HAL
open science

Rethinking the Design of LR-WPAN IoT Systems with Software-Defined Networking

Akram Hakiri, Aniruddha Gokhale

To cite this version:

Akram Hakiri, Aniruddha Gokhale. Rethinking the Design of LR-WPAN IoT Systems with Software-Defined Networking. Distributed Computing in Sensor Systems (DCOSS), 2016 International Conference on, May 2016, Washington, DC, USA, United States. hal-01633328

HAL Id: hal-01633328

https://hal.science/hal-01633328

Submitted on 12 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Rethinking the Design of LR-WPAN IoT Systems with Software-Defined Networking

Akram Hakiri*†, Aniruddha Gokhale †
*Univ de Carthage, SYSCOM ENIT, FST Bizerte, ISSAT Mateur, Tunisia.
† CNRS, LAAS, 7 Avenue du colonel Roche, F-31400 Toulouse, France
†ISIS, Dept of EECS, Vanderbilt University, Nashville, TN, USA.
Email: hakiri@laas.fr and a.gokhale@vanderbilt.edu

***Abstract*—Wireless Sensor Networks (WSNs) are becoming a key enabling technology for Internet of Things (IoT) by virtue of providing a highly unstructured cloud of wireless devices. Despite these advances, the current Internet architecture is not able to cater to the high volume of new traffic patterns delivered by these smart sensing devices. In this context, Software Defined Networking (SDN) has emerged as an intelligent solution to deliver dramatic improvements in network programmability, agility and flexibility. However, SDN was originally targeted to wired networks deployed in cloud data centers, and does not lend itself well to WSNs due primarily to its higher footprint and lack of WSN programmable interfaces.**

To address these challenges, this paper describes an approach to realize software-defined wireless sensor networks by introducing a novel SDN-enabled architecture for WSNs that can be used for diverse IoT systems. Specifically, we propose new control plane services for supporting automatic topology discovery, sensor virtualization as well as managing network policies. Additionally, we introduce new customized SDN-enabled flow tables to meet the requirements of sensor network packets. Finally, we introduce a programmable MAC layer to support fine-grained flow processing.

***Index Terms*—SDN; OpenFlow; IoT; Wireless Sensor Networks; TDMA.**

I. Introduction

Smart grids, smart cities, intelligent transportation, retails, health-care, safety, industrial automation, and environment services are evolving as infrastructure systems that deliver their data through the Internet. Since these systems are able to sense the environment and communicate, they play a key role in realizing the vision of the Internet of Things (IoT) in everyday life, wherein the confluence of inexpensive wireless communication has created a new generation of smart devices. The current estimate for the number of smart things are on the order of 50 billion self-organized devices that utilize Wireless Sensor Networks (WSNs). WSNs consist of wireless entities comprising sensors that gather data from surrounding environment, low-power radio transceivers and micro-controllers, all of which have limited power and computation resources.

Given their traits, WSNs are required to be autonomous and be able to adapt their behavior in accordance with the users' requirements to handle application-driven networks in dynamic and versatile environments. Hence, an intelligent network architecture is required to address the complexity of WSNs to support IoT systems. In this context, Software-Defined Networking (SDN) [1] has emerged as a new architecture to deliver dramatic improvements in network agility and flexibility. However, existing realizations of SDN, such as OpenFlow [2] which is the dominant SDN technology, are geared towards wired networks deployed in settings, such as cloud data centers, and does not support communication within WSNs. For instance, OpenFlow specifies 44 header fields for wired networks, but none of them can be used for the Low-Rate Wireless Personal Area Network (LR-WPAN) such as WSNs.

Extending SDN to WSNs was considered impractical because WSNs run in resource-constrained Internet devices with limited computation resources, small data-storage capabilities and low-power consumption. Specifically, most of the WSN technologies specify packet formats close to 127 Bytes, while the current OpenFlow packet is encapsulated in IP packets, which are close to 1,500 bytes. Besides, the addressing scheme of sensor devices is different from the IP-based addressing format. For example, the addressing scheme of LR-WPAN technologies like IEEE 802.15.4 and ZigBee comprises at most 8 or 16 bits, while IP4 and IPv6 addresses used by OpenFlow comprise 32 and 128 bits, respectively. Subsequently, the existing OpenFlow protocol as is is not able to support the current addressing scheme used by WSNs.

Some recent initiatives attempt to overcome these limitations. For example, Flow-Sensor [3] and Sensor-OpenFlow (SOF) [4] proposed extending the OpenFlow protocol to WSNs, however, these approaches are not suited for real deployment scenarios due to their larger memory footprint, which is no suitable for WSNs. Likewise, TinySDN [5] uses multiple distributed SDN controllers for managing sensor motes that run a single WSN router. However, it depends on a specific operating system, i.e. the TinyOS, which hinders interoperability with Contiki OS or RIOT OS, which are geared towards IoT. Similarly, SDCSN [6] introduced a cluster-based SDN approach for WSNs in which a cluster head is elected to manage a group of sink devices. However, typical sensor motes are equipped with small buffers that can support only a few hundreds of short packets. Conversely, such approaches will need to buffer thousands of packet data in the flow tables before sending them to next hops. The increase in buffered data will lead to an increase in energy consumption, which is a concern in WSNs given their limited

capacities for processing and storage memory. Furthermore, communication in WSNs should occur at low data rates to guarantee low energy consumption. Most WSNs require low-cost nodes communicating over multiple hops to cover a large-scale geographical area. They also need duty cycle management for efficient energy consumption to operate for multi-year lifetimes on modest batteries.

In summary, to make the SDN concept applicable and beneficial to IoT/WSNs, there is a need to develop a novel and customized OpenFlow approach that can fill the current gap for WSNs outlined earlier. Such an approach should provide low software footprint with respect to the limited amount of memory storage and CPU processing of the resource-constrained IoT devices. To address these concerns, we propose the *SensorSDN framework,* which defines a new SDN abstract model that fulfills the specific requirements of diverse WSNs. First, we propose new control plane services for supporting automatic topology discovery, node mobility as well as managing network policies. Second, we introduce new matching fields based on existing LR-WPAN technologies along with new flow rules for allowing routing packets based on the specific matching values we introduce for SensorSDN. Finally, our SensorSDN allows a programmable cross-layer optimization between the MAC link layer and the network layer, as well as data aggregation.

The remainder of this paper is organized as a follows: Section II presents the architectural design decisions of our SensorSDN solution and describes the novel services and flow entries we specifically defined for SensorSDN. Section III presents related works and compares them to our work presented in this paper. Finally, Section IV presents concluding remarks and lessons learned.

II. SensorSDN Protocol Architecture

In this section we describe the architecture and the rationale behind the proposed SensorSDN, which is based on LR-WPAN. First, we introduce the control plane services in the controller side. Second, we present the new abstract flow table entries we introduce in the data plane. Finally, we show the automated resource allocation approach using the programmable TDMA MAC layer.

Figure 1 illustrates the control plane and the data plane of SensorSDN inside sink motes. Each router mote contains a customized flow table that abstracts the software pipelines as well as the programmable MAC layer, i.e., pTDMA. The controller and the sink router exchange OpenFlow (OF) information through a wired OF channel based on USB or serial RS232 protocols.

A. The Controller Services

The controller resides in the centralized control plane that controls the entire sensor network. The controller is part of the PAN (Personal Area Network) coordinator, which is a sink node that boots up and initializes by creating a unique network address called PAN ID. Multiple networks can exist in the same geographic area, however, each one should have

