IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 18, 2019, accepted August 8, 2019, date of publication August 19, 2019, date of current version August 30, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2936127

The Cost of Software-Defining Things: A Scalability Study of Software-Defined Sensor Networks

RENAN C. A. ALVES, DORIEDSON A. G. OLIVEIRA, GUSTAVO A. NUNEZ SEGURA, AND CINTIA B. MARGI, (Member, IEEE)

Departamento de Engenharia de Computação e Sistemas Digitais, Universidade de São Paulo, São Paulo 05508-010, Brazil

Corresponding author: Renan C. A. Alves (renanalves@usp.br)

This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brazil (CAPES)—Finance Code 001, and in part by the FAPESP Research under Grant #2018/12579-7. The work of R. C. A. Alves was supported by the São Paulo Research Foundation (FAPESP) under Grant #2016/21088-1 and Grant #2018/11295-5. The work of G. A. Núñez Segura was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior/Coordination for the Improvement of Higher Education Personnel—Brazil (CAPES)—Finance Code 001, and in part by the Universidad de Costa Rica.

ABSTRACT Software-Defined Networking is a promising paradigm for providing flexibility and programmability to computer networks. Our goal is to assess the performance of this paradigm applied to Wireless Sensor Networks. Previous evaluations are not complete, since they study small networks, do not explore crucial performance metrics, or solely examine light traffic conditions. For this, we execute simulations and a testbed experiment. The testbed shows Software-Defined Networking successfully operates in a real network. We study simulated networks up to 289 data-transmitting nodes, while assessing all the main networks metrics: data delivery, delay, control overhead, and energy consumption. We investigate important parameters for Software-Defined Wireless Sensor Networks, such as controller positioning, radio duty cycling, number of data sinks, and use of source routed control messages. The results indicate that Software-Defined Networking is feasible for Wireless Sensor Networks, presenting competitive data delivery ratio while saving energy in comparison to RPL, the Routing Protocol for Low-power and lossy networks.

INDEX TERMS Software-defined networking, Internet of Things, wireless sensor networks, performance analysis.

I. INTRODUCTION

The main applications of Wireless Sensor Networks (WSN) consist in gathering, processing and relaying information from the environment to data collection points in the network.

Simple applications require only a small number of WSN nodes, while complex applications may require a more intricate network infrastructure. The topology may reach hundreds of nodes, demanding tools to: 1) aid network management, 2) avoid configuration errors, and 3) automate infrastructure-sharing between different applications.

The Software-Defined Networking (SDN) paradigm has been proven to give the aforementioned benefits in wired networks, while reducing operational costs [1]. Therefore, combining the SDN concept with sensor networks holds the potential to facilitate management and to enhance performance, giving birth to Software-Defined Wireless Sensor Networks (SDWSN).

The main challenges in designing and implementing SDWSN are the lossy characteristic of low power networks, and in-band transmission of control packets. The SDN-based networks separate the control plane from the data plane, centralizing routing decisions at the logically-centralized network controller. While wired SDN networks are provided with a low-loss dedicated control channel between the controller and switches, it is unrealistic to make this assumption in SDWSNs.

With these challenges in mind, we study the scalability limits of SDWSN systems, considering a set of key parameters. As surveyed in Section II, previous SDWSN performance analysis papers are not thorough, as they assume light data

The associate editor coordinating the review of this article and approving it for publication was Gongbo Zhou.

traffic, do not explore large networks, or do not assess all the metrics of interest.

With the aid of IT-SDN, our SDWSN implementation (described in Section III), we show the feasibility of SDSWN in networks up to 289 nodes in the light of the following metrics: data packets delivery, delay of data packets, control overhead, and energy consumption.

We examine the influence of using source routed control packets, using radio duty cycling, including multiple data sinks, altering controller position, and changing the network topology. We present a thorough description of our methods in Section IV.

The results are discussed in Section V. As a baseline, we included performance results for the Routing Protocol for Low power an lossy networks (RPL) [2]. Our key findings are: 1) average node distance to controller greatly impacts performance; 2) source-routed control packets are instrumental for scaling SDWSN networks; and 3) SDN flexibility enables more energy-efficient data delivery than RPL.

We performed experiments in a testbed to show the SDWSN feasibility in real-life systems. The outcome of the experiments is discussed in Section VI. Results show IT-SDN is able to operate in a real-life network with 10 nodes with a delivery rate close to 100%. The testbed results are similar to the simulation results.

Our contribution lies in answering the question whether the challenges imposed by the SDN paradigm would allow Software-Defined WSN to scale. In sum, we filled a gap in SDWSN performance evaluation, focusing on scalability under varying network conditions. Our conclusions are summarized in Section VII.

II. RELATED WORK

The concept of applying SDN capabilities to the Internet of Things (IoT) and WSN has been widely discussed in the literature. The most praised benefits are the inherent SDN flexibility, resource reuse, and resource usage efficiency [3]–[8].

From a protocol perspective, a few papers attempted to use OpenFlow as a basis for the proposed SDWSN. However, such protocol is not adequate to the constraints imposed by WSN and IoT devices (such as limited frame sizes, memory constraints, and lack of dedicated control channel), and thus the papers do not include a real device implementation [9]–[11].

Other authors, mindful of the inadequacy of OpenFlow to constrained networks, proposed their own architectures. Some of these works are merely exploratory, in the sense that no actual performance results or implementation details are provided [12]–[14]. Others offer a simplistic description of the proposed protocol to such an extent that the reproducibility is compromised. On the other hand, some authors provided performance results, despite limitations on the scenarios descriptions, small number of nodes and/or lack of important performance metrics [15]–[20].

There are other works that explain the proposed protocol thoroughly, and are thus discussed in more detail.

TinySDN [23] is an SDWSN framework that enables multiple controllers within the network. It is based on TinyOS and it is hardware-independent. TinySDN performance was evaluated by simulations on COOJA [30] considering seven sensor nodes in a linear topology and varying the number of controllers between one and two. TinySDN performance was compared to a traditional WSN using the Collection Tree Protocol (CTP) [31], considering the time to deliver the first packet, the latency for a given node to send a message to the sink, and the memory footprint of the test application.

TinySDN was also used to test the concept of hierarchical controllers, although with the same topology and metrics [32]. In addition, TinySDN was discussed in the context of standardization efforts and IETF protocol stack for IoT [33].

SDWN [21] is another SDN architecture proposal to WSN. The authors argue that duty-cycling, in-network data aggregation and flexible definition of rules should be supported. However, there is no implementation or performance evaluation. SDN-WISE [22] follows the architecture proposed by SDWN [21] with the goal of providing stateful flow tables in the sensor nodes. The paper presents performance results from a 6-node testbed. The metrics evaluated were round trip time, byte efficiency and controller response time, varying the payload size and the distance between sender and sink. The number of nodes generating data packets is not specified, but, from the methods explanation, it seems that there is just one.

Buratti *et al.* [24] compare Zigbee and 6LoWPAN/RPL (distributed solutions) to SDWN (centralized solution) [21]. The research was conducted using Flextop [34], an experimental platform for IEEE 802.15.4 networks, considering an 11-node and a 21-node network. The network configurations evaluated were unicast and multicast for static, quasi-static and dynamic environment conditions. For the unicast configuration, the coordinator sends a query to one specific node, and for the multicast configuration the coordinator sends a query to two nodes, therefore the network is tested only in light traffic conditions. The metrics used for the performance evaluation were packet loss rate, round-trip-time, overhead and throughput. SDWN had the best performance in static and quasi-static environments, but Zigbee and 6LoWPAN/RPL outperformed it in dynamic environments.

SDN-WISE is also the basis for a MapReduce framework for WSN [25]. In short, most nodes are mappers, which produce data from their sensors and forward it to reducers, which are fewer nodes that aggregate the data according to a policy. The authors use SDN to dynamically configure which node is a mapper or reducer according to the resources available. The experimental scenarios are diverse as the number of nodes range from 10 to 100 nodes in grid, bus and ring topologies; also, the reducer allocation method is varied. However, the methods are not clear as the following items are not specified: 1) whether a testbed or simulator was used; 2) how the communication cost was assessed; and 3) the application data rate. In addition, important network metrics, such as delivery rate and delay, are missing.

TABLE 1. Performance evaluation conducted on related work.

Work	Drawbacks	Metrics	Methods	Parameters	Compared to
[10], [21], [3], [4], [5], [6], [7], [11]	Authors do not present performance evaluation results				
[15], [16], [17], [18], [19]	Insufficient details about the protocol specification				
Flow-sensor [9]	Use of OpenFlow	Response time, generated packets	Simulations: PROMELA/SPIN	Toplogy size, density, transmission power	Traditional WSN (not specified)
Coral-SDN [20]	Limited scenarios and metrics	Discovery time, discovery rate	Simulations: COOJA	Network topology	-
SDN-WISE [22]	Limited scenarios	Round trip time, byte efficiency, controller response time	Simulations: OMNeT++, physical testbed	Payload, number of hops, flow table entry TTL, number of nodes	-
Tiny-SDN [23]	Limited scenarios	Delivery time, latency, memory footprint	Simulations: COOJA	Number of controllers	Traditional WSN using CTP
General evaluation based on SDWN [24]	Very light traffic	Packet loss rate, round trip time, overhead, throughput	Physical testbed: Flextop	Payload, number of nodes	Zigbee, 6LoWPAN
SDN based MapReduce framework [25]	Lack metrics, hard to reproduce	Communication cost	Not specified	Number of nodes, topology, sink location	-
Study based on SDN-WISE [26]	Limited scenarios, lack of proper statistics	Loss rate, delay convergence time	Own testbed Simulations: COOJA	Network topology	RPL
μSDN	Limited scenarios, lack of proper statistics	Loss rate, delay convergence time control overhead	Simulations: COOJA	SDN timers	RPL
IT-SDN [27] [28]	Lack comparisons	Delivery rate, delay, control overhead	Simulations: COOJA	Number of nodes	-
IT-SDN [29]	Limited scenarios	Delivery rate, delay, control overhead, energy	Simulations: COOJA	Number of nodes Flow table capacity	RPL

Tsapardakis *et al.* [26] took SDN-WISE as the base SDWSN implementation for their performance study. They implemented their own controller and made several modifications to the original southbound protocol. They compared results from a 7-node testbed with COOJA simulations, showing that the simulations tend to yield better performance. Their comparison between SDN and RPL reveals that using SDN-WISE increases packet loss and delay, but it is able to provide QoS on certain scenarios, considering both a fully connected and a multihop network. Nonetheless, their results lack proper statistical analysis, since there is no information about the standard deviation and the number of times the experiments were replicated.

μSDN [35] is an SDWSN architecture built on top of a RPL network. The authors included mechanisms to curb the extra control overhead added by SDN, including source-routed control packets. Their performance evaluation include important metrics, such as delay, delivery, and radio duty cycle. However, the analysis is limited to one 30-node network, the topology of which is not disclosed. The impact of using source-routed control packets is not demonstrated through experiments. Similar to Tsapardakis *et al.* [26], standard deviation and information about experiment replication are not present.

IT-SDN [27] is an open SDWSN tool inspired by TinySDN [23]. Its SDWSN architecture contains three main communication protocols: southbound, neighbor discovery and controller discovery. Also, its architecture is independent of the operating system and provides source-routed control packets as a way to reduce control overhead and flow table occupancy. Margi *et al.* [28] evaluated IT-SDN (version 0.2) through simulations using the COOJA [30] simulator and emulating the TelosB device. The metrics measured were delivery rate, delay and control overhead, in three topology sizes: 9 nodes, 16 nodes and 25 nodes, every node transmitting one packet per minute. The three scenarios included one controller node and one data sink. Protocol version 0.3 was used to expand the results to networks up to 81 nodes [29]. Although the authors have included RPL as a performance baseline, the range of parameters and largest network are still limited.

To the best of our knowledge, surveys on SDWSN systems discuss protocol design and functionalities, but do not discuss the topic of performance analysis [7], [8]. Therefore, we summarize the information regarding performance evaluation collected from the literature in Table 1. It includes: 1) the main paper drawback in terms of performance analysis; 2) metrics; 3) methods; 4) parameters; and 4) which protocols they have been compared to. We noticed that the literature lacks two main characteristics in the scenarios used: larger network sizes, and several nodes transmitting data within the network simultaneously. It is also important to define a set of metrics capable of capturing the overheads and benefits that each type of approach may provide.

Therefore, our goal is to conduct and to present a comprehensive performance evaluation of the SDWSN approach,

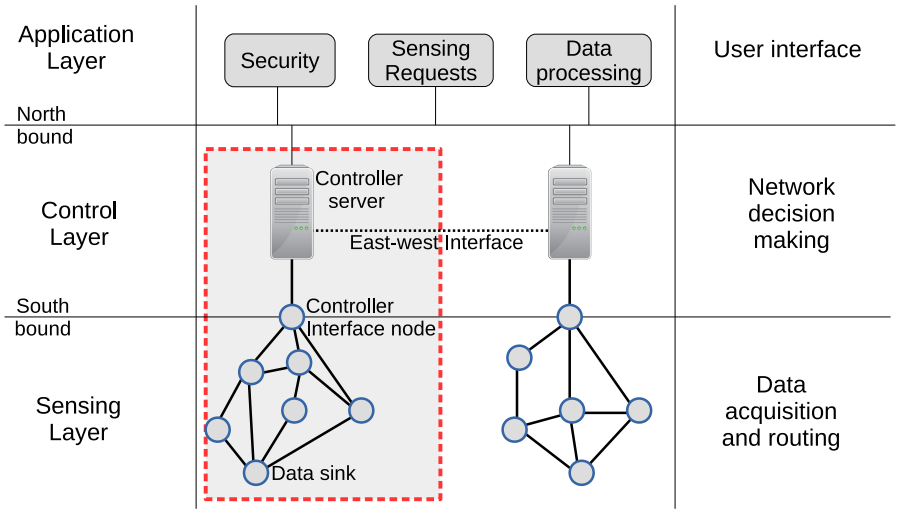


FIGURE 1. SDWSN architecture.

using IT-SDN as a representative, comparing it to RPL [2], the IETF standard for routing in LLNs.

III. SDN ENABLED WSN ARCHITECTURE

This section contains our vision of an SDN-enabled WSN Architecture. The architecture is composed of three layers: the application layer, the control layer and the sensing layer, as shown in Figure 1.

The sensing layer is composed of the data gathering devices, i.e., it is the wireless sensor network itself. The devices collaborate to collect data from the environment and relay the data to one of the network sinks. Due to the separation of control plane and data plane, the packet-forwarding behavior of the nodes is configured by the control layer. The routing decisions are taken by the logically centralized controller, which installs flow rules into the nodes flow tables.

At least one node of the WSN nodes must interface with the network controller to disseminate the configuration information. The control packets are transmitted in-band, that is, there is no dedicated control channel. The data sink is not necessarily the same node as the controller interface node, nor the sink has to be unique. Each flow may be associated with a set of sinks; the nearest sink is used to flush the sensed data out of the WSN.

The WSN nodes communicate with the controller through the southbound interface. This interface, implemented as a network protocol, enables the controller to configure the WSN nodes, and to retrieve their connectivity and resource availability. The controller uses the information received from the nodes to make decisions.

The control layer is the core decision maker of the SDWSN architecture. This layer is composed of controller servers, resourceful computers that calculate and install the flow table rules in the WSN nodes. The criteria to calculate routes may be as simple as the shortest paths in terms of hop count or it could use other information, such as link quality, energy availability, and node capacity.

Large networks may benefit from the service of multiple controllers. The overhead due to node-controller communication tends to increase as the network diameter increases, since the packets are relayed through a larger number of hops. Adding more controllers to the networks alleviates this bottleneck [32]. Nonetheless, the control plane is logically centralized and coordination between the multiple controllers is necessary, which is the purpose of the east-west interface, which is also useful to coordinate disjoint WSNs. Ideally, the controllers should communicate through an infrastructured wired network instead of relying on the ad-hoc wireless connections of the sensing layer.

While the purpose of the southbound interface is to establish the communication between the controller and the WSN, and the purpose of the east-west interface specifies communication between controllers, the goal of the northbound interface is to provide an interface between the controller and the network applications, as illustrated in Figure 1. Thus, through the northbound interface, it is possible to modify the controller decision-making policies.

The network applications lie in the Application Layer. The applications are capable of changing the controller policies and modifying the WSN behavior. An example of application is related to security. Given that the proper telemetries are collected, an application could detect malicious node behavior and, as a consequence, drop packets and avoid routing through that node. Another example is changing the sensing application parameters, such as sampling rate, coverage area and collection granularity. Bulk data processing and analytics also belong to the realm of the application layer.

This concludes the overview of all three layers of the SDN-enabled WSN architecture. Proper design and specification of the layers and their interfaces are the key to enable high quality and reliable sensing systems.

Our objective is to analyze the performance of SDWSN systems regarding the WSN nodes controlled by a single controller, that is, the segment of the architecture delimited by the dotted box in Figure 1. We detail the IT-SDN framework in the next subsections, comprising southbound protocol, WSN node, and controller behavior.

A. SDN ENABLED WSN WITH IT-SDN

This section reviews the main characteristics of IT-SDN (Improved Tiny-SDN) [27], the SDN framework for the WSNs evaluated herein.[1]

In general, an SDWSN framework can be described as a set of protocols and the routing node behavior when receiving packets. The set of protocols comprises the Southbound (SB) protocol, a Neighbor Discovery (ND) protocol and a Controller Discovery (CD) protocol. The Southbound is the core protocol, since it defines the message formats for information exchange between the controller and the sensor nodes.

Neighborhood information is crucial to enable precise route calculation by the controller. Thus, the ND protocol should be able to efficiently gather such information and keep it up-to-date.

At boot, sensor nodes do not have any routing information, and do not know how to reach the controller. Therefore, the CD protocol is used to find a route to the controller. CD and ND are often performed as a joint operation.

According to IT-SDN design, all sensor nodes are SDN-enabled, and at least one of the nodes communicates directly with the controller (e.g. through a serial connection). Packets are routed according to a flow identification number (flow id), defined by the application. By using the flow id concept, the network may accommodate multiple sinks; consequently, the controller is not required to be a data sink.

B. IT-SDN UNDERLYING PROTOCOLS

IT-SDN has its own SB protocol specification, composed of six packet types, defined as follows:

Flow request: Packet used by the nodes to query the controller about an unknown route.

Flow setup: The controller configures routes on the sensor nodes by sending flow setup packets. These packets may be transmitted in response to flow request packets or due to route recalculations performed by the controller. In contrast to other SDWSN proposals, there are three Flow setup versions: regular, source routed and multiple.

The regular flow setup requires setting all intermediate control routes towards the requesting node. For example, consider a linear network with four nodes and a controller, as illustrated in Figure 2. At the moment node 4 asks how to deliver packets to flow `f` (whose destination is node 2), the controller calculates the best path and installs the appropriate rules on the nodes. Before installing the route on node 4, the controller needs to install the intermediate

[1]IT-SDN is available for download at http://www.larc.usp.br/users/cbmargi/www/it-sdn/

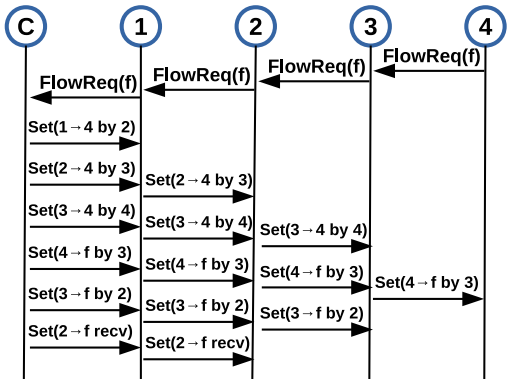


FIGURE 2. Regular flow setup example. Adapted from [27].

