Novel Architecture and Heuristic Algorithms for Software-Defined Wireless Sensor Networks

Ammar Hawbani, Xingfu Wang, *Member, IEEE*, Liang Zhao, Ahmed Al-Dubai, *Senior Member, IEEE*, Geyong Min, *Member, IEEE*, and Omar Busaileh

***Abstract*— This article extends the promising software-defined networking technology to wireless sensor networks to achieve two goals: 1) reducing the information exchange between the control and data planes, and 2) counterbalancing between the sender's waiting-time and the duplicate packets. To this end and beyond the state-of-the-art, this work proposes an SDN-based architecture, namely MINI-SDN, that separates the control and data planes. Moreover, based on MINI-SDN, we propose MINI-FLOW, a communication protocol that orchestrates the computation of flows and data routing between the two planes. MINI-FLOW supports uplink, downlink and intra-link flows. Uplink flows are computed based on a heuristic function that combines four values, the hops to the sink, the Received Signal Strength (RSS), the direction towards the sink, and the remaining energy. As for the downlink flows, two heuristic algorithms are proposed, Optimized Reverse Downlink (ORD) and Location-based Downlink(LD). ORD employs the reverse direction of the uplink while LD instantiates the flows based on a heuristic function that combines three values, the distance to the end node, the remaining energy and RSS value. Intra-link flows employ a combination of uplink/downlink flows. The experimental results show that the proposed architectureand communication protocol perform and scale well with both network size and density, considering the joint problem of routing and load balancing.**

***Index Terms*— Wireless sensor networks, heuristic routing, MINI-FLOW, MINI-SDN.**

I. Introduction

CONVENTIONAL WSNs have been conceived to be application-specific, which makes it incredibly difficult to reconfigure high-level policies and respond to network-wide events. This is because employing these high-level policies necessitates specifying them in terms of distributed low-level configuration [1]. Obviously, task reprogramming or service (e.g., new routing policy) would necessitate each node to be taken out of *field-of-interest* and the embedded software reprogrammed in the node's hardware [6]. Thus, given the need for large-scale WSNs, this routine would not be realistic due to the fact that the network indeed contains a large number of randomly deployed nodes in a harsh environment that makes it almost impossible to manually take out the nodes after being deployed. Such a problem is inherent to conventional WSNs since each node is manufactured to accommodate all the functionalities from the physical layer to the application layer, behaving like an autonomous system that executes both the data forwarding and the network control [6].

In fact, the application nature in WSN reinforces the need to develop sensor nodes that are characterized as being remotely reconfigurable, reprogrammable, maintainable, and self-healing. By utilizing such sensor nodes, new services and routing policies can be introduced in the network as simple as installing a new software on a PC [3].

Inappropriately, today's conventional WSNs involve integration and interconnection of many proprietary, vertically integrated sensor nodes that make it extraordinarily rigid to specify high-level network-wide policies using current technologies. The rigidity of underlying infrastructure offers limited options for innovation or improvement since network nodes mostly have been closed with a low-level vendor-specific configuration that implements complex high-level network policies [1], [9]. As a result, it is very difficult to replace network nodes with any other produced by any vendor. This curbs the operator in vendor lock-in, and restrains the use of commodity hardware. Besides, it constrains the vendors in manufacturing isolated WSN nodes without adequately reusing common functionalities, which in turn adversely affects production and prototyping [9].

Recent technology shift proposes *Software-Defined Networking* (SDN) to dramatically simplify network configuration and resource management [7], [4]. By implementing such technology, WSN behavior can be reprogrammed even after the deployment of nodes by remotely injecting embedded defined software into sensor nodes [16]. SDN is essentially featured by disengaging the control decision (control plane) from the network devices, leaving the devices to perform data forwarding functionality (data plane) [8]. Such disengaging is particularly valuable to network operators and designers. In addition, it attracts

Manuscript received August 15, 2018; revised April 9, 2019, September 13, 2019, January 26, 2020, and April 24, 2020; accepted August 28, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Zhang. Date of publication September 22, 2020; date of current version December 16, 2020. This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant WK2150110012 and Grant WK2150110007 and in part by the National Natural Science Foundation of China under Grant 61772490, Grant 61472382, Grant 61472381, and Grant 61572454. *(Corresponding authors: Xingfu Wang; Liang Zhao.)*

Ammar Hawbani, Xingfu Wang, and Omar Busaileh are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China (e-mail: anmande@ustc.edu.cn; wangxfu@ustc.edu.cn; busaileh@mail.ustc.edu.cn).

Liang Zhao is with the School of Computer Science, Shenyang Aerospace University, Shenyang 110136, China (e-mail: lzhao@sau.edu.cn).

Ahmed Al-Dubai is with the School of Computing, Edinburgh Napier University, Edinburgh EH10 5DT, U.K. (e-mail: a.al-dubai@napier.ac.uk).

Geyong Min is with the Department of Computer Science, University of Exeter, Exeter EX4 4QF, U.K. (e-mail: g.min@exeter.ac.uk).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TNET.2020.3020984

academia, industry and development community to pay increasing attention to specific research aspects in both WSN and SDN [4].

A. Benefits of SDN in WSN

1) By employing SDN, the nodes behave as data plane devices to forward data without interfering network control tasks such as topology management and routing strategies, simplifying their architecture and enhancing their energy efficiency. Moreover, armed with a network-wide view, the controller introduces comprehensive duty-cycling, scheduling, routing and coverage & connectivity solutions [5]. Consequently, such sophisticated strategies will be introduced to WSN as simply as it is to remotely install new software on a PC [3].
2) Freeing WSNs from being vendor-specific, proprietary, vertically integrated networks, allowing different vendors to develop application-customizable devices in a non-isolated manner by sharing common functionalities. Besides accelerating the prototyping, production and protocols innovation [2], this avoids the complicated network management imposed by deploying multi-vendor nodes [10].
3) Releasing WSNs from being application-specific, behaving in a plug-and-play manner since the data plane virtually supports all kinds of forwarding rules. Besides, the control plane separates the application layer from the physical layer, supporting multiple applications with different hardware to work under the same physical network architecture [5].

B. Motivations and Contributions

Based on the limitations of WSNs and the benefits of SDN concept, WSN developers realized the need to shift from conventional WSNs to software-defined WSNs [6]. Recently, a few studies highlighted the importance of extending SDN concepts to WSNs, noticeably focusing on the challenges, the requirements and the logical architecture of network components [2]–[10]. These studies provided convincing motivations for the extension of SDN concept to conventional WSN. However, these studies mostly addressed the challenges and the abstract architecture while the information exchange between the controllers and the end nodes has not been addressed well so far. Motivated by these observations, in this work, we briefly address the WSN-SDN architecture and then deeply address the flows computation and instantiation, aiming at the following two goals. The first goal is to minimize the information exchange (network overhead) between the controller and the end nodes. The second goal is to trade-off between the waiting-time and the duplicate packets when the nodes are duty-cycled. More details about the trade-off between the sender waiting-time and the duplicate packets can be found in [29]. The main contributions of this article are as follows:

1) We propose *MINI-SDN*, a new architecture that integrates the conventional WSN and the *Software-Defined Networking (SDN)*.
2) Based on *MINI-SDN,* we propose *MINI-FLOW*, a communication protocol that facilitates the intercommunication between the elements in the data plane and the control plane.
3) We further develop innovative heuristic algorithms to manage uplink, downlink and intra-link flows while reducing the information exchanged between the control and data planes.
4) We implement MINI-*SDN* & *MINI-FLOW* on the platform proposed in [28], [29]. The source code is publicly available in the link.[1] The simulation results showed that our architecture MINI-*SDN* and communication protocol *MINI-FLOW* perform and scale well with both network size and density.

The remainder of this article is organized as follows. The related works are reviewed in the next Section. Section 3 and Section 4 present the *MINI-SDN* architecture and the *MINI-FLOW* protocol, respectively. Section 5 provides an analysis of the proposed *MINI-FLOW* protocol. The experimental results are evaluated in Section 6. Finally, Section 7 concludes this work.

