Software Defined Wireless Networks (SDWNs): Unbridling SDNs

Salvatore Costanzo, Laura Galluccio, Giacomo Morabito, Sergio Palazzo

Dipartimento di Ingegneria Elettrica, Elettronica, e Informatica
University of Catania – Catania, Italy, 95039
Email: {first_name.given_name}@dieei.unict.it

Abstract

The *software defined networking* (SDN) paradigm promises to simplify network configuration and resource management dramatically. Such features are extremely valuable to network operators and therefore, the industrial and the academic research and development communities are paying increasing attention to SDN. Although manufactures of wireless equipment 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 infrastructure-less wireless communications and how SDN concepts should be adapted to suit the characteristics of these networking environments. This paper is a first attempt to fill this gap. In fact, it aims at analyzing how SDN can be beneficial in infrastructure-less wireless networking environments, and how it should be expanded to take the characteristics of such networking environments into account. Also, a complete SDN solution for wireless personal area networks, named *Software Defined Wireless Network* (SDWN) is presented and some design guidelines are provided.

1. Introduction

Software Defined Networking (SDN) is envisioned as a way to reduce the complexity of network configuration and management while making the introduction of innovation in networks simpler [12].

In this, the network operators envision the possibility to:

- introduce *evolvability* in their networks, which in their perspective means to be able to differentiate their offer when compared to other operators, and
- improve network efficiency given that new, more efficient technical solutions can be easily deployed in their infrastructure using equipment that is already in place.

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 [1] and the institution of large industry-driven organizations focused on SDN such as the Open Networking Foundation (ONF)[2] witness such increasing interest.

The SDN has interested the wireless networking community as well[3]. 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.

However, 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 relevant infrastructure-less wireless networking scenarios and how the SDN concept should be expanded to suit the characteristics of wireless and mobile communications.

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) [16]. In fact, major contributions of this work are the following:

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

Accordingly, the rest of this paper is organized as follows. In Section 2 we discuss the opportunities which can be achieved by applying the SDN paradigm to wireless infrastructure-less networks. In Section 3 we illustrate how the requirements to be considered in LR-WPANs scenarios differ from those utilized to design SDN solutions

[☆]A preliminary version of this paper appeared in [3].

[1]http://www.fp7-ofelia.eu/

[2]http://www.opennetworking.org/

[3]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 [14].

for wired networks. We have used such requirements to develop the *Software Defined Wireless Network* (SDWN) protocol stack which is briefly introduced and implemented in Sections 4 and 5, respectively. Finally, in Section 6 we will draw our conclusions and discuss the future work.

2. 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 overall efficiency in the use of network resource decreases 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,

- 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 infrastructure-less networks, such as the wireless personal area networks (WPANs), as well. However, in wireless infrastructure-less 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 ad hoc 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 [17] and 6LOWPAN [18] are very popular alternatives for the higher layers of the protocol stack but they 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 [19] 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[4] 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 operate into another 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. In fact, in SDN there are network nodes (*SDN switches*) that are responsible for classifying packets in different *flows* and performing the corresponding *actions.* A *flow* could be composed by all packets belonging to a certain TCP connection, or by all packets characterized by a particular VLAN tag and/or a certain MAC source address and/or a given IP destination address [8]. Accordingly, packet classification is performed on the basis of certain *rules.*

SDN switches can be distinguished from other nodes (the *Controllers*) that are responsible for setting the *rules* and corresponding *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. Note that the policies applied by the 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, further advantage of SDN is that it "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 are made available to the Controller

[4]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.

which has a complete vision of network state.

For all the above reasons we believe that the extension of the SDN paradigm to wireless infrastructure-less networks, which we call Software Defined Wireless Networking (SDWN), can have significant advantages in such networking environments.

3. Requirements for the SDWN

Implementations of SDN solutions for traditional wired networks have usually considered velocity as the main performance measure. Indeed, it is necessary to guarantee that SDN nodes can execute switching operations at the line rate. Satisfaction of such need has been paid in terms of low flexibility in the definition of the rules specifying the flows so that rules are implemented through control of specific packets fields. In OpenFlow for example [8], [9], [11], rules only consider the typical TCP (or UDP)/IP header.

In LR-WPAN networking, constraints on 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 [2]. Among them, SDWN uses duty cycles [13], in-network data aggregation [10], and cross-layer optimization [4], [7], [15], through flexible definition of rules and actions.

Accordingly, the new requirements – with respect to the wired counterpart – which have been considered in the design of SDWN are given below:

SDWN must support duty cycles – The most obvious way to reduce the energy consumption is turning the radio off when it is not utilized, that is, using *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 appropriate *actions*, as discussed in Section 5.2.

SDWN must support in-network data aggregation – Energy consumption can be reduced by removing the redundancy 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. For example, in wireless sensor network scenarios the data measured by neighbor nodes is not expected to differ significantly. The same observation applies to the data measured by the same sensor in different time instances. Support of data aggregation functionalities is achieved by SDWN through an appropriate module in the protocol architecture and the definition of an appropriate action as described in Sections 4 and 5.2, respectively.

SDWN must support flexible definition of the *rules* – Existing SDN implementations support 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 the traditional ones. However, analysis of the literature in the field of wireless sensor and actor networks suggests that higher flexibility is required in such scenarios. 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 5.2) for matching purposes. This would allow, for example, to route packets based on the specific value contained 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 can be conditioned also by other relational operators, other than equality. 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 can be 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 – Several other new requirements could be satisfied. Examples include, and are not limited to, the necessity to support mobility of nodes and the resulting topology changes, the need for dealing with the unreliability characterizing wireless links, or to be robust to the failure of generic nodes and/or the sink node.

4. SDWN protocol architecture

In this section we show the protocol architecture we are developing for a sensor and actor network based on the use of 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 sink[5] which is also the node where the network Controller is executed. The IEEE 802.15.4 transceiver of the sink is connected to an embedded system running a Linux operating system. Computing and communication capabilities of such embedded system are significantly highly when compared to the other nodes in the network. The network Controller functionality will be performed by such embedded system.

In Figure 1 we show the proposed protocol architecture 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.

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

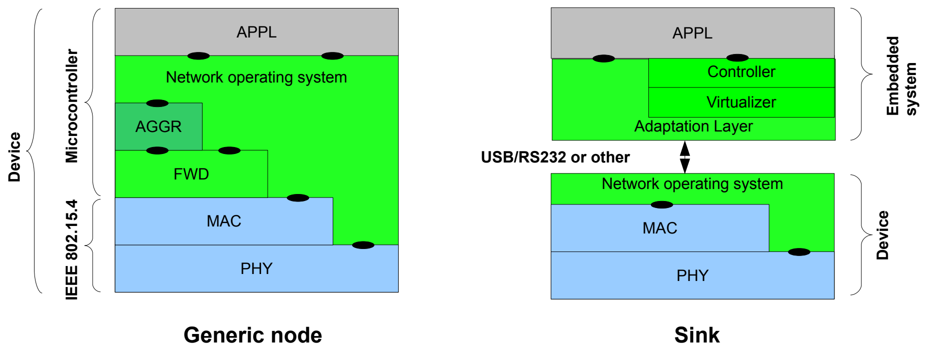

Figure 1: Protocol architecture.

4.1. Generic node

All generic nodes run the basic physical and MAC layer functionalities of the 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*[6] updated. According to the SDN approach, the entries of the Flow Tables, which are called Flow Table Entries, 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 on all packets satisfying the above rule. Finally, the Flow Table Entry specifies the number of received packets which have satisfied the rule, that is, the *statistical information.* In Section 5.2 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, it sends this to the Network Operating System layer [6] which will be described later in this section. Otherwise, i.e., 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.

Pseudocode of the operations run by the Forwarding layer is reported in Pseudocode 1.

The **Aggregation** layer is executed over the Forwarding layer. Its responsibility is to perform the procedures required to aggregate information flowing through the network. Our current implementation of the Aggregation layer is quite straightforward. In fact, one of the actions which can be specified by the flow table entries is to include

Algorithm 1: Forwarding Layer

```
ReceiveFromMacLayer(*packet);
if packet Type == Data Packet then
    Check_if_the_packet_matches_a_rule();
    if packet matches one of the rules then
        Extract_Action_from_Flow_Table();
        Perform_Action();
    else
        Send_packet_to_NOS();
    end
else
    /*packet is a Control Packet*/
    Send_packet_to_NOS();
end
```

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 packetsc and forward them as specified by the corresponding flow table entry.

In the future a more sophisticated behaviors for the Aggregation layer could 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 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 layers as well as the new defined layers and therefore, allow cross-layer operations.

Objectives of the NOS layer are twofold: on the one hand it is responsible for collecting local information from the node and sending such information to the Controller. On the other hand it must set 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

[6]We derive our terminology from OpenFlow.

a simple protocol is applied as described in Section 5.1.

Pseudocode of the operations run by the Network Operating System layer is reported in Pseudocode 2.

Algorithm 2: Network Operating System Layer

```
NOS_Receive_Packet(*packet received);
if (data packet is received from Forwarding Layer)
then
    Create_Rule_request packet();
    Encapsulate_in_Rule_request_packet (packet
    received);
    Rule_request_packet.next_hop=
    =get_next_hop_to_sink();
    Send_to_Sink(Rule_request_packet);
else
    /*the packet is a Control Packet*/
    Control_packet.next hop=
    get_next_hop_to_sink();
    Send_to_Sink( Control_packet);
end
if (packet_received==Rule_Response) then
    Create_Rule_in_Flow_Table();
    Insert_Action_in_FlowTable();
end
```

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.

4.2. 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 on the same type of hardware which is used by generic nodes are the same as explained in the previous subsection. Besides, there are further functionalities 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 interfaces.

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 state (graph, node energy level, link quality, etc).

Furthermore the Virtualizer layer is responsible for allowing the coexistence of different logic networks on the same set of devices. Such networks can use different policies regarding the network management, in other words the 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 state, we use a data structure in which each entry contains all the information regarding a generic node. More specifically, for each node A, the state information that SDWN manages is the following:

- last time instant when the sink node has received a packet generated by A;
- the battery level reported in the last packet 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[7], 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 (be it an entire entry or just one neighbor) should be removed from the map if not otherwise stated.

Finally, besides the Application layer the protocol architecture of the sink node contains one or several **Controller**(s). The Controller is responsible of implementing 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. In this way, the Controller will create flow table entries which are based on the information on the current topology of the network.

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

5. Design and implementation details

In this section we report some implementation details of our SDWN solution. More specifically, in Section 5.1 we present the protocol executed by the NOS to collect topology information. In Section 5.2 we describe how rules and actions are specified. Finally, in Section 5.3 we present the format of relevant SDWN packets.

[7]The link quality is measured in terms of the *received signal strength indication* (RSSI), which is represented using 1 byte.

5.1. 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 (the more convenient) to reach the sink. To achieve this, the sink periodically generates a *beacon* packet. This packet is broadcast throughout the network in a multi-hop fashion. 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 residual battery voltage. 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 node that has 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 to reach the sink. In particular, firstly, it would favor the node that declares the lowest distance from the sink; secondly, the node with the longest battery life and, thirdly, 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 the 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 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).

5.2. 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 rule can operate upon a limited portion of the incoming packet only. In particular, a rule operates on (up to) three windows of bytes of the incoming packet. Each window is 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 1.

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 implemented the operators "=" and "≠" only.
- *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 of the windows, with size higher than zero, matches the corresponding value stored in the "value" field 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 the "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 the "Action Value" field are used to identify the next hop;
- if the action is "modify", the first byte of the "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 (this is 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 aggregation equivalent flow. Definition of the aggregation rules are subject of our future work;
- If the action is "turn off radio", the two bytes of the "Action Value" field are used to specify the duration of the interval during which the radio interface must be off.

Table 1: 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

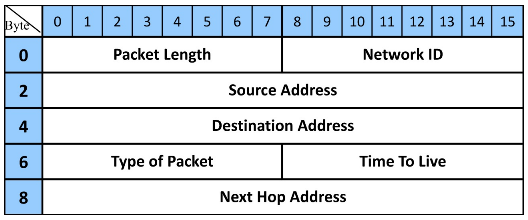

Figure 2: Packet header format.

Table 2: Exemplary data structure reporting the information about the neighboring nodes

3 (Number of neighboring nodes)	
ADDRESS	**RSSI**
121:4	130
121.8	153
121:9	160

For example, the first entry in Table 1 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. Accordingly, the first flow table entry in Table 1 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.

The Flow Table Entry will be populated with the values contained in the "Rule/action Response" packets arriving from the sink.

5.3. Packet format

All the packets circulating in the network use a header that is organized as depicted in Figure 2.

Content of the remaining part of the packet depends 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 2, 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-header depicted in Figure 2 including also a "Hop to sink" byte, another byte reporting an assessment of the current charge level of the battery (several devices support reading of the residual 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. For example, in Table 2 we report such information for a given node (say 121:77), in the case it has three neighboring nodes: 121:4, 121:8 and 121:9, and the RSSI values measured by 121:77 in the link towards the three above nodes are 130, 153, and 160, respectively;

- **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 5.2. 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;

- **Type 5 - Open path packet:** This type of packet[8] is used to build a path using a single packet. Accordingly, it contains the usual header followed by

[8]Our current implementation does not support "Open path" packets.

a Rule and the sequence of the addresses of the nodes in the path.

6. 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 a first step towards the definition of a complete working solution. In fact, several issues remain open. Investigation on the setting of some parameters characterizing the behavior of the proposed solution should be carried out. Even if the values of such parameters can be set dynamically by the Controller, we are currently implementing a detailed simulator to evaluate the more convenient values of the parameters that can be set by default. The simulator will provide appropriate interfaces which enable the interaction with the user–defined Controller. In this way the developers of the Controller software will be able to test the behavior of their solution before the deployment of the real wireless devices.

Acknowledgements

The work of SC was supported by the Sicilian Region under contract "Sentinella". The work of GM was partially supported by the European Commission under contract "OFELIA" (contract number ICT-258365).

References

[1] K. Akkaya and M. Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*. Vol. 3, No. 3. March 2005.

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

[3] S. Costanzo, L. Galluccio, G. Morabito and S. Palazzo. Software Defined Wireless Networks: Unbridling SDNs. *EWSDN 2012*. October 2012.

[4] L. Galluccio, G. Morabito, and S. Palazzo. An analytical study of a tradeoff between transmission power and FEC for TCP optimization in wireless networks *IEEE Infocom 2003*. April 2003.

[5] L. Galluccio, K. Nahrstedt, and V. R. Syrotiuk (Eds). Cross-Layer Design in Ad Hoc and Sensor Networks. *To appear in Elsevier Ad Hoc Networks*. doi:10.1016/j.adhoc.2011.11.001 2012.

[6] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. *ACM Computer Comm. Review*. July 2008.

[7] V. Kawadia and P. R. Kumar. A Cautionary Perspective on Cross Layer Design. *IEEE Wireless Comm.*. Vol. 12, No. 1. February 2005.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *White paper*. March 2008.

[9] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown. Implementing an OpenFlow switch on the NetFPGA platform. *ACM/IEEE ANCS '08*. November 2008.

[10] L. Galluccio, A. T. Campbell, and S. Palazzo. Poster Abstract: CONCERT: aggregation-based CONgestion Control for sEnsoR neTworks. *ACM Sensys 2005*. November 2005.

[11] B. Pfaff *et al.*. OpenFlow Switch Specification – Version 1.1.0 Implemented (Wire Protocol 0x02). February 2011.

[12] S. Shenker *et al.*. The future of networking and the past of protocols. *OpenNetSummit 2011*. October 2011.

[13] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. *ACM MobiCom '98*. October 1998.

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

[15] L.-C. Wang and A. Chen and S.-Y. Huang. A Cross-Layer Investigation for the Throughput Performance of CSMA/CA-Based WLANs With Directional Antennas and Capture Effect. *IEEE Trans. on Vehicular Technology*. Vol. 56, No. 5. September 2007.

[16] IEEE-TG15.4. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). *in IEEE standard for Information Technology*, 2003.

[17] ZigBee Specification. Available at `http://www.zigbee.org/en/spec_download/download_request.asp`

[18] Mulligan, Geoff. The 6LoWPAN architecture. *ACM EmNets '07*. June 2007

[19] IEEE 802.11, Wireless LAN medium access control (MAC) and physical layer (PHY) specification. *in IEEE Computer Society*, 2010.