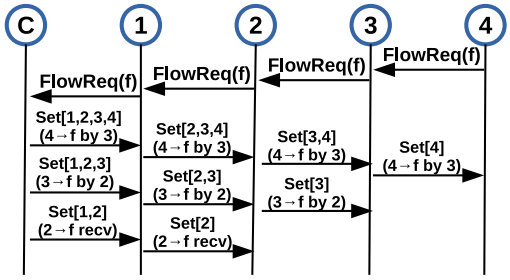


FIGURE 3. Source-routed flow setup example. Adapted from [27].

control forwarding rules; in other words, the controller needs to inform nodes 1, 2 and 3 how to reach node 4. As shown in Figure 2, the controller needs to transmit 6 packets.

The source routed version contains the route to the requesting node in the packet header. Considering the same network as in the former example, the controller installs the flow `f` rule directly on nodes 4, 3 and 2, i.e., it transmits only three packets (as depicted in Figure 3).

The purpose of the multiple flow setup packets is to update many nodes flow table with a single packet. This type is not used herein.

The controller must select which type of flow setup to use. It could be pre-configured to always use the same type or choose the best type according to the situation.

Flow id register: This packet tells the controller that the sender is a destination candidate for the specified flow id.

Acknowledgement: This packet confirms the delivery of the control packets. A packet is acknowledged based on the sequence number.

Neighbor report: This packet contains node neighborhood information, which is sent to the controller at the ND protocol request.

Data packet: These are the application layer packets.

Neighbor discover / controller discovery: These packets are used by the underlying ND/CD protocols.

There are many possibilities to ND and CD protocols [36]. Thus, IT-SDN specification is not coupled with any specific protocol. Instead, interfaces are defined to enable implementing and choosing between alternatives. The ND protocol is

Flow id OR Address	Action	Action Parameter	# uses	age
Flow id: 0	forward	4	4	3
Flow id: 5	receive	-	10	2
Address: 0x1234	drop	-	2	1

FIGURE 4. Example of flow table.

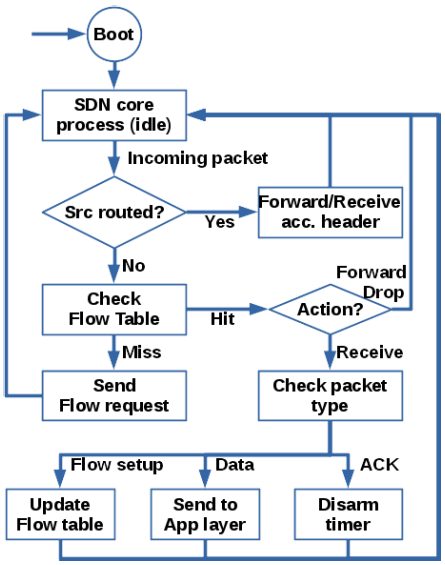


FIGURE 5. IT-SDN node behavior.

responsible for calculating the link metrics, for example, hop count, RSSI, ETX, etc.

C. NODE BEHAVIOR

According to the SDN philosophy, router behavior is as simple as checking flow tables and executing the actions it contains (except source-routed packets). A flow table example is provided in Figure 4. It is composed of the following columns: matching criteria (address or flow id), action taken, action parameter (typically the next hop), the number of times the entry has been used and the age of entry (assessed as the number of updates since the entry was installed).

Figure 5 summarizes the procedure for processing an incoming packet. Source-routed packets bypass the flow table check procedure and are forwarded according to the contents of the source route header.

Non-source-routed packets are routed according to the flow table contents. If there is a matching rule in the flow table, the associated action is executed. The currently available actions are forward, drop, and receive.

The effect of executing a receive action depends on the packet type. Receiving a flow setup packet updates the flow table content, while data packets are forwarded to the application layer. Acknowledgement packets are used to provide reliable delivery of control packets, which is achieved by periodic retransmissions. Therefore, receiving an ACK causes the corresponding retransmission timer to be disarmed.

In case of a flow table miss, the default procedure is to store the packet and to send a flow request, since we aim at maximizing packet delivery. On the one hand, storing the packet increases the delay to deliver the first packet, particularly if the path to the controller has not been discovered yet. On the other hand, discarding the packet decreases the overall packet delivery rate.

TABLE 2. Factor variation and default values.

Parameter	Variation range	Default value
Source routed flow setup	Yes / No	Yes
Radio duty cycle	NullRDC / ContikiMAC	ContikiMAC
Number of data sinks	1, 2, 3, 4	2
Network topology	grid, random	grid
Controller positioning	center, corner	center

TABLE 3. Data payload sizes x number of sinks.

Number of sinks	Payload sizes (bytes)
1	12 (data sink changes after 30 min)
2	12, 96
3	12, 48, 96
4	12, 36, 72, 96

D. CONTROLLER

The network behavior and performance depend on the central controller, as it is the entity that calculates all the routes according to a set of given policies. IT-SDN specifies how the controller communicates with the nodes, but does not specify the internal controller procedures and policies [27].

A few examples of policies that a controller implementation should contemplate are: 1) decide which type of flow setup to use (and when to use it); 2) define if routes are configured reactively or proactively; 3) choose which metrics and route calculation algorithm to use (the controller may query nodes for additional information, e.g. energy).

IV. EXPERIMENTAL METHOD FOR SIMULATIONS

Our objective is to provide a comprehensive performance analysis of SDWSN systems considering the impact of the following factors: use of source routed control packets, underlying radio duty cycle, number of data sinks, data payload size, network topology, and controller positioning. Each factor is evaluated individually while keeping the others constant. The variation range of each factor and their default values are shown in Table 2 and in Table 3.

Source routing control packets is a technique to reduce the overall number of control packets, and limit the required flow table capacity. We assess the impact of this technique in comparison to the traditional approach.

Power efficiency is an important metric in the context of constrained networks. Therefore, Radio Duty Cycling (RDC) is used to reduce the radio energy footprint. However, RDCs

turn the radio off most of the time, potentially reducing the maximum achievable throughput and increasing the delay. We measure the extent of the performance degradation by comparing an always-on IEEE 802.15.4 standard CSMA to ContikiMAC, a widely employed RDC available in Contiki OS.

We analyze grids and random topologies. Grids are regular and simple to reproduce and are commonly used to benchmark WSNs. We assume the nodes radio transmission range is long enough to reach the adjacent nodes in the grid (north/south and east/west neighbors). Random topologies are closer to real-life deployments, but tend to yield worse performance than regular grids, since some nodes are more connected than others, forming bottlenecks. We check this hypothesis by generating random networks with NPART [37], using the default parameters for Berlin networks.

The position of specialized nodes, such as controller and sinks, plays a part in the network performance. In grid topologies, the controller is placed in the center or in the corner. We simulated scenarios with 1, 2, 3, or 4 data sinks, positioned at the center of the upper row, lower row, leftmost column, and rightmost column of the grid.[2] Each regular node transmits data to the i-th sink, in which $i = 1 + nodeid\ mod\ (number\ of\ sinks)$ (nodes are numbered sequentially by column). While controller and sinks positioning is pre-determined in grid topologies, the positioning of these nodes are random in random topologies.

For all the scenarios, we varied the network size according to the following dimensions: 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, and 289. All the nodes in the network transmit data packets at constant intervals, with the exception of data sinks and the controller. A random initial delay is introduced to avoid artificial synchronization of data transmission.

In the case of flow table misses, a flow request is issued to the controller, and the unmatched packet is stored until a suitable flow setup is received.

We used a periodic beaconing neighbor discovery protocol based on the Collect protocol, which uses ETX as a link quality indicator. The frequency of neighbor reports to the controller is limited to 1 per minute, and only if the link quality has changed substantially.[3] We consider a substantial change as a relative difference of 200% or absolute difference of 100. No Controller Discovery protocol is used, that is, the controller actively configures how the nodes will reach it from the gathered neighborhood information.

The controller calculates the shortest route based on the Dijkstra algorithm. Incoming neighbor reports may change the previously calculated best route. The new route is installed in the network nodes only if its cost is at least 20% smaller. Control packet reliability is provided by a periodic retransmission mechanism. The controller maintains a copy of the nodes flow tables, with the addition of an extra field indicating whether the corresponding flow setup packet has been acknowledged. Unacknowledged flow configurations are retransmitted after a fixed amount of time. This mechanism is safe against route recalculations before the former route has been acknowledged, and it ensures all the flows rules are eventually correct in all the flow tables.

We used COOJA [30], a tool able to emulate sky motes firmware and to simulate the radio medium.

To provide a non-SDN routing protocol reference, we also simulated equivalent scenarios with the Routing Protocol for LLN (RPL). We chose RPL as it is an efficient standardized protocol readily implemented in Contiki OS. A brief RPL explanation is provided in Section IV-B.

TABLE 4. Simulation parameters.

Simulation parameters	
Number of nodes	25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289
Simulation duration	3600 s
Node boot interval	$[0, 1]$ s
Number of sinks	1, 2
Data payload size	10 bytes
Data traffic start time	$[2, 3]$ min
ContikiMAC channel check rate	16 Hz

Energy Consumption parameters	
Radio module transmission power	0 dBm
Transmission current consumption	21.70 mA
Receiving current consumption	22.00 mA
Processing current consumption	2.33 mA
Sleeping current consumption	0.18 mA
Operation voltage	3 V

IT-SDN parameters	
Software version	v0.4.1
Controller retransmission timeout	60 s
ND protocol	Collect-based
Link metric	ETX
Neighbor report max frequency	1 packer per minute
CD protocol	none
Route calculation algorithm	Dijkstra
Route recalculation threshold	20%

RPL Parameters	
MOP	storing or P2P-RPL
RPL instances	1
Minimum DIO interval	4.096 s (contiki default)
DIO interval doubling	8 (contiki default)
DAO latency	4 s (contiki default)
DAO expiration	60 s (contiki default)
Objective Function	Minimum rank with hysteresis
DODAG metric	ETX

[2] in the case of even-sided grids, the leftmost or upper node is chosen
[3] the ETX ranges from 0 to 255

Table 4 contains a summary of the simulation parameters. All the graphs in Section V display 95% confidence intervals from 10 simulation runs with randomized seeds.

A. PERFORMANCE METRICS

We assessed the following performance metrics: 1) delivery rate of data packets, 2) delivery delay of data packets, 3) control overhead, and 4) energy consumption.

The overall delivery rate of data packets is defined as the quotient between the total number of packets successfully received by the sinks and the total number of packets transmitted by the application layer of the other nodes.

Similarly, the overall delay is calculated as the average time a data packet takes to reach the destination, i.e, the sum of all individual delays divided by the total number of delivered packets. Queuing and route solicitation delays are accounted for, while undelivered packets are excluded from this metric.

Control overhead represents the extra load of work that the routing protocol introduces in the network. We use the total number of control packets to measure it.

The energy consumption metric represents the average energy consumption of all nodes in the network over the one-hour simulation. This metric considers four states: processing, sleeping, transmitting and receiving. Each node calculates its own energy consumption using Equation 1.

$$E = \sum_{j=1}^{4} I_j T_j V \tag{1}$$

where I_j is the average current consumption in state j, T_j is the time spent in state j, and V is the operation voltage for the TelosB mote [38]. We used Energest [39], a tool embedded in Contiki, to obtain T_j.

We calculate the metrics by processing the nodes serial output, i.e., messages print by the nodes. Outputting to serial is slow and may degrade the measured protocol performance. However, the number of messages should be similar among the assessed protocols and parameters, achieving a fair comparison.

B. RPL PROTOCOL

RPL is the main outcome of IETF Routing Over Low power and Lossy networks Working Group [2]. It is a routing protocol designed on top of 6LoWPAN stack (IPv6 adaptation layer), aimed at providing efficient routing for the Internet of Things.

In sum, RPL operates by building and maintaining a Destination Oriented Directed Acyclic Graph (DODAG). Each node calculates its rank according to an objective function and periodically informs its neighbors by transmitting DIO (DODAG Information Object) control messages. The trickle algorithm controls DIO transmission interval length [40].

The rank of the DODAG root is set to MinHopRankIncrease by default. Other nodes choose the neighbor with the lowest rank as their preferred parent, and keep a list of parent candidates to allow easy parent switching in case of failure. Examples of metrics used in the objective functions are hop count, ETX and RSSI.

Once all nodes in the network compute their preferred parent, the DODAG construction is complete and all the nodes are able to reach the root (upward routes or multipoint-to-point traffic pattern).

The downward routes (or point-to-multipoint traffic pattern) are calculated apart from the DODAG construction. DAO (DODAG Destination Advertisement Object) control messages are transmitted to disseminate reachability information. There are two modes of operation (MOP): storing and non-storing. In the first MOP, routing nodes maintain a routing table with the addresses of their children, whereas in the latter MOP, the DODAG root stores all the reachability information. The downward routes also enable point-to-point communication between nodes.

We consider the DODAG root is always a data sink. In scenarios with more than one sink, the other sinks are not a DODAG root (i.e. there is only one RPL instance and DODAG id); consequently, their reachability information must be propagated through DAO packets.

V. SIMULATION RESULTS AND DISCUSSION

We present the simulation results in this section, displaying a set of four graphs (one for each metric) for each of the studied parameters, namely, (1) source-routed control packets discussed in Section V-A, (2) radio duty cycling presented in Section V-B, (3) number of data sinks addressed in Section V-C, (4) network topologies compared in Section V-D, and (5) controller positioning discussed in Section V-E.

A. SOURCE-ROUTED CONTROL PACKETS

It is clear that source-routed control packets are instrumental to increase the scalability of SDWSN systems, as observed in Figure 6a. Using this feature enables at least 90% data yield in networks up to 225 nodes; otherwise, this level of performance is only achieved in small networks (25 and 36 nodes). From networks of size 81 onwards, data delivery is less than 30% without source routed control packets, while we notice significant performance degradation only in large networks with source-routed packets (as low as 70%).

The main reason behind the performance gap is the number of nodes that join the SDN network and communicate with the controller. Without source routed packets, the peripheral nodes are unable to establish a communication channel with the controller, and are consequently unable to fill their flow tables and deliver data packets from 64-node networks onwards.

The source-routed control packets reduce the overall number of flow setup packets, since intermediate control routes do not have to be set by the controller, as explained in Section III-B. Another consequence is the prevention of flow table overflows, which further reduces the overall control overhead. The combination of these two factors enables peripheral nodes to communicate with the controller, increasing packet delivery ratio.

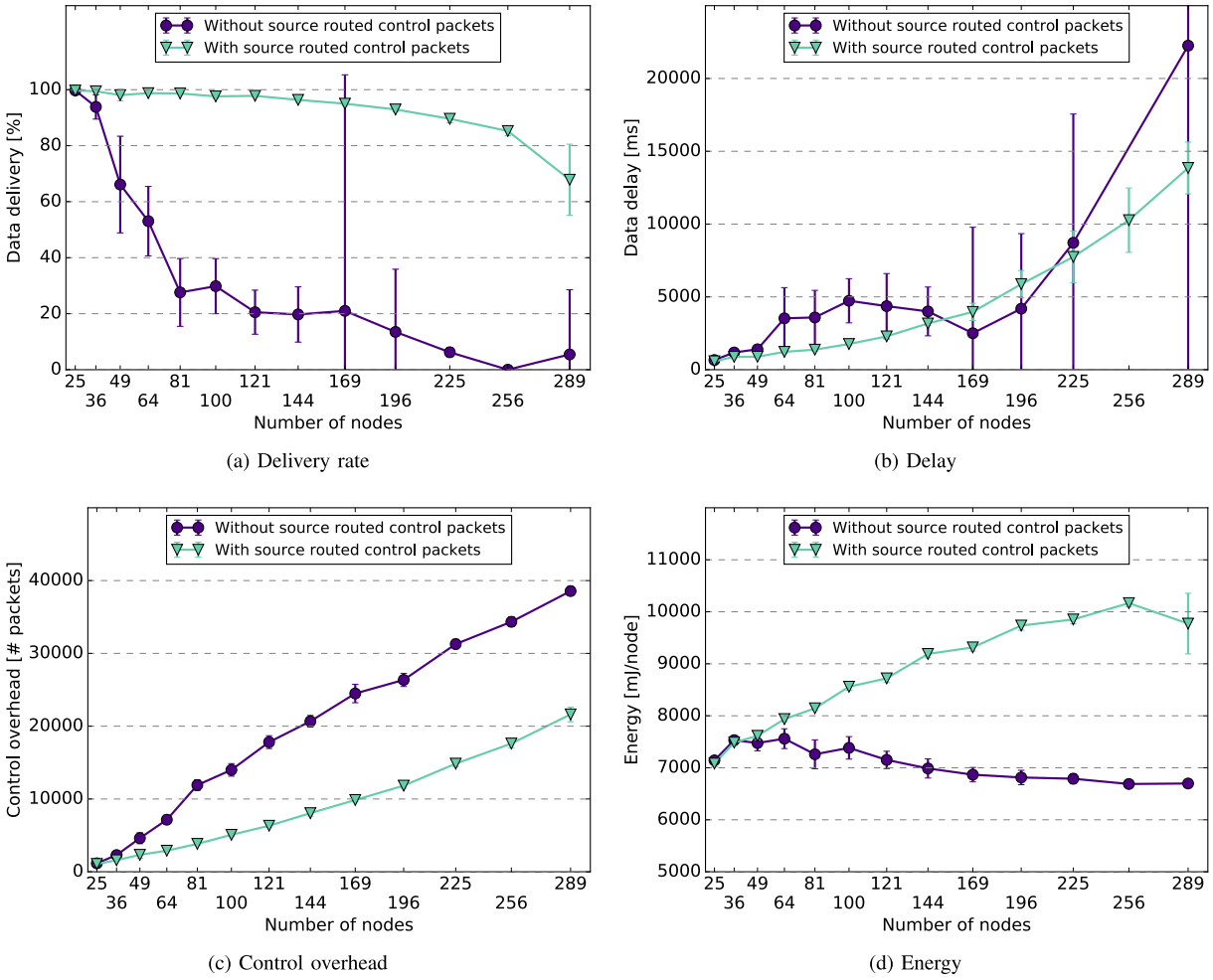


FIGURE 6. Results from the usage of source-routed control packets. Source-routed control packets are instrumental to SDWSN scalability.

The difference in the number of control packets can be observed in Figure 6c. Not using source-routed control packets increases the control overhead by at least 78% in networks larger than 49 nodes. A great increase in flow-configuration packets is noticed. For example, in the 100-node network, we observed the usage of source routed control packets reduces the number of flow configuration packets by 7 times, on average.