II. Related Work

Currently, *OpenFlow* [18] is the most popular instance of *Software-Defined Networking*, which has been proposed to resolve analogous issues in wired and *address-centric* networks. It achieved a high standing position in the networking market and attracted many of the networking leader vendors including *HP*, *NEC*, *NetGear*, and *IBM*, to manufacture *OpenFlow-compatible* devices available in the market. In addition, it attracted developers to design a variety of SDN-based *controllers* e.g., *NOX*, *Floodlight*, and *Maestro,* available online [19]. *OpenFlow* creates one or more *Flow Tables* for each device to execute the packet lookups and forwarding, controls the communication between the *controller* and network devices. The *flow entry* of the *Flow Table* is composed of three sections, *matching rule*, *actions* and *statistical information*. The *matching rule* specifies the values and conditions under which the *flow entry* applies. After matching the *rules*, the node executes *actions* (e.g., drop, update, forward to, etc.) identified in the second section of the *flow entry*. The third section contains *statistical information* about the flows.

OpenFlow could not fulfill the requirements of WSN as its underlying infrastructure is composed of high-speed switches (e.g., Ethernet/MPLS switches and IP routers) whereas the WSN is characterized by low capabilities in terms of memory, processor, and energy source. Furthermore, WSN is a *data-centric* network which means that collecting the sensory data is more important than knowing who sent the data, while *OpenFlow* is designed for *address-centric networks*, which implicitly assume the presence of IP-like addressing to create flows.

Technical challenges of extending *OpenFlow* to WSN are thoroughly articulated in [9]. Motivated by the challenges, including *Creating flows, Openflow channel*, *Overhead of*

[1] https://github.com/howbani/WSNSIM

Control Traffic and *Traffic Generation*, Luo *et al.* [9] proposed *Software-Defined WSN* (SD-WSN), an architecture featuring a clear separation between data and control planes. The architecture was divided into three layers, data, control, and application. Moreover, they provided a suite of preliminary solution, named as *Sensor OpenFlow* (SOF), to overcome the aforementioned challenges. SOF introduces an *OpenFlow*-extension, supporting *SDN-WSN* by plugging new forwarding rules to *OpenFlow.* The rules of SOF slightly meet the requirements of WSN and handle in-network packet processing via various types of WSNs defined addressing.

Although SOF addressed a few technical challenges of extending SDN to WSN, critical requirements such as duty cycles and in-network data aggregation were not addressed. Therefore, these requirements were analyzed in [4] and based on the discussion, Costanzo *et al.*, proposed an SDN-architecture, named as *Software-Defined Wireless Network* (SDWN) which is designed based on IEEE 802.15.4 standard (for low-power wireless nodes that operate in 868 MHz, 915 MHz and 2.4 GHz frequency bands). Unlike SOF [9], the SDWN [4] contains aggregation layer, data forwarding layer and application layer on the top of PHY and MAC layers. Furthermore, to cope with the challenges and requirements, *Smart* [10] undertakes a solution to some inherent problems in WSN such as network management, node mobility, localization and topology discovery. Unlike SOF [9] and SDWN [4], *Smart* [10] composes of five-layer stack, PHY, MAC, NOS (*Network Operating System*), MW (*Middle-Ware*) and application layer. Besides, *Smart* suggests to reside the controller in the sink, and used a localization service for location-based routing.

Furthermore, the requirements such as duty cycles, in-network data aggregation, flexible rules and space constraints, which are essentials in designing *Software Defined Wireless Network, are* not considered in *OpenFlow.* To meet these requirements and inspired by [4] and [9], the WSN-WISE [3], [19] extended *Openflow* to support *data aggregation*, *duty cycling* and *network function virtualization*. It simplifies policy implementation within a reprogrammable and vendor-independent WSN. Besides, WSN-WISE supports multiple controllers serving as a proxy between the two planes, allowing packets to travel following different *flow rules* defined by different controllers. These controllers periodically update the flow tables to let the nodes know their next-hop node towards the controllers. The topology information is collected through the *discovery layer*, which has access to the protocol stack. Above the IEEE 802.15.4 stack, WSN-WISE defines the *forwarding layer* to handle the arriving packets as specified in the *Flow Table*. On top of the *forwarding layer*, WSN-WISE defines the *In-Network Packet Processing* layer to handle data aggregation.

Enabling technologies to implement *Software-Defined Sensor Network* (SDSN) are presented in [2]. Zeng *et al.* [2] introduced an SDSNs based *Cloud Sensing* architecture which consists of a single control server and a set of *software-defined* nodes. To deploy a new sensing task, the server remotely reprograms a few nodes in a distributed manner such that the reprogrammed nodes are admissible to sense and report the related targets. The functions to be activated in sensor nodes are defined in the server, which provides a role generation and delivery mechanism. Later, SDSN was extended in [16] by proposing a reconfigurable node that consists of a *low-power field-programmable gate array* (FPGA) and a *microcontroller unit* (MCU) for changing network behavior.

The rest of this section introduces a few of studies that attempted to address the problems related to the data routing between the two planes, data and control planes. Xiang *et al.* [25] proposed a routing algorithm in which the nodes are divided into clusters each assigned with a control node (cluster head). Based on the residual energy of the nodes and the transmission distance, the controller selects the control node. The selection of control nodes is formulated as an *NP-hard* problem which is optimized by adopting *particle swarm optimization* algorithm. Zeng *et al.*, [26] investigated the coverage sets and nodes activations together with the task assignment and sensing scheduling. These problems are jointly formulated as a mixed-integer with quadratic constraints programming and mixed-integer linear programming. More recently, Li *et al.* [27] presented a Levenberg–Marquardt algorithm for solving the optimization problem of traffic load. They also provided a convergence analysis of the Levenberg–Marquardt algorithm.

LORA [29] is an opportunistic routing protocol that employed *zone* routing, in which each node defines a *candidate's zone.* Candidates within the *zone* are prioritized based on a metric, which is defined as multiplication of direction, transmission distance distribution and residual energy. ZPR [28] modeled the data routing as an in-zone random process. The zone in ZPR is defined by the source, while the zone in LORA is locally defined by each relay node. Candidates within the zone of ZPR are randomly selected. The implementation of LORA and ZPR is available online in the link [2] and the link,[3] respectively.

The aforementioned literature provided significant contributions and offered convincing motivations for expanding SDN concepts to WSN. However, designing an abstracted architecture for WSN was their main research point while the information exchange between the controllers and the end nodes has not been addressed well so far. In addition, the combined problem of data routing and load balancing between the two planes is not practically studied. Moreover, the communication overhead between the two planes is not well evaluated. Thus, we go beyond the state-of-the-art by proposing an SDN based architecture called MINI-SDN which separates the control from data plane and paves the way for the proposed communication protocol called MINI-*FLOW* which undertakes the packet exchange between the two planes.

III. The Proposed Architecture Mini-sdn

This section explains our proposed software-defined architecture for WSN. As shown in Figure 1, the *WSN- SDN* logical architecture contains three components, the sensor network (*data plane*), the sink, and the *controller*. The nodes reach

[2]https://github.com/howbani/lora
[3]https://github.com/howbani/zpr

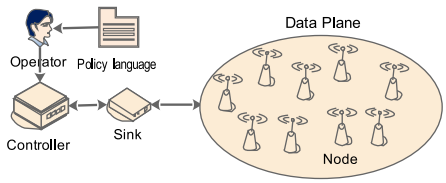

Fig. 1. Simplified view of the software-defined sensor network structure.

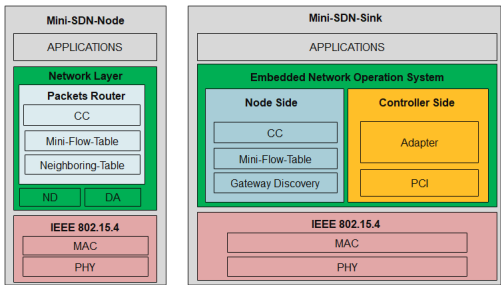

Fig. 2. MINI-SDN protocol stack for sink and node.

the sink either directly by one hop or indirectly through multiple hops. The sink acts as an intermediate agent (gateway) between the *control plane* (*controller*) and the end nodes (*data plane*). The controller could be hosted either internally in the sink or externally on a remote server. We consider the *controller* as an external independent device that communicates with the sink directly but not with the sensor nodes. Like [4], each node in our architecture is equipped with a *micro-control unit* (*IEEE 802.15.4 transceiver*). Furthermore, the nodes have limited processing, memory, communication and energy capabilities. The sink transceiver, based on *IEEE 802.15.4* standard, is connected to an *Embedded Network Operation System* (ENOS) which is armed with high computing and communication capabilities.

MINI-SDN is an architecture that separates the control and data planes of WSNs. It consists of three integrated sub-architectures, *MINI-SDN-Node*, *MINI-SDN-Sink* and *SDN-WSN* controller. Figure 2 and Figure 3 explain the proposed architecture of *MINI-SDN*. Specifically, Figure 2 depicts the proposed architecture for sinks (right side) and the sensor node (left side) while Figure 3 depicts the controller of *MINI-SDN*. The two architectures (*MINI-SDN-Node* and *MINI-SDN-Sink)* are intensely explained in the Subsections 3.1 and 3.2, respectively, while the *MINI-SDN-Controller* is explained in Subsection 3.3.

A. MINI-SDN Architecture for Sensor Nodes

MINI-SDN-Node protocol stack runs the basic **PHY** and **MAC** functionalities defined by the standard IEEE 802.15.4. At the top of IEEE 802.15.4 protocol stack, *MINI-SDN-Node* defines the *network layer*, which runs in the *micro-control unit*, comprising three main blocks *Neighbors Discovery* (**ND**), *Data Aggregation* (*DA*), and *Packets Router,* see Figure 2 (left side). **ND** allows each node to store and discover the information (i.e., ID, battery state and RSSI) of nearby nodes. To achieve this, each node broadcasts a beacon packet and waits for a response from a nearby node that receives the beacon. *DA* executes data aggregation or decision fusion, aiming to reduce data redundancy and conserve network resources [9]. *Packets Router* is adapted to improve the performance of low Duty-cycled WSN by exploiting its broadcast nature (i.e., a node may pick up the packets that are destined to other nodes). Considering the duty cycles where a node randomly switches its mode to active or sleep, the *Packets Router* selects a set of candidates as potential *forwarders* in order to reduce sender *waiting-time* as well as to minimize *duplicate packets*. When a sender has a packet to send, it transmits preambles continuously until either one of its candidates sends back an ACK or the preset timer of active duration expires. Typically, the sender seizes the earliest forwarding opportunity and does not need to wait for the predefined forwarder to wake up. When multiple candidates wake up, to avoid packet duplication the sender should coordinate its candidates and select one to forward the packet. *Packets Router* comprises three blocks, *Candidates Coordination* (CC), *MINI-FLOW-Table* and *Neighboring-Table*. *Candidates Coordination* (CC) implements an important mechanism intended to avoid packets duplication. Given the fact that the nearby nodes can overhear any potential traffic within their ranges, the sender should be able to determine which neighboring node is going to forward the packet to the next hop. Otherwise, a large number of duplicate packets will incur. Note that *CC* has access to edit the content of *MINI-FLOW-Table*. Packets are forwarded based on the matching rules specified in *MINI-FLOW-Table*. The rules are remotely computed in the *controller*. We will deeply explain the *MINI-FLOW-Table* and *Neighboring-Table* in Section 4.

B. MINI-SDN Architecture for Sinks

The *MINI-SDN-Sink* architecture is depicted in Figure 2. Like the protocol stack of *MINI-SDN-Node* explained in the previous subsection, *MINI-SDN-Sink* runs the basic **PHY** and **MAC** functionalities defined by the standard IEEE 802.15.4. At the top of the **MAC** layer, *MINI-SDN-Sink* defines *Embedded Network Operation System* (ENOS), which should have access to all layers including IEEE 802.15.4 protocol stack and application layer, see Figure 2.

The ENOS is an intermediate interface that operates the communications from sink to nodes and from sink to the controller. ENOS involves two main sides, the *Controller-side* and *Node-side*. The *Node-side* is an interface responsible for nodes/sink intercommunications, while the *Controller-side* is responsible for *controller*/sink intercommunications. The *Node-side* contains three parts, the *Candidates Coordination* (**CC**), *MINI-FLOW-Table,* and the *Gateway Discovery.* The CC and *MINI-FLOW-Table* implement the same functionalities as in *Packets Router* of *MINI-SDN-Node*, explained in the previous subsection. *Gateway Discovery* discovers the access nodes (i.e., the sensor nodes that directly communicate via one hop with the sink). The *Controller-side* composes of two components PCI (*Programmable Communication Interface*) and *Adapter*. The PCI component is the communication interface (e.g., USB, RS232, TCP/IP, etc.) that manages the

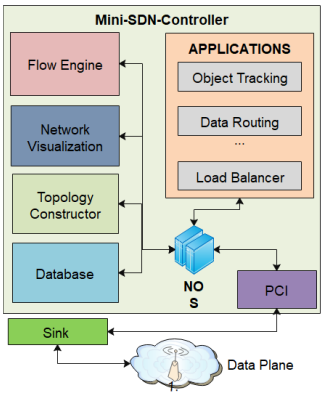

Fig. 3. Simplified view of MINI-SDN-Controller.

communication between the sink and the *controller* while the *adapter* component is responsible for rendering network policies and messages into a format that can be understood by the sensor nodes and the *controller*.

C. The Controller of MINI-SDN

Network services (e.g., high & low levels network-wide policies, the traffic-forwarding decisions, and so on) are considered as the essential functionality that should be provided by the central *controller* which employs a *Network Operating System* (NOS), for example, Linux-based embedded system with high computing and communication capabilities. NOS manages and executes the main network functions such as program execution, I/O operations, security, and communications [13]. Our proposed *Mini-Controller* for WSN, as shown in Figure 3, implements the network service functions in the NOS, which comprises the *Database*, *Flow-Engine* (FE), *Network Visualization* (NV), *Topology Constructor* (TC), PCI and *Applications*. PCI is a communication interface between the controller and the sinks. The *Database* is designed to store the topological collected data such as the node's ID, node's location, node's neighbors, node's battery level, sinks and so on. Based on the data stored in the *Database* component, the *Topology Constructor* constructs a *graph-based* structure for the sensor network by using the vertices and edges. Utilizing the *Topology Constructor* and the topological collected data in the *Database,* the *Network Visualization* constructs a consistent and comprehensive representation of the current state and statistics of the network. The *Flow-Engine* is the essential component of the controller in which all the traffic-forwarding decisions are taken and all the high & low levels network-wide policies are implemented. In addition, *Flow-Engine* translates the network-wide event-driven policies into forwarding rules implemented in the nodes. Besides, it is responsible of updating the *MINI-FLOW-Table* of the nodes and the sinks.

IV. The Proposed Protocol Mini-Flow

MINI-FLOW is designed to manage and compute the flows and the paths between the control and data planes. It involves three mechanisms, the uplink, the downlink and the data routing at the network initialization level. In addition, *MINI-FLOW* proposes intra-link flows to support routing of the traffic within the data plane. Before addressing these mechanisms, we will first elaborate the substructures of *MINI-FLOW*, the *packet header*, *packets types*, *MINI-FLOW-Table* and *Neighbors-Table*. By the end of this subsection, we will introduce the forwarders coordination, the update mechanism of the flows and the exponential system parameters that control the flows. Note that, *MINI-FLOW* is designed based on the low power listening BoX-MAC (i.e., the default MAC protocol in TinyOS) [11]. The common notations used in this article are summarized in Table I.

TABLE I
NOTATIONS

Notation	Description
$\mathbb{N}$	$\mathbb{N} = \{n_0, n_1, n_2 \ldots\}$; $n_i \in \mathbb{N}$ is a sensor node.
$\mathbb{N}_i$	Neighboring set of n_i;
m_i	Size of $\mathbb{N}_i$.
n_t, n_s, n_b	End, source and sink nodes, respectively.
$H(n_i)$	Number of hops from n_i to the sink.
$R(n_j \in \mathbb{N}_i)$	RSSI value from n_i to $n_j \in \mathbb{N}_i$.
L_i	Residual energy of n_i.
L_*	Initial energy.
x_i, y_i	Location of the node n_i.
e	*Euler's constant*, approximately 2.71828.

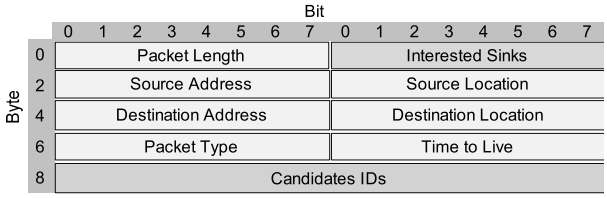

Fig. 4. Packet header format.

A. Packet Header

To reach their intended destinations, the packets travel in the network using the fixed header format (10 bytes long) which is similar to the formats mentioned in the literature [3], [4] with few differences, the *Interested Sinks* and the *Candidates IDs* (see Figure 4). When a network has multiple mobile sinks, the *Interested Sinks* specifies a group of the sinks which are interested in the content of the packet, whereas the *Candidates IDs* specifies the *Candidates* nodes which listened and acknowledged the packet. Furthermore, the header contains the estimated locations of the source and destination. These locations are estimated by utilizing the simplest ranging method, *Received Signal Strength* (RSS) that represents the relationship between the transmission distance and the signal strength degradation. Such relation is modeled by the *path loss* of the signal (radio signal degradation with distance). To calibrate the unstable changes of RSS, the self-calibration protocol, explained in [17], is utilized.

TABLE II
MINI-FLOW-TABLE

Candidate ID	Flow-Direction	Priority	ACK	Action	Statistics
1	Uplink	0.37	1	Forward	37
2	Uplink	0.34	0	Forward	34
3	Uplink	0.29	1	Drop	29
1	Downlink	0.20	1	Drop	20
2	Downlink	0.30	1	Drop	30
3	Downlink	0.50	0	Forward	50
1	Intra-Link	0.46	0	Forward	48
2	Intra-Link	0.44	1	Forward	45
3	Intra-Link	0.10	1	Drop	7

B. Packet Types

Based on the broadcasting nature of WSNs, *MINI-FLOW* supports five types of packets, listed as follows.

Type 1. Beacon Packet: the sink or the sensor nodes periodically broadcast this type of packet, which is intended for neighboring discovery, battery state query, etc. The header of this packet does not encapsulate any of the followings: *address/location* of the destination, *Candidates IDs* and *interested sinks*. The time to live is set to one (hop). The nodes, which hear this type of packets, send back a *Response Packet* to the sender, reporting their information like battery states, number of hops to the sink, ID and RSSI. The header of the *Response Packet* contains the following: time to live (One hop), *Candidates IDs* (the sender of beacon packet).

Type 2. Preamble Packet: When a sender has a *data packet*, it sends preamble packets before transmitting the *data packet*. The nodes in *Neighbors-Table*, which hear this type of packet, send back an *ACK packet* to the sender acknowledging their availability for receiving the *data packet*. The header of this packet contains the *Candidates IDs* and the *address/location* of the sender. It does not contain *interested sinks* and *address/location* of the destination. The time to live is set to one (hop).

Type 3. ACK Packet: The nodes in the *Neighbors-Table*, which hear the *Preamble Packet*, send back an *ACK packet* to the sender acknowledging their availability for receiving the *data packet.* When the sender receives the *ACK packet,* it sets the *ACK* value (Table II) to 1.0.

Type 4. Data Packet: After receiving an *ACK-Packet*, the *Candidates Coordination* (CC) in the *Packets Router* (Figure 2) sends the *data packet* to the next hop based on the *Priority* value (Table II). In this type of packet, the *address/location* of *destination* are attached to the header. In the field of *Candidates IDs*, the sender node specifies one candidate (next hop in the path) based on the *Priority* (Table II).

Type 5. Control Packet: The exchanged packets between the *controller* and the nodes include the rule/action request, rule/action response, topology info, etc.

C. Mini-Flow-Table

Broadcasting nature and resource scarcity in WSNs impose that the WSN-SDN architecture should be characterized by being simplified and highly efficient. In such constrained networks, the nodes periodically switch between active and sleep states according to predefined wake-up intervals. *MINI-FLOW* is designed based on the low power listening BoX-MAC. MINI-FLOW saves the flows as the entries in the *MINI-FLOW-Table*, shown in Table II, in which each flow is defined based on the direction of the flow, uplink, downlink or intra-link.

TABLE III
NEIGHBORS-TABLE

Node ID	Battery Level	Estimated Location	H(i)	R (i)
n1	Li,1	x1,y1	Hi,1	Ri,1
n2	Li,2	x2,y2	Hi,2	Ri,2
n3	Li,3	x3,y3	Hi,3	Ri,3
n4	Li,4	x4,y4	Hi,4	Ri,4

MINI-FLOW-Table is designed to achieve two goals. The first goal is to reduce the exchange of information between the controller and end nodes by using a fixed flow that targets each node rather than each packet. The *controller* delivers the uplink, downlink or intra-link flows to each node after network initialization. The *controller* does not need to define the flows for each packet. This reduces the amount of information exchanged between the two planes. The *controller* updates the flows directly based on the *statistics* received from the end nodes. Flows could be updated whenever an end-node requests the routing policy.

The second goal is, in case of duty-cycled nodes, to reduce the *waiting-time* and to minimize the *duplicate packets*, each node is assigned with multiple flows each has a *Priority* value, which is heuristically computed in the *Flow-Engine.* When the sender-node has a packet, it sends a preamble packet, which will be heard by all awaken neighboring nodes in Table III. The nodes, which received the preamble packet, will send back an ACK to the sender. The sender sets the *ACK* value to 1.0 in the *MINI-FLOW-Table*. Otherwise, the *ACK* value is set to 0.0. Based on the *Priority* value, if the flow is matched (i.e., the candidate is selected), then the corresponding *Action* is executed and the information in the *Statistics* is updated. The *Statistics* information is periodically reported to the *controller.* When the *ACK value is 0.0*, the corresponding *Action* will not be executed even though the *Priority* value is the highest. Table II shows an example of the *flows* in a node that has three neighboring nodes. We will explain how the *controller* computes the *Actions* and *Priority* in the Subsection 4.4. Table III shows the information to be stored in the node n_i. We assume it has four neighboring nodes $\mathrm{N}_i = \{n_1, n_2, n_3, n_4\}$. The function $H(n_j \in \mathrm{N}_i)$ returns the number of hops from n_j to the sink while $R(n_j \in \mathrm{N}_i)$ returns the RSSI value from n_i to n_j.

D. Flows Computing

This section is devoted for computing uplink flows (i.e., the paths from the end nodes to the *controllers*), the downlink flows (i.e., the paths from the controllers to the end nodes), and the intra-link flows (i.e., data traffic within the data plane such as a path from a node to a *cluster head* CH in clustered WSN). *MINI-FLOW* supports both *flat-based* routing (i.e., all nodes of equal role) and *hierarchical-based* routing (i.e., different nodes may assume different roles). The following four subsections (4.4.1 to 4.4.4) elaborate the flows of

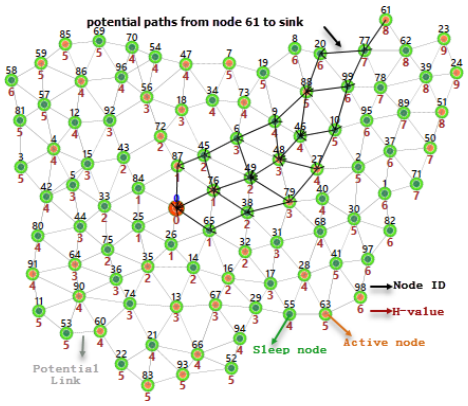

Fig. 5. Network initialization. An example of 100 nodes (indicated by the circle). The digits indicate the *H-value (lower)* and the *ID (upper)* of each node. The arrows show the available paths from the source node n_{61} to the sink n_0. H-value is computed by ALGORITHM 1.

