A Software-Defined Networking framework for IoT based on 6LoWPAN

Fabian Fernando Jurado Lasso, Ken Clarke, Ampalavanapillai Nirmalathas
Department of Electrical and Electronic Engineering
The University of Melbourne
Melbourne, VIC 3010, Australia
fdo.jurado@gmail.com

***Abstract*— The software defined networking framework facilitates flexible and reliable internet of things networks by moving the network intelligence to a centralized location while enabling low power wireless network in the edge. In this paper, we present SD-WSN6Lo, a novel software-defined wireless management solution for 6LoWPAN networks that aims to reduce the management complexity in WSN's. As an example of the technique, a simulation of controlling the power consumption of sensor nodes is presented. The results demonstrate improved energy consumption of approximately 15% on average per node compared to the baseline condition.**

***Index Terms*—Wireless sensor networks; Internet of Things; software-defined networking; 6LoWPAN.**

I. Introduction

The concept of the Internet of Things (IoT) is enabled by the connectivity of a variety of objects, which are embedded with processing, sensing and communication capabilities, so they can work cooperatively to accomplish a common task [1], [2]. The integration of new enabling technologies and innovative applications are forming a genuine IoT [3]. IoT applications include precision agriculture, smart cities and smart transportation systems, require the deployment of thousands of sensor nodes. In fact, in 2011 the number of interconnected objects overtook the number of people [3]. However, smart objects are often constrained by processing power, memory capabilities and power consumption. Moreover, to have an IoT network with WSN devices, these devices must be accessible individually by a unique Internet Protocol (IP) address. To alleviate the aforementioned issues and satisfy the IoT requirements and challenges such as scalability, heterogeneity and bridge the gap between research and practical implementation [4], the IETF (Internet Engineering Task Force) has established the 6LoWPAN [5] (IPv6 over Low-Power Wireless Personal Area Networks) working group. The 6LoWPAN adaptation layer is crucial to enable connectivity of an increasing number of objects since it allows transmission and reception of IPv6 packets over IEEE 802.15.4 based networks. In addition, due to its lightweight implementation and capability to enable interoperability between heterogeneous devices, 6LoWPAN has been adopted by many M2M communication systems [6], [7].

For such applications, there is a need for a WSN management system to ensure the network runs smoothly and is easy to maintain and manage, while also allowing accurate and efficient modification of different parameters in the network as circumstances or requirements change. This is the paradigm of Software-Defined Wireless Sensor Networks (SD-WSN). The principle of SDN is to decouple the control plane from the data plane in the network. The control plane manages complex network operations, while the data plane performs basic operations such as packet forwarding [8], [9].

Previous efforts by other groups in this domain have taken various approaches to enable reprogrammable nodes in WSNs. Luo et al. [10] proposed Sensor OpenFlow (SOF) as a communication protocol between the control and data planes. The data plane is made reprogrammable by customizing the flow-table of every node by using the SOF protocol, and the control plane centralizes the network intelligence in the controller. Two solutions for flow creation have also been proposed by this group. Their first solution redefines flow tables, which classifies WSN addressing by using ZigBee 16-bit network addresses and concatenated value pairs. Their other solution is to augment the WSN with IP. However, they do not provide any performance metrics evidence in the form of a simulation or practical implementation.

Gallucio et al. [11] proposed SDN-WISE to reduce control information exchange between the controller and the sensor nodes. Furthermore, sensor nodes can be programmable as finite state machines which enables them to make decisions to reduce the interaction with the controller. However, this solution has not been implemented in real sensor nodes such as TelosB [12], Zolertia Z1 [13], etc.

De Oliveira et al. [14] proposed TinySDN which enables multiple controllers in the WSN. It is based on TinyOS and it consists of: the SDN-enabled sensor node, which is a data plane component, and the SDN controller node, which is the control plane component where all the intelligence resides. TinySDN was designed to be hardware independent. However, they do not provide any performance metrics regarding the benefits that SDN can offer to WSN's as the evaluation they provide lacks sufficient detail.

In this paper, we demonstrate for the first time and implement on Contiki OS [15], a software-defined wireless sensor network framework for 6LoWPAN (SD-WSN6Lo). It consists of two main components: an SDN Sensor Node, which forwards packets based on received information from the controller, and an SDN Controller Node, where all the

978-1-5386-3395-3/18/$31.00

network intelligence resides. It is different from the prior work presented above in several key areas: it uses the uIPv6 stack, a simulation is provided using emulated Sky [16] and WiSMote [17] devices, and performance metrics are provided to indicate the potential and benefits of SDN in WSN's.

Simulations were conducted using the Contiki COOJA simulator [18] and the results demonstrate the feasibility of SD-WSN6Lo in terms of controlling and reducing the power consumption of the sensor nodes as an example of the system's dynamic capabilities.

The remainder of this paper is organized as follows. In Section II, we introduce the SD-WSN6Lo concept and design. In Section III, we present the implementation and a detailed explanation of the simulation scenario. Section IV examines the simulation results and, finally, Section V summarizes the work and presents the conclusions as well as ideas for future work.

II. System Design

In this section, we present the proposed system model. The description of the SDN controller and SDN nodes are described in detail below.

A. System architecture

The operating system we chose for the sensors nodes is Contiki OS, which is a lightweight and open source operating system for IoT designed for resource-constrained devices [15]. The Contiki OS communication stack has three network stacks: Rime, IPv4, and IPv6. Moreover, Contiki provides a lightweight TCP/IP stack, 'uIPv6', which is an IPv6 stack for memory-constrained devices [19].

The protocol we have designed is built on top of the uIPv6 stack. Our architecture framework follows the SDN principles, as shown in Fig. 1. The Southbound API is the protocol running between the two planes and is designated 'SD-WSN6Lo'. The protocol uses the UDP transport protocol to deliver control messages between SDN Sensor nodes and the SDN Controller.

The packet format used is shown in Fig. 2. The header is comprised of *Packet length*, *message type*, *destination address* and *forwarding address*. As described in Table I.

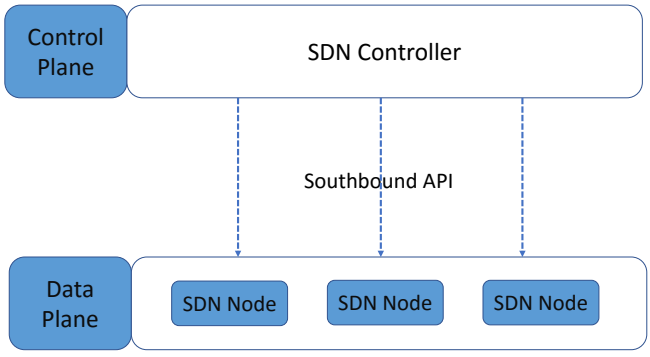

Fig. 1. The SDN architecture framework

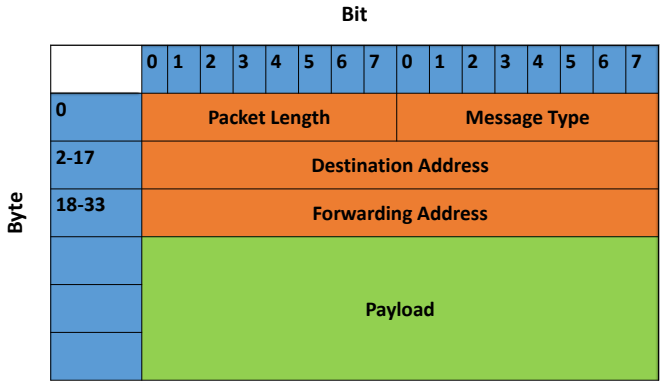

Fig. 2. Packet format

Message type defines the type of control message which can be either *Packet-in*, *Packet-out* or *Neighbors* message. Following the two types of control messages proposed in [10], we defined a *Packet-in* message as a flow set up request sent by SDN Sensor nodes when they receive a packet that is not in their routing table, and *Packet-out* as a control message from the SDN controller as a response to a *Packet-in* message. In addition, the *Neighbors* messages subtype is used, and sent by SDN Sensor nodes to the SDN Controller, advertising its neighbors for further processing.

The *Destination address* field is used by the SDN Sensor nodes in a *Packet-in* message to query the SDN Controller as to how to treat a packet from this specific address. It is also used by the SDN Controller in response to a *Packet-in* message.

The *Forwarding address* field is used by the controller to specify the forwarding address of packets that match the address in the *destination address* field.

Although IPv6 addresses are used in the packet header for illustration of the technique, the large management overhead involved can be greatly reduced in practical implementations by utilization of simple node ID's.

B. SDN Sensor Node

SDN Sensor nodes are devices that run on the data plane of the SDN architecture. They have two roles in the network.

TABLE I
Description of the Packet Header

Type	***Subtype***	***Description***
Packet length		Total length of the packet.
Message Type	Packet-in	Flow set up request.
	Packet-out	Flow set up response.
	Neighbors	Neighbors advertisement.
Destination Address		Used by Sensor Nodes to query the controller on how to handle packets with this address.
Forwarding Address		Used by SDN Controller to specify Forwarding address for packets that matches the address in destination address.

They can either be end devices or packet forwarders. An end device collects data and sends it to the SDN controller. A packet forwarder sends the packet towards the destination based on routing information.

An SDN Sensor node has three main components: *SDN Neighbor Discovery*, *SDN Controller Discovery* and an *SD-WSN6Lo protocol* component.

Immediately on power-up, the node starts the *Neighbor Discovery* component, which allows discovery of neighbors within range. This component starts broadcasting messages to nodes within range and replies to broadcast messages received to complete the Neighbor discovery in the uIPv6 stack.

The *SD-WSN6Lo protocol* is the main component of the SD-WSN, which is responsible for setting up all UDP connections, building and parsing SD-WSN6Lo segments, and implementing *Packet-in*, *Packet-out* and *Neighbors* messages.

C. SDN Controller Node

The SDN Controller Node is a component of the Control Plane. All the network intelligence resides here. This node computes SDN controller tasks such as routing algorithms, user application requirements, and runs the three components as per the SDN Sensor Nodes.

D. SDN Controller Discovery

The path to the controller is found based on the rank of the nodes. Nodes calculate their rank based on the number of hops to the controller and they broadcast this value to neighboring nodes. The receiving node updates its rank only if the received rank is better than its current rank. The frequency of checking rank messages is defined by the user and can be modified by the SDN Controller Node at any time.

E. SDN neighbors message

This component sends neighboring node addresses to the SDN controller along with their Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI). Table II shows an example of the neighbor table sent by Node 4 to the SDN Controller.

The RSSI is a measurement (in dBm) of the power level at the received radio signal, whereas LQI is an estimation of the current quality of the received packet [20]. It is worth mentioning that LQI is meant to be used in practical implementations because in a simulated environment is not yet modeled. The COOJA UDGM (Unit Disk Graph Medium) model obtains the RSSI value based on the distance between sensor nodes and uses 37 as default for all LQI values [21]. Therefore, we only use RSSI in our simulation results.

TABLE II
EXAMPLE OF A NEIGHBOR TABLE SENT TO THE SDN CONTROLLER NODE

Node 4		
Node Address	***RSSI(dBm)***	***LQI***
fe80::212:7401:1:101	-47	37
fe80::212:7405:5:505	-67	37
fe80::212:7406:6:606	-57	37

III. IMPLEMENTATION

The aim of this section is to demonstrate the individual elements, the overall orchestration capabilities, and practicality of the proposed SD-WSN6Lo framework. We look at an example scenario where we wish to control and optimize each nodes' power consumption remotely. This demonstrates the use of SDN in WSN, using the uIP6 stack, by doing complex operations at the SDN Controller Node while keeping the SDN Sensor Nodes as simple as possible. This also demonstrates the power of SD-WSN6Lo in terms of adaptability, and how SDN can positively influence the power consumption performance of a network.

In order to achieve the above goals, we developed the proposed SD-WSN6Lo framework in Contiki OS [15]. The simulations were conducted in COOJA [18] which is the Contiki network simulator. Using COOJA along with standard C for simulation results, significantly reduced the development time. Moreover, COOJA facilitates deployment of different topologies, and the nodes' transmission power outputs can be modified while the simulation is running. The latter is used to provide us with a power consumption analysis.

Two different sensor nodes types were used in the simulation. A Sky node is used as an SDN Sensor Node and a WiSMote as an SDN controller Node. Both types support Contiki OS. A Sky node is an ultra-low power wireless embedded system for sensor networks, using a CC2420 RF IEEE 802.15.4 radio wireless transceiver, an MSP430 microcontroller, 10KB RAM and 48KB of flash memory [16]. A WiSMote is a low consumption wireless embedded system for sensor networks, it has a CC2520 RF IEEE 802.15.4 radio wireless transceiver, an MSP430 5 series microcontroller, 16 KB RAM and up to 256KB of flash memory [17].

A. Simulation Scenario

We have considered a scenario with six sensor nodes running SD-WSN6Lo to demonstrate the key features without getting unnecessarily complicated. One node is a WiSMote sensor node which is the SDN Controller (Node 1), and the other five nodes are Sky motes which are SDN Sensor Nodes, as shown in Fig. 3.

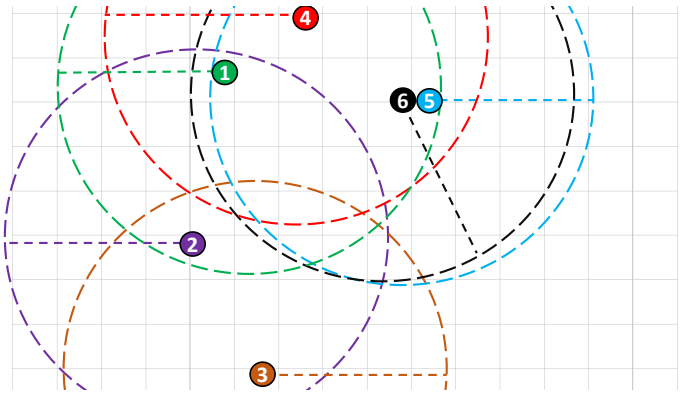

Fig. 3. SDN-WSN6Lo simulation scenario on a 10-meter grid

The SDN Sensor Nodes run the components described in Part II-B above. The Neighbor Discovery component is first to start, followed by the SDN Controller Discovery component, and finally, the SD-WSN6Lo protocol starts and sends the SDN neighbors message to the SDN Controller.

The SDN Controller has been programmed to perform the following tasks;

a) Compute the Minimum Spanning Tree (MST): all vertices and edges in the network are received via the SDN neighbor's messages, and the SDN Controller then runs them in Kruskal's Algorithm [22]. This builds up an MST of the network where the edge weights of the graph are given by the RSSI values. Vertices (Nodes) only forward and receive packets from adjacent vertices in the MST.

b) Send Packet-Out: since nodes in the MST should only receive and forward packets from adjacent nodes, the Breadth First Search (BFS) algorithm [23] has been programmed into the SDN Controller to find the path to the node and deliver the corresponding flow set up to every other node in the network.

c) Transmission power adjustment: having built the MST, we want nodes to only communicate with adjacent vertices in the MST. Thus, the SDN Controller calculates and sends a transmission power adjustment message to each node in the network to adjust their transmission power based on their longest edge in the MST.

