Software Defined Wireless Networks: Unbridling SDNs

Salvatore Costanzo, Laura Galluccio, Giacomo Morabito, Sergio Palazzo

DIEEI – University of Catania – Catania, Italy

***Abstract*—The *software defined networking* (SDN) paradigm promises to dramatically simplify network configuration and resource management. Such features are extremely valuable to network operators and therefore, the industrial (besides the academic) research and development community is paying increasing attention to SDN. Although wireless equipment manufacturers are increasing their involvement in SDN-related activities, to date there is not a clear and comprehensive understanding of what are the opportunities offered by SDN in most common networking scenarios involving wireless infrastructureless communications and how SDN concepts should be adapted to suit the characteristics of wireless and mobile communications. This paper is a first attempt to fill this gap as it aims at analyzing how SDN can be beneficial in wireless infrastructureless networking environments with special emphasis on wireless personal area networks (WPAN). Furthermore, a possible approach (called *SDWN*) for such environments is presented and some design guidelines are provided.**

I. Introduction

Software Defined Networking (SDN) promises to dramatically reduce the complexity of network configuration and management as well as to make the introduction of innovation in the network operations possible.

Network operators envision in such features the possibility to introduce *evolvability* in their networks, which in their perspective means the possibility to differentiate their offer when compared to other operators, and to improve network efficiency given that new, more efficient technical solutions can be easily deployed in existing equipment. Accordingly, SDN design and experimentation is the subject of the increasing attention of the industrial and academic research communities. The deployment of large SDN testbeds, such as the one realized by the OFELIA consortium (http://www.fp7-ofelia.eu/) and the institution of large industry-driven organizations focused on SDN such as the Open Networking Foundation (ONF) (https://www.opennetworking.org/) witness such increasing interest.

The SDN has interested the wireless networking community as well[1]. In fact, an increasing number of enterprises working in the field of wireless and mobile communications have joined SDN-related initiatives. For example, Verizon, Nokia Siemens Networks, Ericsson, and Netgear are current members of ONF.

Regardless of such increasing interest, to the best of our knowledge, there is not a clear and comprehensive understanding of what are the advantages of SDN in the most common wireless infrastructureless networking scenarios and how the SDN concept should be expanded to suit the characteristics of wireless and mobile communications [2], [5].

This paper represents an attempt to achieve such understanding in case of IEEE 802.15.4-based *low rate, wireless personal area networks* (LR-WPANs) [9]. In fact, major contribution of this work are the following:

- We will analyze the advantages of the SDN approach in LR-WPANs.
- We will identify the major differences between the system requirements that should be taken into account in traditional wired networks and LR-WPANs.
- We will present the solution – called *Software Defined Wireless Network* (SDWN) – we are developing to support the SDN approach in LR-WPANs.

The rest of this paper is organized as follows. In Section II we will elaborate on the opportunities which can be achieved by applying the SDN paradigm to wireless infrastructureless networks, while in Section III we will illustrate how the requirements to be considered in LR-WPANs scenario differ from those utilized to design SDN solutions for wired networks. We have used such requirements to develop our *Software Defined Wireless Network* (SDWN) protocol stack which is briefly introduced in Section IV. In Section V design and implementation details will be discussed. Finally, in Section VI we will draw our conclusions.

II. The opportunities of SDWN

Let any false hope (which might have been turned on in the fantasy of wireless networking scientists approaching SDN for the first time) be immediately removed from the table by saying that:

> *There is nothing you can do with SDN which cannot be done without.*

Indeed, it has been extensively discussed and stressed that SDN is not about performance improvements. On the contrary, it might even be that some performance is lost given that higher levels of abstraction are introduced. Rather, SDN is about *simplification* and *evolvability*. New network control and management solutions can be easily deployed on existing equipment, ideally, as simply as it is to install new programs on a computer. Accordingly,

[1]Actually the idea of delegating the management of even the lowest layers of the protocol stack to software programs has been introduced in the wireless domain long time ago through the *Software-Defined Radio* concept [7].