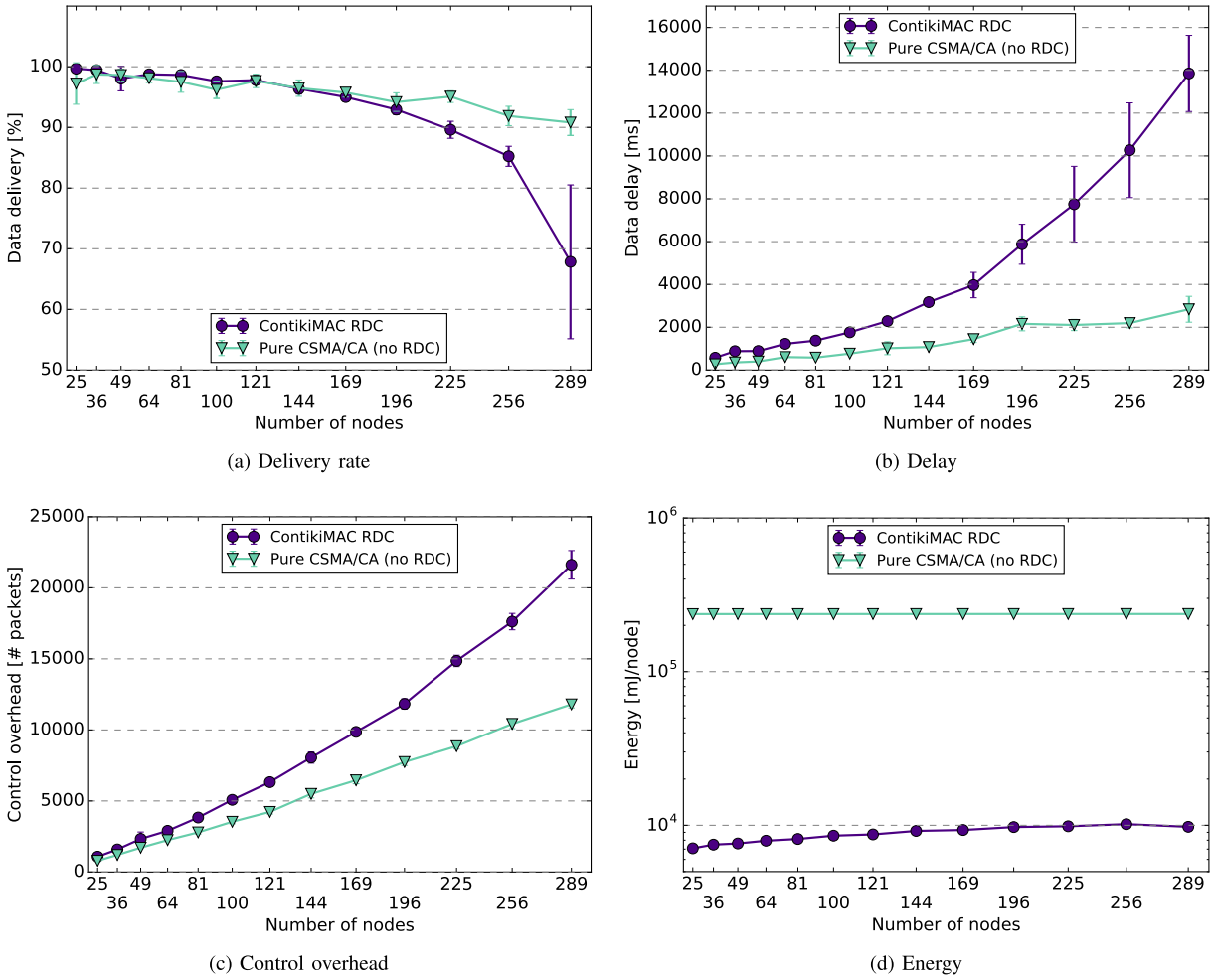
Delay results are displayed in Figure 6b. In the smallest network sizes, not using source-routed packets increases the delay by 14.9% and 33.9%, respectively. In larger networks, the delay metric presents a large standard deviation, especially for the non source-routed version. This is caused by the randomness in the time it takes for the nodes to communicate with the controller and acquire the data flow rules. As a consequence, the data packets that are buffered waiting for a matching flow rule to be installed present an increased delay.

It might seem counter-intuitive that using source-routed packets results in a larger energy consumption per node, as observed in Figure 6d. The non-source routed version presents slightly higher ($< 1\%$) energy consumption in small networks sizes. In the larger topologies, as mentioned before, many nodes do not establish a communication channel with the controller; therefore, these nodes only perform basic neighbor discovery procedures. Consequently, these nodes present low energy consumption, decreasing the average network-wide energy consumption.

B. RADIO DUTY CYCLING

Using radio duty cycling does not influence the packet delivery metric (less than 2.5% difference, Figure 7a) for networks up to 196 nodes. If the network is larger than that, the limitation that ContikiMAC imposes on the maximum link data rate hinders delivery of control packets, delaying timely network configuration.

In particular, in our simulation setup, the data sinks advertise their existence to the controller at the network startup. The advertisement message (*flow_id_register*) takes longer to be successfully delivered in large networks, as ContikiMAC increases medium access contention. As a consequence, the controller takes longer to be aware of the data sinks and instructs the nodes to drop data packets until a valid path is known.



(a) Delivery rate (b) Delay (c) Control overhead (d) Energy

FIGURE 7. Impact of radio duty cycling. RDC is a necessary evil, as energy efficiency comes with performance degradation in large networks.

In this case, the area around the controller becomes a bottleneck, hinting that strategically adding another controller to the network would alleviate the burden on the link layer.

Figure 7b shows delay results. It is noticeable that ContikiMAC slope gets steeper at 144-node network size. This is an effect of increased channel contention and the prolonged delay to first obtain data flow forwarding rules. We study the 90th percentile to exclude the largest delays from the dataset; the values are presented in Table 5 for large networks. For all the cases shown in the table, ContikiMAC 90th percentile is lower than its average, indicating that some packets present a very large delay. Considering the 90th percentile, ContikiMAC is about 70% slower than CSMA, while the average can be up to 5 times larger, indicating that using this RDC increases delay regardless of the extended initial flow configuration.

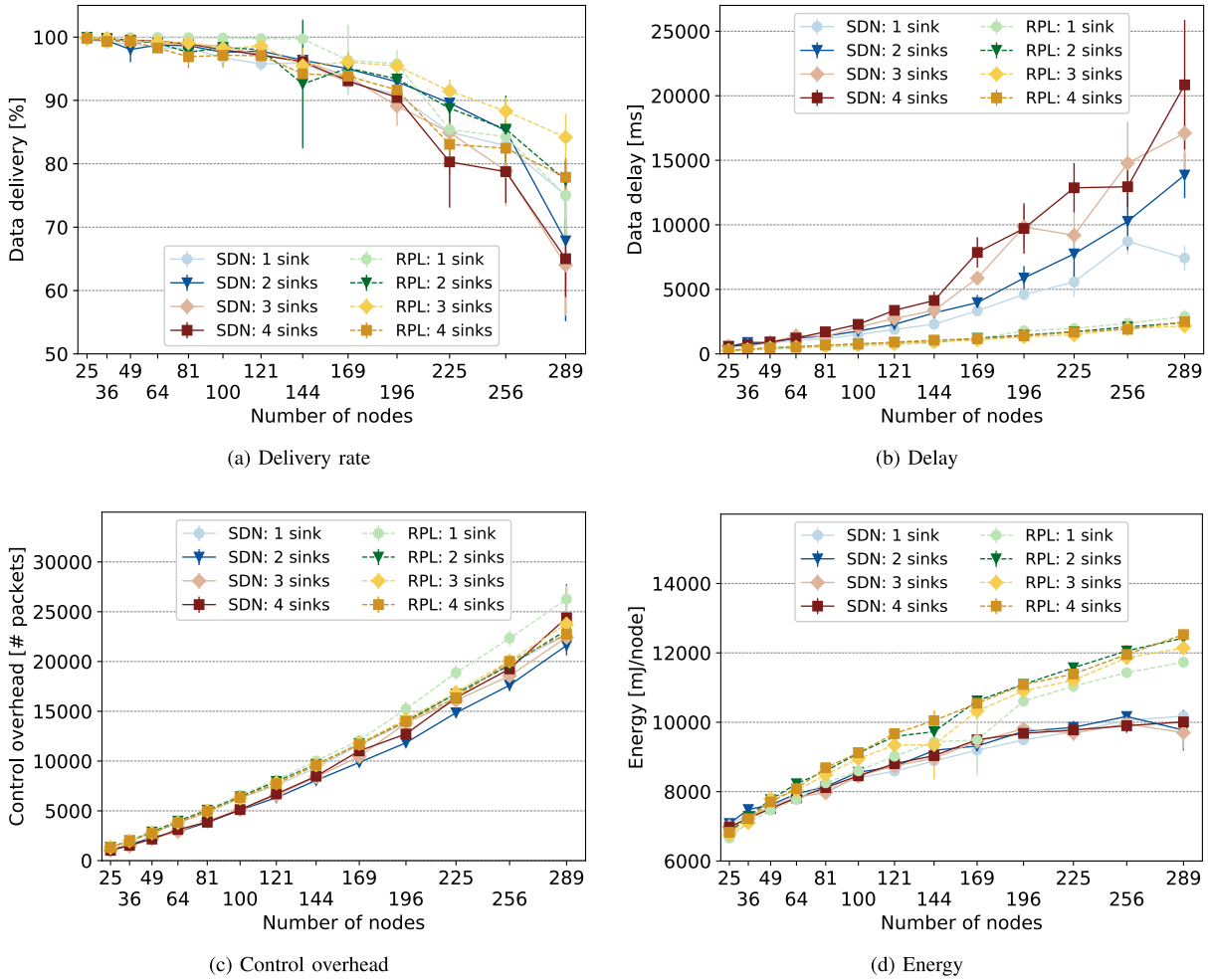
The number of transmitted control messages is larger for ContikiMAC, as observed in Figure 7c. Although the same flow configurations are performed in either scenario, end-to-end SDN control messages require reliability; thus, the extra control messages are due to retransmissions.

TABLE 5. Delay average and 90th percentile (ms). The R column represents the ratio between ContikiMAC and CSMA delays.

	Average			90th percentile		
n	ContikiMAC	CSMA	R	ContikiMAC	CSMA	R
144	3 173	1 076	2.9	2 339	1 436	1.6
169	3 969	1 444	2.7	2 448	1 484	1.6
196	5 877	2 159	2.7	2 796	1 665	1.7
225	7 746	2 105	3.7	3 009	1 731	1.7
256	10 268	2 194	4.7	3 450	1 918	1.8
289	13 854	2 840	4.9	3 479	1 992	1.7

The end-to-end acknowledgements are also prone to delivery failure, triggering unnecessary retransmissions. For example, the 169-node scenario presents 72% delivery out of 3365 transmitted end-to-end control packets (ContikiMAC), while 83% out of 1922 for CSMA. It is noteworthy that at least 60% of the control packets are attributed to discovery algorithms.

As expected, ContikiMAC is significantly more energy efficient than pure CSMA/CA, expending roughly 24 times less energy, as observed in Figure 7d.



(a) Delivery rate

(b) Delay

(c) Control overhead

(d) Energy

FIGURE 8. **Multiple sinks results.**

C. NUMBER OF DATA SINKS AND PACKET SIZES

In this section, we study the network behavior while varying the number of sinks. Each sink receives a difference payload, as described in Table 3.

The delivery rate (Figure 8a) did not show expressive variations between RPL and SDN, considering networks up to 169 nodes and at least 2 sinks (±2%). In larger networks, the SDN approach is worse than RPL, especially with 3 or 4 sinks. Considering 1-sink scenarios, RPL performs better, especially at intermediate networks sizes (up to 5% at 196-node scenario). The cause of SDN performance loss as the network increases is the increment of the average node distance to the controller, particularly the sink distance to the controller. This factor delays the initial network configuration, causing data packet losses.

We also analyzed the data delivery rate of packets transmitted after 20 minutes of simulation. In this case, we observe SDN is competitive with RPL even in large networks. In fact, considering the 1-sink scenario, SDN delivered 15%, 14%, and 26% more than RPL for 225, 256, and 289-node topologies, respectively. This improvement is a consequence of the flexibility provided by SDN, which is able to promptly deal with the sink change.

A similar rationale can be applied to the delay metrics, Figure 8b. The SDN stack buffers the data packets until it receives a matching flow entry. Since the controller sets routes reactively, the initial packet delay is high. In small networks (up to 100 nodes), SDN is 132% slower, on average. In larger networks, the delay can be increased by a factor of 7. Alternatively, we can study the delay 90th percentile. In this case, SDN is on average 29% slower than RPL in networks as large as 169 nodes. In the other topologies, SDN outmatches RPL by 22% on average, which is caused by SDN calculating optimal routes in comparison to the tree-based routing from RPL.

Control overhead is claimed as one the main drawbacks of the SDN approach in the WSN domain [41]. However, our experiments show that SDN presents lower control overhead than RPL (Figure 8c). If there is more than one sink in the network, SDN beats RPL by 15% on average, although the

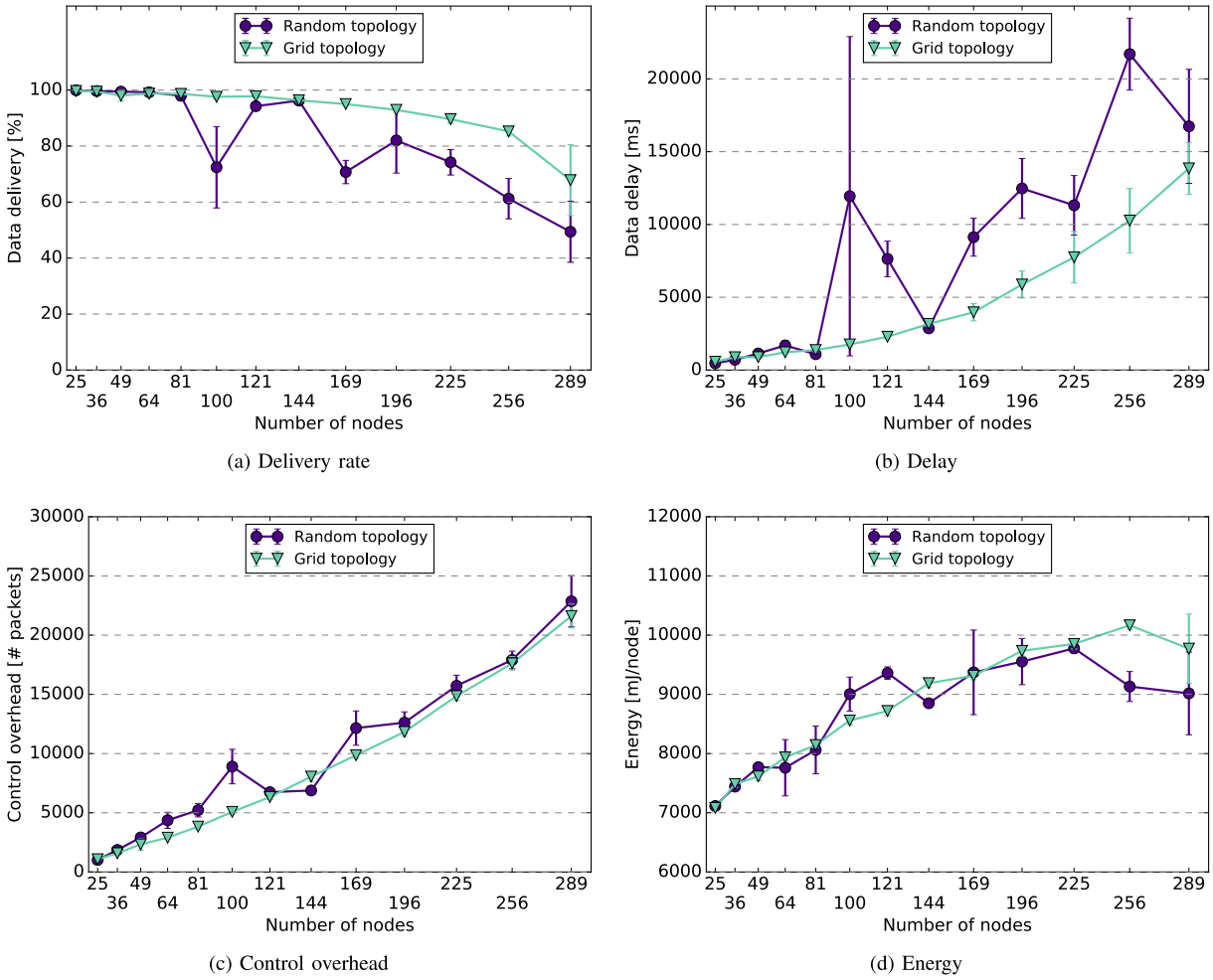


FIGURE 9. Influence of topology on network performance. Randomness tends to decrease network performance.

difference is more prominent in small networks (around 20%) than in large networks (around 5%). In one-sink scenarios, SDN produces 6% fewer control packets on average. SDN is fairly stable after the initial convergence, performing mostly neighbor discovery after the routes are configured. RPL, on the other hand, constantly transmits DIO according to the trickle timer algorithm. Whenever a node changes its parent, it may trigger several DIO transmissions throughout the DODAG. In addition, the DAO messages are sent to keep track of downward routes, which is a separate process from the DODAG construction, further increasing control messages count.

As a consequence of lower control overhead, SDN also presents lower energy consumption in most scenarios, showing an increasing gap as the network gets larger, as observed in Figure 8d. In 25 and 36-node networks, RPL consumes less energy by 3% on average. At 49-node, SDN is slightly more energy efficient ($\approx$ 1.5%). From this point onwards, the energy gap steadily increases up to 21%. In addition to the number of control packets, we also highlight that most of RPL control packets are link layer broadcasts, which are particularly inefficient in comparison to unicast packets when using ContikiMAC as radio duty cycling protocol.

D. TOPOLOGY

In this section, we compare the performance of grid and random topologies. We obtained similar results up to medium sized networks (144 nodes), except for the 100-node random network, in which the controller and sink nodes ended up being badly positioned. In larger networks, randomness tends to decrease the overall performance.

The packet delivery rate in random topology drops up to 28% in comparison to grid topologies (Figure 9a). A similar trend was observed considering only the data packets transmitted after 20 minutes of simulation.

The data delay is displayed in Figure 9b. The random topology presents a characteristic that is a double-edged sword: the average number of neighbors is larger than grid topologies. Conversely, this characteristic reduces the average path length from source to destination, while increasing medium access contention. As a consequence, it takes longer to set the routes in all nodes. Yet, once the routes are set, the packets

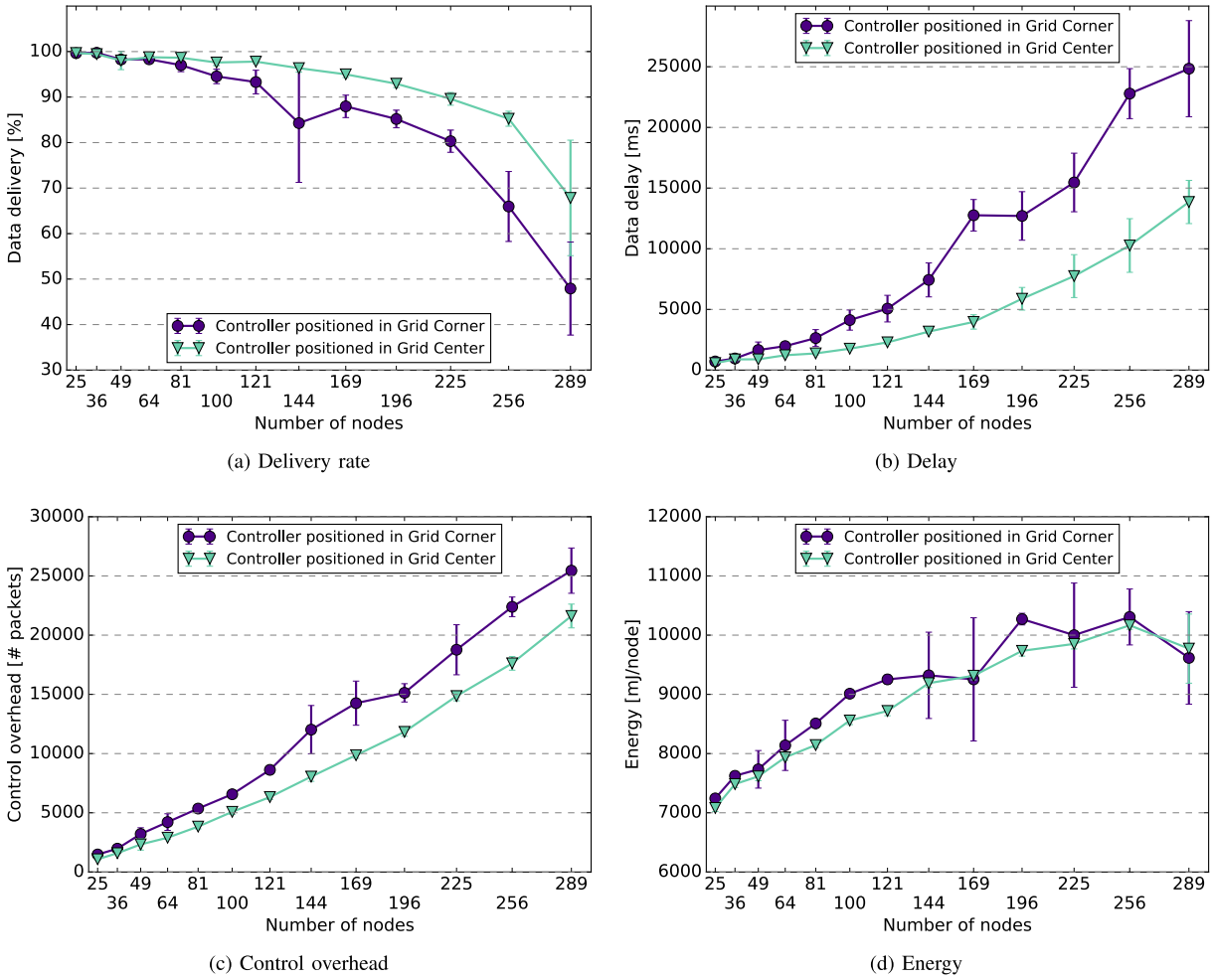


FIGURE 10. Effect of controller positioning. Average distance to controller is inversely correlated to network performance.

can be delivered more quickly. The delay 90th percentile is at least 16% smaller for random topologies up to 100 nodes. In larger topologies, the results were not consistent, as they depend on the controller and sink nodes positioning.

Grid topologies tend to require less control traffic (18% on average), as exhibited in Figure 9c. In some specific cases, the random topology was more efficient, mostly due to the good positioning of controller and sink, and low number of bottleneck nodes.

We did not notice a clear correlation between topology and energy consumption, as observed in Figure 9d. The differences remain in the range of $\pm 10\%$.

E. CONTROLLER POSITIONING

We chose the best and the worse locations, in terms of average node distance to the controller, to study the effect of controller positioning. Asymptotically, the average number of hops to reach a controller positioned in the corner of a grid is doubled in comparison to a controller positioned at the grid center.

The degradation in data delivery and data delay observed in Figures 10a and 10b are due to the increased distance between nodes and controller. Consequently, it takes longer to reach the controller and fill the flow table. The degradation tends to grow with the networks size, although it is not noticeable for small networks (up to 64 nodes).

For studying the network behavior after the initial transient, we calculated the data delivery rate solely considering packets transmitted after 20 minutes of simulation. Independently of the controller positioning, at least 95% data yield was observed in networks up to 225 nodes. In larger topologies, we observed that the controller in the corner entails less data delivery, as one of the sinks can take a long time to successfully deliver the *flow_id_register* to the controller, causing most of the losses.

Analogously, we calculated the 90th percentile of the delay metric. In this case, there are no significant differences regardless of the controller position. Since the sink nodes are in the same position in either case, the packet delay is the same once the routes are properly set.

Placing the controller in the corner increases the control overhead metric, as seen in Figure 10c. The excess, which varies from 18% to 49%, is mainly due to the increased

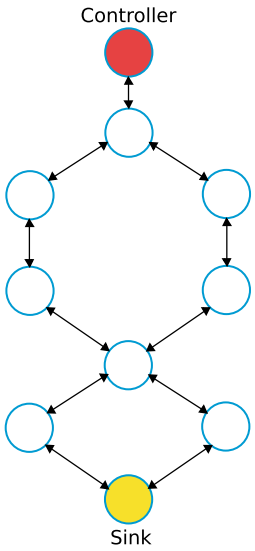


FIGURE 11. Testbed topology.

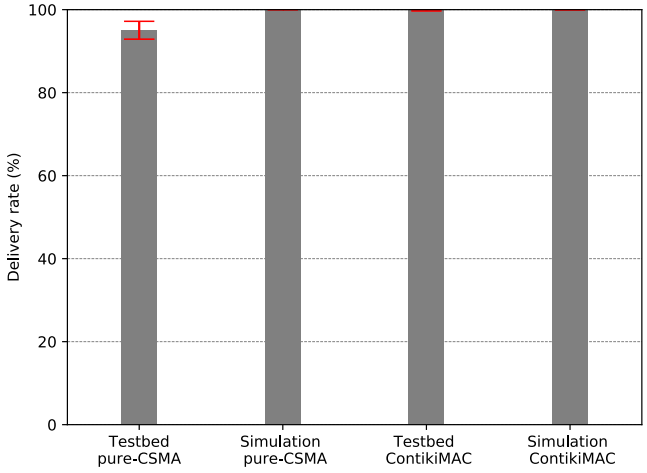


FIGURE 12. Testbed: Data packets delivery rate.

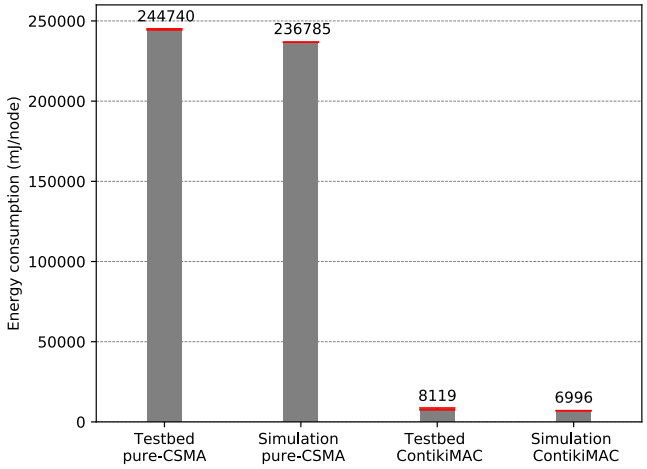


FIGURE 13. Testbed: Energy consumption per node.

retransmission of end-to-end control packets. This behavior is a consequence of medium contention in the area around the controller and larger average hop length to reach the controller.

The differences in energy consumption are bounded by 6% (Figure 10d). This metric is somewhat correlated to the number of control packets in the network. When the controller is in the corner, the farthest nodes may not join the network, decreasing the average energy consumption at the expense of data packet losses. Other than that, we observe an increasing trend (up to 4.5%) in energy consumption in the scenarios with at least 95% data delivery (81 nodes). This is expected, since the end-to-end control packets can reach the controller in a smaller number of hops if the controller is at the center.

VI. TESTBED EVALUATION

We evaluated IT-SDN in a testbed with 10 nodes distributed as shown in Figure 11. The testbed was deployed indoors in our research group facilities. All the devices are TelosB [38], programmed using Contiki OS 3.0 and IT-SDN 0.4.1.

Each sensor node was programmed to send temperature, humidity, luminosity, and energy consumption data to the sink every 60 seconds. The radio module power was set to -7 dBm and each mote was located 0.8 meters apart from its neighbors. During the experiments, the motes were connected to a computer via USB to obtain the serial output and also for energy supply.

The objective of running a testbed evaluation is to assess IT-SDN performance in a network with real devices and to compare these results with simulations results using the same network topology. For this, we conducted experiments using pure CSMA/CA (no RDC) and ContikiMAC. Each scenario was replicated ten times and operated for one hour. Also, we executed simulations with 10 nodes for comparison purposes. The simulations were executed using COOJA and configured with 100% transmission (TX) and reception (RX) success ratio. Data packets delivery rate and energy consumption are the metrics we used to compare the network performance. The energy consumption was calculated using the four-state energy consumption model explained in Section IV-A and shown in Equation 1. As explained in Section IV-A, we used Energest [39] to obtain the time spent on each state.

Figure 12 shows the results for data packets delivery rate. The testbed results show the scenario with ContikiMAC obtained 99.8% and the scenario with pure CSMA/CA obtained 95.0% of data packets successfully delivered. The simulation results show a data delivery rate of 99.9% and 100% when using ContikiMAC and pure CSMA/CA, respectively. Comparing both scenarios (testbed and simulation) using ContikiMAC, we notice they have similar data delivery rate results. Conversely, there is a difference of 5% in the pure CSMA/CA cases. This is a significant difference since the network has a small data traffic (8 data packets per minute) in a network with only 10 nodes. In turn, COOJA does not consider external interference, such as Bluetooth and IEEE

802.11 networks, and it was configured with a 100% TX/RX success ratio.

Observing Figure 13, we notice that the energy consumption when using pure CSMA/CA is 30 times higher than the energy consumption when using ContikiMAC for both simulation and testbed results. These results are similar to the simulation results shown in Figure 7d, where the energy consumption when using pure CSMA/CA is between 24 to 33 times higher than the energy consumption when using ContikiMAC.

This experiment shows IT-SDN is able to operate in a real network using TelosB motes and obtain delivery rate close to 100%. We observe that, when using ContikiMAC in a real-life network, the delivery rate is higher and 30 times less energy is consumed in comparison to using pure CSMA/CA. Furthermore, testbed and simulations results differ less when using ContikiMAC than when using pure CSMA/CA.

VII. CONCLUSION

Software-Defined Networking holds the potential of providing flexibility to Wireless Sensor Networks, while it was unclear whether the challenges imposed by the SDN paradigm would allow Software-Defined WSN to scale.

To answer this question, we performed several experiments in networks up to 289 nodes. We showed that: 1) controller positioning greatly affects performance; 2) radio duty cycling decreases PDR only in large networks; 3) grid topology yields optimistic results; 4) source-routed control packets are instrumental for scaling SDWSN systems; and 5) SDN is flexible to deal with multiple data sinks.

We showed that SDN is feasible for WSN, performing as well as RPL, the de facto standard protocol for IoT. In fact, SDN used less energy while being able to deliver the same number of data packets.

The main drawback introduced by SDN is the initial network setup, which can take longer than distributed algorithms. We studied data delay percentiles to show that SDN is competitive after the network is properly configured. It is worth noticing that this initial network setup depicts the worst case scenario, since this happens in the reactive approach. Despite that, the testbed deployment showed IT-SDN is able to achieve close to 100% delivery rate.

REFERENCES

[1] D. Kreutz, F. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[2] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, document RFC 6550 (Proposed Standard), Internet Engineering Task Force, Mar. 2012.

[3] R. Kirichek, A. Vladyko, A. Paramonov, and A. Koucheryavy, "Software-defined architecture for flying ubiquitous sensor networking," in *Proc. 19th ICACT*, Feb. 2017, pp. 158–162.

[4] I. S. Acharyya and A. Al-Anbuky, "Towards wireless sensor network softwarization," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 378–383.

[5] Y. Choi, Y. Choi, and Y.-G. Hong, "Study on coupling of software-defined networking and wireless sensor networks," in *Proc. 8th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2016, pp. 900–902.

[6] K. Sahoo, B. Sahoo, and A. Panda, "A secured SDN framework for IoT," in *Proc. Int. Conf. Man Mach. Interfacing (MAMI)*, Dec. 2015, pp. 1–4.

[7] 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.

[8] 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. [Online]. Available: http://www.mdpi.com/1424-8220/17/5/1031

[9] A. Mahmud and R. Rahmani, "Exploitation of OpenFlow in wireless sensor networks," in *Proc. Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, vol. 1, Dec. 2011, pp. 594–600.

[10] T. Luo, H.-P. Tan, and T. Q. S. Quek, "Sensor OpenFlow: Enabling software-defined wireless sensor networks," *Commun. Lett.*, vol. 16, no. 11, pp. 1896–1899, Nov. 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/icl/icl16.html#LuoTQ12

[11] P. Jayashree and F. I. Princy, "Leveraging SDN to conserve energy in WSN-AN analysis," in *Proc. 3rd Int. Conf. Signal Process., Commun. Netw. (ICSCN)*, Mar. 2015, pp. 1–6.

[12] H. Akram and A. Gokhale, "Rethinking the design of LR-WPAN IoT systems with software-defined networking," in *Proc. Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, May 2016, pp. 238–243.

[13] F. Olivier, G. Carlos, and N. Florent, "SDN based architecture for clustered WSN," in *Proc. 9th Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput.*, Jul. 2015, pp. 342–347.

[14] M. Sneps-Sneppe and D. Namiot, "Metadata in SDN API for WSN," in *Proc. 7th Int. Conf. New Technol., Mobility Secur. (NTMS)*, Jul. 2015, pp. 1–5.

[15] S. Bera, S. Misra, S. K. Roy, and M. S. Obaidat, "Soft-WSN: Software-defined WSN management system for IoT applications," *IEEE Syst. J.*, vol. 12, no. 3, pp. 2074–2081, Sep. 2018.

[16] T. A. Al-Janabi and H. S. Al-Raweshidy, "Efficient whale optimisation algorithm-based SDN clustering for IoT focused on node density," in *Proc. 16th Annu. Medit. Ad Hoc Netw. Workshop (Med-Hoc-Net)*, Jun. 2017, pp. 1–6.

[17] S. Mahmoud, I. Jawhar, N. Mohamed, and J. Wu, "UAV and WSN softwarization and collaboration using cloud computing," in *Proc. 3rd Smart Cloud Netw. Syst. (SCNS)*, Dec. 2016, pp. 1–8.

[18] N. S. Nafi, K. Ahmed, M. Datta, and M. A. Gregory, "A novel software defined wireless sensor network based grid to vehicle load management system," in *Proc. 10th Int. Conf. Signal Process. Commun. Syst. (ICSPCS)*, Dec. 2016, pp. 1–6.

[19] W. Twayej, H. S. Al-Raweshidy, M. Khan, and S. El-Geder, "Energy-efficient M2M routing protocol based on Tiny-SDCWN with 6LoWPAN," in *Proc. 8th Comput. Sci. Electron. Eng. (CEEC)*, Sep. 2016, pp. 198–203.

[20] T. Theodorou and L. Mamatas, "Software defined topology control strategies for the Internet of Things," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2017, pp. 236–241.

[21] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling SDNs," in *Proc. IEEE Eur. Workshop Softw. Defined Netw. (EWSDN)*, Oct. 2012, pp. 1–6.

[22] 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 *Proc. IEEE INFOCOM*, Apr. 2015, pp. 513–521.

[23] B. T. de Oliveira, C. B. Margi, and L. B. Gabriel, "TinySDN: Enabling multiple controllers for software-defined wireless sensor networks," in *Proc. IEEE LATINCOM*, Nov. 2014, pp. 1–6.

[24] 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 Things J.*, vol. 3, no. 1, pp. 124–133, Feb. 2016.

[25] A.-C. G. Anadiotis, G. Morabito, and S. Palazzo, "An SDN-assisted framework for optimal deployment of MapReduce functions in WSNs," *IEEE Trans. Mobile Comput.*, vol. 15, no. 9, pp. 2165–2178, Sep. 2016.

[26] E. Tsapardakis, M. Ojo, P. Chatzimisios, and S. Giordano, "Performance evaluation of SDN and RPL in wireless sensor networks," in *Proc. Global Inf. Infrastruct. Netw. Symp. (GIIS)*, Oct. 2018, pp. 1–5.

[27] R. C. A. Alves, D. A. G. Oliveira, G. A. N. Segura, and C. B. Margi, "IT-SDN: Improved architecture for SDWSN," in *Proc. 35th Simpósio Brasileiro Redes Computadores Sistemas Distribuídos*, 2017, pp. 1143–1150.

[28] C. B. Margi, R. C. A. Alves, and J. Sepulveda, "Sensing as a service: Secure wireless sensor network infrastructure sharing for the Internet of Things," *Open J. Internet Things*, vol. 3, no. 1, pp. 91–102, 2017.
[29] C. B. Margi, R. C. A. Alves, G. A. N. Segura, and D. A. G. Oliveira, "Software-defined wireless sensor networks approach: Southbound protocol and its performance evaluation," *Open J. Internet Things*, vol. 4, no. 1, pp. 99–108, 2018.
[30] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Proc. IEEE LCN*, Nov. 2006, pp. 641–648.
[31] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection Tree Protocol," in *Proc. 7th ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, Nov. 2009, pp. 1–14.
[32] B. T. de Oliveira and C. B. Margi, "Distributed control plane architecture for software-defined wireless sensor networks," in *Proc. IEEE Int. Symp. Consum. Electron. (ISCE)*, Sep. 2016, pp. 85–86.
[33] B. T. D. Oliveira, R. C. A. Alves, and C. B. Margi, "Software-defined wireless sensor networks and Internet of Things standardization synergism," in *Proc. IEEE Conf. Standards Commun. Netw. (CSCN)*, Oct. 2015, pp. 60–65.
[34] M. D. Abrignani, C. Buratti, D. Dardari, N. El Rachkidy, A. Guitton, F. Martelli, A. Stajkic, and R. Verdone, "The EuWIn testbed for 802.15.4/Zigbee networks: From the simulation to the real world," in *Proc. 10th Int. Symp. Wireless Commun. Syst. (ISWCS)*. Frankfurt, Germany: VDE, Aug. 2013, pp. 1–5.
[35] M. Baddeley, R. Nejabati, G. Oikonomou, M. Sooriyabandara, and D. Simeonidou, "Evolving SDN for low-power IoT networks," in *Proc. 4th IEEE Conf. Netw. Softwarization Workshops (NetSoft)*, Jun. 2018, pp. 71–79.
[36] L. Chen and K. Bian, "Neighbor discovery in mobile sensing applications: A comprehensive survey," *Ad Hoc Netw.*, vol. 48, pp. 38–52, Sep. 2016.
[37] B. Milic and M. Malek, "NPART—Node placement algorithm for realistic topologies in wireless multihop network simulation," in *Proc. Simutools*, 2009, Art. no. 9. [Online]. Available: https://www.informatik.hu-berlin.de/de/Members/milic/NPART. doi: 10.4108/ICST.SIMUTOOLS2009.5669.
[38] MEMSIC, Andover, MA, USA. (2012). *TelosB Product Details*. Accessed: Sep. 2016. [Online]. Available: http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb datasheet.pdf
[39] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based online energy estimation for sensor nodes," in *Proc. ACM 4th workshop Embedded Netw. Sensors*, 2007, pp. 28–32.
[40] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, *The Trickle Algorithm*, document RFC 6206 (Proposed Standard), Internet Engineering Task Force, Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6206.txt
[41] N. Bizanis and F. Kuipers, "SDN and virtualization solutions for the Internet of Things: A survey," *IEEE Access*, vol. 4, pp. 5591–5606, Sep. 2016.

RENAN C. A. ALVES received the B.Sc. degree in electrical engineering with emphasis on computer and digital systems and the M.Sc. degree from the Universidade de São Paulo, Brazil, in 2011 and 2014, respectively, where he is currently pursuing the Ph.D. degree. His main research interests include network protocol modeling and performance analysis, and wireless sensor networks protocols and applications.

DORIEDSON A. G. OLIVEIRA received the Technician degree in digital games from FATEC Carapicuíba, Brazil, in 2015. He is currently pursuing the M.Sc. degree with the Universidade de São Paulo, Brazil. His main research interests include network protocol routing, wireless sensor networks protocols, and artificial intelligence.

GUSTAVO A. NUNEZ SEGURA received the B.Sc. degree in electrical engineering from the Universidad de Costa Rica, and the M.Sc. degree in electrical engineering from the Universidade de São Paulo, Brazil, in 2018, where he is currently pursuing the Ph.D. degree. His main research interests include energy consumption and security in wireless sensor networks and software-defined networking.

CINTIA B. MARGI received a degree in electrical engineering and the M.Sc. degree in electrical engineering from the Universidade de São Paulo, Brazil, in 1997 and 2000, respectively, the Ph.D. degree in computer engineering from the University of California Santa Cruz, USA, in 2006, and the Habilitation degree (Livre Docência) in computer networks from the Universidade de São Paulo, Brazil, in 2015, where she has been an Associate Professor, since 2015, and has been an Assistant Professor, since February 2007. Her research interests include wireless sensor networks (protocols, systems, security, energy consumption trade-offs, and management) and software-defined networking.