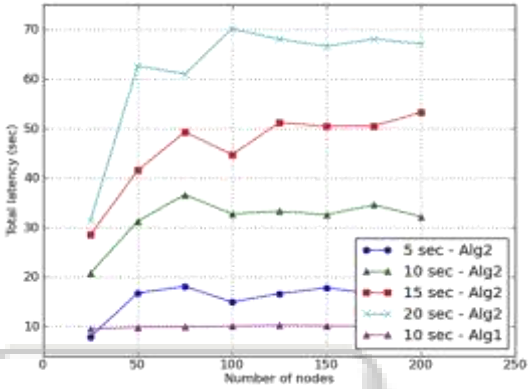

Fig. 6: Total latency in msec. Radio range and max fan-out are fixed respectively at 20 m and 6.

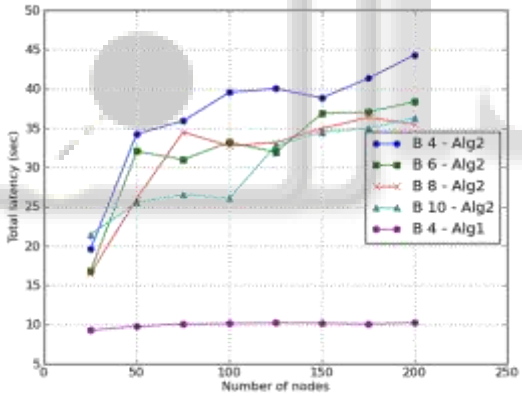

Fig. 7: Total latency in msec. Radio range and max delay are fixed respectively at 20 m and 10 sec.

More collisions may happen. Many packets are lost so the success rate decreases too. For the second algorithm the total number of transmissions is constant with the increasing fan-out. Only a little number of packets are lost due to collision because the number of transmissions is highly reduced.

According to our energy model the number of transmissions should directly influence the energy consumption. Figure 5 confirms what we expected: it compares the total energy consumption of the two algorithms running with the same parameters of the previous simulation (traffic). We can see that it's possible to achieve huge energy savings with data aggregation thanks to the reduction in packet transmissions. As an example in a network of 200 nodes with 4 as maximum fan-out of the tree we can obtain a saving of 70% with algorithm 2.

Finally we tested the latency. As we already mentioned data aggregation introduces delay due to the time that each node has to wait until it collects all (or almost all) the messages from its children, before it can transmit the aggregate data to its parent node. The main factors that introduce delay are: 1) the maximum depth of the tree: higher the number of parallel transmissions, lower is the latency; 2) the maximum delay for jittering.

Figure 6 shows the total latency of the second algorithm varying the number of nodes (from 25 to 200) in the network, for increasing values of the parameters D (5, 10, 15, 20 sec) and compares it to the latency of the first algorithm simulated with D = 10 sec (we have seen before that the fist algorithm doesn't perform well with D > 10 sec). As an example in a network of 100 nodes and for D = 10 sec, algorithm 2 is 3 times slower than algorithm 1. Figure 7 instead shows the total latency with the same setup as before but varying β (from 4 to 10) and fixed D at 10 sec.

Comparing the two figures it comes clear that the parameter D influences latency more than the maximum fan-out. So the time of the data aggregation algorithm is mainly governed by the single delays introduced by each node and only secondly by the height of the broadcast tree.

VII. Conclusions

In this paper we have proposed two algorithms for query based convergecast in WSNs. The first implements a relaying convergecast and we have shown that there is high energy dissipation due to many transmissions. Moreover the high traffic in the network leads to an increase in packet collisions which degrades the reliability of the algorithm.

The second algorithm implements a convergecast with data aggregation. Simulating such algorithm we obtained interesting results: the fewer total transmissions lower the energy consumptions because in sensor networks the tx/rx operations dominates all the other operations in terms of energy dissipation.

However, the energy saving trades-off with higher delay due to data aggregation: in fact every intermediate node must wait for all (or almost) the packets from its child to be received before aggregating the data and send it to its father.

In sensor networks latency is an issue, and in some scenarios it may be as relevant as energy saving. Therefore a possible future work would address the problem of the high latency enhancing our second algorithm.

The second major drawback of our work is that both the algorithms need an already constructed broadcast tree in the network to work. So we fed the simulator with a priori constructed broadcast tree topologies.

We might address also this aspect in a future work.

References

[1] K.-W. Fan, S. Liu, and P. Sinha. Structure-free Data Aggregation in Sensor Networks. Published in Journal IEEE Transactions on Mobile Computing, vol. 6 Issue 8. August 2007.

[2] W. R. Heinzelman, A. Chandrakasan and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. Published

in the Proceedings of the Hawaii International Conference on System Science January 2000.
[3] Krishnamachari, D. Estrin and S. Wicker. The Impact of Data Aggregation in Wireless Sensor Networks. Published in the Proceedingsof the 22nd International Conference of Distributed Computer Systems 2002.
[4] G. J. Pottie and W. J. Kaiser. Wireless Integrated Network Sensors. Communications of the ACM, May 2000.
[5] W. R. Heinzelman, J. Kulik and H. Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. Published in the Proceedings of the 5 Annual CM/IEEE International Conference of Mobile Computing and Networking, August 1999.
[6] Intanagonwiwat, R. Govindan and D. Estrin. Directed Diffusion: A Scalable and Robust Paradigm for Sensor Networks. Published in the Proceedings of the 6 Annual CM/IEEE International Conference of Mobile Computing and Networking, August 2000.
[7] http://code.google.com/p/data-aggregation-wsn/downloads/list

