Application-Driven design to extend WSN lifetime

Bruno F. Marques
Escola Superior de Tecnologia e Gestão
Instituto Superior Politécnico de Viseu, Portugal
bmarq@estv.ipv.pt

Manuel P. Ricardo
INESC Porto
Faculdade de Engenharia da
Universidade do Porto, Portugal
mricardo@inescporto.pt

***Abstract*—The lifetime of a WSN depends on the energy of the nodes. As soon as nodes run out of energy, they get disconnected from the WSN. This paper proposes an *Application Driven* solution that increases the WSN lifetime by limiting the routing and forwarding functions of the network mainly to nodes running the same applications. The solution is evaluated against AODV, and the results obtained show gains of about 30%.**

***Index Terms*—Wireless Sensor Network (WSN); 6LoWPAN; IEEE 802.15.4; Low power routing protocols; Application-Driven WSN.**

I. Introduction

The growth of wireless networks has resulted in part from requirements for connecting people and advances in radio technologies. Wireless Personal Networks (WPANs) are an example of these networks, and Wireless Sensors Networks (WSN) [1] are an example of WPANs. Sensing nodes may be used to measure physical data for multiple applications [2]. Millions of sensing nodes are expected to be deployed in the coming years and they are likely to be associated in networks, interconnected or not to the Internet. Sensor nodes are known to be energy constrained, thereby routing strategies capable of finding energy-efficient paths are demanded.

Energy-efficiency has to be implemented also in large WSN which tend to be self-managed and self-configured. The behavior of these large systems need to be information-aware, where applications play the key role of waking up and enabling parts of the system. Our work is focused on *Application-Driven WSN* (ADWSN). We define an ADWSN as a cross-layer solution aimed to help minimizing the energy consumed by a network of sensors executing a set of applications. We assume that sensors form a large network and that a sensor is enabled to run one or more applications. We also assume that the applications and the nodes to which they are associated, are not always active, alternating between on and off states. By jointly considering the neighbors of each node, the applications each node runs, and the forwarding capabilities of a node, we developed a communications solution which enables the data of every application and node to be transferred while keeping the overall energy consumed low.

In our solution (AD6LoWER) we assume that every node can participate in route discovery and packet forwarding. However, the nodes forwarding a given type of data, will be primarily selected from the set of nodes running the same application to which the data is associated. For that purpose, each application is identified by a *tag*, which is known by the nodes running that application.

This paper provides three contributions. The first contribution is the cross-layer solution which uses the application layer tag in order to select the set of nodes leading to minimum energy consumption. The second contribution is a syn-

chronization mechanism defined at the application layer, which synchronizes nodes with respect to their application life cycles, enabling nodes to wake up and going asleep in synchronism; this mechanism uses an application layer beacon generated by sink nodes, used to inform the other nodes about the time they should wake up in the next time. The third contribution is an energy model which estimates per packet energy consumption; this model allows to compute energy consumed in the transmission and reception of packets, assuming that unicast packets are acknowledged at the MAC layer, while broadcast packets are not.

The paper is organized in eight sections. Sec. II characterizes the problem addressed in this work. Sec. III presents related work. Sec. IV characterizes the applications selected for our study. Sec. V describes our solution in detail. Sec. VI describes the methodology adopted for validating our solution. Finally, Sec. VIII draws the conclusions, and presents the future work.

II. Problem characterization

WSN, being constituted by sensor nodes which are known to be energy constrained, depend of their nodes lifetime. Thus, in order to reduce nodes energy consumption, routing strategies capable of finding energy-efficient paths are demanded. We assume that routing protocols must find routes in which the nodes may be kept asleep the maximum amount of time they can. For that purpose, we introduce the concept of application duty cycle time, characterized by states wake and sleep, and their times. We also assume that WSN forms a mesh network, and that nodes may run multiple applications. Two main questions arise: 1) *How to use mainly the nodes running the application associated to the data being transferred, so that the nodes associated with other applications can continue sleeping?* 2) *How to synchronize nodes so that they can wake and going asleep simultaneously?*

III. Related work

We classified related work in three areas: node energy consumption, routing protocols, and cross-layering. Operating systems for wireless sensor networks, such as Contiki [3] reduce energy consumption by powering off the microcontroller and hardware components when they are not used. The on-line energy estimation mechanism [4] defines the total energy consumption E (J) as

$$E = (I_m \times t_m + I_l \times t_l + I_t \times t_t + I_r \times t_r + \sum_{i=1}^{n} I_{c_i} \times t_{c_i}) \times V \quad (1)$$

being I_m the current consumed by the microprocessor in the time t_m during in which the microprocessor is running, I_l and t_l the current and time when the microprocessor is in low power mode, I_t and t_t the current and the time the communication device is in transmit mode, I_r and t_r the current and the time the communication device is in receive mode, I_{c_i} and t_{c_i} the energy consumed by other components (eg. LEDs, ADCs, DACs), and V the sensor supply voltage.

uAODV [5] is a simplified AODV [6] routing protocol for WPANs. With AODV, *HELLO* messages are used to get information about neighbors. When a source node needs to find a route to a destination, it starts a route discovery process, based on *broadcast*. The source node generates a route request (*RREQ*) packet which intermediate nodes forward until the packet reaches a node that has a route to the destination, or the destination node. Routes to the destination are updated upon reception of a route reply (*RREP*) packet, originated either by the destination itself or by any other intermediate node that has a route to the destination. AODV also uses sequence numbers to prevent the creation of loops. Expiry timers are used to keep the route entries updated. Since nodes reply to the first arriving *RREQ* packet, AODV favors the least congested path instead of the shortest path [7].

