A Secure Middleware for Wireless Sensor Networks *

Claudio Vairo
Department of Computer Science
University of Pisa
Largo B. Pontecorvo, 3
56127, Pisa, Italy
vairo@cli.di.unipi.it

Michele Albano
Department of Computer Science
University of Pisa
Largo B. Pontecorvo, 3
56127, Pisa, Italy
michele@di.unipi.it

Stefano Chessa*
Department of Computer Science
University of Pisa
Largo B. Pontecorvo, 3
56127, Pisa, Italy
ste@di.unipi.it

ABSTRACT

SMEPP Light is a middleware for Wireless Sensor Networks (WSNs) based on mote-class sensors. It is derived from the specification developed under the framework of the SMEPP project, to deal with the hardware and software constraints of WSNs. SMEPP Light features group management, group-level security policies, mechanisms for query injection and data collection based on a subscribe/event mechanism, and adaptable energy efficiency mechanisms. In this paper we present the SMEPP Light specification, its architecture and its main protocols for communication, security, and energy efficiency.

General Terms

Wireless Sensor Networks, Middleware, Embedded Systems, Energy Efficiency

1. INTRODUCTION

A Wireless Sensor Network (*WSN*) is a network composed by a large number of tiny, low-power, inexpensive sensors which self organize into a (multi-hop) ad hoc network [1]. A sensor is a micro-system that comprises a processor, a small memory, one or more sensing units (*transducers*) and a radio transceiver. Sensors are spread in an environment (*sensor field*) without any predetermined infrastructure and cooperate to realize an highly distributed application, whose goal usually consists in sensing environmental data and monitoring a variable set of parameters.

WSNs are flexible enough to be applied in diverse application areas: a WSN can be the architectural support for Ambient Assisted Living applications, that span from health monitoring of patients to independent aging [2]. A "smart house" is usually made up of several intelligent devices that can control home appliances or "feel" their surroundings [3]. Other applications are the industrial plants monitoring, for instance nuclear plants, or monitoring of disaster areas. Despite the different nature of these applications, they share many common mechanisms. These mechanisms can be classified according to the functional requirements they satisfy, namely: network organization, and management mechanisms, security mechanisms, and query dissemination and data collection mechanisms. However, very often (and motivated by the constraints of the sensors in terms of memory and processing capabilities), the software of the sensors is generally a piece of monolithic code encapsulating all the functions belonging to all layers, making thus difficult the re-use of solutions developed for other applications.

One of the critical points to leverage on the potential usefulness of WSNs is the possibility of abstracting common WSNs problems by means of convenient middleware to support the applications development and maintenance, filling the gap between the applications and the network stacks. Using a middleware that provides the right primitives, the application developer can focus on the application business logic, not having to implement the layers that provide the access to the hardware and the networking mechanisms.

This paper describes SMEPP Light, a middleware for WSNs that derives from the Secure Middleware for Embedded Peer-To-Peer (*SMEPP*) project [4],[5]. SMEPP aims at hiding the complexity of the underlying infrastructure while providing open interfaces to third parties for secure application development. SMEPP is specially designed for Embedded P2P scenarios (EP2P) and its objective is to overcome the main problems of existing proposals in domain specific middleware for EP2P: the middleware has to be secure, generic and highly customizable, allowing for its adaptation to heterogeneous devices (from PDAs to embedded sensor/actuator systems) and domains (from critical systems to consumer entertainment).

SMEPP Light is the version of SMEPP tailored for WSNs. This because a sensor can hardly face the technical problems arising in the implementation of the whole SMEPP specification. For this reason SMEPP Light addresses a limited but yet significant and coherent subset of SMEPP. In particular, differently from the full SMEPP specification, SMEPP Light does not support services, on the other hand it provides the same network organization mechanisms and it relies on the SMEPP eventing mechanisms for query dissemination and data collection. These mechanisms are implemented internally according to the directed diffusion paradigm [6].