Algorithm 1 Network Initialization H is a function that returns the number of hops to the sink.

```
1.  NetworkInitialization (n_x)
2.   {
3.  n_x broadcasts a beacon packet;
4.    if(n_i hears the beacon)
5.      if(H(n_x) < H(n_i))
6.        H(n_i) = H(n_x) + 1;
7.        n_i sends response ton_x;
8.         if(n_x received the response from n_i)
9.            N_x = N_x ∪ {n_i};
10.       End if
11.     End if
12.   End if
13. }
```

flat-based routing while Subsection 4.4.5 explains the flows of *hierarchical-based* routing.

1) Network Initialization: The goal of this phase is to have each node identify its *H-value(s),* so that it can simply share the information (i.e., *R-values, H-value* and energy level) with the *controller*. In *flat-based* routing, the initialization phase starts at the sink where the *H-value* is set to 0. The sink starts the initialization by sending a *beacon,* then the nodes that received the *beacon* execute *ALGORITHM 1,* and further send their own new *beacon*. The process is terminated when each node knows its *H-value*. Figure 5 shows an example of implementing *ALGORITHM 1* on 100 nodes. The collected information is reported to the sink by the following four steps. (1) The sender node transmits a preamble packet. (2) The nodes that received the *preamble packet* send back the ACK packets to the sender expressing their availability. (3) The *Packets Router* selects a candidate from the nodes, which expressed their availability providing that the *H-value* of the candidate node is smaller than the *H-value* of the sender. (4) The *Packets Router* specifies the ID of the candidate in the *Candidates IDs* field of the *Control Packet*. This process is repeated until the *Packet* reaches the sink node. The collected information is stored in the *Database* so the *Network Visualization* (NV), *Topology Constructor* (TC) and *Flow-Engine* (FE) can utilize it. In addition, each node saves its neighbors information in the *Neighbors-Table* (Table III).

2) Uplink Flows: Uplink flows are computed through a heuristic function that combines four values, the number of hops to the sink, the *Received Signal Strength* (RSS), the direction towards the sink and the remaining energy. The data routing from nodes to the controllers is determined based on the *Priority* value of each flow in the *MINI-FLOW-Table* (Table II*)*. As shown in Figure 5, each node has multiple candidates at each transmission stage. Accordingly, each node has multiple potential paths to reach the sink. To optimize the path selection, we propose a heuristic algorithm that depends on the four values *H-value*, *R-value*, E-value and *L-value*. E-value represents the *Euclidean distance* from the node to the sink. *H-value* represents the number of hops from the node to the sink, obtained by ALGORITHM 1. *L-value* represents the remaining energy while *R-value* is a function of the distance between the sender and receiver node, which varies due to various in-path interferences [12]. *R-value* is considered as a term in the proposed heuristic function since the energy consumption is strongly related to the transmission distance between the sender and receiver nodes [14]. Each of the four values has an impact on path selection.

(1) *H-Distribution*: This probability distribution prioritizes the nodes, which have smaller *H-value* (i.e., minimum the number of hops to the sink as computed by ALGORITHM 1). Each node n_i expresses the *H-values* of its neighboring nodes N_i as a vector, $H_i = \{H_{i,1}, H_{i,2}, \ldots, H_{i,m_i}\}$ where $H_{i,j} = H(n_j), n_j \in \mathrm{N}_i, m_i = |\mathrm{N}_i|$. Then, H_i is normalized into $\bar{H}_i = \{\bar{H}_{i,1}, \bar{H}_{i,2}, \ldots, \bar{H}_{i,m_i}\}$ by Eq.(1). Finally, the *H-value Distribution* is defined by the *mass function* Eq.(2), denoted by $\tilde{H}_i = \left(\tilde{H}_{i,1}, \tilde{H}_{i,2}, \ldots, \tilde{H}_{i,m_i}\right)$. Note that in Eq.(1), the $\gamma_H \geq 0$ is called the *H-exponent*. Greater value of γ_H offers a higher probability distribution for the nodes with smaller *H-values* to be selected as candidate nodes.

$$\bar{H}_{i,j} = (1 + H(n_j))^{-\gamma_H} \, \forall n_j \in \mathbb{N}_i \tag{1}$$

$$\tilde{H}_{i,j} = \left(1 - e^{\bar{H}_{i,j}}\right) / \sum_{k=1}^{m_i} \left(1 - e^{\bar{H}_{i,k}}\right) \forall n_j \in \mathbb{N}_i \tag{2}$$

Figure 6 shows the impact of γ_H on *H-Distribution* when node n_i has 5 neighbor nodes, $\mathrm{N}_i = \{n_1, n_2, \ldots, n_5\}$. For simplicity, the *H-Values* of the neighboring nodes are assumed to be $H_i = \{H_{i,1} = 1, H_{i,2} = 2, \ldots, H_{i,5} = 5\}$, whereas γ_H varies from 0 to 1. The *H-Distribution* curve shows that the larger the value of *H-exponent* (γ_H), the greater the probability of selecting the node having a smaller *H-value*.

(2) *R-Distribution*: This distribution prioritizes the node with larger *R-value*. We assume that the RSSI value is represented in a positive form, the closer the value is to zero, the stronger the received signal has been. RSSI can vary greatly and affect the functionality in wireless networking. It is derived in the intermediate frequency (IF) stage before the IF amplifier. Node n_i expresses the *R-values* of its neighbor nodes N_i as a vector $\mathrm{R}_i = \{R_{i,1}, R_{i,2}, \ldots, R_{i,m_i}\}$ where

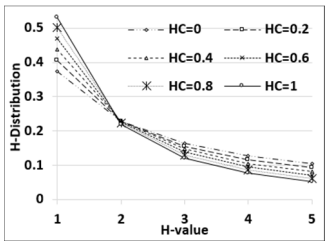

Fig. 6. The curve of H-Distribution when the H-exponent control (HC) varies from 0.0 to 1.0.

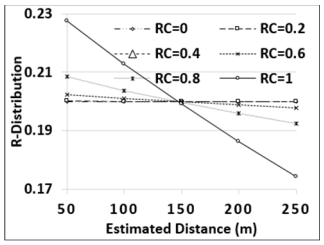

Fig. 7. The curve of R-Distribution when the R-exponent control (RC) varies from 0.0 to 1.0.

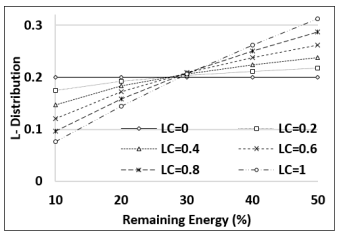

Fig. 8. The curve of L-Distribution when the L-exponent γ_L varies from 0.0 to 1.0.

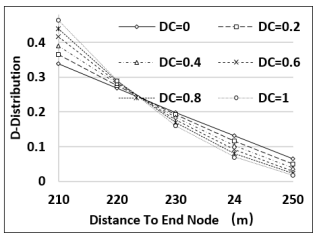

Fig. 9. The curve of D-Distribution when the D-exponent control (DC) varies from 0.0 to 1.0.

$R_{i,j} = R(n_j), n_j \in \mathrm{N}_i, m_i = |\mathrm{N}_i|$. Then, R_i is normalized into $\bar{R}_i = \left\{\bar{R}_{i,1}, \bar{R}_{i,2}, \ldots, \bar{R}_{i,m_i}\right\}$ by Eq.(3). Finally, the *R-value Distribution* is defined by the *mass function* Eq.(4), denoted by $\tilde{R}_i = \left(\tilde{R}_{i,1}, \tilde{R}_{i,2}, \ldots, \tilde{R}_{i,m_i}\right)$. Note that in Eq.(4), the $\gamma_R \geq 0$ is called the *R- exponent*. Larger value of γ_R offers higher probability for the nodes, which have greater *R-value* to be selected as forwarders.

$$\bar{R}_{i,j} = 1 - \left([R(n_j)^{\gamma_R}] / \sum_{k=1}^{m_i} R(n_k) \right) \forall n_j \in \mathbb{N}_i \quad (3)$$

$$\tilde{R}_{i,j} = e^{\bar{R}_{i,j}} / \sum_{k=1}^{m_i} e^{\bar{R}_{i,k}} \forall n_j \in \mathbb{N}_i \quad (4)$$

During simulation experiments, RSSI-based distance estimation model, as explained in [15], is employed to obtain the expected transmission distance. We implemented the *free space propagation model* [24] that assumes the ideal propagation condition by representing the communication range as a circle around the transmitter. If a receiver is within the circle, it receives all packets. Otherwise, it loses all packets [17]. Figure 7 shows the impact of *R- exponent* γ_R on the *R-Distribution* ($\tilde{R}$) when n_i has 5 neighbor nodes $\mathrm{N}_i = \{n_1, n_2, \ldots, n_5\}$ with transmission distance varying from 50m to 250m. The *R-Distribution Control (RC)* varies from 0 to 1. The *R-Distribution* curve shows that as the value of γ_R becomes larger, the probability of selecting the nodes having larger R-value becomes higher.

(3)*L-Distribution*: The goal of this distribution is to assign a higher priority to the nodes with a greater remaining energy level. Node n_i defines an energy-normalized vector $\bar{L}_i = \left(\bar{L}_{i,1}, \bar{L}_{i,3}, \ldots, \bar{L}_{i,m_i}\right)$ by Eq.(5). Then, the values on vector $\bar{L}_i$ are plugged into the mass function Eq.(6) to obtain a discrete random variable, $\tilde{L}_i = \left(\tilde{L}_{i,1}, \tilde{L}_{i,2}, \ldots, \tilde{L}_{i,m_i}\right)$. Note that L_j denotes the residual energy of node n_j while L_* denotes the initial energy of n_j. Also, in Eq.(6), the $\gamma_{\mathrm{L}} \geq 0$ is called the *L-exponent*. Greater value of γ_L offers greater probability distribution for the nodes, which have more remaining energy to be selected as candidates.

$$\bar{L}_{i,j} = \left[\frac{L_j}{L_*}\right]^{\gamma_{\mathrm{L}}} \forall n_j \in \mathbb{N}_i \quad (5)$$

$$\tilde{L}_{i,j} = \left(1 - e^{-\bar{L}_{i,j}}\right) / \sum_{k=1}^{m_i} \left(1 - e^{-\bar{L}_{i,k}}\right) n_j \in \mathbb{N}_i; \gamma_L \geq 0 \quad (6)$$

Figure 8 shows the impact of *L-exponent* (γ_{L}) on the *L-Distribution* ($\tilde{L}_i$) when node n_i has 5 neighbor nodes $\mathrm{N}_i = \{n_1, n_2, \ldots, n_5\}$. For simplicity, the *L-Values* of the neighboring nodes are assumed to be $L_i = \{L_{i,1} = 10, L_{i,2} = 20, \ldots, L_{i,5} = 50\}$ whereas γ_L varies from 0 to 1. The *L-Distribution* curve revealed that when the value of *L-exponent* γ_L is greater, the probability of selecting the nodes which have more residual energy (*L-value*) gets higher.

(4) Euclidean Distance to the Sink (E-*Distribution*): The E-*Distribution* aims to assign a higher priority in each transmission session for the nodes that are in the direction towards the sink n_b. Sender n_i expresses the E-*value* of its neighbor nodes (i.e., $\forall n_j \in \mathrm{N}_i$) as a vector, $\mathrm{E}_i = \{\mathrm{E}_{i,1}, \mathrm{E}_{i,2}, \ldots, \mathrm{E}_{i,m_i}\}$, where $\mathrm{E}_{i,j}$ represents *Euclidean distance* from relay node n_j to sink n_b as expressed in Eq. (7). The vector E_i is normalized into $\bar{E}_i = \left\{\bar{E}_{i,1}, \bar{E}_{i,2}, \ldots, \bar{E}_{i,m_i}\right\}$ by Eq. (8), where φ_i denotes the communication radius of n_i. Finally, the E-*Distribution* is obtained by Eq. (9), $\tilde{E}_i = \left(\tilde{E}_{i,1}, \tilde{E}_{i,2}, \ldots, \tilde{E}_{i,m_i}\right)$. In Eq. (8), $\gamma_{\mathrm{E}} \geq 0$ is a system parameter, called the E- exponent. Greater value of γ_{E} offers a higher probability for the nodes that have smaller E-*value*

(closer to the sink node) to be selected as candidates.

$$E_{i,j} = \sqrt{(x_b - x_j)^2 + (y_b - y_j)^2} n_j \in \mathbb{N}_i \tag{7}$$

$$\bar{E}_{i,j} = \left(1 - \sqrt{\frac{E_{i,j}}{E_{i,b} + \varphi_i}}\right)^{\gamma_E}, n_j \in \mathbb{N}_i \tag{8}$$

$$\tilde{E}_{i,j} = \bar{E}_{i,j} \sum_{k=0}^{m_i} \bar{E}_{i,k}, n_j \in \mathbb{N}_i \tag{9}$$

(5)*Uplink Flow Priority***:** The uplink *Priority* is computed by the *heuristic function* defined as the average sum of the four distributions, *H-Distribution, R-Distribution,* E*-Distribution* and *L-Distribution* as formulated in Eq. (10). The four probability *distributions* are considered as four forces controlling the forwarding process at the same time. Each force has a different impact on the forwarding process, so the greater the force, the greater the influence on the forwarding process. The *Priority* value $\tilde{P}_{i,j}$ indicates the *probability* of transmitting the data packet from the sender n_i to the receiver n_j in the flow entries (Table II).

$$\begin{aligned}
&\tilde{P}_{i,j} \\
&= \left(\tilde{H}_{i,j} + \tilde{R}_{i,j} + \tilde{L}_{i,j} + \tilde{E}_{i,j}\right) \Big/ 4 \\
&= \frac{1}{4}\left[\left(1 - e^{\bar{H}_{i,j}}\right) \sum_{k=1}^{m_i} \left(1 - e^{\bar{H}_{i,k}}\right) + e^{\bar{R}_{i,j}} \Big/ \sum_{k=1}^{m_i} e^{\bar{R}_{i,k}}\right. \\
&\quad \left. + \left(1 - e^{-\bar{L}_{i,j}}\right) \Big/ \sum_{k=1}^{m_i} \left(1 - e^{-\bar{L}_{i,k}}\right) + \bar{E}_{i,j} \Big/ \sum_{k=0}^{m_i} \bar{E}_{i,k}\right] \\
&\quad \times n_j \in \mathbb{N}_i
\end{aligned} \tag{10}$$

3) Downlink Flows: We propose two heuristic algorithms for the downlink. The first algorithm simply considers the downlink flows as the reverse of the uplink flows. The paths should be specified in the header of the packet. In the second algorithm, the paths are computed based on a heuristic function that considers three values, distance to the end node, the remaining energy and RSSI value. The location of the end node is specified in the header of the packet. The two heuristic algorithms are precisely clarified in the following subsections.

(1) Optimized Reverse Downlink (ORD): The paths from the *controller* to end nodes are locally computed in the *controller* rather than in each node. There are multiple reverse paths from the *controller* to each end node (e.g., see Figure 10, the reverse paths from the sink node n_0 to the end node n_{61}). To optimize the downlink selection, based on the paths computed in the initialization process, the *controller* constructs a *sub-graph* $G_i = (V_i, E_i)$ for the target end node n_t and applies both the *R-value* and the *L-value* based on the collected information in the *database*. The priority of each potential path is computed as follows. Let $P_t^b = \{n_b, \ldots, n_t\}$ be a path from the sink node n_b to the end node n_t. The ordered pair (n_i, n_j) represents one hop from n_i to n_j while the set $E(n_i)$ represents the adjacent edges of n_i. The priority of the path P_t^b, denoted by $\tilde{P}_t^b$, is computed by Eq. (11) where $\tilde{R}_{i,j}$ and $\tilde{L}_{i,j}$ are computed by Eq. (4) and Eq. (6), respectively. The controller sorts the obtained potential paths according to their priority. Higher priority implies higher path utility.

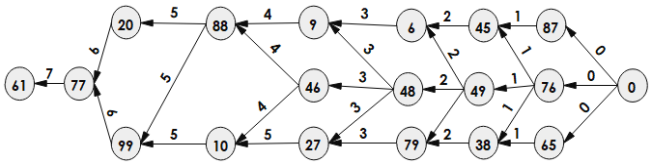

Fig. 10. The Sub-graph of the end node n_{61}. It shows the potential reverse paths from the sink n_0 to the node n_{61}(the network topology is depicted in Figure 5). Each path contains eight hops. The weight of the edge (link) indicates the *H-value* (obtained by ALGORITHM 1) while the vertex ID indicates the ID of the sensor node.

$$\tilde{P}_t^b = \prod_{\forall (n_i, n_j) \in P_t^b} \left(\left(\tilde{R}_{i,j} + \tilde{L}_{i,j}\right) \Big/ \sum_{\forall (n_i, n_k) \in E(n_i)} \left(\tilde{R}_{i,k}.\tilde{L}_{i,k}\right)\right) \tag{11}$$

(2) Location-based Downlink (LD) Algorithm: considering the values which have an influence on the selection of the path from the *controller* to the end node, the priority of the downlink is computed as an average sum of three values, *R-value*, *D-value* (the distance to the end node) and *L-value*. The *R-Distribution* and *L-Distribution* are explained in the previous subsection while the *D-Distribution* is explained below.

(a) Distance to End Node (D-Distribution): The goal of *D-Distribution* is to allocate a higher priority for the relay nodes that are closer to the end node n_t. The sender node n_i expresses the *D-values* of its neighbor nodes $\forall n_j \in \mathrm{N}_i$ as a vector, $\mathrm{D}_i = \{D_{i,1,t}, D_{i,2,\mathrm{t}}, \ldots, D_{i,m_i,t}\}$ where $D_{i,j,t} = D_{j,t}$ denotes the *Euclidean distance* from the relay node n_j to the end node n_t, expressed in Eq. (12). D_i is normalized into $\bar{D}_i = \{\bar{D}_{i,1,\mathrm{t}}, \bar{D}_{i,2,\mathrm{t}}, \ldots, \bar{D}_{i,m_i,t}\}$ by Eq. (13) where φ_i is the communication radius of n_i. Finally, the *D-Distribution* is defined by the mass function Eq. (14), denoted by $\tilde{D}_i = \left(\tilde{D}_{i,1,\mathrm{t}}, \tilde{D}_{i,2,\mathrm{t}}, \ldots, \tilde{D}_{i,m_i,t}\right)$. In Eq. (14), $\gamma_D \geq 0$ is called the D-exponent. Greater γ_D offers a higher probability for the nodes, which have smaller *D-value* (closer to the end node) to be selected as candidates. Figure 9 shows the impact of γ_D on the *D-Distribution* when n_i has 5 neighbor nodes $\mathrm{N}_i = \{n_1, n_2, \ldots, n_5\}$. The distance from n_i to n_t is *225m* while the distances from the neighbor nodes to n_t are expressed by $D_i = \{D_{i,1,t} = 210, D_{i,2,\mathrm{t}} = 220, \ldots, D_{i,5_i,t} = 250\}$. The value of γ_D varies from 0 to 1. The curve of the *D-Distribution* shows that as the value of *D-exponent* (γ_D) becomes larger, the probability of selecting the nodes closer to n_t becomes higher.

$$D_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{12}$$

$$\bar{D}_{i,j,t} = \left(\frac{D_{j,t}}{D_{i,t} + \varphi_i}\right), n_j \in \mathbb{N}_i \tag{13}$$

$$\tilde{D}_{i,j,t} = (1 - \sqrt{\bar{D}}_{i,j,t})^{1+\gamma_D} \sum_{k=0}^{m_i} \left(1 - \sqrt{\bar{D}}_{i,k,t}\right)^{1+\gamma_D}, n_j \in \mathbb{N}_i \tag{14}$$

(b)Downlink Flow Priority: The three distributions (quantities), *R-Distribution*, *D-Distribution* and *L-Distribution*, are considered as three forces that control the downlink flow. The impacts of these forces are tuned by the exponential parameters such that a greater value of the exponent increases the impact of the force. Consequently, to counterbalance the impact of these forces, the *Priority* of the downlink is computed as an average sum of these three forces, formulated in Eq. (15).

$$\begin{aligned}\tilde{P}_{i,j,t} &= \left(\tilde{R}_{i,j} + \tilde{D}_{i,j,t} + \tilde{L}_{i,j}\right)\Big/3 \\ &= \frac{1}{3}\left[\left(1-e^{\bar{H}_{i,j}}\right)\Big/\sum_{k=1}^{m_i}\left(1-e^{\bar{H}_{i,k}}\right)\right. \\ &\quad + \left(1-\sqrt{\bar{D}}_{i,j,t}\right)^{1+\gamma_D}\sum_{k=0}^{m_i}\left(1-\sqrt{\bar{D}}_{i,k,t}\right)^{1+\gamma_D} \\ &\quad \left. + \left(1-e^{-\bar{L}_{i,j}}\right)\Big/\sum_{k=1}^{m_i}\left(1-e^{-\bar{L}_{i,k}}\right)\right], n_j \in \mathbb{N}_i \quad (15)\end{aligned}$$

4) Intra-Link Flows: In *hierarchical-based* routing e.g., clustered-based, the members hand the collected data to the CH which then delivers it further to the sink directly or by another CH (multi-hops). MINI-FLOW instantiates the intra-link flows to support the data traffic within the data plane devices. The initialization of the network in *hierarchical-based* routing starts at a CH where the *H-value* is set to 0. The CH starts the initialization phase by sending *beacons*. The nodes that receive the *beacon* within the cluster execute ALGORITHM 1 and further send their new *beacons*. The uplink/downlink flows from/to the CH to/from the sink in *hierarchical-based* routing are computed exactly as in *flat-based* routing. Similarly, the intra-link flows from the end node to the CH can be computed based on a heuristic function that combines four values, the number of hops to the CH, the *Received Signal Strength* (RSS), the direction towards the CH and the remaining energy. Similar to the downlink, the intra-link from CH to the end node is computed as an average sum of the three values, *R-value*, *D-value* (the distance from CH to the end node) and *L-value*.

E. Candidates and Coordination

Commonly, the MAC layer coordinates the active/sleep states by one of the two approaches: synchronous or asynchronous [21]. In the synchronous approach, the MAC protocol synchronizes the wake-up intervals and active periods such that multiple nodes wake up at the same time and have the same active period. While in the asynchronous approach, the nodes randomly define their wake up intervals and active periods [23]. Sender and receiver nodes should be involved in an active state in order to undertake the communication task. This roughly runs in three steps [22]. First, the Network Layer determines the next hop (forwarder) using a predefined routing metric e.g., link quality, number of hops, etc. Second, the MAC Layer waits for the intended forwarder to wake up and receive the packet. Third, the forwarder sends back an ACK to the sender, acknowledging the received packet. These requirements are neglected in *OpenFlow* wired domains. Coordination among candidates is an important mechanism to avoid the negative impact of packets duplication. When more candidates (neighboring nodes in active mode) have received the preamble packet, the next step is to ensure that the data packet will be sent to one candidate. The *Candidates Coordination* (CC) selects the node, which has the maximum *Priority* in Table II. The number of candidates in uplink and downlink are controlled by network density in a distributed manner.