According to [8], cross-layer optimization in WSN has been addressed by multiple studies in different scenarios. The main idea is to design communication layers such that they can share and react to information from other layers. In recent years, cross-layer design was used to increase the efficiency of WSN communication systems. MiLAN [9], is an example of cross-layering that receives a description of application requirements, monitors network conditions, and optimizes sensor and network configurations to maximize application lifetime.

IV. Applications Characterization

Two different applications are used in our experiments: *environment monitoring* and *structural monitoring*. These applications and their underlying network are characterized by the following aspects: static, organized, pre-planned, no mobility, 16 nodes deployed in a square lattice topology, all nodes are battery-powered. We also consider multi-hop communication. The traffic pattern is point-to-multipoint when data is queried, and point-to-point when queries are replied.

The *Environmental Monitoring* Application (Application A) aims to monitor gas emissions, and acoustic noise in an urban tunnel. The queries are made regularly, at a rate of one query per hour. The *Structural Monitoring* Application (Application B) aims at monitoring vibration levels on facilities for safety purposes (e.g. tunnels). Nodes are static and data is obtained regularly.

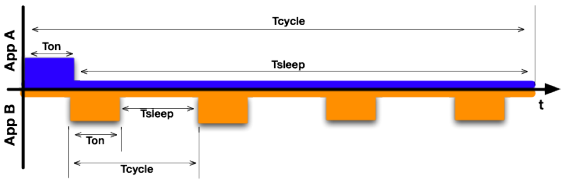

Fig. 1. Applications life cycle within the AD6LoWER solution

Application A has a duty cycle of one hour. Every node running this application wakes every hour remaining awake during 15 seconds for activity including asking for sensed data if it is a sink, or replying to asked data if it is a sensor node. Application B has a duty cycle of 15 minutes. The nodes also wake during 15 seconds to sense data and to communicate. As shown in Figure 1, the period of application A is 4 times the period of application B, thus a node of Application B wakes up 4 times while a node of application A wakes 1 time. All nodes are awake when data is queried and replied, and sleeping when there is no activity.

V. Proposed Solution

The ADWSN paradigm assumes that each application defines its own network and set of nodes so that the exchanged information can be confined to the set of nodes associated to the application. The nodes are *"tagged"* in order to mark them with respect to the applications they run. Our routing scheme tries to insure that data of an application is relayed mainly by the nodes running that application. When the sink node queries the other nodes running the same application, routing paths must be found. The routing scheme will use mainly the nodes *"tagged"* with that application; the nodes not associated to this application will not participate in routing process, in a first attempt. The node is put asleep when there is no activity related to its applications. The wake up mechanism is based on the applications time cycle, and in an application layer beacon which is sent by the sink nodes. When a node receives a query packet it knows exactly when it must wake up on the next period.

In our solution the nodes alternate between the wake and sleep states. The amount of time of each phase is determined by the applications duty cycle. When a node is awake it performs activities including waiting for a sink query and forwarding packets to neighbors. When the wake up time expires, the node switches to the sleep state, waking up again by the time indicated by the application beacon.

```
foreach (query to be sent) do
    if (no route available) then
        RREQ.oN ⟵ 0;
        send RREQ(TAG, oN) packet;
        activate PATH_DISCOVERY timer;
    end
end
foreach (PATH_DISCOVERY timeout) do
    if (RREQ.TAG != 0) then
        if (RREQ.oN != 1) then
            RREQ.oN ⟵ 1;
        else
            RREQ.TAG ⟵ 0;
        end
        send RREQ(TAG, oN) packet;
        activate PATH_DISCOVERY timer;
    else
        sleep and retry next wakeup time;
    end
end
foreach (RREP packet received) do
    if (first packet) then
        update Neighbors Table;
        install new route entry;
    end
end
```

Algorithm 1: *Pseudocode of the proposed solution for the originator node*

Our solution follows the AODV routing scheme, except that in a first approach only the nodes tagged with the same application will be used. *HELLO* messages are sent regularly to get information about neighbors and the applications they run, and kept in a *Neighbors Table*. This information will further be used by the routing scheme in order to choose, if necessary, intermediate nodes which do not run the application but whose forwarding function is also demanded. For this purpose, route request (*RREQ*) and route reply (*RREP*) packets have a *oN* (*OtherNode*) flag. When a node needs to send a packet, it must first find a route to the destination. It then generates a *RREQ* packet and waits for the correspondent *RREP* packet. If the $PATH\_DISCOVERY$ time, which is the maximum amount of time to wait for the reception of a *RREP* packet, times out, and no *RREP* packet has been received, the sending node sets the *oN* flag and sends again the *RREQ* packet. If a second $PATH\_DISCOVERY$ timeout occurs, the sender will set the *tag* to "0" and send a third *RREQ* packet. Intermediate nodes, upon receiving the *RREQ* packet, will extract cross-layer information about the application, and the time the nodes running that application will wakeup next time (*"tag","Application beacon"*). The nodes running the same application will forward the *RREQ* packet until it reaches its destination or it reaches a node which has a route to the destination. If the *oN* flag is set, the nodes not running the same application but having neighbors running the same application, will forward the *RREQ* packet to neighbors, and reconfigure themselves as running that application. If the *tag* is set to "0", any node, running or not this application, will forward the *RREQ* packet, and the nodes in the selected path will reconfigure themselves as running this application. *RREP* packets are sent back to the originator using reverse paths and updating routes in intermediate. The pseudocode of the proposed solution for a originator, and intermediate and destination nodes is shown respectively in Algorithms 1 and 2.

In the normal AODV solution, when a sink node issues a query it *"broadcasts"* the entire WSN, and when nodes reply, shortest paths are used. In our case, mainly the nodes running the application associated to the query will participate in packet forwarding.

```
foreach (HELLO packet received) do
    process packet;
    update Entry (Neighbor,TAG) in Neighbors Table;
end
foreach (RREQ packet received) do
    process packet;
    if ((node has same TAG) or (RREQ.oN = 1 and
    node has neighbors with same TAG) or
    (RREQ.TAG = 0)) then
        if (route found) then
            send back RREP packet;
        else
            forward RREQ packet to neighbors;
        end
    end
    if ((RREQ.oN = 1 and node has neighbors with
    same TAG) then
        reconfigure as running the application;
    end
end
foreach (RREP packet received) do
    process packet;
    if (first RREP packet) then
        update Neighbors Table;
        install new route entry;
        send back RREP packet;
        if (RREP.TAG = 0) then
            reconfigure as running the application;
        end
    end
end
```

Algorithm 2: *Pseudocode of the proposed solution for intermediate and destination nodes*

VI. Solution evaluation

In order to evaluate the proposed solution, a square lattice of 4x4 nodes was chosen. The nodes are distributed as shown in Figure 2, where the four scenarios evaluated are also shown. In those scenarios the network supports two applications, A and B, each running in eight nodes. Each node runs a single application. Sink nodes placements where chosen in order to allow long routing paths, since long paths consume more energy. There is a cost associated to the transmission and reception of packets. In scenario 1, the nodes running application A were selected so that a long path could be obtained. In the scenario 2, both applications have the same node distribution; in these scenarios we want to investigate the influence of the application duty cycle in energy consumption. Scenarios 3 and 4 are used

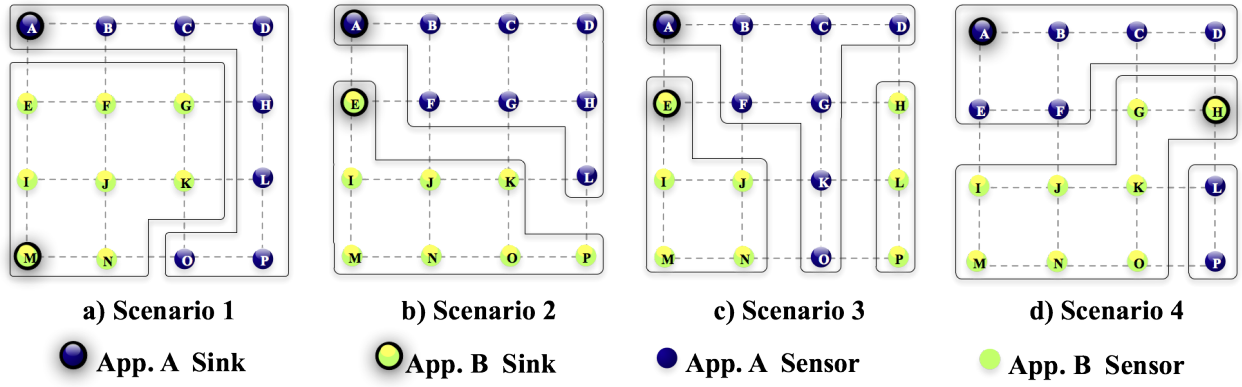

Fig. 2. Nodes deployment in different square lattice mesh topologies

to investigate situations where at least one node from other application is required to relay data.

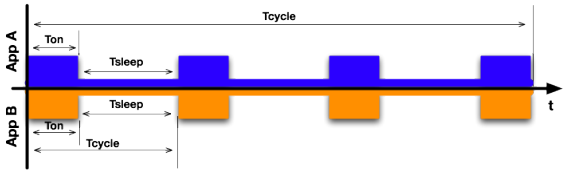

Fig. 3. Applications life cycle within the AODV solution

Since our solution constrains the paths to the nodes associated to the application, a node needs to be waked up only when its applications run and not for generic routing and forwarding purposes. In contrast, AODV was used as shown in Figure 3; in this case despite the applications having different periods, nodes would have to wake up every 15 minutes in order to process routing messages.

In our solution, when the sink node issues a query, it is *"multicasted"* only to the nodes associated to the application and not the entire WSN. When the application nodes reply, only the nodes running the application will reply to the data query and forward layer 3 packets. So, with the our solution, the routing paths are chosen not only according to the minimum hop, but also considering the constraint of a node belonging to an application. At least in a first attempt.

Figure 2.c shows a particular node deployment, where nodes H, L, and P running application B are unable to receive sink queries in a first attempt because they are isolated. In this case, the nodes K and O will be selected, in a second attempt, to participate in the routing and forwarding process of application B. Figure 2.d shows a second case of node deployment where some of the nodes of application A (nodes L and P) are also isolated.

As described before, the nodes of application A are awake for 15s every hour while nodes of application B are wake for 15s every 15 minutes. During a period a node may be sending or receiving a packet, idle, or sleeping. The energy consumed when a node is sending a packet is computed as the time required to send the packet, times the current consumption by the node in transmitting mode (times de voltage); the energy consumed when a node is receiving a packet is computed as the time required to receive the packet, times the current consumption of the node in receiving mode; the energy consumed when a node is idle is computed as the amount of time the node is idle, times the current consumption in idle mode; the energy consumed when a node is sleeping is computed as the amount of time the node is sleeping, times the current consumption of the node in sleep mode.

Considering that a node is implemented as a CrossBow TelosB [10] sensor hardware, and the consumption of its CPU and RF transceiver, the total energy consumed can be described as follows:

$$E = E_{on} + E_{TX_{Bcast}} + E_{RX_{Bcast}} + E_{TX_{Ucast}} + E_{RX_{Ucast}} + E_{Idle} + E_{Sleep} \quad (2)$$

where E_{on} is the energy consumed during the time the node waked, $E_{TX_{Bcast}}$ is the energy consumed when sending broadcast packets, $E_{RX_{Bcast}}$ is the energy consumed when receiving broadcasted packets, $E_{TX_{Ucast}}$ is the energy consumed

when sending unicast packets, $E_{RX_{Ucast}}$ is the energy consumed when receiving unicast packets, E_{Idle} is the total energy consumed when the node is in the idle state (the state where a node has its radio on and waiting to send or to receive a data packet), and E_{Sleep} is the total energy consumed when the node is sleeping.

	Nominal
Current in Transmit (0 dBm) mode (mA)	19.5
Current in Receive mode (mA)	21.8
Current in MCU on, radio off (mA)	1.8
Current in MCU on, radio on - idle mode (μA)	365
Current in Sleep mode (μA)	5.1
Power supply (V)	3.6
Transmit bit rate (kbit/s)	250
Transmit symbol rate (ksymbol/s)	62.5

TABLE I
TELOSB SPECIFICATION

The energy consumed by a node in idle state is computed as $E_{Idle} = I_{Idle}(A) \times V \times t_{Idle}(s)$, considering $I_{Idle} = 365\mu$A, $V = 3.6$ V , and t_{Idle} the time the node is idle, which depends on the scenario. The energy consumed by a sleeping node is computed as $E_{Sleep} = I_{Sleep}(A) \times V \times t_{Sleep}(s)$, considering $I_{Sleep} = 5.1\mu$A, $V = 3.6$ V, and t_{Sleep} the total time the node is sleeping. The values are extracted from Table I, extracted from [11].

For theoretical evaluation purposes, we assume the simplest case of having no collisions and all the packets being correctly received. We also assume that a unicast packet is acknowledged at the MAC Layer, whilst a broadcast packet is not. The energy consumed per packet considering the information of Table I, and the IEEE 802.15.4 specification [12], can be computed as follows:

- **Transmission of broadcast packet**: non-beacon enabled IEEE 802.15.4 networks use an unslotted CSMA-CA channel access mechanism [13], [14]. We assume that each time a device needs to transmit, it waits for a random number of unit backoff periods in the range $\{0, 2^{BE} - 1\}$ before performing the Clear Channel Assessment *CCA*. If the channel is found to be idle, the device transmits. If the channel is found to be busy, the device waits another random period before trying to access the channel again. Assuming the channel is found to be free, and also assuming that the backoff exponent BE is set to *macMinBE* which has the default value of 3, and the access time can be computed as

$$\begin{aligned} T_{CA} &= InitialBackoffPeriod + CCA \\ &= (2^3 - 1) \times aUnitBackoffPeriod \times CCA \\ &= 7 \times 320\mu s + 128\mu s \\ &= 2.37ms \end{aligned} \quad (3)$$

The *CCA* detection time is defined as 8 symbol periods. *aUnitBackoffPeriod* is defined as 20 symbol periods, corresponding 1 symbol to 16 μs.

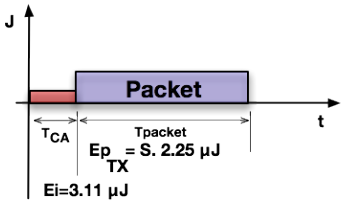

Fig. 4. Energy consumed by a node when transmitting a broadcast packet of size S

As shown in Figure 4, the energy consumed is computed as $E_{TX_{Bcast}} = E_i + E_{P_{TX}}$; E_i is the energy consumed during the *Channel Access* period (CA) which is $T_{CA} \times P_{Idle}$; P_{Idle} is the power consumed by the node in the idle mode which is 1.31mW. $E_{P_{TX}}$ is the energy consumed during the time required to send the packet of size *S* (in octets). $E_{P_{TX}} = S \times T_{octet} \times P_{TX}$; T_{octet} is the time required to send one octet, which is 32μs, and P_{TX} is the power consumed in the transmission of the same octet, which is 70.2mW.

- **Reception of broadcast packet**: when a node receives a broadcast packet of size *S*, the energy consumed corresponds to the energy consumed during the time required to receive the packet of size *S* (in octets) which is $E_{RX_{Bcast}} = E_{P_{RX}} = S \times T_{octet} \times P_{RX}$; T_{octet} is the time required to receive one

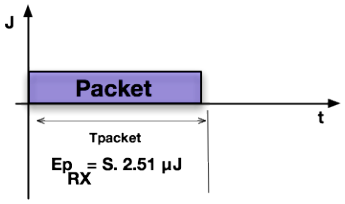

Fig. 5. Energy consumed by a node when receiving a broadcast packet of size S

octet, which has the same value as the time required to send one octet. $P_{RX} = 78.5$ mW is the power consumed by receiving the same octet, as shown in Figure 5.

- **Transmission of unicast packet**: When a node sends a unicast packet, the amount of energy consumed is computed as the energy required to transmit the packet plus the energy consumed during the reception of the acknowledge frame. The transmission of an acknowledgment frame in a non-beacon enabled network commences *aTurnaroundTime* symbols after the reception of the data frame, where *aTurnaroundTime* is equal to 192μs. This gives the device enough time to switch between transmit and receive mode. As shown in Figure 6, the total energy consumed is $E_{TX_{Ucast}} = E_i + E_{P_{TX}} + E_{T_{Ack}} + E_{M_{Ack}}$; E_i is the energy consumed during the *Channel Access* period; $E_{P_{TX}}$ is the energy consumed during the time required to transmit the packet of size S; $E_{T_{Ack}}$ is the energy consumed while waiting for the reception of the acknowledgment, which is 0.252μJ, and corresponds to *aTurnaroundTime* times the power consumed in the idle mode which is 1.31 mW; $E_{M_{Ack}}$ is the energy consumed during the time required to receive the complete MAC acknowledgment frame which has a size of 11 bytes, and corresponds to 27.6μJ.

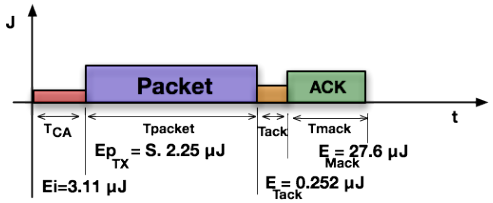

Fig. 6. Energy consumed by a node when transmitting a unicast packet of size S

- **Reception of unicast packet**: the energy consumed to receive a unicast packet of size S is $E_{RX_{Ucast}} = E_{P_{RX}} + E_{T_{Ack}} + E_{M_{Ack}}$. $E_{P_{RX}}$ is the energy consumed during the time needed to receive the packet of size S; $E_{T_{Ack}}$ is the energy consumed during the T_{Ack} time to wait before sending the acknowledge packet (this value is the same as the waiting time before receiving the acknowledge packet); $E_{M_{Ack}}$ is the energy consumed during the time of the transmission of the *acknowledge MAC frame* - T_{Mack} times the power consumed in the transmission mode which is 70.2mW, corresponding to 24.7μJ, (see Figure 7).

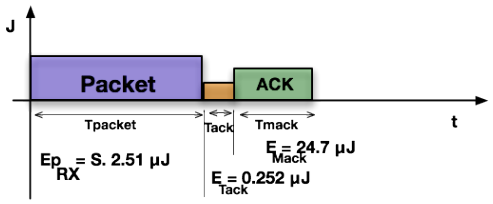

Fig. 7. Energy consumed by a node when receiving a unicast packet of size S

VII. Results and Discussion

We characterized the energy consumed by the nodes running simultaneously both applications, the number of broadcast and unicast packets, and the time the nodes are wake, sleeping, and in idle mode. We compared our solution with the generic on-demand routing solution. The analysis was performed based on packets of 127 octets (application data size of 81 octets, plus the 2.4 GHz IEEE 802.15.4 MAC layer header, and PHY layer header size of 46 octets) as there is a energy cost associated to packet sizes: the size chosen reflects worst cases in the analysis performed. Also, MAC layer collisions were not considered; for simplicity, we assumed that all packets were sent and received with no errors and no retransmissions. The calculus was made using a C program that implements both solutions.

Figure 8 shows the distribution of the broadcast packets generated by each node in the four scenarios considered. As shown, the number of broadcast packets sent using our solution is lower than the number of packets sent using the AODV solution. In scenarios 3 and 4, the AD6LoWER (our) solution needs node K and node H, respectively, to forward packets. Node K and node H forward 5 broadcast packets, respectively in scenario 3 and scenario 4. The total number of broadcast packets sent in each scenario corresponds to the packets sent by the sink nodes when they issue queries. In our solution, only the nodes belonging to the application forward the packets, thus network flooding is bounded

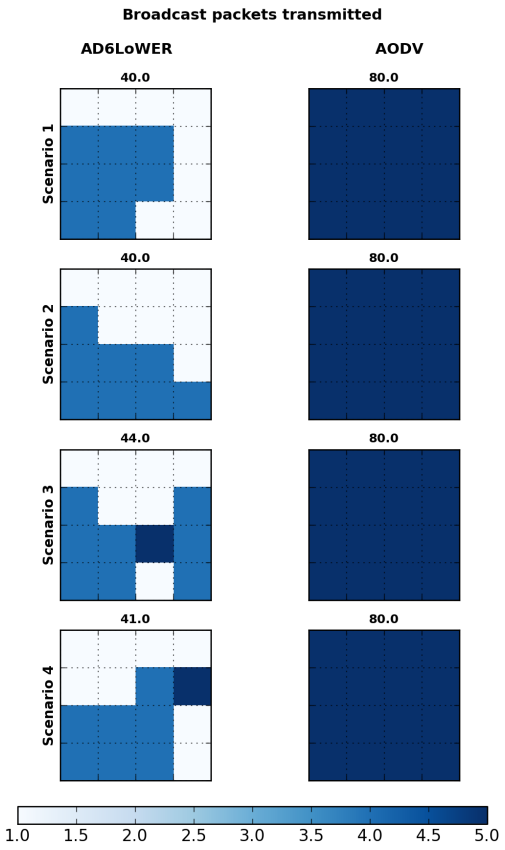

Fig. 8. Total number of broadcast packets generated in each scenario for the AD6LoWER and the AODV solutions

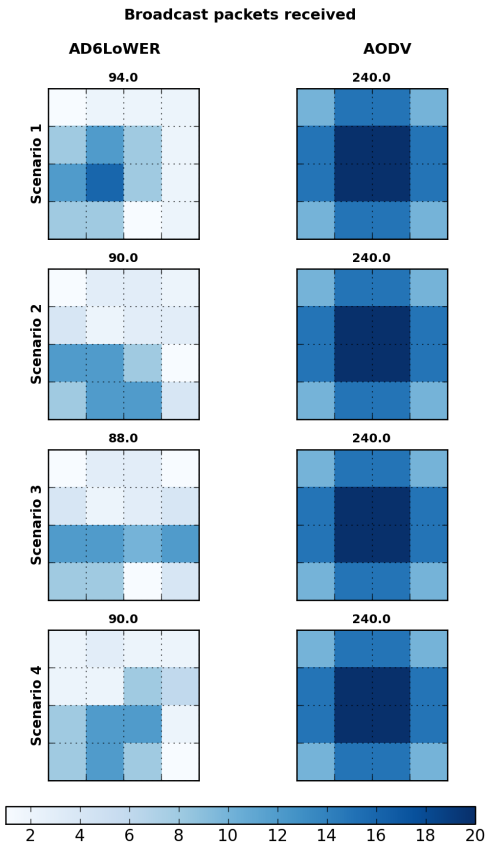

Fig. 9. Total number of broadcast packets received in each scenario for the AD6LoWER and the AODV solutions

to the nodes of the application. In the case of the AODV solution, when a sink issues a query the packets are broadcasted to the network. Since there are two applications running, the network is broadcasted twice. As an example look at scenario 1. Our solution sends 40 packets, while the AODV solution sends 80 packets. Figure 9 also shows the distribution of broadcast packets received by each node. As shown, the number of broadcast packets received with our solution is lower than the number of packets received using the AODV solution. For the AODV solution, in the scenarios considered, 4 nodes are used more often (nodes F, G, J, and K). They are placed in the middle of the topology and receive more packets than the others, since they have more neighbors. With the AD6LoWER solution, the distribution of the nodes inside the topology has influence. Looking at scenario 1, node J receives more packets than the others nodes (16 packets). This node runs application B, thus it is used 4 times per hour. It also receives more packets since has more neighbors. In the case of scenario 4, there are 3 nodes (J, K, and N) which receive 12 packets, because node K relays packets from nodes running application A.

When analyzing Figures 8 and 9, we also conclude that, with the AD6LoWER solution, application B generates more broadcast packets than application A, since queries are four fold. With AODV, all nodes are waked in order to receive and to send broadcast packets, so the node distribution does not influence the number of broadcast packets. With AODV, queries are issued 5 times per hour (one from application A, and 4 from application B). Figures 8 and 9 also show the broadcast *hotspots*, that is, the nodes that send, and receive more broadcast packets.

Figure 10 shows the distribution of unicast packets sent by each node in the four scenarios considered. We can verify that the total number of unicast packets sent in each scenario is higher

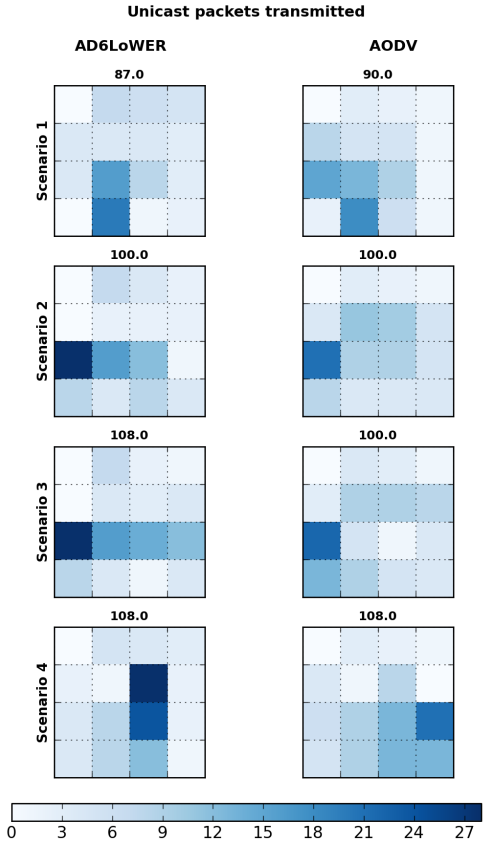

Fig. 10. Total number of unicast packets transmitted in each scenario for the AD6LoWER and the AODV solutions

for AD6LoWER than for AODV. AD6LoWER needs to send more unicast packets in scenarios 1 and 3. In these scenarios nodes need to forward more packets in order to reply to a sink query. In scenario 2 and 3, node I sends more packets (28). Since AD6LoWER uses mainly the nodes belonging to the applications, node I is more often used. In the scenarios considered, the paths selected by the AD6LoWER solution are longer than those selected by the AODV solution, that makes no distinction between nodes. In scenario 4, nodes G and K are also used to forward packets from nodes running other application. In this situation, node G transmits 28 packets, while node K transmits 24 packets. Since for the AODV solution all nodes must be awake, in scenario 4, node L is the node with more activity, and it transmits 21 packets.

The number of the unicast packets received in our solution is smaller than in AODV, as shown in Figure 11). In case of scenario 1 the total number of unicast packets received for the AD6LoWER solution is 209, while for the AODV solution is 298.

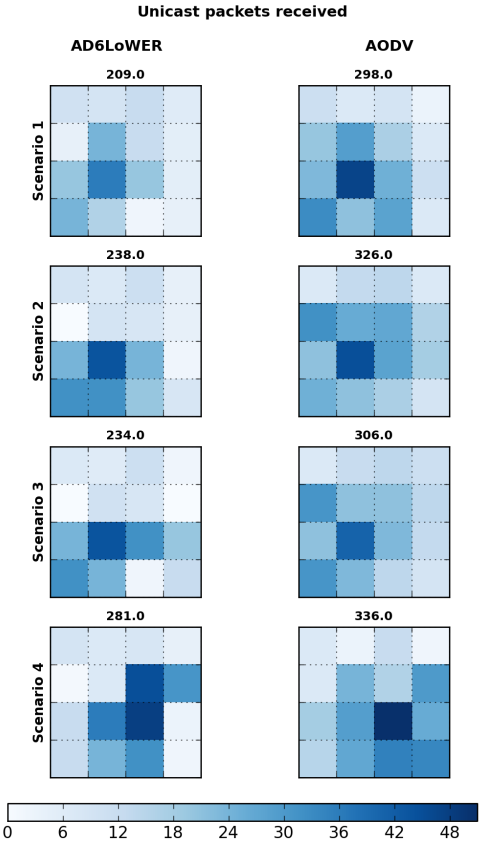

Fig. 11. Total number of unicast packets received in each scenario for the AD6LoWER and the AODV solutions

Figure 12 shows the total energy consumed by each node. Our solution always consumes less energy. The total of energy consumed is computed as the sum of the energies consumed by individual nodes. The energy consumed by each node is given by Eq. 2. Figure 13 shows the total of energy gains using de AD6LoWER solution in each scenario. The gain is computed as:

$$Gain = \frac{E_{AODV} - E_{AD6LoWER}}{E_{AODV}} \times 100 \quad (4)$$

where E_{AODV} is the total of energy consumed by the nodes for the AODV solution, and $E_{AD6LoWER}$ is the total of energy consumed by the nodes for our solution. In the selected scenarios the mean energy consumed for the AD6LoWER solution is *5.96J*, and for the AODV solution is *8.76J*. The mean gain, considering the 4 scenarios, is *31.9%*. Concerning to the time

	Scenario 1				Scenario 2				Scenario 3				Scenario 4			
	AD6LoWER		AODV		AD6LoWER		AODV		AD6LoWER		AODV		AD6LoWER		AODV	
	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx
BCast (packets)	40	94	80	240	40	90	80	240	44	88	80	240	41	90	80	240
UCast (packets)	87	209	90	298	100	238	100	326	108	234	100	306	108	281	108	336
Time waked (s)	600		960		600		960		600		960		600		960	
Time sleeping (s)	57,000		56,640		57,000		56,640		57,000		56,640		57,000		56,640	
Time idle (s)	597.8		955.8		597.0		955.5		641.9		955.7		596.7		955.4	
Energy consumed (J)	5.87		8.75		5.87		8.76		6.22		8.75		5.89		8.77	
Energy Gain (%)	32.9				33.0				28.9				32.8			

TABLE II
SIMULATION RESULTS

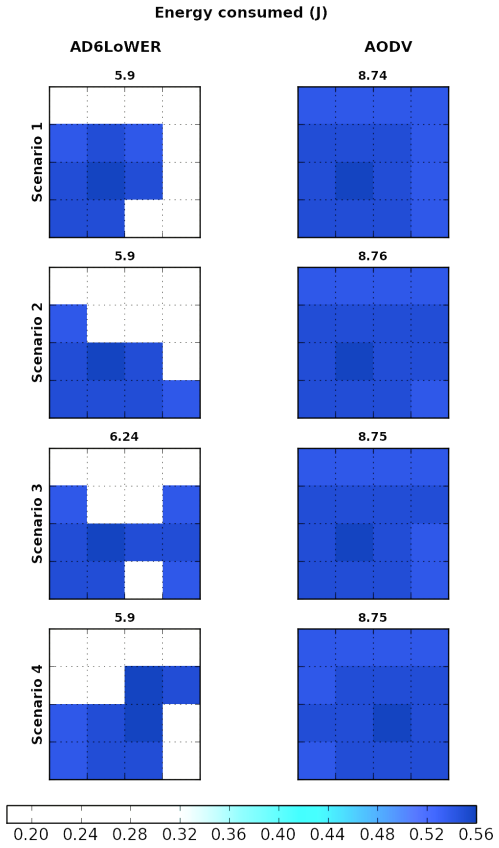

Fig. 12. Total of energy consumed in each scenario for the AD6LoWER and the AODV solutions

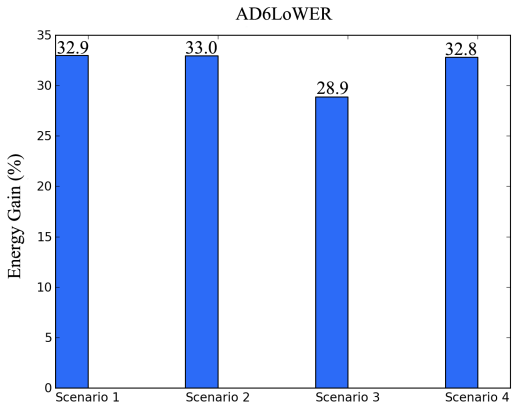

Fig. 13. AD6LoWER energy gains in each scenario (in %)

the nodes are sleeping, results show that, for the AD6LoWER solution, nodes sleep more time than for the AODV solution. For the selected scenarios, and for the AD6LoWER solution, the sum of time the nodes are sleeping is 57,000 s, while for the AODV solution is 56,640 s. In the idle mode energy is also consumed. The results are summarized in Table II. They show that, for the scenarios studied, our solution provides significant energy gains.

VIII. CONCLUSIONS AND FUTURE WORK

This paper proposes a solution (AD6LoWER) to reduce the energy consumed by WSN nodes. The work presented reflects a theoretical analysis of a solution which assumes that the nodes may run different applications inside the same large WSN with mesh connectivity. The solution tackles the problem by restricting routing and forwarding functions mainly to the nodes running the same applications, letting other nodes sleeping and avoiding their utilization. AD6LoWER was evaluated against AODV. Our results show that when designing WSN applications with the AD6LoWER solution, the nodes will not retransmit data packets from applications which they do not run; nodes not involved in communications are kept sleeping as much as possible. The overall energy consumption is lower than with AODV, and the final results prove that with the AD6LoWER solution the WSN lifetime is extended.

Since this evaluation was not performed on real sensor nodes, the results obtained still are theoretical, and may not yet reflect realistic issues

associated to MAC collisions or packet retransmissions caused by errors, and delay added due to application tagging and routing based on such tagging. We will address theses issues in future studies, as well as the study additional of topologies, including sensors capable of performing two different applications at the same time. We expect to implement a testbed in ContikiOS, and compare it against the RPL routing protocol using COOJA [15] as simulation tool.

REFERENCES

[1] I. F. Akyildiz, M. C. Vuran, O. B. Akan, and W. Su, "Wireless sensor networks: a survey revisited," *Computer Networks Journal (ELSEVIER SCIENCE)*, 2006.

[2] M. D. (Ed.), T. W. (Ed.), and F. T. (R&D), "Urban wsns routing requirements in low power and lossy networks," Routing Over Low power and Lossy networks (Active WG), April 2008, iETF, draft-ietf-roll-urban-routing-reqs.

[3] A. Dunkels, "Contiki OS, open source, highly portable, multi-tasking operating system for memory-efficient networked embedded systems and wireless sensor networks ," available at http://www.sics.se/contiki/.

[4] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*. New York, NY, USA: ACM, 2007, pp. 28–32.

[5] F. O. (ed, RWTH) and B. G. (SICS), "Reconfigurable Ubiquitous Networked Embedded Systems: D1.8 Contiki development report," SIXTH FRAMEWORK PROGRAMME PRIORITY 2 - Information Society Technologies, September 2007.

[6] IETF Network Working Group, "Ad-hoc on-demand distance vector (aodv) routing algorithm," Nokia Research Center, University of California, Santa Barbara and University of Cincinnati, available at http://tools.ietf.org/html/rfc3561.

[7] G. Ferrari, S. A. Malvassori, M. Bragalini, and O. K. Tonguz, "Physical layer-constrained routing in ad-hoc wireless networks: A modified aodv protocol with power control," in *Proc. Int. Workshop on Wireless Ad-hoc Networks 2005 (IWWAN'05)*, May 2005, pp. 2–2.

[8] M. R. H. Khan, M. A. Hossain, and M. S. H. Mukta, "Zigbee cross layer optimization and protocol stack analysis on wireless sensor network for video surveillance," in *International Conference on Electronics, Computer and Communication (ICECC 2008)*. Bangladesh: University of Rajshahi, 2008, pp. 795–799.

[9] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, "Middleware to support sensor network applications," *Network, IEEE*, vol. 18, no. 1, pp. 6 – 14, jan/feb 2004.

[10] "Wireless Modules, TelosB," available at http://www.xbow.com/Products/productdetails.aspx?sid=252.

[11] Chipcon Products and Texas Instruments, "2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver," 2006, document SWRS041.

[12] IEEE Computer Society, "IEEE Standard for Information technology - Telecommunications and information exchange between systems Local and metropolitan area networks Specific requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)," September 2006.

[13] JENNIC, "Calculating 802.15.4 data rates," Jennic, August 2006, application Note: JN-AN-1035.

[14] B. Latré, P. D. Mil, I. Moerman, B. Dhoedt, P. Demeester, and N. V. Dierdonck, "Throughput and Delay Analysis of Unslotted IEEE 802.15.4," *JNW*, vol. 1, no. 1, pp. 20–28, 2006.

[15] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, 14-16 2006, pp. 641 –648.

ACKNOWLEDGMENT

The authors would like to thank the support from the Portuguese Foundation for Science and Technology (FCT) under the fellowship SFRH/BD/36221/2007, to thank the support from the INESC Porto, and to thank the support from the School of Technology and Management of Viseu's Electrical Engineering Department. Also, they would like to thank to the anonymous reviewers for their valuable comments that did contribute to improve the quality of the paper.

Bruno Marques received a BsC degree in 1994, and a Diploma degree in 1998, all in Electrical Engineering from Polytechnic Institute's School of Technology of Viseu, Portugal. In 2001 he received an M.S in Electrical and Computer Engineering from University of Porto, Portugal. He is presently studying for a Ph.D degree. He is an assistant professor at School of Technology and Management of Viseu, where he gives courses in industrial and computer networks, and embedded systems.

Manuel Ricardo received a Diploma degree in 1988, an M.S. in 1992, and a Ph.D. in 2000, all in Electrical and Computers Engineering from University of Porto, Portugal. He is an associate professor at University of Porto, where he gives courses in mobile communications and computer networks. He also leads the Wireless Networks Area at INESC Porto.).