*Work funded in part by the European Commission in the framework of the SMEPP project (contract N. 033563).
*Also affiliated with ISTI-CNR, Pisa

MobiQuitous 2008 July 21-25, 2008, Dublin, Ireland

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3888
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3888

2. RELATED WORK

The problem of defining a middleware for WSN has been addressed in many recent works. TinyLime [7] defines the middleware based on the concept of tuple spaces where data is represented by elementary data structures called tuples, and the memory is a multiset of tuples called a tuple space, that is shared among all the sensors. Another proposal, TCMote [9], is based on tuple channels. A tuple channel is a FIFO structure that allows one-to-many and many-to-one communication of data structures, represented by tuples. By using the tuple channels the applications can decide which application components exchange data with each other. In this way the communication overhead can be optimized. A different approach, based on the notion of events, is employed by the Mires middleware [10]. This middleware exploits a publish/subscribe paradigm to let the applications specify interests in certain state changes of the real world. Upon detecting such an event, a sensor node sends a so-called event notification towards interested applications. All of these approaches however do not address the issues related to security. This aspect significantly differentiates our work, since our approach inherits from SMEPP the security concepts related to the use of groups, that can be used to build secure multicast and unicast communications with a fine granularity.

A different approach to WSN middleware is given by the ZigBee standard [1], [8]. At the network layer it has an inherently node centric behavior, but it offers service-oriented mechanisms to the applications. The ZigBee specification includes mechanisms aimed at limiting the sensors duty cycle, which however are configurable at network creation and that can not be adapted dynamically to the application. With respect to ZigBee the aim of SMEPP Light is more focused to monitoring/control applications. With this restriction the number and kind of functionalities to be offered to the applications are rather different. In particular SMEPP Light seeks for a reduced core of primitives, which includes also mechanisms for query dissemination and data aggregation that are not primitive in ZigBee, and it offers energy efficiency mechanisms that dynamically adapt to the applications' needs. Another difference of SMEPP Light with respect to ZigBee is that SMEPP Light offers security mechanisms based on groups to implement fine-grained secure communications.

3. HARDWARE AND SOFTWARE TARGET PLATFORMS

SMEPP Light targets a Mote-class hardware platform for sensors. A typical example of sensors in this class is the Crossbow MicaZ sensor [11]. It has an 8 bit, 8 MHz processor, 128 KBytes of program memory, 4 KBytes of RAM and 512 KBytes of storage memory.

SMEPP Light is developed on top of the TinyOS operating system (that is the de-facto standard for this class of sensors). A TinyOS application is a set of components linked together to form an executable. Each component consists of an interface and its implementation. The interface specifies a set of commands implemented by the component and a set of events that the component can signal. Hereafter, to avoid confusion with the term event that is also used for other purposes in SMEPP Light, we will call signals the TinyOS events. TinyOS relies on the concept of split function for energy saving. Split functions are functionalities split into pairs command/signal: when a component A invokes a split function of another component B implemented by a command, the command enqueues the request and returns immediately. Only when the result is available B generates the corresponding signal to give the result to A.

4. REQUIREMENTS

The main feature of SMEPP (and thus of SMEPP Light) is that the peers in the same network organize themselves into groups. The existence of different groups is useful since it enables the definition of different security and communication domains, that involve only peers owning the appropriate credentials. This is, for example, quite different from ZigBee where the security domain is either the whole network or an individual communication link between two peers.

The peers of a group in SMEPP Light interact via a publish/subscribe mechanism. A peer can subscribe for events of other peers that belong to the same group, so that it automatically receive the relevant events whenever they become available. From the security point of view, a group can be open or closed, private or public. In closed groups, a key is necessary to access the group, while the appropriate key is necessary to discover private groups.

SMEPP Light also provides a two-level security based on symmetric cryptography: network-level and group-level. The network-level security exploits two keys: one for packets' confidentiality (used to encrypt the packet) and one for packets' integrity (used to compute a MAC to be attached to the packet). The two keys are set by the application and can be changed at run time. The group-level security exploits three keys, namely the masterKey, the sessionKey, and the sessionMAC. The masterKey is used to restrict the access to the closed group, so that a peer can join a closed group only if it owns the right masterKey. Once a peer joins a group it receives the sessionKey and the sessionMAC. These keys are used to enforce data confidentiality and data integrity in all the communications within the group, and they can be changed at run time, however the masterKey must be known in advance and is set at compile time. Since a peer can join several groups, in all its communications it must specify in clear text the identifier of the group to which the message is directed, so that each peer receiving that message can use the right key to check the message integrity and to decrypt it.

Another important requirement of SMEPP is that each peer must be described by an XML document. In SMEPP Light, a peer description also comprises the list of the transducers the sensor is equipped with. The peer description is compressed into a bitmask that the sensors can easily store and exchange.

For energy management purposes, each group defines its own duty cycle that drives the radio activity of the peers. Thus each peer in the group operates the energy management according to this duty cycle (e.g., it responds to the messages only when it is active). However, any received subscription can request the peer to use also another duty cycle for environmental sampling. For this reason SMEPP Light manages all the duty cycles (which should coexist) by turning on or off the peer (and thus the radio) whenever necessary. This policy of energy management is rather different from the policies used by ZigBee. In fact, in the case of ZigBee, the duty cycle is defined a priory by the

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3888
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3888

network coordinator and routers (normally according to the expected behavior of the application), and it is common to all the network.

Table 1: Interface of SMEPP Light

command peerId smepp_newPeer(netwKey, netwMAC)
command groupId smepp_createGroup(groupDescription) command smepp_getGroups(groupDescription) signal getGroups_result(groupId[]) command smepp_getGroupDescription(groupDescription) command smepp_joinGroup(groupId, masterKey) signal peerJoined(peerDescription) signal joinGroup_result(groupId, subscriptions[], result) command peerDescr[] smepp_getPeers(groupId) command smepp_leaveGroup(groupId) signal peerLeft(peerId, groupId)
command smepp_subscribe(eventName, groupId, expirationTime?, rate?) signal subscribed(eventName, groupId, expirationTime, rate, offset) command smepp_unsubscribe(eventName?, groupId) signal unsubscribed(eventName, groupId) command smepp_event(groupId, eventName, value) command smepp_receive(groupId, eventName, frequency) signal receive_result(sender, groupId, eventName, value)

5. SMEPP LIGHT SPECIFICATION

Interface – SMEPP Light provides primitives for peer initialization, group management, and event transmission. The set of the (main) primitives is shown in Table 1. The peer initialization is executed by means of the primitive `smepp_newPeer`. It takes in input the network and the MAC keys (hence these keys are established by the application) and returns to the application the peer identifier. This identifier is unique within the network and corresponds to the sensor identifier used by TinyOS and assigned at compile time to the sensor.

The group management primitives support the creation of groups, the search for existing groups and the join to existing groups. The `smepp_createGroup` primitive creates a group according to the group description taken in input. The description contains the security keys of the group and a set of flags expressing the group security policies in terms of closeness and privacy.

The primitives for group discovery are used to retrieve groups that match a certain search criteria. The command `smepp_getGroups` accepts a group description partially filled up, and returns, via the signal `getGroups_result`, the id of matching groups. Then `smepp_getGroupDescription` is used to read the group descriptions. Group discovery can be based on group name or on the security properties.

The command `smepp_joinGroup` takes in input the master key of the group to be accessed. The result of the join protocol (either success or failure) is returned to the application by means of the `joinGroup_result` signal. If the join is successful, the peer also receives the session keys used for the communications in the group, the list of the peers in the group, and the list of the subscriptions that are active within the group. The list of subscriptions is also notified to the application layer that can thus begin raising any relevant event. Furthermore, as a result of the join protocol, all the peers in the group are notified with the signal `peer_joined` reporting the identifier of the newly joined peer. The descriptions of the peers into the group can be accessed via the `smepp_getPeers` primitive.

The command `smepp_leaveGroup` is a split function that enables a peer to leave a group. This disassociation is automatically notified to all the peers in the group with the `peerLeft` signal.

The main event management primitives offer functionalities for event subscription and event notification. The event subscription can be invoked by any peer in a group and is issued to all the peers in the group. When one of the peers detects an event matching the subscription it sends the event back to the peer (or the peers) that subscribed for it. The command `smepp_subscribe` takes in input the name of the event to be subscribed and a group id to which the subscribe is directed. The event name may encode an arbitrary monitoring task to be run on the peers in the group. The primitive also takes two optional parameters: the expiration time and the rate of the subscribe. The rate defines the sampling rate of the monitoring task associated to the event name, which also implies the maximum rate at which the events can be sent back to the subscriber. Each subscribe results in the creation of a routing tree spanning on all the peers in the group and rooted in the subscriber. This tree is used to route the events to the subscriber. After the expiration time the subscribe (and consequently the associated routing tree) expires and the subscriber should issue again another subscribe if it is still interested. The presence of a subscribe request is notified to the application layer of all the peers in the group by means of the `subscribed` signal that provides to the application layer the parameters of the subscription (event name, group id, rate, expiration time) and an offset time, that is the time in which the subscribe has been generated and that is used to synchronize all the peers in the group. After the invocation of the command `smepp_subscribe`, the subscriber can invoke the command `smepp_receive` to start waiting for the corresponding events.

If a peer detects an event matching a subscribe, it sends the event to the subscriber by using command `smepp_event`, that routes the event using the routing tree constructed by the subscribe. When the event reaches the subscriber, SMEPP Light provides the event to the application layer by raising the `receive_result` signal, that provides to the application layer the value associated with the event along with the event name, the identifier of the peer that detected the event, and the identifier of the group where the event was detected.

Architecture – SMEPP Light is composed by three main components, namely the Peer Identification, Group Management and Event Management, that implement the SMEPP Light primitives, and three components that provide support to security, networking, and energy efficiency. The interaction among these components is shown in Figure 1.

The Peer Identification component maps to the peer initialization primitives and it interacts with the Security component to set the network keys.

The Group Management component manages the topologies of the groups and maps to the group management primitives. It is in charge of the group descriptions and duty cycle management and it interacts with all the support components: it sets the group master key, it sends and receive data from the Network component when most of the primitives are executed, and it interacts with the Energy Efficiency

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3888
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3888

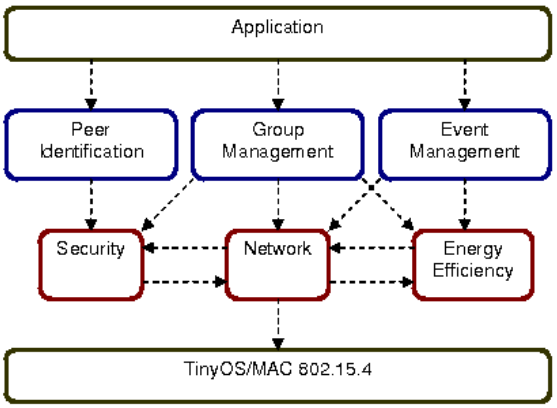

Figure 1: Components in the SMEPP Light architecture

component to set the information for the management of the peers' duty cycle.

The Event Management component maps to the event management primitives and it is in charge of subscriptions and events. This component interacts with the Network component to access the wireless medium and to set up the routing trees associated with subscriptions, and it interacts with the Energy Efficiency component to configure the duty cycle of the peer according to the subscribes generated or received.

The Security component manages the keys for all the security issues related to the network and to the group layers. At the current stage of development it keeps the network keys set by the Peer Identification component, the group master key set by the Group Management component, and the group session keys received from the Network component during the join protocol. This component will also manage the (planned) protocols for the dynamic refresh of the session keys.

The Network component implements the communication between peers. Its main mechanisms are the network broadcast used to implement the subscribe and the management of the routing trees associated to the subscribes. It also provides one-hop broadcast used to implement the group discovery and the join mechanisms.

The Energy Efficiency component manages the duty cycle of the peer. In particular it manages the the on/off periods of the radio interface according to the duty cycles associated to the subscribe messages received or generated by the peer. It should be observed that the management of the radio is transparent to the other components, since this component makes sure that the radio is activated before it is used by other components and that it is turned off soon after its use. Details on how this is implemented are reported in Section 6.