Uplink-Flow Action: The action of the uplink flow, which must be executed on the packets (i.e., drop, forward), is defined by Eq. (16) where $\vec{C}_i$ is the threshold of uplink candidates that are allowed to be selected by the node n_i.

$$\vec{Act}\,(n_i) = \begin{cases} Forward\vec{C}_i \leq \sqrt{m}_i; \dfrac{\tilde{P}_{i,j}}{\sum_{k=0}^{m_i}\tilde{P}_{i,k}} \geq \dfrac{1}{m_i}; \forall n_j \in \mathbb{N}_i \\ Drop\vec{C}_i > \sqrt{m}_i; \dfrac{\tilde{P}_{i,j}}{\sum_{k=0}^{m_i}\tilde{P}_{i,k}} < \dfrac{1}{m_i}; \forall n_j \in \mathbb{N}_i \end{cases} \quad (16)$$

Down-Flow Action: The downlink action, which must be executed on the packets (i.e., drop, forward), is defined by Eq. (17) where $\overleftarrow{C}_i$ is the threshold value of downlink candidates that are allowed to be selected by the node n_i.

$$\overleftarrow{Act}\,(n_i) = \begin{cases} Forward\overleftarrow{C}_i \leq \sqrt{m}_i; \dfrac{\tilde{P}_{i,j,t}}{\sum_{k=0}^{m_i}\tilde{P}_{i,k,t}} \geq \dfrac{1}{m_i}; \forall n_j \in \mathbb{N}_i \\ Drop\overleftarrow{C}_i > \sqrt{m}_i; \dfrac{\tilde{P}_{i,j,t}}{\sum_{k=0}^{m_i}\tilde{P}_{i,k,t}} < \dfrac{1}{m_i}; \forall n_j \in \mathbb{N}_i \end{cases} \quad (17)$$

F. System Parameters

The flows are controlled through the five system parameters, *H-exponent*γ_H, *R-exponent* γ_R, E-*exponent* γ_E, *L-exponent* γ_L and *D-exponent* γ_D. The uplink flows are controlled through the four parameters, *H-exponent* γ_H, *R-exponent* γ_R, E-*exponent* γ_E and *L-exponent* γ_L while the downlink flows are controlled through the three parameters *D-exponent* γ_D, *R-exponent* γ_R and *L-exponent* γ_L.

To maximize the network lifetime, the value of *L-exponent* γ_L is set to be greater than the values of the other parameters in both uplink and downlink flows. *L-exponent* γ_L is designed to avoid choosing the nodes with low-energy in each transmission session. Greater value of *L-exponent* γ_L increases the network lifetime, but it enforces the packets to travel via longer paths as it prefers the nodes with high-energy. In order to avoid loops, the values of *H-exponent* γ_H and *D-exponent* γ_D should continuously set to be greater than the value of *R-exponent* γ_R. *R-exponent* γ_R induces the packets to be transmitted through shorter transmission distance in each transmission session (single hop) without considering the direction towards the sink

nor considering the energy level of the relay nodes. *H-exponent* γ_H enforces the packet to reach the sink via the minimum number of hops without considering the energy level of the relay nodes. *D-exponent* γ_D and *E-exponent* γ_E induce the packets to travel towards the direction of the destination node via a shorter routing distance without considering the energy level of the relay nodes nor considering the transmission distance in each transmission session. The values of system parameters are strongly related to the network density and should be set in accordance with the application specifications and requirements. Considering the duty cycle of the nodes, to counterbalance between the sender waiting time and the duplicate packet, the default values for uplink and downlink flows are obtained by Eq. (18) and Eq. (19), respectively. We assign higher priority for energy parameter in both uplink and downlink flows.

$$\vec{Par}(n_i) = \begin{cases} \gamma_L = (1+\sqrt{m_i}) \\ \gamma_H = (1+\sqrt{m_i})/2\pi \\ \gamma_E = (1+\sqrt{m_i})/2\pi \\ \gamma_R = (1+\sqrt{m_i})/3\pi \end{cases} \tag{18}$$

$$\overleftarrow{Par}(n_i) = \begin{cases} \gamma_L = (1+\sqrt{m_i}) \\ \gamma_D = (1+\sqrt{m_i})/2\pi \\ \gamma_R = (1+\sqrt{m_i})/3\pi \end{cases} \tag{19}$$

G. Flow Update

The priority value should be updated periodically based on the remaining energy of nodes. Otherwise, higher priority nodes will exhaust their energy earlier. The *controller* updates the flows based on the *statistics* reported by the end nodes. In this work, the *controller* updates the flows for the end node each time the node loses 5% of its energy.

V. Analysis

The performance analysis of MINI-FLOW is provided in the supplementary file.

VI. Simulation and Results

A. Simulation Settings

We evaluated the performance of the proposed protocol using the simulator [28], [29] which is developed in *visual studio 2015 (C# WPF)*. The documentations of the simulator are available online in the link.[4] We assumed that the nodes are randomly deployed in a fair distribution, and the sink node is placed in the center of the sensing field. For simplicity, at the simulation level, the controller is located in the sink node. We utilized one sink and one controller. Each node runs *BOX-MAC*. The nodes have the same active (1s)/sleep (2s) periods. Each node has a battery of 0.5 Jouls and consumes energy according to the *First Order Radio Model* as in [20]. We employed *free space propagation model* that assumes the ideal propagation condition by representing the communication range as a circle around the transmitter. The packet generation rate is adjusted to 1/0.1s (one packet in 0.1 second). In each 0.1s, a random node generates a packet and routes it towards the sink. The size of the data packet is 1024bits while the size of the control packet is 512bits.

TABLE IV
Simulation Parameters

(A)

Parameter	Value
Packet Rate	1/0.1 s
Simulation Time	480 s
Active Time	1 s
Sleep Time	2 s
Number of Nodes	100

(B)

Parameter	Value
Packet Rate	1/0.1 s
Simulation Time	300 s
Active Time	1 s
Sleep Time	2 s
CR	80 m

(C)

Parameter	Value
Packet Rate	1/0.1 s
Simulation Time	300 s
Sleep Time	2 s
CR	80 m
Number of Nodes	100

Comparisons to the State-of-the-art: Two well-known WSN protocols are selected:

(1) **ORR**, a load-balanced opportunistic routing protocol [21]. ORR runs in two steps. Step 1: the optimal number of candidates is computed based on a cost estimation function, which is derived from duty-cycle and network density. Step 2: the residual energy is used during the selection of forwarders [21]. During the simulation experiments, we set the *alpha* parameter to 1.0. The authors of ORR evaluated ORR by varying the *alpha* parameter from 0.0 to 4.0.

(2) **ORW**, a practical opportunistic routing protocol [22]. ORW utilized the *Expected Duty Cycled* wakeups (EDC) as a global routing metric. EDC computes the expected number of wakeups until the packet reaches its intended destination (via single hop or multiple hops). Explicitly, EDC defines the number of wakeups as the sum of three terms. First, the expected time needed to forward the packet (one hop). Second, the time needed to forward the packet from the selected forwarder to the intended destination. Third, a small constant accounting the cost of forwarding the packet. These two protocols are implemented using *C# WPF Visual Studio 2015* in the link.[5]

B. Simulation Results

Evaluation Metrics: We considered the following *evaluation metrics*. *(1) Energy Consumption*(**EC**): The total energy consumption required to deliver the packets from the sources to the destination at a given simulation time. The energy unit is *Joule*. **(2)***Average Number of Redundant Packets (AxRP)***:** When the sender node broadcasts a packet, all its awoken candidates will hear the packet and receive it. However, after coordination, one candidate will be selected to forward the packet while the other candidates will abort the received packet. The aborted packets are considered as redundant. AxRP is the average number of the aborted packets at a given simulation time. **(3)***Average Routing Distance Efficiency (RDE)***:** The *Routing Distance Efficiency* (RDE), measures the ratio of the *Euclidean Distance* $d_{s,b}$ (between the sources to the sink) to the actual *Routing Distance* d_b^s for a data packet that travelled along the path. $RDE = d_{s,b}/d_b^s$. **(4)***Average Waiting Times (AWT)***:** The average number of times the sender

[4]https://github.com/howbani/WSNSIM

[5]https://github.com/howbani/WSNSIM

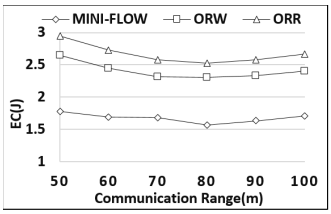

Fig. 11. Energy consumption vs. varying communication range.

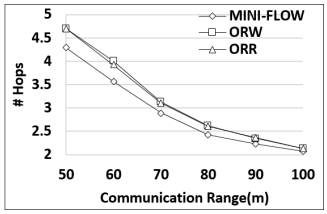

Fig. 12. