DISSense: An Adaptive Ultralow-power Communication Protocol for Wireless Sensor Networks

Ugo Maria Colesanti*, Silvia Santini† and Andrea Vitaletti*,
*Dipartimento di Informatica e Sistemistica, “Sapienza” Università di Roma, Rome, Italy
Email: colesanti,vitaletti@dis.uniroma1.it
†Institute for Pervasive Computing, ETH Zurich, Zurich, Switzerland
Email: santinis@inf.ethz.ch

Abstract—**This paper presents DISSense, an adaptive, ultralow-power communication protocol for wireless sensor networks. DISSense is specifically designed for long-term environmental monitoring applications and it provides for both data collection and data dissemination services. By automatically adapting the length of its active phases, DISSense can guarantee for both a very low duty cycle and reliable data delivery. We tested the performance of DISSense on both a testbed and on the TOSSIM simulation environment. Our experimental results show that a sensor network running DISSense can provide for average data delivery ratios above 98% and at the same time achieve a lifetime of several years. Our TinyOS 2.1 implementation of DISSense is publicly available.**

I. Introduction

Environmental monitoring [1], [2], [3], [4] is one of the many envisioned application scenarios for wireless sensor networks (WSNs). Enabling fine-grained, long-term monitoring, however, requires tens or hundreds of battery-powered nodes to be deployed over a large area. The number of nodes and the dimension of the network clearly make battery replacement a cumbersome and time consuming task. Minimizing energy consumption to extend network lifetime is thus a primary requirement entering the design of protocols and applications for WSNs. In particular, the operation of the wireless transceiver causes high energy consumption on sensor nodes [22], and, thus, energy saving techniques typically aim at optimizing communication. For instance, sensor selection and data aggregation algorithms aim at reducing the overall amount of data and, thus, data packets, the network needs to relay to a central collector [18], [19]. Energy-aware routing protocols try to minimize the overall communication cost of relaying a packet from a remote source to a sink node [12]. Finally, energy-aware medium access control (MAC) protocols can reduce the cost of packet transmissions by making the nodes persist in to the so-called *sleep* mode whenever possible [7], [8], [11]. While in sleep mode, the wireless transceiver drains nearly no current but it is still able to quickly resume operation. The longer a node persists in sleep mode, the lower will be the *duty cycle* of its radio, which is the percentage of time during which the wireless transceiver is actively powered for communication. Many MAC protocols for WSNs may indeed achieve very good performance in terms of duty cycle [7], [8], [11]. However, the actual duty cycle of the network strongly depends on how efficiently MAC and routing protocols can cooperate. Some approaches do offer optimized cross-layer solutions [9], [10] but they have been tailored to specific platforms, are not publicly available, or do not offer upper bounds for the latency of delivered packets [9], [10].

In this paper, we present the design and implementation of DISSense, an adaptive, ultralow-power communication protocol that has been designed taking into account cross-layer optimization issues. DISSense offers both a collection and dissemination service and it is particularly suited to support long-term environmental monitoring applications. We implemented DISSense in TinyOS 2.1 and tested it on both an indoor testbed and on the TOSSIM simulator [17]. Our experimental results show that thanks to its ability to adapt to changing network conditions, DISSense aggressively reduces the duty cycle of the network, thereby enabling a mote-based network to reach a lifetime of several years. At the same time, DISSense is able to deliver, on average, more than 98% of the data packets injected into the network. Other protocols presented in literature may indeed achieve similar performance in terms of data delivery ratio [9], [10], [12]. However, unlike other approaches [9], [10], DISSense runs on several platforms, including Tmote Sky and MicaZ motes [16], and its implementation in TinyOS 2.1 is publicly available. Furthermore, for scenarios in which the sampling rate of the network is sufficiently low, DISSense may provide for low latency and long network lifetime. Last but not least, DISSense is able to adaptively compute its key parameters, thereby relieving the user from a tedious, manual protocol tuning.

II. Background and Related Work

A. Periodic Environmental Monitoring

Periodic environmental monitoring (PEM) represents one of the most popular application scenarios for WSNs [1], [2], [3], [4]. In PEM deployments, sensor nodes are scattered over a large area and periodically collect and report sensor data to a central sink. Examples of measured quantity include temperature, electric conductivity, or moisture levels. The sampling period may vary from few minutes to several hours [1], [2], [3], [4]. The per-node amount of data collected during each sampling period is usually low enough to fit into a single data packet. The required data delivery ratio typically exceeds 95% [6]. Latency requirements may vary depending on the scenario. For off-line analysis, collecting the data every day or even at slower pace typically suffice. For other scenarios, such as cultural heritage monitoring or fire detection, the latency must be sufficiently low (seconds or minutes) in order to enable human or automatic intervention. As for most application scenarios for WSNs, the lifetime of the network may be required to be of several years. Because replacement of, or intervention on, sensor nodes is unfeasible, the nodes must rely on their own energy supply, which often consists of common AA batteries. Critical application scenarios may also require higher than average performance in terms of delivery ratio and data rate [24], [25]. In these cases, long network lifetime may be achieved either by using high performing and possibly bulkier batteries [24] or by leveraging energy harvesting techniques (e.g., photovoltaic panels) [25]. It is important to note that although WSNs are envisioned to encompass thousands or millions of nodes [5], [23], common PEM deployments build upon networks of tens or hundred of nodes, which is also the reference network size for DISSense.

B. Routing and MAC Protocols

There exist a plethora of energy-aware MAC protocols for WSNs. In [28], these are classified in *slotted*, *random*, *frame-based*, and *hybrid* protocols. Slotted protocols make the nodes share a common schedule that alternates sleep and active phases. The length of active slots typically ranges between tens and hundreds of milliseconds, while sleeps slots last significantly longer resulting in low duty cycles. Within each active slot CSMA-based techniques are used to manage channel contention. Random access protocols avoid the use of a shared schedule. Instead, they demand most of the communication effort to the transmitter, which must inform the receiver if a transmission will take place. Frame-based protocols group slots into frames and assign one or more slots to each node. In this way, nodes can avoid collisions and channel contention. Keeping this schedule requires tight synchronization between nodes and induces a large memory footprint, making frame-based protocols hard to use in practice. Hybrid protocols aim at combining the advantages of both random and frame-based protocols. With respect their random counterparts, hybrid protocols limit collisions but cause more control overhead.

T-MAC [7] is a slotted protocol that uses a CSMA/CA MAC with RTS/CTS mechanism for packet transmission. T-MAC achieves energy efficiency by keeping the active phase as short as possible with respect to the sleep period. If no traffic is sensed after a pre-specified timeout, a node can switch to sleep mode until the next active period will start. T-MAC sets the default length of the timeout to 15ms for a period of 610ms. Thus, the duty cycle of the radio is 2.4%, which is considerably high for PEM scenarios. Longer sleep periods may cause synchronization problems and high latencies that may hamper the correct operation of routing protocols.