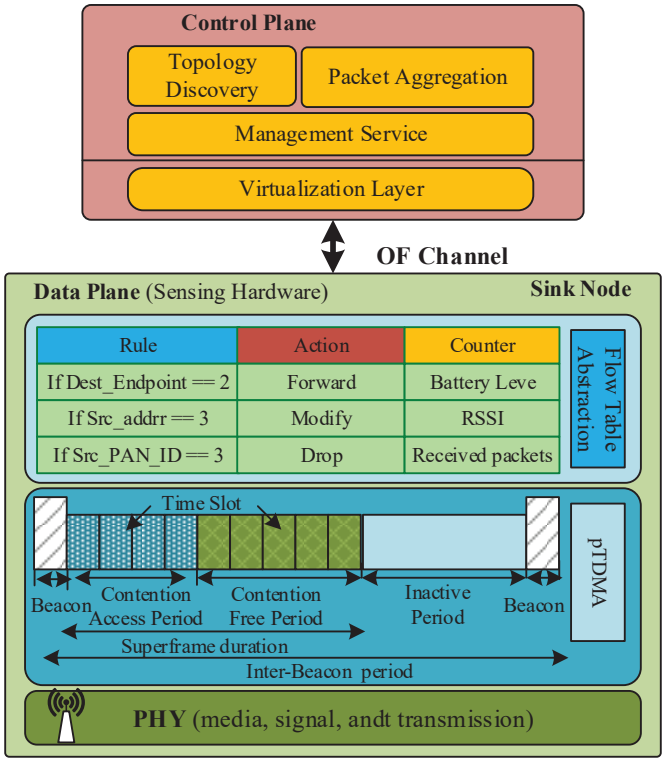

Fig. 1: Architectural Overview of SensorSDN

its own PAN ID. When a user application wants to create a new network instead of joining an existing one, the controller selects a router to act as a PAN coordinator and provides it with a list of available channels.

The controller provides the following services: (i) the topology discovery service, which is responsible for discovering other sinks, perform packet aggregation, and facilitate traffic engineering; (ii) the management service, which includes network policy control and security enforcement capabilities; and (iii) the virtualization service, which enables sharing the same WSN infrastructure and abstracts the physical computing resources into logical isolated and interconnected units.

1) Topology Discovery: In order to discover the topology, the controller uses the discovery function to invoke the active scan service at the MAC layer, which will broadcast beacon frames to discover any suitable router on the same advertised channel to join the newly formed network. When other devices see the beacon frame, they respond with their own beacon frame to join the network. The responding device adds network information into the frame and communicates its distance from the controller and from other devices in terms of number of hops and their battery levels. When the controller receives all the request and response messages, each device will add the MAC and network addresses of its neighbor to its descriptor list and its neighbor table. Subsequently, a list containing all the scan descriptors is sent to the topology discovery service to create a topology graph based on this list.

New devices can join the network later by sending join

requests to their "parent" devices. For example, a temperature sensor or a pressure sensor can join its parent router; or a new router can join the network by sending a request to the sink device. If incoming packets are not recognized by the sink or the sinks are unable to process them according to their flow entries, they send them to the controller for redefining new flow rules. Thereafter, the controller replies with a notification message to add the address of the new device.

Subsequently, appropriate actions can be applied to those packets to ensure that they match against their packet header. To this end, the controller inspects the headers of incoming packets that have not yet been matched and compares them to its aggregation flow information base. If a match is found in this database, it will applied to the packet header; otherwise, packets will be dropped.

Once the device has successfully joined the network, it will broadcast an announcement message to advertise to all other devices that it has joined the network and sends its short 16 bit address as well as its 48-bit IEEE MAC Address. Receiver devices will then search in their neighbor's tables if the 48-bit IEEE MAC address already exists. If it does, then it means that the requesting device has already previously joined so it just will issue the same network address; otherwise, it will add it as a child device and generate a new network address for it.

Besides, each router stores a neighbor table containing a list of all single-hop sinks within its transmission/reception range. The neighbor table is populated during the discovery process when a router is searching to join a network group. In order to keep the neighbor table up-to-date, each router should compare the incoming *Beacon* frame to its neighbor table and add it if an entry does not exist. All neighbors within a router's earshot will resend it to other neighbors and multiple copies of the same frame will be forwarded. If a router has already received and processed the same frame, it will discard the copies. Otherwise, it increases the distance from its neighbor node by 1, overwrite the existing value of battery level by its own battery level, and then send the packet to next hop. The battery level is overwritten each time a packet is received because each router will change it with its battery level and resend the packet.

Accordingly, the controller is able to estimate the link quality, and the number of packets that satisfy a given rule. In such a way, the controller is able to build the network topology and select the best path between routers based on the Receive Signal Strength Indicator (RSSI) information. Best path selection is based on three metrics: the nearest distance is considered first, then the energy level and finally the link quality provided by the RSSI information.

2) Management Service: The management service in Figure 1 is used by the controller to support frequency agility at the sink node. This is required because multiple frequencies allowed for use for the multiple channels. The management service will perform channel scanning to select the best one in terms of signal-to-noise ratio, and can also dynamically change the channel in case of interferance, etc. In particular, it offers the ability to prevent PAN ID conflicts and be able to change channels in order to avoid tearing down the network or rebuild another channel. The management service calls energy scan and active scan services from the MAC layer to perform scanning on the supplied channels. Once the scan is performed, the network formation function is called to decide on the channel to join. The channel selection can be decided according to different criteria, such as the lowest energy reading, the lowest amount of traffic, and the fewest networks or some weighted function of these criteria.

Furthermore, the management service provides a comprehensive network security service by setting up authentic relationships between wireless motes using cryptographic key management. The cryptography scheme uses 128-bits keys of the AES encryption to secure transmitted frames and prevent unauthorized nodes from joining the network.

The management service also tracks the state of the network in order to discover, configure, and maintain routers on the network and instruct them when to form or join a network or when to leave it. Specifically, it can detect choke points in routing devices by looking at their routing tables. It sends requests on behalf of the controller to read the contents of the router neighbor table. It can also query the routing table to figure out the table entries, and sends and reports beacon frames to determine which networks or devices are in the transmission or reception range of the other nodes. It then populates the neighbor list in the topology discovery service to advertise it about the node association/disassociation, i.e., the manamenet service sends information to the topology discovery service in the controller to publish the current/actualized list of devices so this is how node association/disassociation is "reflected" back to the controller.

3) Virtualization Service: The virtualization service allows external applications sharing data collected from one or multiple shared sensor motes. In particular, multiple sensing motes can be arranged in the form of an unstructured cloud that acts like gateways for sharing data between multiple applications running in remote servers in the cloud. Additionally, the virtualization service can use the individual capabilities of each sink mote to run multiple application tasks concurrently. We call this process of distributing and consolidating virtualized tasks as Sensor Function Virtualization (SFV).

SFV allows deploying multiple sensing tasks into software packages inside virtualized motes. By enabling such a task chaining over WSNs we can easily create, modify and remove new services. SFV can be achieved by either creating multiple virtualized functions over the same router hardware or by consolidating several sensing tasks among distributed routers. Virtualization allows creating dynamic groups of WSNs that are able to execute isolated concurrent tasks seamlessly without being tied to specific target hardware platforms, operating systems, or customized interfaces.

We surmise that recent advances in hardware is making the motes powerful enough to support some degree of virtualization. Moreover, current standardization efforts and test trials are aiming at better performance and coverage up to 14 kms. Thus, by enabling SFV inside the virtualization service,

WSNs can profit from the elasticity, scalability, and flexibility provided in the physical motes. It also allows offloading sensing nodes since computation will be shared between remote motes. As a result, virtualized functions can be added to or removed from motes seamlessly to users/applications without interrupting existing communicating systems, and appear like sensing devices offering these functions by themselves. Such capability is the result of sensor service chaining, which makes it possible to divide several functions between remote sensors, each function inside a given sensor node and where nodes can collaborate together in a way to create a larger, complex function which appears to be provided by a single node.

B. The Data Plane

A SDN data plane should forward incoming packets among different motes based on matching their headers against entries in its flow table by applying the required actions to those packets. As the current SDN does not support WSNs packet header, we defined new header fields for packet matching. We also introduce a new abstract flow table to program the flash memory of the router's hardware. Finally, we introduce the programmable TDMA layer that interacts with the network layer to provide dynamic and flexible data forwarding.

1) Matching Fields: The matching fields described in Table I are grouped into three different layers: the MAC layer, the network layer, and the application layer. The MAC matching fields include the sink and router addressing fields. Packets are matched against either the source/destination address of routers or sinks. The former allows routers to process received packets based on the MAC addresses similar to using learning switches in IP networks. Packet matching can also be performed similar to VLAN tagging (VLAN ID) in the Ethernet packet header by parsing source/destination PAN addresses.

Layer	Field 1	Field 2	Field 3	Field 4
MAC	Dest PAN ID	Dest Addr	Src PAN ID	Src Addr
Network	Dest NwkAddr	Src NwkAddr	Radius	–
Application	Dest Endpoint	Src Endpoint	Cluster ID	Profile ID

TABLE I: SensorSDN Matching Fields

Similarly, packets are also matched against the network layer. Routing devices can perform actions based on the source/destination network addresses of its neighboring devices, as well as the radius value. Likewise, packets can be processed against application layer header fields. In particular, as LR-WPAN combines the transport layer and the application layer in a single Application Support Layer (APS), the source and destination endpoints are compared to source/destination ports in the TCP/IP transport layer.

The cluster ID and the profile ID fields allow packet matching at the application layer. Table II illustrates each field we introduced along with the size inside the packet header and a description of each field.

2) The Abstracted Flow Table: The abstracted flow table depicted in Table III contains the common requirements to process packets whose headers match the pattern defined by the fields described in the previous section. The controller can control the behavior of the flow table by manipulating entries inside the flow tables. The flow table entries are defined by *rules*, *actions*, a *counter*, and a *priority*:

Layer	Field Name	Size (bits)	Description
MAC	PAN ID	16	src/dest coordinator address
	Device Addr	64-bit	IEEE MAC long device address
		16	Short device address
Network	NwkAddr	16	src/dest node network address
	Radius	8	Maximum hop-count
Application	Endpoint	8	src/dest Endpoint ID
	Cluster ID	8	Application Object address
	Profile ID	16	Application domain space.

TABLE II: SensorSDN Addressing Fields

- *Rules* specify how packets should be matched against their header fields. For example, in Table IV packets with same *Dest_PID == 0xF1FE* should receive the same processing in the flow table pipeline.
- *Actions* specify the processing to perform on all packets satisfying one or multiple rules. In Table IV, we introduced five rules, *i.e. Forward, Drop, Modify, SendToCTRL, and Flood*. For example, all packets with the MAC address is $\neq$ "01:23:45:67:89:ab" should be modified by MAC Addr 01:03:45:67:89:DE and forwarded to router with network address 0xfff7. Similarly, packets having the destination PAN ID 0xF1FE should be discarded.
- The *counter* allows gathering different statistics from a given node including battery levels, time stamps of the last received packets, lost packets, link quality, etc. The collected data allows the management service to monitor the network performance and bottlenecks.
- The *Priority* field defines the order of executing the instructions by the data plane. Instructions with higher priority should have immediate actions to handle header fields and those with lower priority will be delayed by changing their software pipeline.

3) Programmable TDMA: The media access protocol (MAC) for WSN can be classified into two categories: i) contention-based and ii) scheduling-based protocols. The former relaxes the time synchronization constraints to allow dynamic topology adjustment as some nodes may join or leave the network. Carrier Sense Multiple Access with Collision Avoidance (CMSA-CA), which are the basis for various contention-based techniques, have higher costs in terms of message collisions, network overhead and power consumption. The latter uses time division multiplexing, where designated time slots are used to assign a transmission period to avoid packet collisions and overlapping during concurrent transmission. TDMA, which is the basis of different scheduling-based techniques, allocates independents time slots with respect to the network topology as well as the nodes packet generation rate. In this way, collisions can be avoided by silencing interfering nodes in each time slot and ensuring minimum network latencies as interference between adjacent wireless links are avoided. Moreover, in TDMA there is no hidden

Match Fields	Actions	Counter	Priority
Src/Dest PAN ID	Forward	Battery Level	32
Src/Dest MAC Address	Drop	NodeID	68
Radius	Modify	TimeStamp of loss Packets	68
Src/Dest Network address	SendToCTRL	Last time a node received a packet	37
Src/Dest Endpoint	Flood	Link Quality (RSSI)	28
Cluster Id		Per-Flow counter:	27
Profile Id		• Received packets/bytes • Duration (sec, nanosec)	

TABLE III: SensorSDN Flow Table

Header Fields	Action	Counter	Priority
if Dest_PID == 0xF1FE	Drop Packet		32
if MAC Addr $\neq$ 01:23:45:67:89:ab	Modify MAC Addr 01:03:45:67:89:DE & forward to 0xfff7		60
if cluster ID == 0x13	SendToCTRL		27
if Endpoint Addr == ff02:1	Flood multicast packets		68

TABLE IV: Example of Flow Table Entries in SensorSDN

node problems since all nodes will transmit during different non-overlapping time slots.

However, the current TDMA scheme allocates fixed time slots for every node so that no node is allowed to transmit in the superframe if its data rates change dynamically. To address this issue, SensorSDN introduces a programmable TDMA scheme that improves the QoS in WSNs and allows predictable network performance. Indeed, the controller exchanges control messages, i.e. *beacon* messages, among mote routers to reflect how the MAC layer is scheduled between different routers. In particular, control packets reflect the structure of the superframe and the dynamic changes between the Contention Free Period and the Contention Access Period.

The pTDMA algorithm allows allocating time slots dynamically to transmit/receive multiple packets according to application requirements. It allows scheduling multiple node transmissions in both the contention access period and the guaranteed time slot period. If the number of time slots allocated to a given sink at the contention period are not sufficient to send its data, it can dynamically reserve time slots from the guaranteed communication period. In traditional TDMA approaches, a sink is not allowed to request time slots in the guaranteed period inside the same superframe even if those are available. Instead it should wait for the next superframe to request additional ones. This approach taken by SensorSDN maximizes the channel utilization and allocates time slots between consecutive time windows based on the traffic demands to ensure the required QoS.

In most IEEE 802.15.4 systems, the MAC is implemented in software that run on some MCU core, each device provides dozens of primitives to program the MAC layer and allow data transfer. In TDMA systems, the beacon begins a time-interval – the so-called superframe – which are divided into time slots of equal length to fill the time between two beacons. The superframe is composed of two periods: the Contention Access Period (CAP) and the Contention Free Period (CFP). The CFR forms the so-called Guaranteed Time Slot (GTS) is assigned by the Sink node (PAN coordinator) to let devices sending their data any time they want instead of using the CAP in which any device can send data with respect to its neighbor desire. For latency sensitive applications, it is possible to use one of the first slots because the smaller the slot number is, the shorter is the delay. The number of slots (N_{MAX}), the duration of the time slot (T_{SLOT}) and the duration of the superframe (T_{FRAME}) can be changed dynamically by the sink node.

III. Related Works

Recently there has been increased research in applying SDN concepts to wireless sensor networks. Flow-Sensor [3] builds sensor nodes with unmodified OpenFlow and uses model checking to verify the reliability and the consistency of Flow-Sensor. However, Flow-Sensor cannot be deployed into physical motes because its resource requirements are too much for the wireless devices, which cannot be provided in WSNs due to their constrained computation resources and memory. Similarly, Sensor OpenFlow [4] introduced additional fields to OpenFlow by employing OpenFlow Extensible Match (OXM) called Type-Length-Value (TLV) structure to define flow matches. However, TLV format parsing introduces irrelevant header fields for WSNs such as Ethernet, MPLS, and IP tagging that should be parsed by WSN routers. Those routers have limited buffers and they neither can process all those details at line rate nor require them to forward LR-WPAN packets.

Likewise, authors in [5] introduced the TinySDN framework atop of the TinyOS operating system. TinySDN separates the control logic from data forwarding by enabling the use of multiple distributed SDN controllers for managing sensor motes that run a single WSN router. Although the abstracted router uses a modified version of the standard OpenFlow protocol to support customizable flow tables on each sensor mote, it suffers from several drawbacks. Indeed, using multiple distributed controllers may present a scalability bottleneck

in large-scale sensor deployment because all the intelligence should be moved inside each physical mote. Such a scenario requires additional sensor virtualization layer to orchestrate the hardware and application resources among distributed sensing nodes. However, sensor virtualization is not supported in TinySDN. Moreover, it has been shown that increasing the number of distributed controllers in a wired networks does not necessarily increase the performance of the communication [7]. Besides, because it is hard to coordinate actions between distributed control planes, it is very hard to keep a consistent and optimal global network state. For example, inconsistent node discovery decisions could generate inconsistent topology information, which could involve serious performance and correctness problems.