To calculate the transmission power used by each node in the network, we have characterized the RSSI as a function of distance. By measuring the strength of wireless signal between two nodes, the distance between nodes can be obtained by a function of the RSSI attenuation and the distance [24]. The UDGM channel model degrades the signal strength with the increase of the distance between sensor nodes [21]. The RF output power of the CC2420 radio chip in the sky node can be controlled by programming the 4-bit PA_LEVEL field of the Transmit Control Register (TXCTRL) [20] as shown in Table III. We are deliberately using the ideal cases of power level range at the moment, to demonstrate the basic principles involved. But, we will revisit this issue in future work when we refine the model to care of practical issues like shadowing, interference, etc. Yet, For practical deployments with high obstacle dense propagation topologies, path loss models such as MultiWall-Floor model [25] can be used to achieve greater accuracy for the distance reached by each output power set-up.

The simulation to characterize the RSSI as a function of distance has been done in COOJA using two sky nodes. One node is fixed and configured as a receiver node, and the other node is configured as a sender node. It is then moved in five-meter increments. The receiving node then measures the RSSI value at each increment. The simulation was done for every PA LEVEL. It was found that a sky node transmitting at 0dBm (the maximum power) cannot be farther than 45 meters from an adjacent node.

The relationship between different transmitting power levels and distance range is also shown in Table III.

TABLE III
OUTPUT POWER SETTINGS OF THE CC2420 RADIO AND MAXIMUM DISTANCE REACHED

PA_LEVEL	***Output Power (dBm)***	***Current Consumption (mA)***	***Maximum Distance(m)***
31	0	17.4	< 45
27	-1	16.5	< 40
23	-3	15.2	< 35
19	-5	13.9	< 29
15	-7	12.5	< 22.7
11	-10	11.2	< 16.5
7	-15	9.9	< 10
3	-25	8.5	< 4.5

B. Measurement of energy consumption

The measurement of energy consumption is done using Powertrace, which is a network-level power profiling for low-power wireless networks [26]. It tracks the duration of each power state: CPU, Low Power Mode (LPM), transmitting, listening, idle transmitting and idle listening.

The *energy consumption* of a node in a particular power state is calculated as follows.

$$Energy(mJ) = \sum_{k=1}^{n} P_k T_k$$

Where n is the total number of samples. T_k is the time, at sample k, during the node has been in a particular power state. P_k is the power consumption (in mW), at sample k, in a particular power state.

The total energy consumption of a node is the sum of the consumptions in each power state.

IV. SIMULATION RESULTS

Calculations are done based on the nominal values of the operating conditions shown in Table IV for Sky [16] and WiSMote [27], [28] motes. We use 3 V as the nominal value for the power supply in both mote types.

The SDN Controller computes the MST based on a total of eight edges received from all nodes. The edges in the MST are the links between nodes (5, 6), (4, 1), (4, 6), (2, 3) and (2, 1). The nodes adjust their transmission range accordingly, as

TABLE IV
OPERATING CONDITIONS OF SENSOR NODES

		Min	***Nom***	***Max***	***Unit***
Sky	Supply Voltage	2.1		3.6	V
	Receiving		21.8	23	mA
	Transmitting		19.5	21	mA
	Active Mode		1800	2400	μA
	Low Power Mode		54.5	1200	μA
WisMote	Supply Voltage	2.2		3.6	V
	Receiving		18.5		mA
	Transmitting		33.6	37.2	mA
	Active Mode		2200	2600	μA
	Low Power Mode		1.69	2.2	μA

indicated in Fig. 4 by reduced radius circles compared with those of Fig. 3.

The total energy consumed by each node in the simulation for a runtime of 20 minutes is shown in Fig. 5. The simulation time was chosen to allow a sufficient number of exchanged messages between nodes to occur, which would then allow reasonable conclusions to be drawn about the practicalities of the new scheme. This figure shows a comparison between the energy consumed when nodes are running at full transmission power (0dBm) and when the nodes adjust their transmission power based on messages from the controller node. The figure shows that nodes 2, 3, 4, 5 and 6 modified their transmission power from 0dBm to -1dBm, - 3dBm, -5dBm, -15dBm and -5dBm, respectively. As expected in an MST, intermediate nodes consume more energy than leaf nodes. Note that we assumed that the SDN Controller has no limitations on power resources, so its radio transmission power was not reduced in this example scenario. Thus, its energy consumption was excluded from the analysis.

