A Software-Defined Management System for IP-enabled WSNs

F. Fernando Jurado-Lasso, *Graduate Student Member, IEEE,* Ken Clarke, and Ampalavanapillai Nirmalathas, *Senior Member, IEEE*

***Abstract*—Software-defined networking (SDN) offers potential pathways to overcome the management complexity of the Internet of Things (IoT). Previous studies have often been limited to software simulations or general proposals only. In this work, we design and evaluate an SDN-based management system for Wireless Sensor Networks (WSNs) using IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN). The framework is described in detail covering different data-, control- and application-plane implementations, and includes a novel addressing scheme and packet format. It also uses a centralized routing protocol, located at the SDN controller, based on the shortest path algorithm. We compare our approach with the routing protocol for low-power and lossy networks (RPL), which uses a distributed routing protocol. Hardware tests were carried out in a dynamic environment, with multiple sources of interference for different payload sizes to evaluate the impacts and practicality of SDN in WSNs. The performance comparison shows that the proposed SDN management system for IP-enabled WSNs using a centralized routing protocol outperforms the RPL protocol in terms of round-trip time (RTT), jitter, memory consumption, and packet loss rate (PLR), despite the control overhead introduced.**

***Index Terms*—Internet of Things, wireless sensor networks, software-defined networks, WSN management, topology management, task management.**

I. Introduction

THE Internet of Things (IoT) is the network of billions of interconnected devices that share information to enable services to users. IoT comprises multiple types of interconnected devices and networks [1]. Small deployments such as smart homes can have a range of different interconnected devices involving Wireless Sensor Networks (WSNs), smartphones or computers. Large deployments, such as those for smart cities, involve multiple interconnected devices as well as different types of communication networks. Thus, WSNs, transport networks, mobile networks and other communications technologies must somehow all be brought seamlessly together to provide future services to users. In an IoT network, interconnected devices and networks need to be remotely managed and reconfigured, regardless of vendor or type, to provide an interoperable, scalable and reconfigurable network.

The Software-Defined Networking (SDN) paradigm has emerged as a promising solution to the aforementioned problems. SDN simplifies the creation and introduction of new abstractions into the network by separating the control logic, implemented in a logically centralized controller, from the network infrastructure [2]. The control plane then runs the most energy intensive functions, leaving the data plane to act as a simple forwarding device. This communicates with the upper layer, or application layer, using s so-called *northbound Application Programming Interface (API)* and with the lower layer, called the data plane, using the *southbound API.*

WSNs are considered as an enabling technology of IoT. These are composed of wireless sensor nodes which can work cooperatively to achieve a common goal [3]. Wireless sensor nodes have a complete embedded system integrated with sensors, power source, communication radios and processing power. They are considered tiny computers because of size and communication capabilities, and are ideal to enable the connectivity between objects.

The implementation of SDN in WSNs is a challenge. SDN was originally designed for wired networks, so control packets are sent through a dedicated channel. However, in WSNs there is only a single channel for transmission of both data and control packets. Additionally, resources in WSNs are usually scarce making the practical implementation of SDN challenging.

The implementation of SDN-based management approaches in WSNs have been surveyed by several authors [4], [5], [6]. However, many publications lack practical comparison and performance evaluation against rival WSN protocols. Most of the research undertaken to date has been limited to software simulations or proposed general frameworks. Few analyses have been done and little evidence provided regarding the benefits that SDN could potentially bring to IoT. Many works in the literature use the Zigbee [7] protocol for WSNs. However, we propose here a software-defined wireless sensor networks (SD-WSNs) management system that brings the flexibility to manage and reprogram the data plane using 6LoWPAN technology. This work differs from existing solutions, and that first introduced in [8], by providing the following novel features: (i) a complete description and implementation of the communication protocols between planes, (ii) use of an operating system, (iii) a control plane able to reconfigure the network topology, vary the transmission power of nodes, and handle task requests, (iv) an application plane having management software to provide secure access and network information and, lastly, (v) an experimental performance comparison with the RPL protocol, the routing protocol of choice for IoT. The aim of this paper is to demonstrate the benefits and impacts of removing the process-intensive and energy-consuming functions from the 6LoWPAN protocol stack and adding a

Manuscript received January 15, 2019.

The authors are with the Department of Electrical and Electronic Engineering, The University of Melbourne, Victoria 3010, Australia (e-mail: fjurado@student.unimelb.edu.au, clak@unimelb.edu.au, nirmalat@unimelb.edu.au).

management system to the wireless network infrastructure. We then compare it with the traditional WSNs approach.

In this work, we first review previous research efforts in SD-WSNs, before we then describe extensions to our previous work done in [8] to further improve a novel and practical software-defined management system for IP-enabled WSNs. This is achieved in several ways: (i) the packet format has been altered to avoid fragmentation due to large overheads, (ii) the application layer has been incorporated into the scheme to establish the northbound API which enables communications with the control plane, (iii) we have set up a practical evaluation scenario to test our SDN approach and, (iv) a performance comparison, in terms of round trip time (RTT), Jitter, Packet Loss Rate (PLR) and memory consumption between our SDN approach and the IETF RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks) routing protocol [9] is presented .

The remainder of this paper is organized as follows. Section II reports research efforts to enable SDN in WSNs. In Section III we present a detailed description of each layer of the new framework. In Section IV, the hardware used and the testbed topology are described. Section V explains the performance metrics used and provides the results analysis and discussion while, finally, in Section VI the conclusions are drawn.

II. Related Works

SDN has emerged as a novel solution to solve the management complexity of WSNs and IoT. Previous efforts to enable SDN in WSNs can be found in the literature but the majority lack experimental validation.

Luo et al. [10] proposed Sensor OpenFlow as a southbound protocol with the goal of making the WSN infrastructure reprogrammable by customizing the flow tables. The fact that WSNs are considered to be data-centric and attribute-based in comparison to traditional address-centric networks, the authors proposed two solutions: (i) to use ZigBee 16-bit network addresses, and concatenated value pairs which let packets be routed based on the attributes, and (ii) to enable the Internet Protocol (IP) protocol in WSNs. In contrast with OpenFlow, Sensor OpenFlow supports in-networking processing, but no evidence is provided for any type of improved performance with their proposed southbound protocol.

TinySDN [11] was presented as an approach to reduce the latency of multihop networks, and eliminate the WSNs dependence on a single SDN controller, by using multiple controllers. Researchers adopted TinyOS [12] as the operating system for wireless sensor nodes. Their analysis was done based on the latency for a sensor node to obtain an SDN controller, and this was then compared to the Collection Tree Protocol (CTP) [13]. Even though TinySDN improved the latency of controller assignments by using multiple controllers, CTP still outperformed TinySDN in latency when sending a packet to the sink after the establishment of a flow had concluded.

In [14] SDN-WISE was proposed as a solution to reduce the large control overhead in the network. Two objectives were defined: (i) reduce the number of control packets in the southbound API and (ii) program the sensors as finite state machines (FSM's) to support operations that cannot be supported by stateless solutions. The performance evaluation was achieved by measuring: (i) RTT, (ii) the efficiency (the ratio of payload bytes to total bytes in the network), and (iii) the controller responses to requests from sensor nodes for new table entries. Although their evaluation was compared to previous SDN approaches, it was not evaluated against other WSN protocols. This needs to be done to provide proof that SD-WSN will improve the quality of network management rather than introduce greater, or unexpected, inefficiencies in the network.