SDCSN [6] consists of a cluster-based SDN approach for WSN in which a cluster head is elected to manage a group of sink devices. Each cluster head can act as a local gateway in its SDN domain and can communicate with other clusters to perform hierarchical inter-domain routing, which can enforce the reliability and the availability of the SDCSN. However, SDCSN uses the standard OpenFlow protocol over Linux containers running over powerful machines, which makes it unsuitable for embedded resource-constrained devices.

Similarly, authors in SDMN [8] and SDN-WISE [9] define a customizable flow table and handle packets delivery as finite state machine by using mathematical and logic operators. While both proposals are interesting to apply SDN concepts to WSN, however, they do not provide any differentiation between data flow and control flow. They also provide programmability at the data plane without paying attention to the dynamic management of the MAC layer. Thus, cross-layering optimization and flow differentiation is still required for sensed data.

In contrast to those contributions, our solution provides a cross-layer optimization within each physical mote by introducing new flow rules that takes into account the dynamic resource reservation and the wireless medium sharing between virtualized and physical sensing devices. Indeed, our approach allows defining a common packet header for heterogeneous target platforms, while ensuring their interoperability and scalability. Furthermore, our solution provides a management service at the centralized control plane to access the WSN information base and enables automatic topology discovery through autonomic association/dissociation of remote wireless sensors. It also provides guaranteed time slot reservations to enable flow differentiation and QoS provisioning, and security policy enforcement at the controller side.

IV. Conclusion

Wireless Sensor Networks are becoming one of the most promising wireless technologies for realizing the vision of diverse IoT systems such as smart cities, health-care, transportation, retail, safety, environment services, and industrial automation. However, current WSNs have imposed complex requirements on both the underlying network layer and the MAC layer to coordinate and control the access to the shared medium. To address these limitations, this paper introduced a novel and intelligent SDN architecture towards the future generation IoT systems to address their complexity and interoperability. This work-in-progress paper provides a flexible solution that helps to cater for network heterogeneity. Our current ongoing work is focusing on implementing the proposed solution in ZigBee/6LowPAN hardware platforms. Additionally, security, resiliency, robustness and data integrity will be a key requirement of future WSN networks, which form additional dimensions of future work.

Acknowledgment

This work was partially funded by the Fulbright Visiting Scholars Program and NSF CNS US Ignite 1531079. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF, DGA, CNES or Fulbright program.

References

[1] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 3, pp. 1617–1634, 2014.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, and J. Rexford, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, pp. 69–74, 2008.

[3] A. Mahmud, R. Rahmani, and T. Kanter, "Deployment of flow-sensors in internet of things' virtualization via openflow," in *Mobile, Ubiquitous, and Intelligent Computing (MUSIC), 2012 Third FTRA International Conference on*, June 2012, pp. 195–200.

[4] T. Luo, H.-P. Tan, and T. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," *Communications Letters, IEEE*, vol. 16, no. 11, pp. 1896–1899, November 2012.

[5] B. Trevizan de Oliveira, C. Borges Margi, and L. Batista Gabriel, "Tinysdn: Enabling multiple controllers for software-defined wireless sensor networks," in *Communications (LATINCOM), 2014 IEEE Latin-America Conference on*, Nov 2014, pp. 1–6.

[6] F. Olivier, G. Carlos, and N. Florent, "Sdn based architecture for clustered wsn," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2015 9th International Conference on*, 2015, pp. 342–347.

[7] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12, 2012, pp. 7–12.

[8] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling sdns," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, Oct 2012, pp. 1–6.

[9] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks," in *Computer Communications WorFOCOM WKSHPS), 2015 IEEE Cce on*, April 2015, pp. 513–521.