The total amount of energy saved by each node after the SDN Controller sends the power adjustment messages are shown in Fig. 6. Note that the greatest reduction is achieved by node 5 because its transmission power can be reduced by the largest amount given its close proximity to its nearest neighbor.

The percentage of the energy consumed by each node per power state is shown in Fig. 7. Depending upon the location of the node in the MST, it can spend more or less time, and hence energy, in a particular power state. Moreover, SDN Sensor Nodes spend most energy in their transmitting and receiving states. Therefore, using the proposed SD-WSN6Lo framework it would be possible to further reduce the energy consumption by centrally managing the times of nodes listening and transmitting depending upon, for example, the time of day or the real-time application requirements.

The description of memory usage of nodes is shown in Table V. The SDN Sensor Node, running in a Sky device, used approximately 84% and 89% of RAM and ROM memory, respectively. The SDN Controller Node, running in a WisMote device with 256 KB of flash memory, used approximately 63% and 18% of RAM and ROM memory, respectively. The code resides in the memory segment called *text*. In the *data* memory segment is stored initialized variables and in *bss* memory segment resides variables that are uninitialized [29]. Due to the memory constraints of sky mote [16], a more powerful sensor node such as [30] can be used to add new features for instance security algorithm, in-network processing, etc.

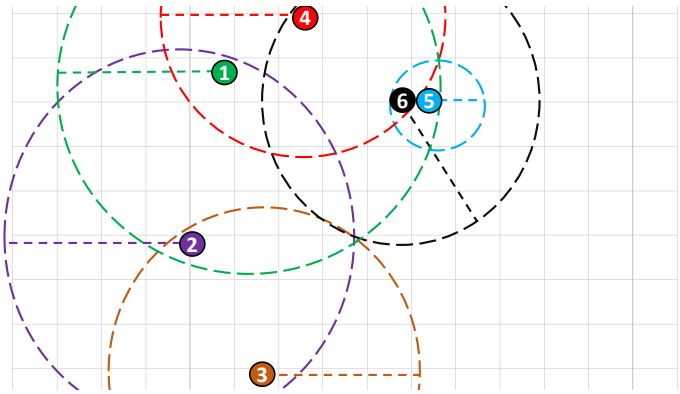

Fig. 4. Simulation scenario - Transmission power adjusted

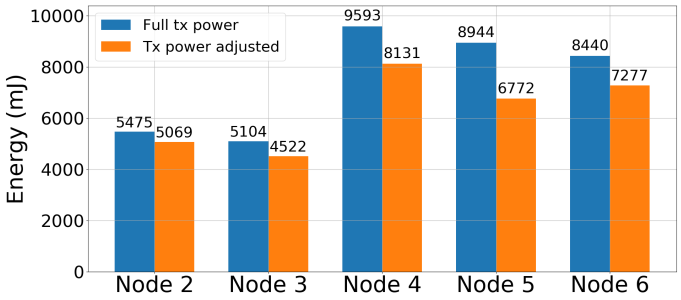

Fig. 5. Total energy consumed by each node in 20 minutes

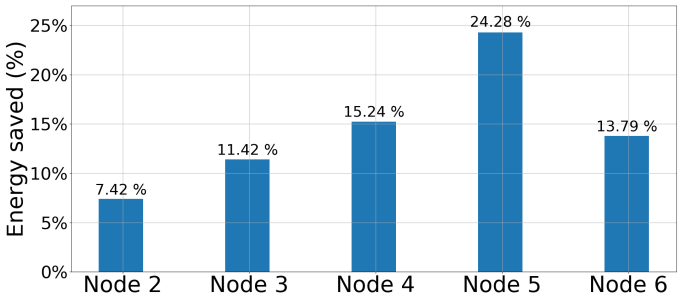

Fig. 6. Total percentage of energy saved by each node

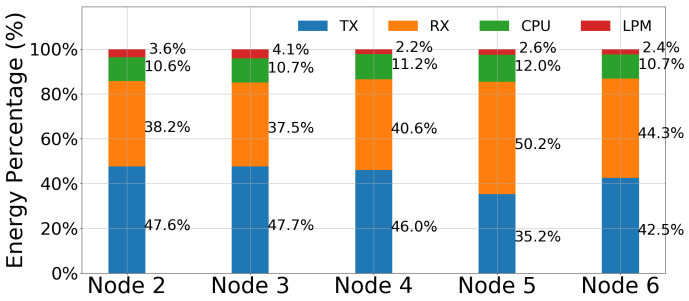

Fig. 7. Percentage energy consumed per node in each power state

V. Conclusions and Future Works

In this work, we have presented SD-WSN6Lo, a Software-Defined WSN framework for 6LoWPAN networks. It is built

TABLE V
Memory Consumption

Application	text (B)	data (B)	bss (B)	Total (B)
SDN Sensor Node	42656	242	8206	51104
SDN Controller Node	47657	268	10144	58069

on the application layer of the uIP6 stack provided by Contiki OS. SD-WSN6Lo follows the paradigm of SD-WSN, which decouples the data plane from the control plane. Moreover, the SDN Controller can be easily programmed to implement different network functions such as routing, QoS, firewall, etc. This novel WSN management application not only provides a management framework but also conserves the IP network structure which allows the WSN to use the TCP/IP stack in 6LoWPAN networks. This enables internet connectivity (IoT), thus allowing additional capabilities such as, for example, communication between different sensor node manufacturers that run Contiki OS, and enabling communication with other networked devices such as routers, M2M, cell phones, etc.

The simulation presented above showed the ease of changing the network topology through the SDN Controller without making any firmware modification in the SDN Sensor Nodes. Moreover, changing parameters, such as the transmission power, in the SDN Sensor Nodes can be easily achieved centrally by programming the SDN Controller which oversees the delivery of messages to each node.

Energy consumption was also considered as a metric to evaluate the performance of the SD-WSN6Lo. The results presented show how the WSN can be configured to deliver messages to every node and how their individual power consumption can be easily managed. In this case, an MST routing algorithm was deployed in the SDN Controller and it calculated the optimal transmission power for each node. It was found that using this scheme, large energy consumption savings could be made, for example achieving power reductions of over 24% for nodes in close proximity, or approximately 15% on average across all nodes.

To the authors' knowledge, this is the first attempt to use SD-WSN in a 6LoWPAN network. Several issues are open to further investigation such as security, scalability, and latency. For example, since all the network intelligence is centralized, an attacker may compromise the entire network by targeting the controller. Therefore, there is a need for adequate security and countermeasures against cyber-attacks as researched in [31]–[33]. Furthermore, due to the potentially large amount of information exchange between the nodes and controller, both the latency and network management overhead need to be examined in more detail.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] S. Vashi, J. Ram, J. Modi, S. Verma, and C. Prakash, "Internet of things (iot): A vision, architectural elements, and security issues," in *I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2017 International Conference on*. IEEE, Conference Proceedings, pp. 492–496.

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[4] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "A survey on facilities for experimental internet of things research," *IEEE Communications Magazine*, vol. 49, no. 11, 2011.

[5] I. L. W. Group, "Ipv6 over low power wpan (6lowpan)."

[6] J. Kim, J. Lee, J. Kim, and J. Yun, "M2m service platforms: Survey, issues, and enabling technologies," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 61–76, 2014.

[7] A. Aijaz and A. H. Aghvami, "Cognitive machine-to-machine communications for internet-of-things: A protocol stack perspective," *IEEE Internet of Things Journal*, vol. 2, no. 2, pp. 103–112, 2015.

[8] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A survey on software-defined wireless sensor networks: Challenges and design requirements," *IEEE Access*, vol. 5, pp. 1872–1899, 2017.

[9] M. Ndiaye, G. P. Hancke, and A. M. Abu-Mahfouz, "Software defined networking for improved wireless sensor network management: A survey," *Sensors*, vol. 17, no. 5, p. 1031, 2017.

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

[11] 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 (INFOCOM), 2015 IEEE Conference on*. IEEE, Conference Proceedings, pp. 513–521.

[12] T. Datasheet, "Crossbow inc," 2013.

[13] W. Zolertia, "platform, z1 datasheet."

[14] B. T. De Oliveira, L. B. Gabriel, and C. B. Margi, "Tinysdn: Enabling multiple controllers for software-defined wireless sensor networks," *IEEE Latin America Transactions*, vol. 13, no. 11, pp. 3690–3696, 2015.

[15] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*. IEEE, Conference Proceedings, pp. 455–462.

[16] M. Corporaton, "Tmote sky: Datasheet," 2006.

[17] WiSMote. [Online]. Available: http://www.aragosystems.com/produits/wisnet/wismote/

[18] F. Osterlind, "A sensor network simulator for the contiki os," *Swedish Institute of Computer Science (SICS), Tech. Rep. T2006-05*, 2006.

[19] M. Durvy, J. Abeill, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, and N. Finne, "Making sensor networks ipv6 ready," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, Conference Proceedings, pp. 421–422.

[20] T. Instruments, "Cc2420: 2.4 ghz ieee 802.15. 4/zigbee-ready rf transceiver," *Available at Available at http://www. ti. com/lit/gpn/cc2420*, vol. 53, 2006.

[21] P. Ruckebusch, J. Devloo, D. Carels, E. De Poorter, and I. Moerman, "An evaluation of link estimation algorithms for rpl in dynamic wireless sensor networks," in *International Internet of Things Summit*. Springer, Conference Proceedings, pp. 349–361.

[22] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network flows: theory, algorithms, and applications," 1993.

[23] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[24] L. Panxing and W. Tong, "A method for adjusting transmit power of zigbee network node based on rssi," in *Signal Processing, Communications and Computing (ICSPCC), 2015 IEEE International Conference on*. IEEE, Conference Proceedings, pp. 1–4.

[25] T. Chrysikos, G. Georgopoulos, and S. Kotsopoulos, "Wireless channel characterization for a home indoor propagation topology at 2.4 ghz," in *Wireless Telecommunications Symposium (WTS), 2011*. IEEE, Conference Proceedings, pp. 1–10.

[26] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, "Powertrace: Network-level power profiling for low-power wireless networks," 2011.

[27] T. Instruments, "Cc2520 datasheet, 2007," 2014.

[28] ——, *MSP430F543x and MSP430F541x Mixed-Signal Microcontrollers*. Datasheet, Aug. 2009 (revised Aug. 2014), vol. 2017. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430f5437.pdf

[29] Contiki, "Reducing contiki os firmware size." [Online]. Available: https://github.com/contiki-os/contiki/wiki/Reducing-Contiki-OS-firmware-size

[30] T. Instruments, "Cc2538 development kit." [Online]. Available: http://www.ti.com/tool/CC2538DK

[31] D. B. Rawat and S. Reddy, "Recent advances on software defined wireless networking," in *SoutheastCon, 2016*. IEEE, Conference Proceedings, pp. 1–8.

[32] T. Kgogo, B. Isong, and A. M. Abu-Mahfouz, "Software defined wireless sensor networks security challenges," in *AFRICON, 2017 IEEE*. IEEE, Conference Proceedings, pp. 1508–1513.

[33] S. W. Pritchard, G. P. Hancke, and A. M. Abu-Mahfouz, “Security in software-defined wireless sensor networks: Threats, challenges and potential solutions,” in *IEEE Int. Conf. of Ind. Informat., Emden, Germany*, Conference Proceedings.