Protocols – For the sake of brevity we describe only the main protocols used by the SMEPP Light middleware, i.e. the group creation/join protocols (Figure 2) and the subscribe/event protocols. We illustrate the protocol of group creation and join referring to the diagram of Figure 2, where it is shown the case where Node B creates a group and Node A joins the group of B. Node B creates the group using the `createGroup` command. The group creation does not involve communications since it consists in setting a few data structures in SMEPP Light and in setting the master and the session keys in the B's Security component, hence it can immediately provide the result of the operation without resorting to the split function mechanism. The application layer of Node A performs the search for existing groups by invoking the `getGroups` command. This command sends to all the A's neighbors (in local broadcast) a message requesting the group descriptions of the existing groups. At Node B, SMEPP Light replies to this request (without involving the application layer) by sending to A the descriptions of all the groups known to Node B (in this case only one). At Node A SMEPP Light keeps all the received descriptions and after a timeout it notifies to the application layer the list of identifiers of all the groups detected. The application at Node A can then choose a group ID and it can access the corresponding group description by invoking the `getGroupDescription` primitive.

Now the application layer of Node A can invoke the `joinGroup` primitive to join the group. As a consequence, SMEPP Light sends in unicast to Node B the request to join the group. This message is notified to SMEPP Light in Node B, that, in turn, decides whether to accept the request of A. In our case the request is accepted, hence SMEPP Light in Node B sends to A a set of messages containing the session keys, the list of peers belonging to the group, and the list of subscriptions that are currently active into the group. All of these messages are used by SMEPP Light in Node A to update its internal data structures, and once this phase is completed SMEPP Light raises a `joinGroup_result` signal to the application layer of the same node to notify that the join protocol is completed. In the meantime, SMEPP Light in Node B sends in broadcast to all the other peers in the group a message notifying that Node A joined the group.

The subscribe protocol is initiated by any peer in a group that wants to receive a given type of events generated by other peers in the group. Consider for example the case where Node A subscribes for events that are generated by a Node B. To this purpose Node A invokes a `subscribe` command, that broadcasts the subscribe to all the peer in the group. Then Node A invokes the `receive` command to prepare SMEPP Light in Node A to receive events related to this subscribe. When Node B receive the subscribe message, SMEPP Light in Node B notifies this request to the application layer by means of the `subscribed` signal. This signal gives to the application layer the event name for which the subscribe holds, hence it is responsibility of the application to start any relevant monitoring task to detect the events matching the subscribe. This monitoring task should be activated at the sampling rate contained in the subscribe message. Whenever the application in Node B detects an event it sends the event to the subscriber using the `event` command. This primitive sends a message containing the event to SMEPP Light in Node A, that, in turn, notifies the application in Node A by means of the `receive_result` signal. This protocol continues until the subscribe expires or Node A cancels it using the `unsubscribe` command.

If a primitive fails, the middleware notifies the error to the application to let it implement fallback policies. There are however some exceptions to this general behavior when the middleware can not identify the fault. One example is the removal of a peer without the `leave` primitive. This case may

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3888
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3888

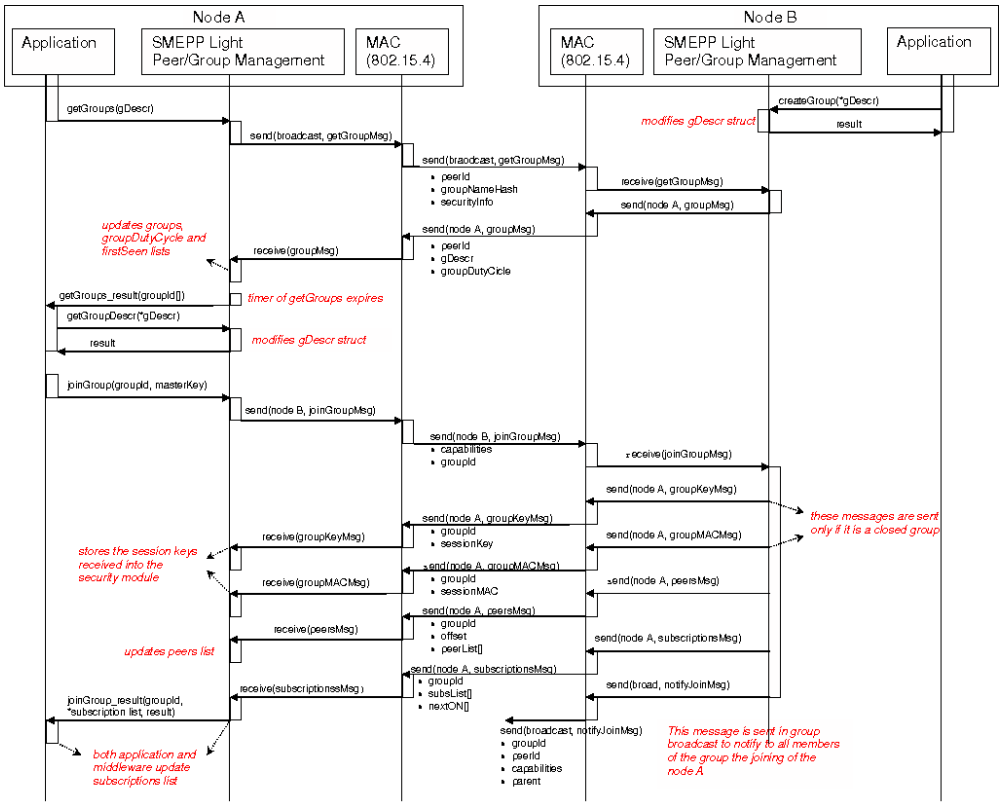

Figure 2: Creating a group

result in broken routing trees, stopping data flows related to some subscribe. SMEPP Light copes with this problem via regular refreshing of subscriptions, as also does directed diffusion [6]. There are also some situations that can not be coped with, for example when the sudden removal of a peer results in a group partition. In this particular case, the group splits up, but the peers that remain connected continue working without the sensors in the partitioned branch.

6. ENERGY EFFICIENCY

As anticipated in Section 5 the Energy Efficiency component of SMEPP Light saves sensors' energy by keeping the radio off whenever possible, i.e. when the sensors do not expect to receive or send data. This component manages the radio by means of *user* duty cycles that are implicitly determined by the subscribe messages (recall that these messages contain the subscribe rate and an expiration time), and a *management* duty cycle that enables the sensors in the group to exchange control messages (in particular join and subscribe messages). Each of these duty cycles defines periodic intervals when the radio should be turned on by all the sensors.

As a side-effect of this approach, when a sensor executes the `getGroups` primitive it needs to keep sending request messages until one of these messages is sent during a period of activity of a group. However during the `getGroups` protocol the sensor receives enough information to synchronize with the other sensors of the group, so the next communications happen only according to the *management* duty cycle of the group. To make this approach effective, the sensors need to be (weakly) synchronized, for this reason synchronization information is periodically exchanged among the peers in the group.

Figure 3 shows an example of the status of the radio of a node that belongs to a group in which are active two subscribes (subs1 and subs2). The three lines on the top show the activity windows of the radio for the *user* duty cycles corresponding to the two subscribes and for the *management* duty cycle. The last line shows the overall radio activity. The Energy Efficiency component calculates the union of all the duty cycles, and decides when the radio should be turned off and on, according to two parameters that provide some tolerance to the system. One parameter specifies the minimum distance (in milliseconds) between the end of a window of radio activity and the start of the next one: if two windows are too close the radio is kept on until the second window ends. The other parameter expresses the time used to anticipate and delay the radio commutations from off to on and from on to off, respectively.

The performance of the energy efficiency mechanism was evaluated by measuring the periods of radio activity of a sensor. In the experiments we used 4 MicaZ motes [11] (s1,s2,s3,s4) connected in a line, i.e. s1 is connected to s2, s2 to s3 and s3 to s4. We measured the radio activity on node s2, s1 is the node that produced the subscribes and s4

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3888
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3888

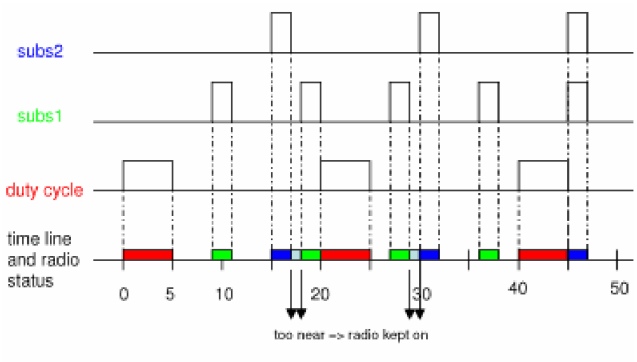

Figure 3: Duty cycles and subscriptions

produced events. We repeated four sets of experiments with a number of subscribes ranging from 0 to 4. The rate of each subscribe was set randomly in each experiment. Each experiment was repeated 10 times for 180 seconds, and in each experiment was measured the average period of time in which the radio of sensor s2 was in the state of off, ready, receive and send. From these data we computed the average energy consumption of sensor s2 (expressed in mA-hr) in all the set of experiments, as shown in Figure 4. For a comparison the figure reports the energy consumption estimated with the TOSSIM simulator and the energy consumption in the case where the energy efficiency is disabled. From the figure it is seen that the energy efficiency strategy enables significant energy savings, and that the energy consumed grows sublinearly with the number of subscribes.

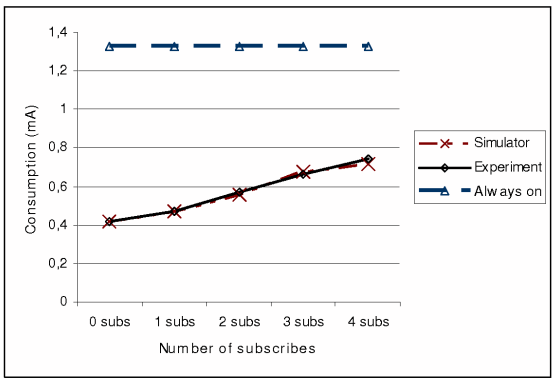

Figure 4: Energy consumption (in mA-hr)

7. CONCLUDING REMARKS

We presented the main features of SMEPP Light, a middleware for the organization, management and query of WSNs. In particular we described the main requirements that are at the base of the SMEPP Light design, namely the sensors organization in terms of groups and the security model also based on groups, the query injection and data collection based on a subscribe/event model, and the energy efficiency strategy based on this model. We performed some preliminary measurement on the performance of the energy efficiency strategy that shows that SMEPP Light can significantly contribute to energy savings. Future work includes the extension of SMEPP Light to a service oriented interaction model, and mechanisms for the dynamic refresh of the security keys.

8. REFERENCES

[1] P. Baronti et. Al. "Wireless Sensor Networks: a Survey on the State of the Art and the 802.15.4 and ZigBee Standards", Computer Communications, 30 (7), May 2007, pp. 1655-1695.

[2] EU FP6 IP project "PERSONA", http://www.aal-persona.org

[3] S. K. Das and D. J. Cook ,"Designing and Modeling Smart Environments", WoWMoM'06

[4] EU FP6 "SMEPP" project, http://www.smepp.org/

[5] M. Albano et. Al., "Towards Secure Middleware for Embedded Peer-to-Peer Systems: Objectives and Requirements", RSPSI 07, Innsbruck (Austria),2007

[6] Intanagonwiwat et. Al. "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks", MobiCom 2000, Boston, pp. 56–67

[7] C. Giani et. Al. "Data collection in Sensor Networks: the TinyLime Middleware", J. of Pervasive and Mobile Computing, 4(1), pp. 449–469

[8] ZigBee Specifications 2006, http://www.ZigBee.org

[9] M. Díaz et. Al. "A Coordination Middleware for Wireless Sensor Networks", IEEE SENET 05, Montreal, Aug 05, pp. 377–382.

[10] E. Soutoet. et Al. "A Message-Oriented Middleware for Sensor Networks", MPAC 04, Toronto, Oct. 04, pp. 127–134,

[11] Crossbow Technology - www.xbow.com.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3888
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3888