Soft-WSN [15] was proposed as an SD-WSN management system for IoT. The architecture is based on an SDN controller and two different management policies [15]. Sensor nodes used the IEEE 802.15.4 [16] protocol to communicate among themselves and the IEEE 802.11 [17] protocol for communication between access points (APs), the controller and server. Soft-WSN outperformed traditional WSN protocols in terms of packet delivery ratio (PDR) and energy consumption, but had large message overhead for control packets. In addition, sensor nodes and AP's have different radio communication technologies, making communications between the SDN controller and server complex.

Buratti et al. [18] are one of the few groups to compare different protocols for the Internet of Things (IoT). These were: (i) Software-Defined Wireless Networking (SDWN) which had a centralized network layer protocol and routing policies running on an external controller, (ii) ZigBee [7] and (iii) 6LoWPAN [19]. The protocols were tested and compared based on PLR, RTT, overhead and throughput. The results showed that SDWN performs better for RTT and PLR in applications where sensor nodes are fixed, whereas ZigBee and 6LoWPAN both outperform SDWN in a dynamic environment or where there is node mobility. However, the SDWN protocol modifies layer three of the TCP/IP protocol stack by adding a proprietary network layer. This leads to non-compliance with the essential IoT requirements of scalability and interoperability.

Minimizing network interference using SD-WSNs has also been investigated. Orfanidis et al. [20] planned to improve the robustness of their network by targeting different sources of interference impacting the network. They used a statistical machine learning approach to identify periodic interferences affecting the WSN performance. A test-bed with multiple sources of interference, such as Bluetooth [21] and WiFi [17] networks, was proposed. However, the proposal lacks practical details regarding an actual physical implementation or performance metrics.

There are several works in the literature that contemplate alternative schemes where the WSN infrastructure is fully reprogrammable [22], [23]. Even though these implementations bring full reconfiguration capabilities to wireless sensor nodes, the use of reprogrammable hardware increases the development complexity and cost. In addition, the high energy consumption in FPGAs is a concern as discussed in [24].

Since sensor nodes are seen as small-scale computers, they require a lightweight operating system (OS) to work [5], [25]. The two OS's that have achieved most attention so far are: (i) Contiki OS which is a lightweight and open source OS for IoT, devised for sensor nodes with limited resources [26]. It is based on the C programming language and supports three different network stacks; RIME, IPv4 and IPv6. (ii) TinyOS was designed for low power sensor nodes with limited resources but it is based on the nesC programming language [12] and supports IPv6 in its IP based protocol stack, namely, Berkeley Low-power IP (BLIP).

In this work, we will compare the performance of our SDN approach against the RPL protocol, which is the routing protocol for low-power and lossy networks (LLNs), standarized by the IETF under RFC6550 [9], supported by Contiki OS [26]. RPL was first proposed by the ROLL (Routing Over Low-power and Lossy networks) working group at IETF (Internet Engineering Force Task). The RPL protocol targets large WSNs deployments and supports applications such as industrial, commercial, home and urban networks [27]. The RPL routing protocol sits on top of the 6LoWPAN layer and is devised as the routing protocol of choice for the IoT. RPL is based on a vector distance that builds the WSN as a Direct Acyclic Graph (DAG) rooted at the sink (DAG ROOT) forming a Destination Oriented DAGs (DODAGs). These DODAGs are optimized, to minimize the cost to the root from any node, given an objective function that specifies the constraints and metrics such as latency, hop count, energy, etc [28]. Nodes within a DODAG are assigned a rank that dictates a relative position to the root node and other nodes in the DODAG. The construction and maintenance of DODAGs is achieved by sending DODAG Information Object (DIO) messages. A DIO message can contain RPL instance, RANK, DODAGID. Before joining a DODAG, every node in the network listen to neighbors DIO messages. All nodes will reach the root, once the DODAG construction has finalized.

Our previous work on SD-WSN6Lo [8] was the first attempt at a southbound protocol, integrating SDN with the uIPv6 protocol stack of Contiki OS [29]. This work aimed to reduce the management complexity of WSNs by moving the energy intensive tasks from the sensor nodes to the SDN controller. The SDN controller was programmed to reconfigure the network topology and adjust the transmission power of sensor nodes without the need for any firmware modification in the sensor nodes. SD-WSN6Lo was demonstrated to reduce the overall energy consumption of the network, but at the time no performance evaluation was carried out against other WSN protocols. The other main issue concerned the large overhead of large IPv6 addresses that are included in the packet. This made it impossible to carry large amounts of data, without fragmentation, since the IEEE 802.15.4 standard supports a maximum frame size of 127 bytes. The paper was also based on software simulations only and did not attempt to describe details of the architecture or offer any experimental evaluation. The work presented below addresses these shortcomings by examining the results from a combined hardware and software implementation of SD-WSN6Lo, involving an SDN controller, a web application and multiple wireless sensor nodes.

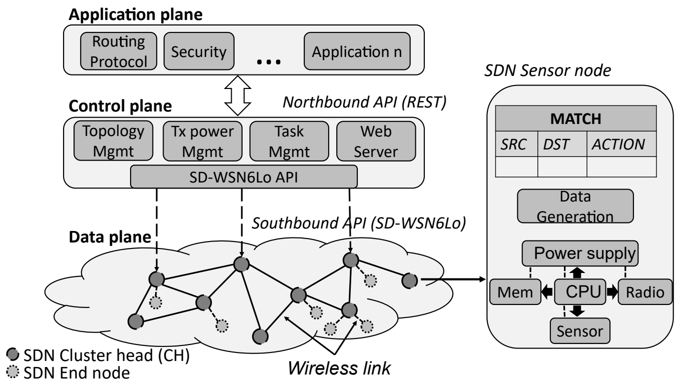


Fig. 1. Proposed SD-WSN architecture

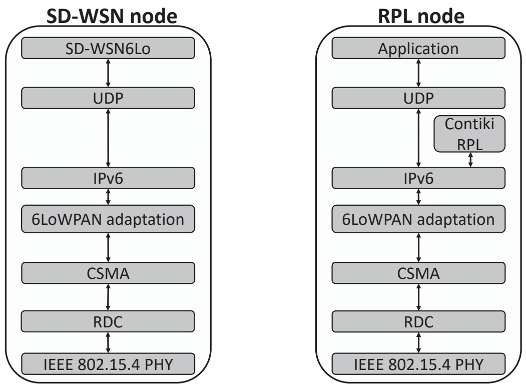


Fig. 2. SD-WSN and RPL node protocol stack

III. Proposed architecture

The proposed framework adopts an SDN architecture as shown in Fig. 1. It provides a complete and practical SDN framework for IoT using 6LoWPAN. The framework is divided into *data-*, *control-* and *application*-planes.

A. Data Plane

The data plane is the 6LoWPAN network formed by the sensor nodes, using Contiki OS as the operating system and the lightweight TCP/IP uIPv6 stack, represented by the lowest layer of the architecture in Fig. 1. Sensor nodes can be configured as either *cluster heads*, dark gray circles, or *end nodes*, light gray circles. The protocol stacks of the SD-WSN and RPL nodes are shown in Fig. 2. RPL nodes differ from SD-WSN nodes in the location of processing the routing algorithm. RPL runs the routing algorithm in each sensor node in the network, whereas SD-WSN runs the routing algorithm only in the centralized controller. Therefore, SD-WSN nodes act as a forwarding device whose forwarding table is updated by the centralized controller. Before going into the details of sensor nodes, we first explain the addressing scheme and the packet format.

1) Addressing scheme: the addressing scheme used is as follows: The first 64 bits of the IPv6 address are used for the network address. the first hextet is used for the network prefix set by the controller, followed by the two zero hextet and the node id of the sensor node. The last 64 bits are

TABLE I
DESCRIPTION OF THE FIELDS IN THE HEADER

Type	Subtype	Description
Packet length		Total length of the SDN packet.
Message Type	Packet-in	Flow setup request.
	Packet-out	Flow setup response.
	Packet-out-ack	Acknowledgment for *packet-out.*
	Neighbors	Neighbors advertisement.
Freq		Neighbor and discovery frequency.
Seq		Sequence number.
ACK		Acknowledgment.
PA		Power level.
Flags		Flags.
Task		Task configuration.

used for the host address. For example, the IPv6 address *2001:0:0:3:212:7403:3:303* of a cluster head has a network prefix of *2001:0:0:3* and host address of *212:7403:3:303*. End nodes belonging to the cluster, share the same network prefix but different host addresses. This allows the architecture to support *subnets*, an important feature which reduces the size of routing tables, and is extremely useful when sending configuration packets to sensor nodes.

2) Packet format: the SDN packet structure consist of an eight-field header, each 8-bits long. The header fields are described in Table I.

3) Cluster head nodes: these nodes can forward packets to other nodes in the network, enabling connectivity of the network and their forwarding tables can be modified on runtime. They run three different algorithms based on protothreads, which are lightweight, stack-less, low-overhead threads designed for memory constrained devices [30]. Sequential flow control is done using protothreads, which avoids the use of complex state machines:

(i) *Neighbor discovery algorithm:* discovers neighbors within the neighborhood. In order to send neighbor discovery packets, Algorithm 1 is implemented. This algorithm waits for the timer of the discovering period to expire and then sends the discovery packet containing the sensor *node ID* as payload. The smaller the discovery period is, the higher the overhead generated. But, in dynamic environments, the frequency cannot be very low in order to adapt rapid changes in the topology. To process incoming discovery packets, Algorithm 2 is implemented. This algorithm frees the CPU until a new discovery packet is received. Then it adds the neighboring cluster node and maps the link-local address to the node id received. Lastly, if the prefix has already been set then we build the IPv6 global address of the cluster head and add the route if this does not already exist.

(ii) Cluster heads use the *controller discovery* algorithm to discover the path to the controller. The Algorithm is based on ranks, and it provides the number of hops to the controller. Each cluster head broadcasts its rank. The receiving cluster head (Algorithm 3) processes it and updates its rank and path to the controller only if the received rank has a lower rank value. To send rank packets an algorithm similar to Algorithm 1 was used, but instead of sending a node id, it sends the rank and network prefix set by the controller. The Algorithm waits until the node obtains the network prefix and at least one neighbor has been found before sending rank packets.

(iii) Lastly, the algorithm of the *SD-WSN6Lo protocol*, which sits on top of the uIP6 protocol stack provided by Contiki OS, comes into play. This protocol provides various services, depending of the type of sensor node. In the case for end nodes it; (i) builds, parses and processes SD-WSN packets such as packet-in and packet-out control messages, (ii) builds, sends and processes neighbor and rank advertisement messages, (iii) keeps track of the state of neighbors (the neighbor table attributes are: *node ID, RSSI*), (iv) manages the forwarding table and (v) sets the transmission power of nodes. For the controller, the protocol additionally manages the queueing system and the communication with the northbound API. Removing and adding additional types of service is done by using pre-processor flags. This frees up memory from sensor nodes for services no longer required. The SD-WSN6Lo protocol uses the UDP transport protocol to deliver control messages in the southbound API. The uip6 protocol routes IPv6 packets. The 6LoWPAN adaptation layer enables the transmission and reception of IPv6 packets over IEEE 802.15.4 radios. Contiki OS provides multiple protocols for the MAC, RDC (Radio Duty Cycling) and

Algorithm 1 Neighbor discovery

```
Input: $t_{discovery}, node\_id$
Output: sends neighbor discovery packet.
1: $timer \leftarrow t_{discovery}$
2: while true do                                  ▷ Loop forever
3:     wait for $timer$ to expire
4:     send_broadcast($node\_id$)
5:     $timer \leftarrow t_{discovery} + t_{random}$
6: end while
```

Algorithm 2 Input neighbor discovery

```
Require: $discovery\_packet, prefix$
Ensure: adds neighbors, builds IPv6 address.
1: while true do
2:     wait for $discovery\_packet$ to arrive
3:     $nbr\_lladdr \leftarrow discovery\_packet.lladdr$
4:     $nbr\_mac \leftarrow discovery\_packet.mac$
5:     $nbr\_id \leftarrow discovery\_packet.node\_id$
6:     if add_neighbor($nbr\_lladdr, nbr\_mac$) then
7:         map($nbr\_lladdr, nbr\_id$)
8:         if Prefix set then
9:             $nbr\_ipaddr \leftarrow build\_global\_addr(prefix, nbr\_mac, nbr\_id)$
10:            if !route exist then
11:                add_route($nbr\_ipaddr, nbr\_lladdr$)
12:            end if
13:        end if
14:    end if
15: end while
```

Algorithm 3 Input rank packets

```
Require: $rank\_packet$
Ensure: Finds path to controller
 1: while true do
 2:     wait for $rank\_packet$                              ▷ Received rank
 3:     $nrank \leftarrow rank\_packet.rank$
 4:     $nbr\_lladdr \leftarrow rank\_packet.lladdr$
 5:     $nprefix \leftarrow rank\_packet.prefix$
 6:     $nbr\_id \leftarrow rank\_packet.node\_id$
 7:     if $nrank < rank$ then
 8:         $prefix \leftarrow nprefix$
 9:         $rank \leftarrow nrank + 1$
10:         map($nbr\_lladdr, nbr\_id$)
11:         build_my_global_addr($prefix, mac, node\_id$)
12:         add_route($ctlr\_ipaddr, nbr\_lladdr$)       ▷ update controller route
13:         $timer \leftarrow t_{random}$
14:         wait for $timer$ to expire
15:         send_rank($prefix, rank$)
16:     end if
17: end while
```

radio layers.

4) End nodes: sensor devices configured as end nodes only send and receive packets. They do not forward packets to other nodes in the network and have a smaller firmware size than cluster heads. They also do not perform cluster discovery, controller discovery, and forwarding of packets, etc. Tasks in end nodes are reconfigurable via control packets coming from the SDN controller.

The discovery of cluster heads is done using the *Neighbor Discovery Protocol (NDP)* for IPv6 [19]. NDP uses the *ICMPv6 (Internet Control Message Protocol version 6)* [31] protocol to perform functions for the goal of router solicitation, router advertisement, neighbour solicitation, and neighbour advertisement. *End nodes* receive NDP router advertisements and retrieve the prefix set by the controller. Then, they create their own global IPv6 addresses from it. The creation of the global IPv6 address is done in such a way that the node belongs to the subnet of the cluster head, as discussed in Section III-A1.

5) Communication between SD-WSN nodes: to enable communication between SD-WSN nodes, we use IEEE 802.15.4 radios, commonly used for low power and lossy networks, along with the 6LoWPAN adaption layer to send and receive IPv6 packets over IEEE 802.15.4. In addition, the vast majority of sensor nodes available in the market use IEEE 802.15.4 radio technology. Consequently, we assume sensor nodes support the IEEE 802.15.4 communication technology.

B. Control Plane

This plane hosts all of the network intelligence. Most of the energy intensive functions of the network also live in the control plane. The protocol stack of the controller is shown in Fig. 3. The Controller sends and receives packets to both the 6LoWPAN infrastructure (Data plane) and application layer.

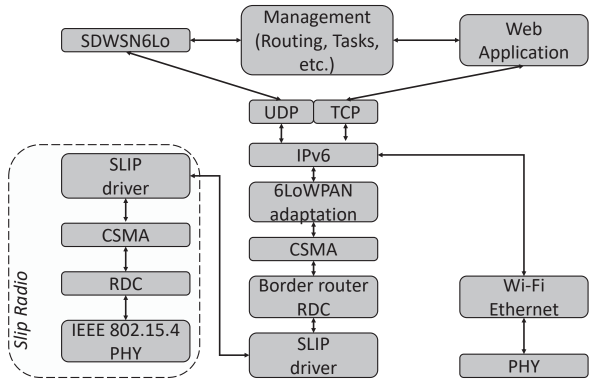


Fig. 3. The controller's protocol stack

This plane is programmed to perform the following tasks: handle communication between planes, run routing algorithms, build control packets, and provide a queuing system to reliable deliver control packets. This plane makes use of the SD-WSN6Lo protocol to deliver control packets to the 6LoWPAN infrastructure. The tasks in detail are:

1) Communication between planes: the control plane communicates with the plane above via the internet, as shown in Fig. 1, using HTTP request and HTTP Post messages [32].

The communication with the sensor infrastructure (Data plane) is done using a *slip radio*. A slip radio enables the controller to send IEEE 802.15.4 packets to the sensor nodes. The controller, hosted in a Linux computer, uses the Serial Line Internet Protocol (SLIP) to encapsulate Internet Protocol packets and send them over serial ports [33]. The slip radio forwards the packets using the IEEE 802.15.4 radio. When receiving a packet, the slip radio processes the incoming packet and sends it to the controller through the serial interface. The controller then de-encapsulates the packet, processes and sends it to the layer above.

SD-WSN nodes can generate control packets either using a periodical or reactive approach. The periodical approach is configured by the controller to advise SD-WSN nodes to send an update of their neighbors with a specified frequency. The reactive approach is generated by SD-WSN nodes either when detecting a change in their neighbors, or when they receive a packet whose destination route is unknown. This approach immediately warns the controller about potential changes in the network topology (e.g. due to interference, battery depletion, etc.).

2) Routing Algorithm: to demonstrate the reconfiguration capabilities of the proposed architecture, we have implemented the *Shortest Path* algorithm [34]. This algorithm finds the shortest path, which can be measured using various means, from a source to a destination in a given graph. This routing algorithm is often used to reduce the number of hops to destination. We also included the capability to intelligently adapt the transmission power of the nodes based on the longest link in the solution of the shortest path problem. In this paper, we used the Received Signal Strength Indicator (RSSI) as the cost matrix for the shortest path algorithm, which is an estimation of the power level of the received signal. The implementation uses the Dijkstra algorithm [34] and it is shown in Algorithm 4.

Algorithm 4 Shortest Path based on RSSI

```
Input: $ntwk\_table$
Output: Finds shortest path
 1: $Edges \leftarrow nbr\_table\_edges$
 2: $n_{vertex} \leftarrow number\_vertices(ntwk\_table)$
 3: $n_{edges} \leftarrow number\_edges(ntwk\_table)$
 4: Let C be a $n_{vertex} \times n_{vertex}$ cost matrix
 5: $C \leftarrow ntwk\_table.rssi$
 6: if $n_{vertex} > 1$ then
 7:     if Compute_Dijkstra($C, n_{vertex}, n_{edges}$) then
 8:         return edges
 9:     else
10:         return NULL
11:     end if
12: end if
```

3) Construction of control packets: this finds the correct paths to deliver control packets to each sensor node in the network and builds them. The main complexity is in delivering control routing packets to nodes without a direct link to the controller. How can this be done when every node on the path to the destination must first be configured?

The algorithm proposed to deliver and build control packets to the 6LoWPAN network is shown in Algorithm 5. The following assumptions are made:

- The controller node ID is equal to one, and all other nodes will have ID's of a higher number.
- All cluster heads are placed first.

Algorithm 5 Generation of control packets

```
Require: $edges, n_{vertex}, FLAG$
Ensure: Control packets for network.
 1: for $i \leftarrow 2, n_{vertex}$ do                          ▷ avoid controller id
 2:     $nbr\_ctrl \leftarrow$ find_path($i, 1, n_{vertex}, edges$)
 3:     $nbr\_node \leftarrow$ find_path($1, i, n_{vertex}, edges$)
 4:     $nearest\_nbr \leftarrow nbr\_node$
 5:     while $nbr\_node \neq i$ do
 6:         $nbr \leftarrow nbr\_node$
 7:         $nbr\_node \leftarrow$ find_path($nbr, i, n_{vertex}, edges$)
 8:         if FLAG then                                     ▷ adjust power?
 9:             $PA \leftarrow cal\_pa(edges, n_{vertex}, nbr)$
10:         end if
11:
12:         $pkt \leftarrow pkt(nbr, i, nbr\_node, nearest\_nbr, PA)$     ▷
    pkt(node_id, dest_node, forward_node, nbr_node, pa)
13:         add_queue($pkt$)
14:     end while
15:     if FLAG then
16:         $PA \leftarrow cal\_pa(edges, n_{vertex}, i)$
17:     end if
18:     $pkt \leftarrow pkt(i, 1, nbr\_ctrl, nearest\_nbr, PA)$
19:     add_queue($pkt$)
20: end for
```

To find a path from node v to node w, we use the Depth First Search Algorithm (DFS) [35]. The DFS algorithm traverses the tree data structure by exploring each branch as far as possible before backtracking.

4) Queuing system: the distribution of control packets is of high priority as we want to ensure that configuration packets sent to update the routing table of a node are correctly acknowledged. To achieve this, we propose a FIFO (First In First Out) *queuing* system along with an *acknowledgement*. The queue will store configuration packets built by the controller in order of arrival and the protothread will try to deliver the packet to the destination until it gets acknowledged by the receiver. The algorithm is also based on protothreads and it is shown in Algorithm 6.

The delivery of control packets can be done in realtime by the control plane. This can be triggered either by the application- or data-plane. The process of reconfiguring the network directly affects the PDR and delay performance via collision of control- and data-packets in the wireless communication medium. These data packets may also have been delayed in queues and multiple transmissions, as discussed in [36]. As the frequency of control information exchanged with the data plane increases, then the control overhead will also increase.

Algorithm 6 Delivery of a control packet

```
Require: $queue$
Ensure: Reliable delivery of a control packet.
 1: $retrans \leftarrow 0$
 2: do
 3:     wait until something in $queue$
 4:     $pkt \leftarrow head(queue)$                ▷ first element in queue
 5:     $seq\_number \leftarrow random$                 ▷ 8-bit number
 6:     $ack \leftarrow 0$
 7:     $payload \leftarrow get\_pkt(dest\_node, forward\_node)$
 8:     $build\_sdwsn6lo\_pkt(payload, pkt.node\_addr)$
 9:     $retrans \leftarrow retrans + 1$
10:     $timer \leftarrow timeout$
11:     wait for $timer$ expire or good $ack$
12: while $timer$ expired and $retrans < 5$)
13: remove $pkt$ from $queue$
```

C. Application Plane

This plane makes use of the communication channel between the application and control plane to send instructions or received data from the control plane.

The application plane hosts a management software that handles three different tasks: secure access (middleware), collecting network information and communication with the controller.

1) Secure Access: due to the sensitive configuration functions of the IoT network that reside in the control plane, some form of protection must be used to filter out unauthorized connections to the controller. To achieve this, a *middleware* function is proposed. It uses a *MongoDB* database [37] to store authorized network administrators. Only after successful authentication will the middleware allow access to the network configuration menu.

2) Collection of Network Information: this function overcomes a common problem in many prior proposals where little is known about the state of the network. It provides information about network routes and neighboring nodes along with RSSI, which is a useful measurement to build cost matrices in routing protocols.

Many previously suggested network architectures are inflexible, mainly because sensor nodes are considered to be autonomous systems that do not allow the operator to easily reconfigure or customize the network without changing hardware. The proposed architecture here allows easy, central reconfiguration of the routing protocol with a single click in the management software.

3) Communication with the controller: to achieve this, the application layer communicates with the controller using *HTTP request* and *HTTP Post* events. This layer has to be configured with the controller IP address to function, and for security reasons the network administrator has to previously authenticate with the server via the middleware function. The management software can sit at any location of the network but, to avoid and prevent security issues, it is recommended that it is placed inside the local network of the controller.

IV. Experimental Platform and testbed

To evaluate the actual performance of the proposed framework architecture, SD-WSN infrastructure was set up using the TI CC2538 Evaluation Module (EM) [38] for cluster heads, end nodes and the slip radio. The CC2538 EM was chosen because of its full software support for Contiki OS and 6LoWPAN. The CC2538 EM has a CC2538 System-On-Chip (SOC) for 2.4-GHz IEEE 802.15.4-2006 and ZigBee applications [39], a PCB antenna, and a micro USB connector for USB testing. It can be powered by either USB, batteries or an externally regulated power supply. The CC2538 SoC has a powerful ARM Cortex-M3 microcontroller with up to 32 KB on-chip RAM and up to 512KB on-chip flash memory, a clock speed of up to 32 MHz, support of On-Chip Over-the-Air Upgrade (OTA) and an IEEE 802.15.4 radio [39]. The EM has enough resources to support Contiki OS and enough room to introduce new applications. The SmartRF06EB [40] was also used for debugging in the development stage.

The controller sends and receives packets from the 6LoWPAN infrastructure using a slip radio that is directly connected to a laptop, with a USB 2.0 interface, hosting the controller. The laptop is equipped with an Intel Core i7 CPU, 16 GB of RAM and running Linux Mint Rosa 64 bit in a virtual machine. The controller runs on Contiki OS natively in Linux Mint. For experimental evaluation regarding the southbound API and to simplify the deployment, we placed the controller and the application layer in the same Linux machine as this does not interfere in the evaluation performance. They communicate using HTTP request and post events.

The testbed was constructed in the facilities of the Networked Society Institute, located in the Department of Electrical and Electronic Engineering at the University of Melbourne, Australia. Many staff and students in multiple offices are located around the laboratory, which provides a large amount of dynamic interference from users on the university wireless network and via their Bluetooth transmissions. Four wireless sensor nodes and one slip radio were deployed from approximately five to seven meters apart. The topology and the average RSSI values of the 6LoWPAN infrastructure are shown in Fig. 4. The location of $R4$ was chosen deliberately to ensure the receiver was close to its performance limits. This enabled us to examine the capabilities of both protocols in a WSN with both strong and weak links. This network deployment closely represents a real WSN deployment where links are impacted by the surrounding environment.

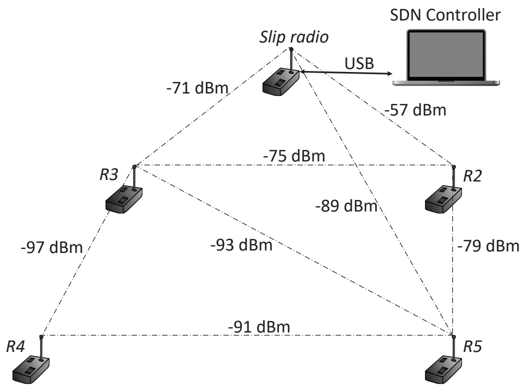


Fig. 4. Testbed setup

TABLE II
SD-WSN6Lo sensor node parameters

Parameter		Value
Protocols	Application Layer	SD-WSN6Lo
	Transport Layer	udp
	Internet Layer	uip6
	Adaptation Layer	sicslowpan
	MAC Layer	nullmac
	RDC Layer	nullrdc
	Radio Layer	IEEE 802.15.4
Payload size		20, 40, 60 and 80 Bytes
Packet rate		30 Packets/min
Total packets per payload size		1000 packets
Tx power		3 dBm
Rx sensitivity		-97 dBm
Radio Baud rate		250 kbps

V. Experimental Performance Evaluation

The performance evaluation is carried out using the hardware presented in Section IV and the network topology shown in Fig. 4. The parameters of the experiment setup for the SD-WSN6Lo sensor nodes are shown in Table II.

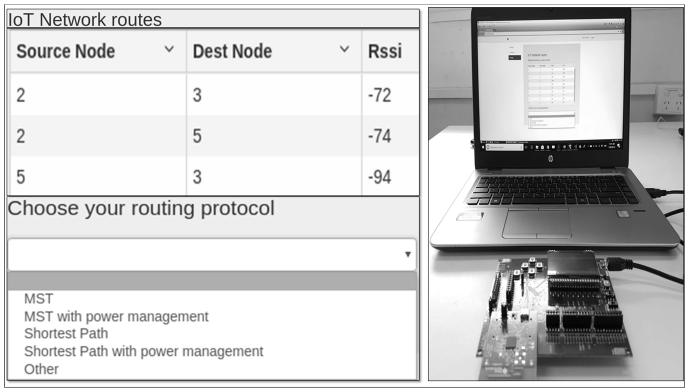
A. Performance metrics

Similar performance metrics to those used by other workers in [14], [15], [18], were also used in this work. We considered five metrics: (i) Round Trip Time (RTT), which is the time it takes for a packet to go from the source to destination node and back again. RTT is measured using the ping6 command, which uses the ICMP6 [31] protocol to send an ECHO_REQUEST datagram to invoke an ICMP ECHO_RESPONSE, with different payload sizes. Measurement of RTT allows us to assess how the latency is affected by removing the processing- and energy-intensive functions from the sensor nodes. (ii) jitter, which is the variation in the packet arrival times and is measured by calculating the standard deviation of the RTT [41]. Measurement of jitter is important because it reveals underlying problems such as network congestion and route changes. (iii) Packet Loss Rate (PLR) is the ratio of the total of packets received versus the total of packets transmitted [42]. This allows us to compare the efficiency of the two protocols in delivering data. (iv) Control overhead, which is calculated as the number of messages flowing in the network other than data packets and, lastly, (v) memory consumption, which is the total size of memory used in terms of RAM (Random Access Memory) and ROM (Read Only Memory). This metric reveals the amount of memory released when removing the processing- and energy-intensive functions from sensor nodes. Freeing up memory allow us to host extra applications in the sensor nodes. Lastly, this paper does not present any results related to energy consumption. We are currently investigating the impacts of SDN in power consumption performance and scalability in detail for our next paper. However, it is expected that as the network size increases, that more control packets will need to flow in the network. When nodes send control packets periodically, the number of control packets will increase as demonstrated in [15]. Additionally, if the control packets' time interval is small, control message overhead in the network will increase. In general, the control packets' time interval mainly depends on the network-specific requirements and network deployment [36].

B. Results Discussion

The practical evaluation is performed based on the metrics and RPL protocol discussed previously in Section V-A. We compare the performance of our SDN approach against the RPL protocol to show the impacts of SDN in WSNs. In this case, we ran the shortest path algorithm, as the routing algorithm, in the controller. As discussed formerly in Section III, we removed the energy intensive functions from the wireless sensor nodes to a centralized controller. We adopted Contiki OS as the operating system for the wireless sensor nodes, slip radio and controller. In contrast, the RPL protocol network uses a distributed architecture in which sensors act as an autonomous system but they are also using Contiki OS as their operating system.

The experiment is carried out based on the parameters shown in Table II. Since reliability for control packets is provided by the protocol, the *nullmac* protocol for MAC layer was chosen and since no analysis regarding energy consumption is provided, then the *nullrdc* protocol for RDC layer was chosen [26]. RPL nodes also use *nullmac* and *nullrdc* for the MAC and RDC layer, respectively. For the controller, same MAC protocol is used but a null RDC implementation, that uses framer for headers and sends the packets over the slip radio instead of IEEE 802.15.4 radio, is used [26]. We deployed the WSN infrastructure as shown in Fig. 4. We start evaluating the performance of the RPL protocol, by sending 1000 ping IPv6 packets with 20 bytes of payload at a packet rate of 30 packets/min to each sensor node in the network. Then, we proceed to the next payload. We performed multiple runs and used a confidence interval of 95%. We repeat the process for our SDN approach. The performance evaluation lasts for almost nine hours, and 1600 ping IPv6 packets were sent for each routing protocol. Fig. 5 shows the developed SD-WSN platform with the management software and the controller (PC) directly connected to the slip radio by a USB cable.



IoT Network routes

Source Node	Dest Node	Rssi
2	3	-72
2	5	-74
5	3	-94

Choose your routing protocol

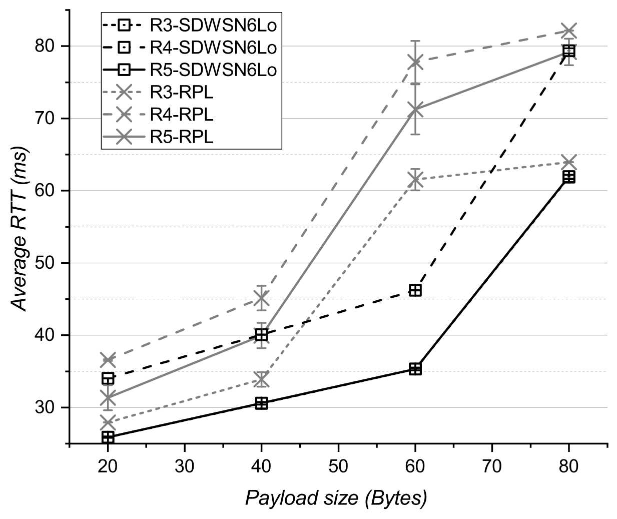
MST
MST with power management
Shortest Path
Shortest Path with power management
Other

Fig. 5. SD-WSN management interface (left), and the controller and slip radio (right)

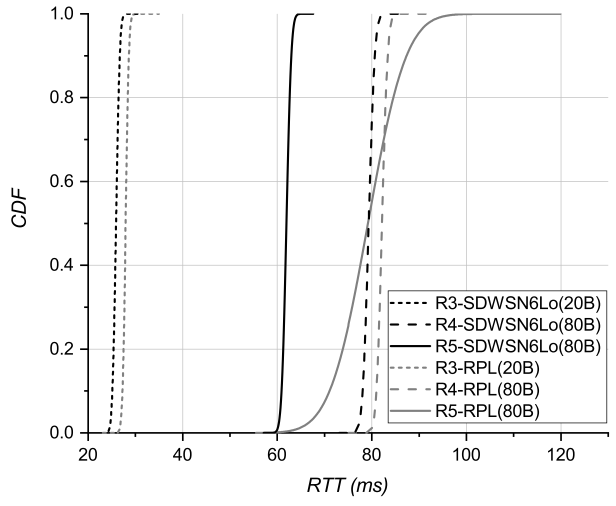
1) Round Trip Time: Fig. 6 shows the performance comparison in terms of RTT for both protocols. In Fig. 6(a), we can observe that SD-WSN6Lo outperforms RPL protocol for all payload sizes. Most of the RTT for SD-WSN6Lo, except R4, lie on the bottom line in the graph, whereas all RTT values for RPL are above it. Even though the R4 line for SD-WSN6Lo is not at the bottom line, it is below the R4 line of RPL. It was expected that R4 would have the worst RTT performance, in comparison to other nodes, because it has two of the worst RSSI links, two hops away from the controller. In addition, Fig. 6(b) represents the cumulative distribution function (CDF) of RTT for two payload sizes. The curve shape, for R5 in RPL, shows larger distributions of RTT compare to SD-WSN6Lo. SD-WSN6Lo outperforms RPL at each of the sensor nodes and payload sizes.

SD-WSN6Lo shows better performance against RPL, in terms of RTT, mainly because the controller has a global view of the WSN infrastructure and can define forwarding rules for each sensor node in the network, whereas sensor nodes in RPL act as autonomous systems and make their own routing decisions. Moreover, the controller maintains good network performance via the frequent neighbor messages sent by cluster heads.

2) Jitter: The variation in the packet arrival time is shown in Fig. 7. It is clear that SD-WSN6Lo has a smaller and a more stable jitter than RPL, as shown in Fig. 7(a). For SD-WSN6Lo, the jitter slightly increases with the payload size, whereas RPL has highly variable jitter. Fig. 7(b) illustrates this via the

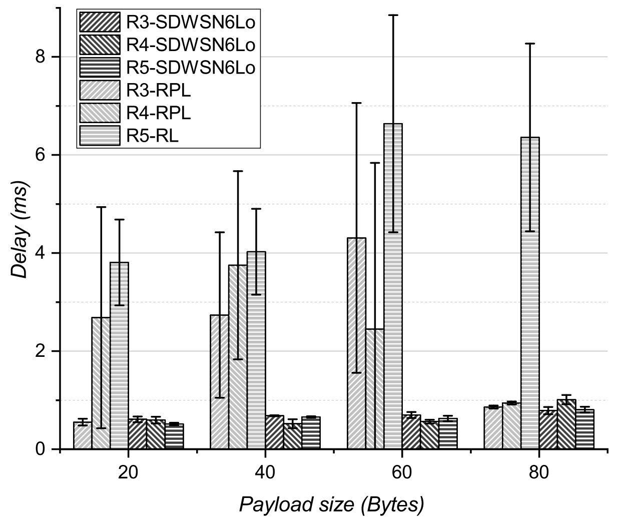


(a) RTT vs Payload size

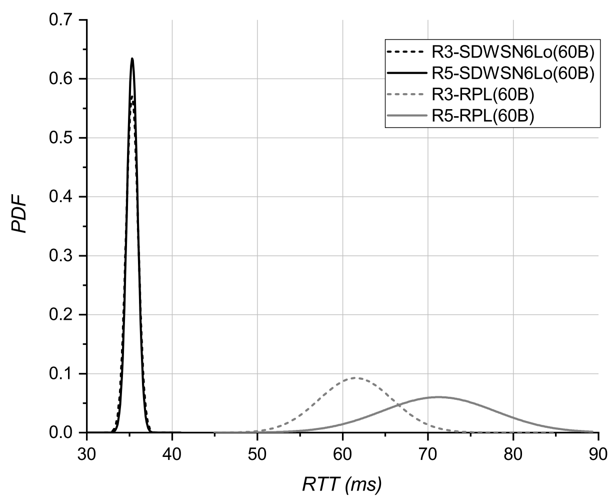


(b) CDF of RTT for various payloads for nodes R3, R4 and R5

Fig. 6. Performance comparison of RTT



(a) Jitter for different payload sizes



(b) PDF of the RTT for a 60 byte payload

Fig. 7. Performance comparison of the variation in packet arrival times

probability distribution function (PDF) of the RTT for 60 and 80 bytes of payload. The better performance of SD-WSN6Lo is mainly due to the SDN controller managing and maintaining the network connectivity in a centralized manner, which means that the processing of the routing algorithm is removed from the nodes with their limited resources and relatively unreliable wireless infrastructure. In RPL, routing between nodes can change often and the sensor nodes perform routing processing individually, which adds extra delay and jitter into the network. Thus, SD-WSN6Lo can be used for more critical applications that require smaller and more stable jitter.

3) Packet Loss Rate: Fig. 8 shows the comparison of the number of ping6 packets received divided by the total number of ping6 packets sent by the SD-WSN6Lo and RPL protocols. The figure shows the packet loss rate percentage for each sensor node at different payload sizes. We see that the sensor node with the smallest PLR percentage is R3, which is located close to the controller and so has good signal strength as shown in Fig 4. The PLR percentage for R3 increases slightly with payload size in comparison to the PLR for R4, as expected, which is located two hops away from the controller and has two of the weakest RSSI values (-91 and -97 dBm) links of the network increasing the probability of packet loss. The PLR for both protocols can be improved by including a MAC layer protocol that takes care of retransmission of lost packets. The packet loss experienced by SD-WSN6Lo is mainly due to the decoupling of the control- and data- planes, which requires control information to flow from the WSN infrastructure to the controller in both directions. This increases the congestion and collisions probabilities. Also, it forwards packets based on the routing algorithm deployed without measuring any link quality. Additionally, for applications with low PLR, a proper routing algorithm has to be run and deployed by the controller. Yet SD-WSN6Lo was still able to maintain the PLR low because of the reconfiguration capabilities via the regular of interaction between the controller and neighboring nodes.

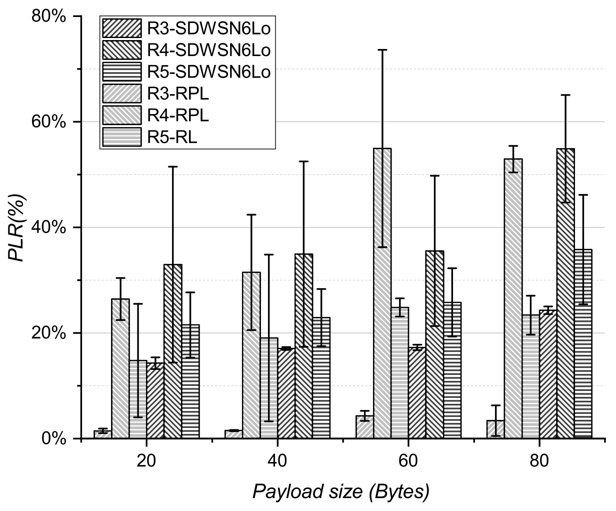


Fig. 8. PLR comparison for different payload sizes

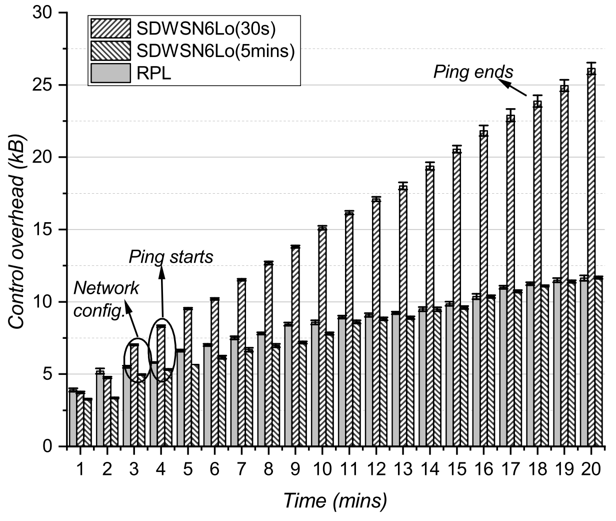


Fig. 9. Control overhead comparison

4) *Control overhead:* To measure the control overhead introduced by both protocols, we set up the network architecture in a Contiki COOJA [43] simulator and simulated the RPL protocol and the SDWSN6Lo protocol with two different advertisement periods, 30 sec and 5 mins. We then captured packets flowing in the network for 20 mins. Fig. 9 shows that even though the SDWSN6Lo approach introduced extra overhead to make the data plane reprogrammable, its performance is as good as or, in some cases, better than RPL for an advertisement period of 5 mins. As the advertisement period of SDWSN6Lo reduces, more overhead will be flowing in the network. The choice of the advertisement period mainly depends on the network-specific requirements and network deployment. For highly dynamic WSNs a small advertisement period is required to keep the controller aware of any changes in the network.

5) *Memory consumption:* Table III shows the memory usage in terms of RAM and ROM for each node type. The value called *text* in the table refers to the size of the code stored in ROM. The *data* and *bss* memory segments are stored in RAM. *data* contains the initialized variables and *bss* contains the uninitialized variables [44].

TABLE III
MEMORY CONSUMPTION

Sensor Type	text (B)	data (B)	bss (B)	Total (B)
SDN End Node	34735	364	10871	45970
SDN Cluster Head	37858	397	11607	49862
SDN Controller	158996	3370	67792	230158
RPL node	42734	429	11095	54258
Lasso et al. [8]	42656	242	8206	51104

TABLE IV
PERFORMANCE COMPARISON OF TOTAL MEMORY CONSUMPTION

Sensor Type	SDN End Node	SDN Cluster Node	RPL Node	Lasso et al. [8]
SDN End Node	-	-7.8%	-15.3%	-10.1%
SDN Cluster Head	7.8%	-	-8.1%	-2.43%
RPL Node	15.3%	8.1%	-	5.81%
Lasso et al. [8]	10.1%	2.43%	-5.81	-

Table IV compares the memory consumption of the new approach with our previous work in [8] and the RPL protocol. We can see that the SD-WSN approach significantly reduces the usage of RAM and ROM, which can either lower costs or free memory for alternative functionality in the sensor nodes. The greatest reduction is achieved in SDN End Nodes. Compared to RPL, it can save up to 15% of total memory used. This is achieved due to the separation of the control- and data-planes, which allows the processing intensive functions, such as the routing algorithms, to be moved from the node to a centralized controller.

VI. CONCLUSION

In this work, we presented a detailed software-defined networking management system to enable topology and task reconfiguration capabilities for IP-enabled WSNs. We investigated this new framework's performance via a test-bed implementation with multiple nodes in the presence of real-world sources of interference. The results presented proved the practicality and utility of the framework, and also showed its superior performance, in the testbed, against the industry-standard RPL protocol for the key attributes of round trip time, jitter, packet loss rate, and memory consumption. It achieves this by having a controller that manages and maintains the network infrastructure in a centralized manner. The controller's global view of the network infrastructure allows it to dynamically define forwarding rules for each individual node in the network. Although there was some control overhead introduced by bidirectional flow of the necessary control data packets, packet losses were still as good as, or better than, RPL in the test-bed comparison. This is a positive outcome when considering scalability of this scheme beyond the proof-of-concept presented here, and our future work will focus on this area, as well as the energy consumption introduced by SD-WSNs approaches.

This work has demonstrated the advantages of using SDN to reconfigure an IP-enabled WSN as it removes the need for site visits to upgrade firmware which are expensive. The use of SDN in large-scale deployments can potentially be achieved by the use of multiple controllers as investigated in [11].

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] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[3] 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.

[4] 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.

[5] 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. 5:1031, pp. 1–32, 2017.

[6] S. Bera, S. Misra, and A. V. Vasilakos, "Software-defined networking for Internet of Things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, 2017.

[7] Z. Specification, "ZigBee alliance IEEE standard 802.15.4k2013," 2014. [Online]. Available: https://www.zigbee.org/zigbee-for-developers/network-specifications/

[8] F. F. J. Lasso, K. Clarke, and A. Nirmalathas, "A software-defined networking framework for IoT based on 6LoWPAN," in *Wireless Telecommunications Symposium (WTS), 2018*. IEEE, Conference Proceedings, pp. 1–7.

[9] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. P. Vasseur, and R. Alexander, "RPL: IPv6 routing protocol for low-power and lossy networks," Report 2070-1721, 2012.

[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] 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.

[12] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, and E. Brewer, "TinyOS: An operating system for sensor networks," *Ambient intelligence*, vol. 35, pp. 115–148, 2005.

[13] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo, "The collection tree protocol (CTP)," *TinyOS TEP*, vol. 123, no. 2, 2006.

[14] 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.

[15] S. Bera, S. Misra, S. K. Roy, and M. S. Obaidat, "Soft-WSN: Software-defined WSN management system for IoT applications," *IEEE Systems Journal*, 2016.

[16] J. A. Gutierrez, E. H. Callaway, and R. L. Barrett, *Low-rate wireless personal area networks: enabling wireless sensors with IEEE 802.15. 4*. IEEE Standards Association, 2004.

[17] IEEE, "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," 2012.

[18] C. Buratti, A. Stajkic, G. Gardasevic, S. Milardo, M. D. Abrignani, S. Mijovic, G. Morabito, and R. Verdone, "Testing protocols for the Internet of Things on the EuWIn platform," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 124–133, 2016.

[19] I. L. W. Group, "IPv6 over low power WPAN (6LoWPAN)." [Online]. Available: https://datatracker.ietf.org/wg/6lowpan/charter/

[20] C. Orfanidis, "Ph. D. forum abstract: Increasing robustness in WSN using software defined network architecture," in *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*. IEEE, Conference Proceedings, pp. 1–2.

[21] IEEE, "Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPAN)," 2004.

[22] J. Portilla, A. De Castro, E. De La Torre, and T. Riesgo, "A modular architecture for nodes in wireless sensor networks," *J. UCS*, vol. 12, no. 3, pp. 328–339, 2006.

[23] S. Natheswaran and G. Athisha, "Remote reconfigurable wireless sensor node design for wireless sensor network," in *Communications and Signal Processing (ICCSP), 2014 International Conference on*. IEEE, Conference Proceedings, pp. 649–652.

[24] K. Goh, S. Ong, Y. Joe, P. Kusolpalin, W. Moh, and K. V. Ling, "FPGA based wireless sensor node for distributed process monitoring," in *Industrial Electronics and Applications (ICIEA), 2012 7th IEEE Conference on*. IEEE, Conference Proceedings, pp. 1934–1939.

[25] D. Zeng, T. Miyazaki, S. Guo, T. Tsukahara, J. Kitamichi, and T. Hayashi, "Evolution of software-defined sensor networks," in *Mobile Ad-hoc and Sensor Networks (MSN), 2013 IEEE Ninth International Conference on*. IEEE, Conference Proceedings, pp. 410–413.

[26] 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.

[27] J. Martocci, P. De Mil, N. Riou, and W. Vermeylen, "Building automation routing requirements in low-power and lossy networks," Report 2070-1721, 2010.

[28] J.-P. Vasseur, M. Kim, K. Pister, N. Dejean, and D. Barthel, "Routing metrics used for path calculation in low-power and lossy networks," Report 2070-1721, 2012.

[29] M. Durvy, J. Abeille, 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.

[30] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, "Protothreads: Simplifying event-driven programming of memory-constrained embedded systems," in *Proceedings of the 4th international conference on Embedded networked sensor systems*. Acm, Conference Proceedings, pp. 29–42.

[31] A. Conta, S. Deering, and M. Gupta, "Internet control message protocol (ICMPv6) for the internet protocol version 6 (IPv6) specification," Report 2070-1721, 2006.

[32] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–HTTP/1.1," Report 2070-1721, 1999.

[33] J. Romkey, "Nonstandard for transmission of IP datagrams over serial lines: SLIP," Report 2070-1721, 1988.

[34] R. K. Ahuja, *Network flows: theory, algorithms, and applications*. Pearson Education, 2017.

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

[36] S. Misra, S. Bera, M. Achuthananda, S. K. Pal, and M. S. Obaidat, "Situation-aware protocol switching in software-defined wireless sensor network systems," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2353–2360, 2018.

[37] K. Chodorow, *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. " O'Reilly Media, Inc.", 2013.

[38] T. Instruments, "CC2538 evaluation module kit - CC2538EMK," 2015. [Online]. Available: http://www.ti.com/tool/CC2538EMK

[39] ——, "CC2538 powerful wireless microcontroller system-on-chip for 2.4-GHz IEEE 802.15. 4, 6LoWPAN, and Zigbee applications," *CC2538 datasheet (April 2015)*, 2015.

[40] ——, "SmartRF06 evaluation board user's guide," Report, 2017. [Online]. Available: http://www.ti.com/lit/ug/swru321b/swru321b.pdf

[41] D. Vir, D. S. Agarwal, and D. S. Imam, "A simulation study on node energy constraints of routing protocols of mobile ad hoc networks use of qualnet simulator," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 1, no. 5, pp. 401–410, 2012.

[42] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks (5th)*. Prentice Hall, 2010.

[43] F. Osterlind, "A sensor network simulator for the contiki os," Swedish Institute of Computer Science, Report T2006-05, 2006. [Online]. Available: http://eprints.sics.se/2296/1/SICS-T–2006-05–SE.pdf

[44] T. R. usage and ROM, "contiki-ng." [Online]. Available: https://github.com/contiki-ng/contiki-ng/wiki/Tutorial:-RAM-and-ROM-usage

F. Fernando Jurado-Lasso (GSM'18) received the B.Sc. degree in electronic engineering from the Universidad del Valle, Cali, Colombia, in 2012. He also received the M.Sc. in Telecommunications Engineering degree from The University of Melbourne, Australia, in 2015. He is currently pursuing the Ph.D. degree with the Department of Electrical and Electronic Engineering, University of Melbourne.

His research interests include software-defined particularly focus on wireless sensor networks, protocols and applications for the Internet of Things.

Ken Clarke received his B.Sc.(hons) in Applied Physics from Heriot-Watt University, Edinburgh, Scotland, in 1985.

He worked in the optoelectronic and telecommunications industries in various R&D and engineering roles in both the UK and Australia from 1985-2010, before moving to the University of Melbourne where he is currently Deputy Director of the Networked Society Institute. He has published over 50 articles and book chapters, and produced five patents.

Ampalavanapillai Nirmalathas (M'98 - SM'03) received the B.Eng. and Ph.D. degrees in electrical engineering from The University of Melbourne, Australia, in 1993 and 1998, respectively. He is currently a Professor with the Electrical and Electronic Engineering Department, The University of Melbourne. He is also the Director of the Melbourne Networked Society Institute, which is an interdisciplinary research institute focusing on challenges and opportunities arising from society's transition to a networked society. He also provides academic leadership to the Melbourne Accelerator Program, which he co-founded to support entrepreneurial activities of the university community through business acceleration models. He holds two active international patents and one provisional application in the process. He has authored over 400 technical articles.

His research interests include microwave photonics, optical wireless network integration, broadband networks, and stability of Internet and telecom services. He is a member of OSA and a Fellow of the Institution of Engineers Australia.