BoX-MAC-1 [8] is a random protocol that is part of the standard low-power MAC of the TinyOS operating system [13]. Each node running BoX-MAC-1 wakes up periodically from the sleep mode and checks for channel activity. This mechanism, called *Low Power Listening* (LPL), enables a transmitter to communicate with a receiver by continuously transmitting data packets during an *activity* period. As soon as all transmissions are acknowledged, nodes go back to sleep mode. BoX-MAC-1 moves energy consumption for communication from receivers to transmitters and works well on 1-hop scenarios with low traffic loads. However, in multi-hop scenarios that require a routing protocol, overhearing and collisions as well as the transmission overhead of broadcast messages such as routing beacons, significantly degrade the energy efficiency of LPL-based protocols [28].

WiseMac [27] is a random protocol that uses LPL but, unlike BoX-MAC-1, it maintains a neighbor table with poll schedules, updated each time a packet is received, which allows a node to send short preambles only. The preamble length also takes into account the maximum clock drift from the last message exchange and, if no poll schedule is available, it simply falls back to long LPL preambles. Despite in [28] WiseMac is considered the most performing MAC protocol for low data rate applications, it is also mentioned that, similarly to Box-MAC-1, its performances quickly degrades when broadcast communication pattern is required.

Z-MAC [11] is a hybrid approach that works as a contention-based protocol for low traffic levels, but it turns into TDMA mode for high levels. Z-MAC uses global time synchronization once during setup. Subsequently, only local synchronization between sender and receiver is required. Despite the hybrid design, Z-MAC becomes energy efficient only for high traffic load only (more than 3 packets per second per node), but it is still far from reaching the ultralow energy efficiency required by PEM applications.

CTP is a collection protocol for WSNs. It uses *beacon* messages for building and maintaining a routing tree, and *data* messages to report application data to the sink. The standard implementation of CTP consists of three main logical software components: the *Link Estimator*, the *Routing Engine* and the *Forwarding Engine*. The Routing Engine takes care of sending and receiving beacons as well as creating and updating the routing table. The Forwarding Engine forwards data frames. Each transmitted data frame is acknowledged at the link layer, enhancing the reliability of the protocol. Furthermore, the FE implements a duplicate-detection mechanism and it has the ability to detect and repair routing loops. The Link Estimator is mainly responsible for determining the inbound and outbound quality of a communication link. CTP is compatible with the BoX-MAC-1 MAC. Both these protocols have been tested in a 100 node network with 5 minutes sampling period [12]: In this work, the overall network lifetime reached approximately 4 months, which is far below the PEM requirements.

*C. **Full Layered Protocols**.*

Koala [10] implements an efficient asynchronous wake up strategy and on-the-fly route computation whenever data download is requested by the sink. The energy saved avoiding the control overhead during inactivity periods compensates the higher cost of wake up and route construction. The sampled data is logged on the flash memory of each node and can be sent when requested. To ensure energy efficiency, Koala needs to log a significant amount of data before initiating the wake up strategy. However, the low data rate of PEM applications implies that several days are required to log a sufficiently large volume of data such that the energy efficiency of Koala becomes reasonably high.

Dozer [9] is a data-gathering protocol designed for Environmental Monitoring. It integrates MAC layer, topology control and routing to reduce energy wastage of the communication subsystem. The data gathering in Dozer relies on a tree-based network, while the data exchange is enabled by a TDMA protocol. To avoid global synchronization, each node has two independent TDMA schedules, one for its parent role and one for its child role. Link-layer acknowledgement is enabled for each packet transmission to enhance the protocol reliability. Dozer provides mechanisms for load balancing, parent selection and hidden-node collision avoidance. With its *lazy* TDMA approach, Dozer is able to reach an ultralow-power consumption, which increases network lifetime up to 8-10 years. To the best of our knowledge, Dozer is the most performing ultralow power communication protocol available nowadays. However, Dozer is a commercial closed-source protocol available only for the TinyNode platform [14]. Moreover, it cannot provide any guarantees on data latency and requires a fine tuning of its parameters during setup.

III. DISSense: Protocol Overview

DISSense provides for both data dissemination and collection services for WSNs. It targets environmental monitoring applications requiring periodic sampling of a given physical phenomenon. In particular, DISSense takes as input the desired sampling period and computes an adaptive time schedule for the nodes to coordinate in order to build a data collection tree. The schedule alternates short activity phases, during which nodes deliver sensed data to the sink, and long intervals, during which nodes operate in an ultralow-power mode. To disseminate the shared schedule to all nodes in the network, DISSense provides an efficient, one-to-many backward channel .

A. Adaptation

DISSense achieves energy-efficient operation by adaptively shortening the length of the time interval during which nodes must activate their radio transceivers. Reducing the length of the active phase clearly enables DISSense to reduce the duty cycle of the network and, thus, to extend its lifetime. The main challenge arising in this context consists in making the protocol able to timely and reliably deliver data to the sink despite the shortening of the active phase. The diameter, density, and overall link quality of the network also affects protocol behavior. For example, reliable protocols such as DISSense may require several (re)transmission attempts over a bad link before at least one succeeds. Moreover, the denser the network the longer it takes to settle channel contention. And the higher the diameter of the network the higher the average number of hops packets must be relayed through before reaching the sink. By taking into account all these factors we make DISSense able to autonomously adapt its duty cycle to the actual dynamics of the network, and to ensure both high delivery ratios and energy efficiency. To control DISSense's adaptive behavior, we define two metrics: the Time To Resync (TTR) and Time To Receive Data (TTRD), which we will describe in detail in section IV-D.

B. Schedule

The sink is responsible for determining and disseminating the schedule according to which nodes send and receive their packets. Figure 1 illustrates the different phases of the DISSense schedule: active phases are scheduled at each sampling period. Because of clock drifts and of the long inactivity period between active phases, a Guard Time Interval (GT) is foreseen at the beginning of each phase. Moreover, a re-synchronization procedure periodically takes place during the Re-synchronization Interval (RI), so as to realign the schedule and compensate for clock drifts. Depending on the sampling period and intervals length, DISSense is able to skip the RI for one or more sampling periods, so as to optimize the overall protocol duty cycle. The skip functionality depends on the parameter σ_{sk}, whose computation is described in section IV-D. During the RI, nodes exchanges routing beacons and collect the information needed to build a collection tree having the sink as its root. At the end of the RI, DISSense ensure that the nodes share a common wake-up time for the next active phase, and have a parent selected in the collection tree for data transmission. After the RI, the Data Collection Interval (DCI) begins. During the DCI each node sends its data over the already built collection tree, and also act as forwarders for other nodes of the network. Between two active phases, DISSense turns into an Ultra-Low-Power State (ULPS) by switching the radio to LPL mode with a 0.1% duty cycle. In ULPS the radio is not turned fully off since some nodes may be added and other ones can go out of synchronization. Both these nodes need to retrieve the protocol schedule in order to participate to the network. The value of the duty cycle during ULPS is low such that it does not significantly affect the overall protocol duty cycle. Note that the sink schedule only adopts an *ACTIVE* interval, since it does not need to discriminate the different intervals of the active phase. The active phase of DISSense runs on a CSMA/CA MAC with 100% duty cycled radio. The benefits of such a solution are twofold. First, it accelerates the construction of the collection tree and the data collection process itself, thereby shortening the length of the active phase. Second, it prevents the inefficiencies, described in section II, related to broadcast transmissions (e.g. routing beacons) under duty cycled MAC protocols.

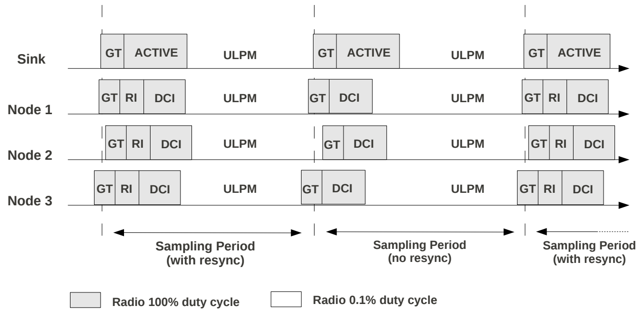
Sink
Node 1
Node 2
Node 3
GT
ACTIVE
RI
DCI
ULPM
Sampling Period (with resync)
Sampling Period (no resync)
Sampling Period (with resync)
Radio 100% duty cycle
Radio 0.1% duty cycle

Fig. 1: DISSense schedule with $\sigma_{sk} = 1$

C. Collection and Backward Channel

Data collection in DISSense is achieved by leveraging and extending the CTP Collection protocol [12]. CTP is a popular and highly reliable collection protocol. When running CTP each node computes a metric, called ETX, which represents the estimated number of transmissions a packet from this node will go through before reaching the sink. CTP also supports loop detection, duplicate transmission and quick reaction to topology changes. However, CTP is not optimized for applications requiring short active phase sessions interleaved with inactivity periods, such as the scenario we are taking into account. Instead, DISSense allows to stop, start, pause, and reset the construction and maintenance of the collection tree at any time.

The backward channel in DISSense is used by the sink to resynchronize the network and to send schedule changes to nodes (e.g. intervals length and changes in sampling period). A node missing a schedule update is likely to loose synchronization with the other nodes. DISSense implements a backward channel, namely the Implicit Backward Channel (ICB), that guarantees that each node node having selected a parent in the collection tree, also shares the same values sent by the sink over the ICB during the active phase. The ICB runs during the RI and uses the same beacons required for the collection tree construction. The main advantage of this solution is that the RI interval can be tailored on the collection tree construction only since the ICB execution does not require any additional time. Further details are discussed in section IV-A and IV-B.

D. Architecture

Figure 2 illustrates DISSense's architecture. DISSense main modules are the Manager, the Adaptive Engine, the LplManager, and the NtpManager, which are described below. As mentioned above, DISSense relies on CTP to build and maintain the routing tree. Thus, DISSenses embeds CTP's Link Estimator, Forwarding Engine, and Routing Engine modules.

1) Manager: The Manager handles DISSense core functionalities, such as network re-synchronization and schedule management. The module also provides a Send/Receive interface to the application layer that enables the send of a single data packet during each active phase and collects statistics for the Adaptive Engine. The Manager has the ability to start,

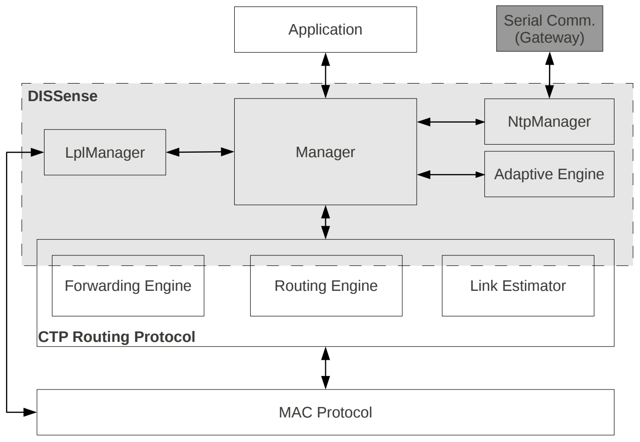

Fig. 2: DISSense Architecture

pause and reset the underlying CTP protocol. It can also change the radio duty cycle.

2) Adaptive Engine: The Adaptive Engine has a dual functionality. On the sink, it computes intervals length for GT, RI, DCI and skip parameter σ_{sk} . These values are then transmitted over the ICB. On the nodes, the Adaptive Engine retrieves and stores the values, which can later be read by the manager module.

3) LplManager: The LplManager module is responsible for radio communications during Ultra-Low-Power Mode. As mentioned in section III-B, during ULPM phase, the nodes turn their radio to a 0.1% duty cycle rather than switching it off. The LplManager enables a node that looses synchronization or a newly added node to efficiently retrieve synchronization information from its neighbors during their inactivity period. This mechanism avoids the need to scan for an active phase, which requires high power consumption. The LplManager also supports the additional functionality of state transmission, which consists in sending a snapshot of the node state for debugging purposes.

4) NtpManager: The NtpManager module that is active only on the sink, interacts with an external gateway to synchronize the sink with an external entity (e.g. Ntp Time). The NtpManager provides an additional functionality that enables the user acting on the gateway, to dynamically change the sampling period, and to determine the hour of the day at which data samples need to be generated.

IV. DISSense: Implementation

DISSense is implemented in TinyOS 2.1 [13], a lightweight, open-source operating system for WSNs. It supports the TelosB/TmoteSky and MICAz platforms. Support for Iris and TinyNode 184 is planned in future releases [16], [14]. DISSense also runs on the TOSSIM simulation environment.

As described in section III, the sink in DISSense acts as an orchestrator for the network. In particular, the sink is responsible to determine the adaptive schedule, share it over the network, and periodically collect the sampled data from the nodes. To this end, DISSense runs a deeply modified version of CTP. On one hand, DISSense adapts CTP to run on the specific schedule described in section III-B, which requires CTP to be paused and resumed periodically following the duty cycle of the protocol. On the other hand, DISSense embeds a backward channel in CTP namely, the Implicit Backward Channel, which enables DISSense to send controls for initiating synchronization and tree construction.

A. Data Collection

In DISSense the CTP protocol is stopped during Ultra-Low-Power Mode and resumed during the active phase. When resumed, DISSense runs CTP in a *stateful* or *stateless* mode. In stateful mode the routing information of the previous sampling period used. This mode is employed when the DCI is scheduled without the RI and there is the need to directly collect the sampled data. Instead, in stateless mode the CTP state (neighbor table, selected parent) is reset in order to refresh the topology information of CTP and enable the IBC to share new synchronization information as much as the new schedule parameters values. The stateless mode is run when the RI is scheduled.

B. Implicit Backward Channel

The IBC guarantees, during each RI interval, that a node having selected a new parent also shares the values transmitted by the sink at the beginning of the RI. DISSense uses the IBC to share schedule's information updates. In particular, at each RI, the sink transmit over the IBC the sampling period, the length of RI and DCI intervals, the skip parameter σ_{sk} and the next wake up time. These values are appended as a footer to each routing beacon. For each incoming beacon, the Routing Engine follows the algorithm in figure 3 which enables a node to store the information carried by the beacon (and retransmit it as a footer in each subsequent transmitted beacon) if and only if the sender has been selected as a parent.

Note that during beacons transmission, each node fills the footer with its local values thus, at the beginning of the RI, each node that has not yet selected a parent, transmits stale content. The algorithm, however, implicitly solve this issue thanks to the parent selection algorithm of CTP. The parent selection algorithm guarantees that each parent candidate must already have selected a parent in the tree. Thus, each parent candidate must have run the IBC update process of algorithm 3 hence, carry updated values. As a consequence, at the end of the RI, each node will be in one of these two states:

- The node belongs to the collection tree, is resynchronized and have updated schedule parameters values;
- The node does not belong to the collection tree, is out of synchronization and needs to retrieve the schedule through the LplManager during ULPM.

C. Resynchronization

The re-synchronization procedure uses the IBC to reliably propagate the timestamp related to the beginning of the next active phase. DISSense manages the Next Wake Up Time field of the IBC footer using the *TimeSyncAMSend* interface implemented in TinyOS 2.1 by the *CC2420TimeSyncMessage* component. This component is a submodule of the *Flooding Time Synchronization Protocol* [15]. It has been written for ChipCon2420 transceivers and allows a sender to piggyback a local timestamp t_e, related to an event e, to each transmitted packet. The receiver, in turn, decodes the timestamp t_e as a new timestamp $t'_e = t_e + \delta$ representing the event e expressed as the receiver's timestamp t'_e with an error δ corresponding to the packet propagation time.

```
**Require:** node $n$
  **upon** rx beacon $b_i$ from node $i$
  **if** parent($n$) == null **then**
    $temp \leftarrow footer(b_i)$
    process($b_i$)
    **if** parent(n) == $i$ **then**
      resync($temp.nwu$)
      store($temp.ri, temp.dci$)
      store($temp.sp, temp.\sigma_{sk}$ )
    **end if**
    $temp \leftarrow null$
  **else**
    process($b_i$)
  **end if**
```

Fig. 3: IBC Algorithm

Using the *TimeSyncAMSend* interface for CTP beacons transmission enables DISSense to reliably propagate the next wake up time over the network with a maximum error $\Delta = d{\cdot}\delta$ where d is the network diameter. Note that, over short distances, the packet propagation time is in the order of few microseconds and, as it will be shown in section V-B, Δ is several orders of magnitude lower than the size of the active phase. Thus, as figure 3 shows, when the parent is selected, the *resync* commands is called over the *Next Wake Up Time* field that realigns the schedule of the next wake up.

D. Adaptation

As described in section III-A, DISSense uses two metrics, TTR and TTRD, in order to find a good tradeoff between duty cycle minimization and correct protocol execution. These metrics are computed by the Manager, which sends them to the Adaptive Engine along with the protocol sampling period. Based on these values, the Adaptive Engine computes the intervals length of GT, RI, and DCI as well as the skip parameter σ_{sk} .

Time To Resynchronize (TTR): The TTR represents the maximum time required by DISSense to resynchronize all the nodes of the network. The value is computed by the Manager module of the sink during each sampling period where the RI is scheduled. Let n be the number of nodes of the network, the Manager retrieves TTR_i from each node $i = 1 \ldots n-1$ and computes the TTR as $max\{TTR_1, \ldots TTR_{n-1}\}$. The TTR_i of each node i is computed as the time elapsed from the beginning of the active period and the resynchronization event(c.f. *resync* command in algorithm of figure 3). Each locally computed TTR_i is piggybacked to data packets transmitted at each sampling period.

Time To Receive Data (TTRD): The TTRD represents the time required by DISSense to collect the sampling data from the network. The Manager module of the sink computes the $TTRD$, during each sampling period, as the time elapsed between the reception of the first and the last data packet during active phase. Note that the computed $TTRD$ is an approximation of the actual time required by DISSense to collect sampling period because it ignores the node-to-sink latency of the first packet received. However, the first packet usually arrives from a 1 hop neighbor of the sink and thus, the node-to-sink latency of the first packet is very small compared to the overall computed TTRD.

Duty Cycle: Let T_{period} the sampling period and T_{GT}, T_{RI}, and T_{DCI} the GT, RI and DCI lengths. Also let $T_{on}(\sigma_{sk})$ be the fraction of time, within a sampling period, during which the radio is in active phase. Similarly, let $T_{ulpm} = 1 - T_{on}(\sigma_{sk})$ represent the same ratio for the Ultra-Low-Power Mode phase. We compute $T_{on}(\sigma_{sk})$ as:

$$T_{on}(\sigma_{sk}) = \frac{T_{GT} + T_{RI} + T_{DCI}}{T_{period} \cdot (\sigma_{sk} + 1)} + \frac{T_{GT} + T_{DCI}}{T_{period} \cdot (\sigma_{sk} + 1)} \cdot \sigma_{sk} \tag{1}$$

Recalling section III-B, during $T_{on}(\sigma_{sk})$ the duty cycle is 100% while during T_{ulpm} it is set to 0.1%. We define the protocol duty cycle $P_{dc}(\sigma_{sk})$ as weighted sum of $T_{on}(\sigma_{sk})$ and T_{ulpm} with their corresponding duty cycle, hence:

$$P_{dc}(\sigma_{sk}) = 1 \cdot T_{on}(\sigma_{sk}) + 10^{-3} \cdot T_{ulpm} \approx T_{on}(\sigma_{sk}) + 10^{-3} \tag{2}$$

Guard Time Interval: The Guard Time interval must be greater than the maximum drift produced within two resynchronization procedures. It depends on clock precision, sampling period and Skip value. For instance, the clock precision of TelosB motes is 50 ppm [16] that becomes $100ppm$ assuming clocks drift of two nodes in opposite direction. Further, resynchronization takes place each $\sigma_{sk} + 1$ sampling periods. Thus, if T_{period} represents the sampling period, the Adaptive Engine computes the Guard Time Interval as:

$$T_{GT} = 3 \cdot 50 \cdot T_{period} \cdot (\sigma_{sk} + 1) \cdot 10^{-6} \tag{3}$$

Note that to ensure a safety margin (e.g. unpredictable behavior related to temperature changes) the interval is equal to the maximum drift increased by 50%.

Skip period: The Adaptive Engine computes the skip value σ_{sk} so as to minimize the overall duty cycle of the protocol $P_{dc}(\sigma_{sk})$. Recalling equation (2), the minimum duty cycle corresponds to the minimum $T_{on}(\sigma_{sk})$. Thus, replacing (3) in (1) we have:

$$T_{on}(\sigma_{sk}) = \frac{150 \cdot (\sigma_{sk} + 1)}{10^6} + \frac{T_{DCI}}{T_{period}} + \frac{T_{RI}}{T_{period} \cdot (\sigma_{sk} + 1)} \tag{4}$$

It is easy to demonstrate that the minimum of (4) is reached for:

$$\tilde{\sigma}_{sk} = \sqrt{\frac{T_{RI} \cdot 10^6}{T_{period} \cdot 150}} - 1 \tag{5}$$

However, σ_{sk} must be an integer thus, after having observed that the second derivative of (4) is positive, we can assess that the minimum duty cycle is reached for:

$$\sigma_{sk} = \arg\min(T_{on}(\lfloor \tilde{\sigma}_{sk} \rfloor), T_{on}(\lceil \tilde{\sigma}_{sk} \rceil)) \tag{6}$$

Resynchronization and Data Collection Intervals: The RI must be long enough to allow DISSense to resynchronize all the nodes of the network and build the collection tree. Recalling the resynchronization algorithm in figure 3, a resynchronized node has already selected the parent thus, the RI depends uniquely on TTR. Similarly, the DCI must be long enough to allow each generated sample to be routed to the sink thus, it depends on TTRD.

For each TTR and TTRD update, the Adaptive Engine increases them by 50% to catch values fluctuation and inserts them in an Exponential Weighted Moving Average (EWMA) obtaining:

$$T_{RI} = \frac{1.5 \cdot TTR_{new} + 9 \cdot T_{RI_{old}}}{10} \tag{7}$$

$$T_{DCI} = \frac{1.5 \cdot TTRD_{new} + 9 \cdot T_{DCI_{old}}}{10} \tag{8}$$

Where TTR_{new} and $TTRD_{new}$ are the newly inserted values while $T_{RI_{old}}$ and $T_{DCI_{old}}$ the intervals length before the updates.

V. Evaluation

We evaluated the performance of DISSense on both an indoor testbed and on the TOSSIM simulator. While the latter allowed us to test DISSense on different network topologies, the testbed-based evaluation demonstrates DISSense feasibility for real-world deployments. As detailed below, our experimental results show that DISSense can guarantee for reliable data delivery and, thanks to its power-efficiency, it allows to operate a Tmote Sky-based WSN with common AA type alkaline batteries for several years.

A. Metrics

We evaluate the performance of DISSense in terms of achieved duty cycle and data delivery, as well as in terms of number of generated duplicate packets. For the definition of DISSense's duty cycle we refer to equations 1 and 2. We derive an estimation of the lifetime of the network given a specific duty cycle using the methodology suggested in [9], thus ignoring power consumption of application-specific sensors. We further assume that a node is powered by two common AA Alkaline batteries of 2500mAh capacity each and that the current drain of the TmoteSky during 100% duty cycle and active CPU is 24.8mA [22]. Taking into account battery self-discharge equal to 15% in 4 years [26], a network running DISSense with a duty cycle of 1% can operate for 1 year. Reducing the duty cycle to 0.2% allows extending the network lifetime to 5 years. These values show the significant impact of the duty cycle on the lifetime of a WSN, although they only represent rough estimates of the total lifetime. Indeed, temperature, intermittent power drain, and chemical

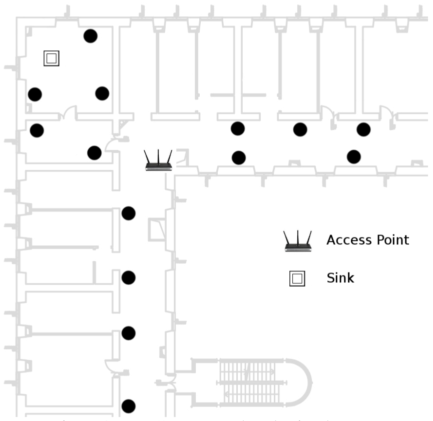

Fig. 4: WSN Testbed (indoor)

deterioration over time can increase or decrease the speed of battery discharge. Nonetheless, the lifetime can be increased by using high-performance batteries [24] or by exploiting energy harvesting techniques (e.g., photovoltaic panels [25]).

Besides being able to operate a WSN for long periods of time, DISSense must also provide for reliable data delivery. We thus evaluate its performance in terms of data delivery ratio (DDR), which we define as the ratio between the number of data packets injected by the nodes into the network and the number of packets successfully collected at the sink. As a further metric to describe the performance of DISSensee, we also also consider the number of duplicate packets that eventually reach the sink (the lower this number, the higher DISSense's efficiency).

We do not report results about performance of DISSense in terms of latency, as its value is upper-bounded by the length of the active phase. In all our experiments, latency never exceeded 4.5 seconds, which represents the default active phase interval at startup.

B. Testbed

Setup: We run DISSense on a testbed of 15 nodes deployed on the first floor of an office building, as depicted in Fig. 4. The presence of walls, a WiFi access point and intense wireless communication activity contributed in creating an unreliable, unpredictable, and thus realistic wireless communication channel. We run DISSense on this testbed for 2 months keeping the sampling period to: 1 minute for the first 31 days; 15 minutes for next 10 days; 1 hour for the last 20 days. In the following, we indicate with DISSense-x the instance of DISSense having a sampling period of x minutes.

Results: Fig. 5 shows how the length of the RI, DCI, and GT intervals, as well as the parameter σ_{sk}, varies over time when running DISSense-1. The default startup value of both T_{RI} and T_{DCI} is 2 seconds. Using equations (6) and (3), the default values of σ_{sk} and T_{GT} result being 14 and 139ms, respectively. During each RI interval, the sink's Adaptive Engine recomputes the values of these parameters thus enabling DISSense adaptive behavior. Indeed, Fig. 5 shows that the length of both the RI and DCI intervals quickly

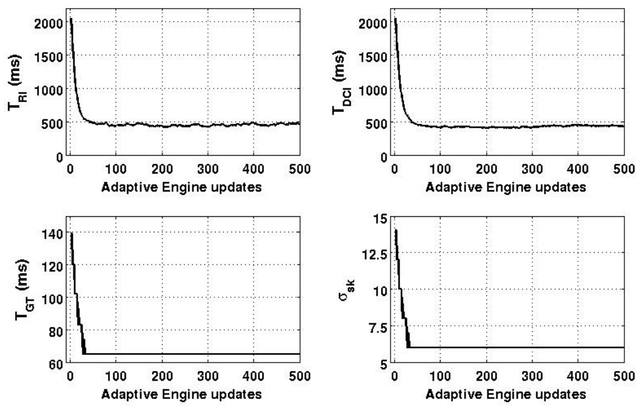

Fig. 5: DISSense-1 Adaptive Engine Parameters (Testbed)

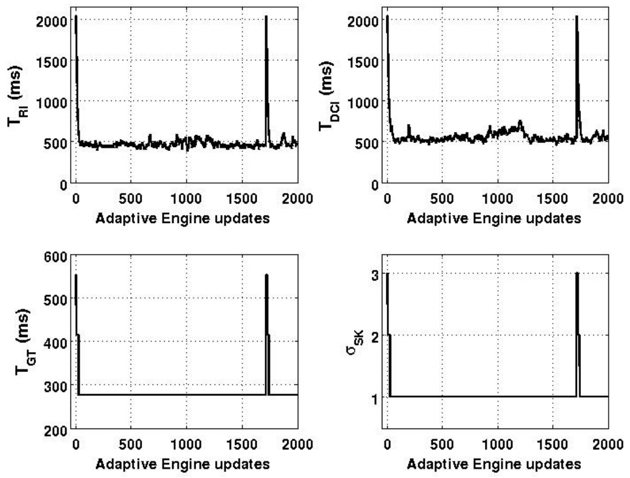

Fig. 6: DISSense-15 Adaptive Engine Parameters (sink reset)

converges to values included in the ranges $[450ms, 650ms]$ and $[400ms, 500ms]$, respectively. It is interesting to point out that since the sampling period is fixed and since equation (3), which controls the evolution of T_{GT}, depends on the values of the sampling period and skip parameter only, the evolution of T_{GT} follows that of σ_{sk} .

Fig. 6 reports the same data as Fig. 5 but for DISSense-15. In this experiment the sink has been reset to make the nodes go out of synchronization. After this reset, the sink loads the default values for the schedule and shares them with the nodes. Fig. 6 shows that the Adaptive Engine is able to quickly recompute new optimal values for the key protocol parameters.

Table I summarizes the performance of DISSense. As expected, increasing the sampling interval makes the duty cycle shrink. In particular, it decreases from 1.09% for DISSense-1 to 0.15% for DISSense-60. This is due to the fact that, although a longer sampling period induces an increase in the length of the guard interval T_{GT}, the period during which the nodes are inactive has a proportionally higher increase. This results in an overall lower duty cycle. Table I also shows that DISSense's average delivery ratio oscillates around 98% irrespectively of the sampling period. We found that that most of the packet losses are due to the occurrence of routing

	Sampling Period (min.)		
	1	15	60
Duty Cycle (%)	1.09	0.22	0.15
Data Delivery Ratio (%)	97.8	98.6	98.9
Duplicate Packets (%)	0.16	0.24	0.03

TABLE I: DISSense performance (testbed)

Path Loss Exponent	4.7
Shadowing Standard Dev.	3.2
Reference distance d_0 (m)	1
Path Loss (d_0) (dBm)	25.6
Noise Floor (dBm)	-105
White Gaussian Noise	4

TABLE II: Parameters of the Network Generator

loops. Although DISSense can rely on CTP's loop detection mechanism, the time necessary to re-establish a valid route is typically larger than the length of the active phase. Thus, looping packets get dropped because they cannot reach the sink before the network goes back to sleep.

The last row in Table I shows that, thank to the fact that DISSense relies on CTP's effective duplicates suppression mechanism, the number of duplicate packets that reach the sink is negligible with respect to the total data traffic.

C. Simulation Study

Setup: We run DISSense-1, DISSense-2 and DISSense-5 within the the TOSSIM simulation environment. To this end, we generated networks having 10, 20, 30, 40, and 50 nodes (excluding the sink) using TOSSIM's Network Generator tool with parameters set as summarized in Table II. For each network size, we generated 20 different topologies. To reproduce the vagaries of the wireless channel we resort to the *casino-lab* noise model [17]. Furthermore, the queue size of CTP has been increased to 40 packets so as to avoid packet losses due to full buffers.

Results: Fig. 7 shows the value of the duty cycle of DISSense as the number of nodes in the network increases. The duty cycle increases with the number of nodes and decreases as the sampling period increases. In particular, the duty cycle of DISSense-1 increases from 0.8% for 10 nodes to 3.3% for 50 nodes while for DISSense-2 it increases from 0.5% to 1.75% and for DISSense-5 from 0.35% to 0.8%. The duty cycle further decreases when the sampling period progressively grows to 60 minutes. From these results we can discuss the major strengths and weakness of DISSense. As described in section V-B, DISSense is very efficient for relatively long sampling periods (larger than 5 minutes), for which it can achieve duty cycles as low as 0.15%. However, the performance of DISSense degrades as the sampling period decreases and the network size increases. This is due to the fact that the CSMA/CA-based design of DISSense makes the time needed for channel contention increase with the density of the network. This, in turn, makes the length of DISSense synchronization and data collection intervals, and, thus, the length of the active phase, increase, causing the duty cycle to increase too. For long sampling periods this effect is mitigated by the largely predominant ULPM interval. When the sampling period is short, the effect becomes less negligible.

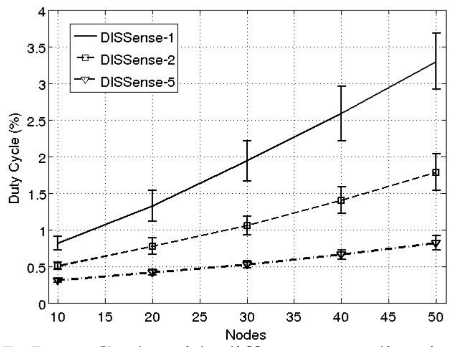

Fig. 7: Duty Cycle with different sampling intervals

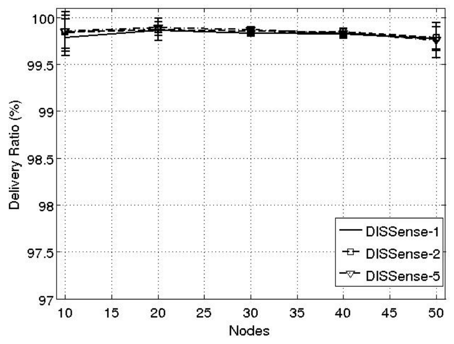

Fig. 8: Delivery Ratio with different sampling intervals

On the other hand, data delivery rates are not affected by changes in sampling period or network size. In fact, as Fig. 8 shows, DISSense achieves a DDR higher than 99% irrespectively of the sampling period or network size. DISSense achieves this good performance by combining CTP's inherent reliability and the ability of the Adaptive Engine to estimate the appropriate length of the RI, DCI, and GT intervals.

D. Multi-sink Support

To mitigate DISSense's loss in performance for large network sizes, it is possible to resort to a multi-sink approach. If several sinks are defined, CTP will naturally construct several collection trees, each having one of the sinks as its root. This allows to reduce the overall diameter and density of the network and thus to reduce the length of the active phase. To this end, though, we have to assume that the sinks are all synchronized (through the NtpManager). Multi-sink DISSense works as the single-sink version except for the presence of multiple schedules, one for each sink. In fact, each sink shares its schedule with the nodes belonging to its subtree and adapts it following the same principles described in sections IV-B and IV-D. As the schedules depend on the number of nodes of each subtree, each schedule will show different values for T_{GT} , T_{DCI} , T_{RI} , and σ_{sk} . As a consequence, a node switching to a new subtree may have to adapt to a new schedule and is thus likely to go out of synchronization. This typically happens to nodes halfway between two sinks. One possible solution to this problem consists in making the sinks able to share their respective schedules through the NtpManager. In particular, choosing the largest schedule

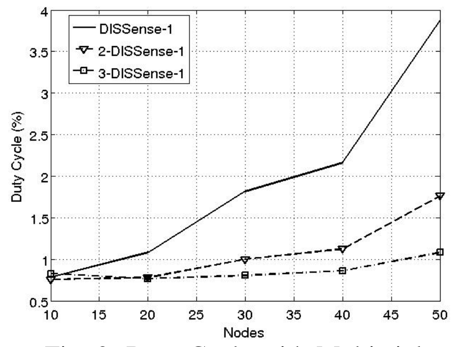

Fig. 9: Duty Cycle with Multi-sink

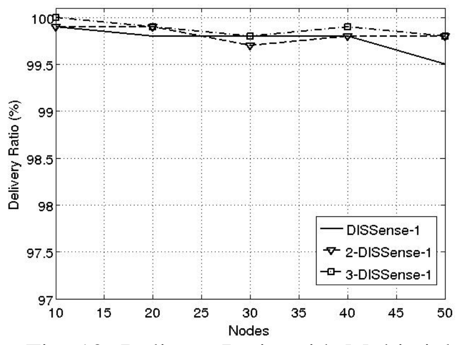

Fig. 10: Delivery Ratio with Multi-sink

within those computed by each sink, a node changing its subtree will be able to easily integrate in the new subtree. This approach requires precise synchronization between the sinks running DISSense, which can be guaranteed as the sinks are likely to be powerful, internet-enabled gateways. An alternative solution to the subtree-switching problem consists in simply letting nodes retrieve the correct schedule through the LplManager.

Fig. 9 and Fig. 10 show the duty cycle and delivery ratio reached by two multi-sink versions of DISSense compared to the single-sink one. In particular, we have selected one topology for each network size of the previous experiments and compared DISSense-1 to 2-DISSense-1 and 3-DISSense-1, where the prefix number corresponds to the number of sinks. No schedule sharing services has been adopted and thus the nodes changing their subtree retrieve the new schedule through the LplManager. As Fig. 9 shows, the duty cycle is greatly reduced for multi-sink DISSense. In particular, for the 50 nodes topology, the duty cycle decreases from 3.8% of DISSense-1 to 1.75% of 2-DISSense-1 and 1.1% of 3-DISSense-1. Fig. 10 shows that also for the multi-sink version of DISSense the DDR is always higher than 99%.

E. Comparison to Dozer [9] and Koala [10]

We finally provide a qualitative comparison between DISSense and its closest competitors: Dozer [9] and Koala [10]. In particular, we compare the performance of the three protocols in terms of data delivery ratio, latency, duty cycle, adaptability, and openness. Table III summarizes our findings.

Data Delivery Ratio: DISSense, Dozer, and Koala all show high performance in terms of data delivery ratio. In particular, the end-to-end acknowledgement mechanism of Koala enables the protocol to achieve a DDR of 99.99%. Instead, both Dozer and DISSense make use of link-layer acknowledgements thereby implicitly accepting some packet losses. In particular, DISSense automatically drops packets not delivered during the past active phase while Dozer overwrites packets matching the same origin inside forwarding queues. Nonetheless, both protocols achieve data delivery ratios of 98-99%.

Latency: DISSense has an overall lower latency than Dozer and Koala. As mentioned in V-A, latency in DISSense is upper-bounded by the length of the active period, which is typically lower than 5 seconds. Packet latency in Dozer (in the worst-case scenario) is equal to the number of hops in the collection tree times the length of the period of the TDMA schedule, the latter being fixed and equal to 30 seconds. Additionally, if an acknowledgement gets lost Dozer makes the transmitting node to stop and wait for the next TDMA round before retransmitting. This mechanism additionally increases the latency in Dozer. Koala has an overall higher latency. To keep its duty cycle short, Koala buffers packets and limits the number of time it needs to wake-up the network and perform a bulk download of the collected the data. The minimum buffer size is 32 KB [10]. Considering a 2-minutes sampling period and 35 bytes of payload per packet (as done in [9]), a 32 KB buffer would get filled in approximately 1.3 days. This value clearly increases as the length of the sampling period increases.

Duty Cycle: Dozer has an overall lower duty cycle than Koala and DISSense. As reported in [9], Dozer can achieve an average duty cycle of 0.168% on a network of 40 nodes and a 2 minutes sampling period. The actual duty cycle of each node varies depending on the role the node has in the collection tree. A node with many children needs to assign accordingly many communication slots to its children, thus incurring in a high duty cycle. In the setup described in [9] leaf nodes have a duty cycle of 0.07% while nodes with many children achieve 0.32%. If the network topology does not change frequently enough, this difference may induce significantly uneven lifetimes on different nodes of the network. Koala manages to keep the duty cycle below 1% but, as discussed above, at the cost of high data latency. In particular, waking-up the network too often (e.g., each hour) would produce a sensible increase in Koala's duty cycle due to the high energy cost of network wake-up, route computation and bulk data download that the protocol requires. DISSense's duty cycle depends on the network size and on the sampling period. As discussed previously, DISSense's performance in terms of duty cycle is comparable to that of Dozer for small network sizes (<20 nodes) or high sampling period (>5 min.). Also, DISSense induces the same duty cycle to all nodes, which translates in an homogeneous power consumption amd, hence, a predictable overall network lifetime.

Adaptability: Through its Adaptive Engine DISSense can determine the optimal schedule for the network irrespectively

	DISSense	Dozer	Koala
Duty Cycle	0.1% - 4%	0.168%	0.1% - 1%
Data Delivery Ratio	98-99%	98-99%	>99.99%
Latency	<5s	minutes	days
Platform Dependent	no	yes	no
Open Source	yes	no	yes
Adaptability	yes	no	no

TABLE III: Protocols qualitative comparison

of its size, topology, or state of the wireless channel. In particular, the Adaptive Engine autonomously collects statistics while the protocols runs and thereby determines the key protocol parameters. Instead, both Koala and Dozer depend on the a priori specification of crucial parameters. For Koala these include the probe interval for network wake-up and the buffer size, while Dozer relies on knowledge of the round period, parents update interval, overhearing phase length and frequency, and slot length.

Openness: Last but not least, DISSense and Koala are platform independent while, due to commercial agreements, Dozer is implemented on the TinyNode platform only. Also, Dozer is closed-source while DISSense and Koala are open-source protocols with publicly available implementations.

VI. CONCLUSIONS

In this paper, we described the design and implementation of DISSense, an adaptive, low-power communication protocol for WSNs-based periodical environmental monitoring applications. DISSense is easy to setup thanks to its adaptive engine that automatically updates the protocol parameters in order to minimize its power consumption. We tested DISSense on both a testbed and the TOSSIM WSN simulator. Our experimental results show that DISSense can guarantee for high data delivery and, thanks to its power-efficiency, it is able to operate a Tmote Sky-based WSN for several years.

ACKNOWLEDGMENTS

This work has been partially supported by the EU-funded project VITRO (contract number INFSOICT-257245) and by the CHIRON project (JU ARTEMIS Grant Agreement #2009-1-100228).

REFERENCES

[1] Brooke, T., Burrell, J.: From ethnography to design in a vineyard. In: Conference on Designing for user experiences, pp. 1–4. 2003.

[2] Tiano, P., Pardini, C.: Book of proceedings of the international workshop SMW08. Edizioni Firenze, Florence (2008).

[3] Beutel, J., Gruber, S., Gubler, S., Hasler, A., Keller, M., Lim, R., Talzi, I., Thiele, L., Tschudin, C., Yuecel, M.: The PermaSense Remote Monitoring Infrastructure. In: ISSW 09 Europe, pp. 187–191, Switzerland (2009).

[4] Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., Anderson, J.: Wireless sensor networks for habitat monitoring. In: 1st ACM international Workshop on Wireless Sensor Networks and Applications. 2002.

[5] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. J. Computer Networks, Vol. 38, Issue 4. 2002.

[6] Gnawali O., Welsh, M.: Sensor networks architectures and protocols. In: Emerging Wireless Technologies and the Future Mobile Internet, Chapter 5, Cambridge University Press, 2011.

[7] van Dam, T., Langendoen, K.: An adaptive energy-efficient MAC protocol for wireless sensor networks. In: 1st international Conference on Embedded Networked Sensor Systems, pp. 171–180. 2003.

[8] Moss D., Levis P.: BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking. Technical Report SING-08-00, Stanford University (2008)

[9] Burri, N., Von Rickenbach P., Wattenhofer R.: Dozer: ultra-low power data gathering in sensor networks. In: 6th international Conference on information Processing in Sensor Networks, pp. 450–459. 2007.

[10] Musaloiu-E., R., Liang, C. M., and Terzis, A.: Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks. In: 7th international Conference on information Processing in Sensor Networks, pp. 421–432. IEEE Computer Society, Washington, DC (2008).

[11] Rhee, I., Warrier, A., Aia, M., and Min, J.: Z-MAC: a hybrid MAC for wireless sensor networks. In: 3rd international Conference on Embedded Networked Sensor Systems, pp. 90–101. 2005.

[12] Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., and Levis, P.: Collection tree protocol. In: 7th ACM Conference on Embedded Networked Sensor Systems, pp. 1–14. 2009.

[13] TinyOs Home Page, http://www.tinyos.net

[14] TinyNode Home Page, http://tinynode.com

[15] Maróti, M., Kusy, B., Simon, G., and Lédeczi, Á. The flooding time synchronization protocol. In: 2nd international Conference on Embedded Networked Sensor Systems, pp. 39–49. ACM, New York (2004)

[16] Memsic Wireless Modules,http://www.memsic.com

[17] Tossim Tutorial, http://docs.tinyos.net/index.php/TOSSIM

[18] Rowaihy, H., Eswaran, S., Johnson, M., Verma, D., Bar-Noy, A., Brown, T., La Porta, T.: A survey of sensor selection schemes in wireless sensor networks. In: SPIE Defense and Security Symposium Conference on Unattended Ground, Sea, and Air Sensor Technologies and Applications IX, vol. 6562, 2007.

[19] Hasan Cam, Suat Ozdemir, Prashant Nair, Devasenapathy Muthuavinashiappan, H. Ozgur Sanli, Energy-efficient secure pattern based data aggregation for wireless sensor networks, Computer Communications, pp. 446–455, Volume 29, Issue 4, 2006.

[20] Sang Hyuk Lee, Soobin Lee, Heecheol Song, Hwang Soo Lee: Wireless sensor network design for tactical military applications : Remote large-scale environments.In: Military Communications Conference.2009.

[21] Molina-Garcia, A., Fuentes, J.A., Gomez-Lazaro, E., Bonastre, A., Campelo, J.C., Serrano, J.J.: Application of Wireless Sensor Network to Direct Load Control in Residential Areas. In: Industrial Electronics.2007

[22] Polastre, J., Szewczyk, R., Culler, D.: Telos: enabling ultra-low power wireless research. In: Information Processing in Sensor Networks.2005.

[23] Yick, J., Mukherjee, B., Ghosal D.: Wireless sensor network survey. Computer Networks, Volume 52, Issue 12, 22 August 2008

[24] Ceriotti, M., Mottola L., Picco G.P., Murphy, A.L., Gun S., Corr M., Pozzi M., Zonta D. and Zanon P.: Monitoring Heritage Buildings with Wireless Sensor Networks: The Torre Aquila Deployment. In: 8th Int. Conf. on Information Processing in Sensor Networks. 2009.

[25] Barrenetxea, G., Ingelrest, F., Schaefer, G. and Vetterli, M.: Wireless Sensor Networks for Environmental Monitoring: The SensorScope Experience. In: 20th IEEE International Zurich Seminar on Communications (IZS). 2008.

[26] Duracell Alkaline Technical Bulletin. Chapter 5. p.9. http://www1.duracell.com/oem/Pdf/others/ATB-full.pdf

[27] El-Hoiydi, A. and Decotignie, J. D.: WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks. In: 9th International Symposium on Computers and Communications. 2004.

[28] Langendoen, K. and Meier, A.: Analyzing MAC protocols for low data-rate applications. In: ACM Transaction on Sensor Networks. New York (USA). 2010.