978-0-7695-4870-8/12 $26.00 © 2012 IEEE
DOI 10.1109/EWSDN.2012.12

- if new and more efficient control and management solutions become available, these can simply be installed to replace the old ones;
- if the context in which the network operates changes and consequently the requirements change, then other control and management solutions can be deployed that are more effective for the new conditions.

As a consequence, higher efficiency of network equipment can be achieved on the long term. However, again, the same efficiency could be achieved by replacing the old network equipment with new one implementing the desired new functionalities or, if supported by the equipment, by deploying the new solutions in each relevant piece of the network.

The above claims are valid for any networking environment and therefore for wireless infrastructureless networks, such as the wireless personal area networks (WPANs), as well. However, in wireless infrastructureless networking environments the separation of the network control and management functionality from the forwarding operations (as envisioned in SDNs) offers further possibilities. In fact, while there is almost total consensus about the technical solutions that should be used at the first two layers of the protocol stack, at higher layers and in the management plan several candidates are available and networks deployed for different scenarios are likely to use different alternatives.

For example, in wireless sensor networks there is a universal consensus on the access technology to be used – which should be based on IEEE 802.15.4 – but there is still a debate going on for the characterization of the higher layers of the protocol stack. In fact, ZigBee and 6LOWPAN are very popular alternatives which are not compatible with each others.

Similar discussions could be carried out about vehicular networks for which the *Wireless Access Vehicular Environment* "WAVE" has been standardized as IEEE 802.11p or about wireless mesh networks for which the IEEE 802.11s amendment has been standardized. In fact, in both environments there is no consensus on the routing protocols[2] to be used as well as on most of the network management operations to be performed.

As a consequence, it may happen that nodes applying the same access technology utilized in a given network cannot enter a new network because of differences at the higher layers of the protocol stack. This gives strict limitations to the possibility for nodes to migrate from network to network.

Such problem can be easily solved by applying the SDN paradigm which envisions that the functionality performed at the network and higher layers of the protocol stack are defined through software and can be changed easily and "on the fly".

In fact, in SDN the network management operations are centralized – at least in the earliest implementations – and physically separated from the forwarding operations. Indeed, in SDN there are network nodes (*SDN switches*) that are responsible for classifying packets according to some *rules* and performing the corresponding *actions*[3]. The above nodes can be distinguished from others (the *Controllers*) that are responsible for setting the *rules* and *actions*. A network node that has no information available to classify an incoming packet, requests the assistance from one (or several) of the Controllers which should provide an appropriate rule/action pair based. Note that the policies applied by Controller to set the rule/action pair (which in the end define the entire network behavior) can be easily and rapidly modified by *installing* a new Controller software.

As it is evident from the above discussion, SDN "enforces" a centralized approach to network control. This raises crucial problems about which network element(s) should run network control operations but makes network optimization easier – as all relevant information could be made available to the network Controller – and therefore may provide several advantages as will be discussed in the following.

For all the above reasons we believe that the extension of the SDN paradigm to WPANs, which we call Software Defined Wireless Networking (SDWN) paradigm, can have significant impact on such networking environment.

III. Requirements for the SDWN

Implementations of the SDN solutions for traditional wired networks have considered velocity as the major performance measure. Indeed, it is necessary to guarantee that SDN nodes can execute switching operations at line rate. Satisfaction of such necessity has been paid in terms of low flexibility in the definition of the rules specifying the flows, like it will be further discussed later.

In LR-WPAN networking scenarios constraints about the velocity can be relaxed. In fact, communications in LR-WPANs occur at low rate by definition. On the contrary it is extremely important to guarantee low energy consumption. By looking at the scientific literature it becomes clear that reduction of energy consumption can be achieved in several ways [1]. Among them, SDWN uses duty cycles [3], in-network data aggregation [4], and flexible definition of rules and actions to allow cross-layer optimization.

Accordingly, the new requirements which have been considered in the design of SDWN are given below:

SDWN must support duty cycles – The most obvious way for reducing the energy consumption is turning the radio off when it is not utilized, that is, to use *duty cycles*. The radio interface of generic nodes is turned off periodically. This would result in topology modifications which should be considered by the modules that are responsible for network control. Obviously control of the duty cycle requires appropriate primitives. To this purpose, SDWN defines an appropriate *action*, like we will explain in Section V-B.

SDWN must support in-network data aggregation – Energy consumption can be reduced by removing the redundancy

[2]Note that IEEE 802.11s identifies the *Hybrid Wireless Mesh Protocol* (HWMP) – which is derived by the AODV – as the default routing protocol; however, vendors are allowed to implement their own solutions.

[3]Here we derive our terminology from OpenFlow [6], which is to date the most popular SDN implementation.

from the data circulating in the network, that is, by using *data aggregation* techniques. In fact, it is well known that in many relevant scenarios data circulating in the network is highly correlated both in time and space [8]. Support of data aggregation functionalities is achieved by SDWN through an appropriate module in the protocol architecture and the definition of a new action like the one described in Sections IV and V-B, respectively.

SDWN must support flexible definition of the *rules* – OpenFlow supports the definition of rules which consider the traditional TCP (or UDP)/IP header fields only. This allows a definition of switching and routing strategies which are much more flexible when compared to traditional switching/routing strategies. However, analysis of the literature suggests that higher flexibility is required in wireless sensor and actor networks. In SDWN higher flexibility is achieved along two orthogonal dimensions.

In fact, on the one hand SDWN allows to define rules which consider any byte in the packet (obviously there are some limitations like we will explain in Section V-B) for matching purposes. This would allow, for example, to route packets based on the specific value in the payload they carry (which would be extremely useful in several wireless sensor and actor networking scenarios). On the other hand, SDWN allows to define rules in which matching must not be necessarily conditioned by the equality between the bytes to be considered in the packet and some reference values. In fact, SDWN allows to define rules in which matching can be conditioned by other relational operators. For example, a route can be configured for packets in which the value contained in the payload is higher than a given threshold while another route is configured for packets in which the above value is below the threshold.

Again several application scenarios can be figured out in which the above flexibility is extremely useful.

Other requirements – There are several other obvious new requirements that must be satisfied, which we will not specifically analyze for space constraints. Examples include the necessity to support nodes mobility and the resulting topology changes, the necessity to deal with the unreliability characterizing wireless links, and the need to be robust to the failure of generic nodes and the Control node.

IV. SDWN PROTOCOL ARCHITECTURE

In this section we show the protocol architecture we are developing for a sensor and actor network based on IEEE 802.15.4 communication nodes. Besides the transceiver a micro-control unit is deployed in such nodes with limited computing and energy capabilities. In the network there is also one (or several) sink node(s)[4] which is (are) also the node(s) where the network Controller is executed. The IEEE 802.15.4 transceiver of the sink is connected to an embedded system running, for example, a Linux operating system. Computing/communication capabilities of such embedded system are significantly high when compared to the other nodes of the network. The network Controller functionality will be performed by such embedded system.

In Figure 1 we show the proposed protocol architectures for SDWN nodes. More specifically, in Figure 1 (left) we show the protocol architecture for generic nodes, whereas in Figure 1 (right) we show the protocol architecture for the sink nodes.

A. Generic node

All generic nodes run the basic physical and MAC layer functionalities of standard IEEE 802.15.4, required to form a peer-to-peer topology. On top of the MAC layer, a **Forwarding** layer is executed which is responsible for treating arriving packets as specified by the controller. To achieve such goal, the Forwarding layer maintains a *flow table*[5] updated. According to the SDN approach, the entries of the flow tables, which are called, are defined by a rule, an action, and statistic information.

A *rule* is a description of the characteristics which are featured by packets belonging to a *flow* and that must be treated by the node in the same way. Indeed, each flow table entry specifies the *action* which should be executed to all packets satisfying the above rule. Finally, the table flow entry specifies the number of received packets which have satisfied the rule, that is, the *statistic information*. In SectionV-B we will provide more details on the implementation of the flow table entries.

Arriving packets are provided by the MAC layer to the Forwarding layer. This identifies the type of packet and if it is a control packet, then sends it to the Network Operating System layer which will be described later in this section. Otherwise, i.e., if the packet is a data packet, the Forwarding layer controls whether the packet matches one of the rules specified in the flow table. If this is the case, then the packet is treated according to the corresponding action. Otherwise the packet is given to the Network Operating System layer.

The **Aggregation** layer is executed over the Forwarding layer. Its responsibility is to perform the action required to aggregate information flowing through the network. Our current implementation of the Aggregation layer is quite straightforward. In fact, on of the actions which can be specified by the flow table entries is to include the packet in an *aggregation equivalent flow* (AEF). Packets belonging to the same AEF can be aggregated with each others and sent to the Aggregation layer for this purpose. The Forwarding layer will just concatenate arriving packets, if these are sufficiently small, and forward them as specified by the corresponsing flow table entry.

In the future, a more sophisticated behavior for the Aggregation layer can be designed.

Finally, in the architecture shown in Figure 1 (left) we can distinguish a **Network Operating System** (NOS) layer which runs on top of the IEEE 802.15.4 standard physical and link layers and has access to all the new protocol layers described

[4]In our scenario we have a sink however, the proposed architecture can be utilized also in the case there are several sinks.

[5]We derive our terminology from OpenFlow.

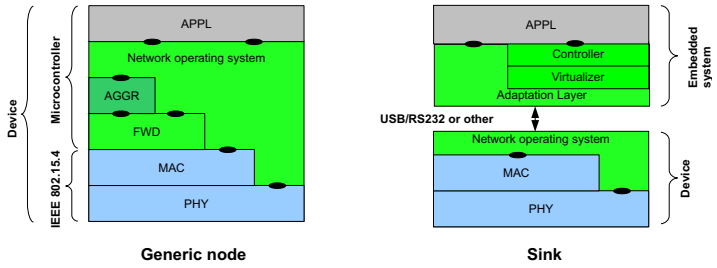

Fig. 1. Protocol architecture.

above. Indeed, we observe that the NOS layer has access to APIs offered by all layers. Such APIs enable to control the behavior of the physical and link protocol layers as well as the new defined layers and therefore, allow cross-layer operations.

Objectives of the NOS layer are, on the one hand, to collect local information from the node and send such information to the Controller; on the other hand, to control the behavior of all other layers of the protocol stack as specified by the Controller. Achievement of the above objectives requires NOS to be able to reach the sink node at any time. To this purpose a simple protocol is applied as described in Section V-A. A Flow Chart of the operations run by the NOS layer is reported in Figure 2.

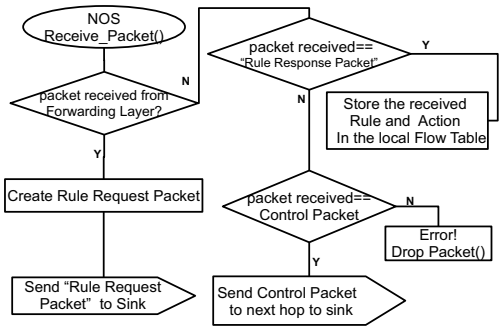

Fig. 2. Flow Chart for NOS operations.

Finally, note that the Application layer runs on top of the NOS layer, which is thus expected to provide an appropriate API. In order to support legacy applications, the current API generalizes the IEEE 802 APIs.

B. Sink node

The architecture of the Sink node can be split into two different parts as shown in Figure 1 (right). In fact, the bottom layers which run in the same device which is used by generic nodes are the same as explained in the previous subsection. Besides, there are further functionality which require high computing and communication capabilities and therefore are executed in the Linux-based embedded system.

The *device* and the embedded system are usually connected through USB, RS232 or some other communication interface.

In the embedded system, an **Adaptation** layer is executed which is responsible of formatting the messages in such a way that they can be handled by the WPAN devices.

Another key element of the sink node is the **Virtualizer** layer. This layer uses the local information collected by the generic nodes to build a consistent and detailed representation of the current network topology (graph, energy level of all generic nodes, quality of the links, etc). The Virtualizer layer is responsible for allowing the coexistence of different logic networks on the same devices. Such networks can use different policies regarding the network management, i.e. they use different Controllers. Each coexisting network is characterized by a network rule which is used to filter packets that will be treated according to the specific management policies defined by the corresponding Controller.

For what concerns the representation of the network topology, a map is used to collect all the information regarding a generic node. More specifically, for each generic node A the topology information that we need to manage is the following:

- Last time instants when the sink node has received a packet generated by A.
- The battery level reported in the last packets received from A.
- A list of nodes that are neighbors of A. For each of such neighbors, say B, we need to represent the address (at this moment this is a 2 byte field), the quality of the link between nodes A and B[6], a time stamp reporting the last time instant when the sink has received a packet from A that reports B among A's neighbors.

In our current implementation we represent the network conditions by exploiting the Java Map interfaces. For the Map we define an obsolescence timeout which is the time after which an entry (being it an entire list of neighbors or just one neighbor) should be removed from the map unless there are evidences that this action should not be taken.

Finally, besides the Application layer the protocol architecture of the sink node contains one or several **Controller**(s). The Controller is responsible to implement the desired network management policy. More specifically, the Controller will receive packets that generic nodes have not been able to classify in an existing flow and for such packets must define a rule along with the appropriate actions. The Controller will

[6]The link quality is represented by means of the *received signal strength indication* (RSSI), which is represented through 1 byte.

create flow table entries in this way which are based on the information on the current topology of the network.

In our current implementation Controllers can be implemented by using Java.

V. Design and implementation details

In this section we report some implementation details of our SDWN solution. More specifically, in Section V-A we present the protocol executed by the NOS to support communications between a generic node and the controller. In Section V-B we describe how rules and actions are specified. Finally, in Section V-C we present the format of relevant SDWN packets.

A. Collection of topology information

One of the major goals of the NOS layer is to collect the topology information which is needed by each node to communicate with the sink node and by the sink node to provide a representation of the current topology to the Virtualizer and the Controller.

In order to support communication between a generic node and the Controller, each node must learn a path (as convenient as possible) to reach the sink. To achieve this, the sink periodically generates a *beacon* packet. This packet is broadcast throughout the network in multi-hop. At each hop the beacon packet contains information about the current distance from the sink (expressed as the number of hops to reach it) and a measurement of the local battery level. Upon receiving a *beacon* packet each node increases the value of the current distance from the sink by one, overwrites the value of the current level of the battery, and, then, forwards the packet. Also, upon receiving a *beacon* packet each node measures the RSSI value in the link towards the nodes that have just transmitted the beacon.

The information contained in the *beacon* packets and the RSSI measurement allow each node to identify the most convenient next hop node to reach the sink. In particular, at first, it would choose the node that declares the lowest distance from the sink; then, the node with the longest battery life and, alternatively, the node toward which the highest RSSI value was measured.

Each node also stores in a local table, called *neighbors table*, the list of nodes from which it received a beacon, and for each of them stores the measured value of RSSI as well. This list of neighbors will be sent periodically to the sink node using a packet, called *report packet*, which will be forwarded to the best next hop node identified as explained above.

The sink node will receive the list of neighbors from any network node and will be able to infer the global topology of the network. In fact, it will receive information by all nodes about their one-hop neighbors and the link quality towards each of them (expressed in terms of RSSI).

B. Specification of rules and actions

Differently from the wired network case in our scenario we have strict limitations in terms of available memory. Accordingly, we need to develop a methodology to define rules which is memory efficient. In order to limit the space required to store the rule in the flow table, our rules can operate only upon a limited portion of the incoming packet. In particular, a rule operates on (up to) three windows of bytes of the incoming packet. Each windows in no longer than 2 bytes (if this is set to zero the window is ignored). An example of how the rules are stored in the flow-table is reported in Table I.

TABLE I
Examplary Flow Table

Window I				**Window II**				**...**	**Action**		**Stats**
Size	*Op*	*Addr*	*Value*	*Size*	*Op*	*Addr*	*Value*	*...*	*Type*	*Value*	*Count*
2	=	2	170.24	2	$\neq$	4	170.11	..	Forward	170.23	17
2	=	2	170.16	1	=	1	3	..	Drop	1	3
2	$\neq$	2	170.24	1	=	7	25	..	Modify	7/26	3
2	=	2	170.17	0	=	0	0	..	Forward	170.21	11

Each flow table entry contains three blocks (*window blocks*) related to three windows of bytes to match against packets, a block (*action block*) related to the action that specifies the way in which the packets of a flow will be processed and a *counter* used for statistical purposes.

Each window block is composed of the following fields:

- size: indicates the number of bytes of the window. It can assume a value between 0 and 2. More specifically, if this is set to zero, then the window is ignored.
- operator: defines the relational operator that must be checked to consider the matching valid. In our first release we implement only the operators "=" and "$\neq$".
- address: indicates the position of the first byte of the window.
- value: indicates the value which must be matched against the byte(s) of the packet window.

Upon receiving a packet, the matching algorithm checks whether each window matches the value stored in the field "value" and if so, it performs the action that is identified by the value stored in the "Action Type" field.

The type of action can be "forward", "modify", "drop", "aggregate" and "turn off radio". The action will be performed according to the value contained in "Action Value" field. This field length is two bytes and assumes a different meaning depending on the type of action:

- If the action is "forward", the two bytes of "Action Value" field are used to identify the next hop.
- If the action is "modify" the first byte of "Action Value" field is used to identify the specific byte of the packet which must be modified and the second byte to indicate the value to be assigned to such byte.
- If the action is "drop" the first byte is used to give the probability with which the packet should be dropped (by default this probability is set to one). The second byte indicates the second byte of the address of the node to which the packet should be forwarded in case it is not

dropped. Note that, if there are several nodes with the same last byte in their address, the node will choose randomly (implementation specific) the one to which the packet will be actually forwarded.
- If the action is "aggregate" the two bytes are used to identify the next hop to which the aggregated packet must be forwarded. Definition of the aggregation rules will be explored in our future work.
- If the action is "turn off radio" the two bytes of "Action Value" field are used to specify the duration of the interval during which the radio interface must be off.

For example, the first entry in Table I specifies that packets that have in bytes 2 and 3 the values 170 and 24 and that do not have in bytes 3 and 4 the values 170 and 11 must be forwarded to node 170.23. Finally, the last value in the line specifies that 17 packets have been classified according to this rule up to now.

The flow table entry will be populated with the values contained in the "Rule/action Response" packets received by the sink. We will provided a more detailed description in Section V-C.

C. Packet format

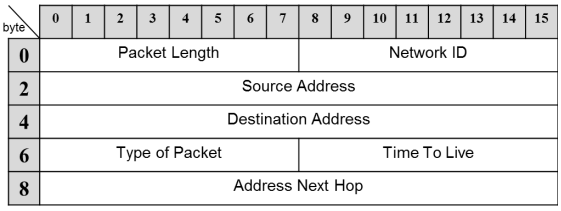

Fig. 3. Packet Header Format.

All the packets circulating in the network use a fixed header that is organized as depicted in Figure 3. Accordingly, the first flow table entry in Table I, which we have already discussed in the previous section, can be interpreted as follows. All packets generated by node 170.24 and that are not directed to node 170.11 must be forwarded to node 170.23.

Additional fields may be included based on the specific type of packet, as explained below:

Type 0 - Data packet: This packet is generated/delivered by/to the application layer and contains the header as depicted in Figure 3 plus the payload.

Type 1 - Beacon packet: This is the packet which is broadcast periodically by the sink. Such packet contains the header fields as reported above and an additional byte (referred to as "Hops to sink") providing the number of hops required to reach the sink from the node which has *transmitted* (not *generated*) the packet.

Type 2 - Report packet: This packet is generated periodically by each node and is transmitted to the sink upon receiving a Beacon Packet. Report packets contain the 10 bytes of header depicted in Figure 3 plus the "Hop to sink" byte, another byte reporting an assessment of the current charge level of the battery (several devices support the reading of the battery voltage) and information about the current neighboring nodes. Such information is structured as follows. The first byte contains the number of current neighbors. Then for each of the above neighbors 2 bytes are used for the address and an additional byte for the RSSI.

Type 3 - Rule/action Request: This packet is generated by a node upon receiving a packet which does not match any of the rules stored in the flow table. It is generated as a copy of the incoming packet in which, however, the Type of Packet field (i.e., byte 6 in the header) is set to 3 and the original Type of Packet value is inserted at byte 10.

Type 4 - Rule/action Response: Besides the usual header this packet contains a pair Rule/Action which is described as presented in Section V-B. More specifically, we use 2 bits to identify the window size, 3 bits to identify the relational operator, one byte to identify the position in the packet of the first byte of the window, and two bytes for the "value" field.

VI. Conclusions

In this paper we have made a first attempt to analyze the opportunities and challenges of applying the SDN paradigm in IEEE 802.15.4 networks. Currently, we are implementing and testing a specific solution for such scenario which we call SDWN.

In this paper we have presented the general architecture of SDWN along with some of its most relevant design and implementation details.

This paper must be considered as a first step towards the definition of a complete working solution. In fact, several issues remain open. Significant examples are related to the setting of some parameters characterizing the behavior of the proposed solution. Even if the values of such parameters can be set dynamically by the controller, we are currently running simulations to evaluate some optimal values that can be used by default.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci. A survey on sensor networks. *IEEE Comm. Magazine*. Vol. 40, No. 8. Aug. 2002.

[2] M. Boc, A. Fladenmuller, M. Dias de Amorim, L. Galluccio, S. Palazzo. Price: Hybrid geographic and contact-based forwarding in delay-tolerant networks. *Computer Networks*. Vol. 55, No. 9, Jun. 2011.

[3] B. Chen, K. Jamieson, H. Balakrishnan, R. Morris. Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks *Wireless Networks*. Vol. 8, No. 5. Sep. 2002.

[4] L. Galluccio, S. Palazzo, and A. T. Campbell. Modeling and designing efficient data aggregation in wireless sensor networks under entropy and energy bounds. *International Journal of Wireless Information Networks*. Vol. 16, No. 3, Sep. 2009.

[5] L. Galluccio, T. Melodia, S. Palazzo, G. E. Santagati. Challenges and Implications of Using Ultrasonic Communications in Intra-body Area Networks. *WONS 2012*. Jan. 2012.

[6] N. McKeown, T. Andreson, H. Balakrishnan, et. al. OpenFlow: Enabling Innovation in Campus Networks. *White paper*. Mar. 2008.

[7] W. Tuttlebee. *Software Defined Radio: Origins, Drivers, and International Perspectives*. Wiley. 2002.

[8] M. C. Vuran, O. B. Akan, I. F. Akyildiz. Spatio-temporal correlation: theory and applications for wireless sensor networks. *Computer Networks*. Vol. 45, No. 3. Jun. 2004.

[9] IEEE 2003 version of 802.15.4 MAC & Phy standard. `http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf`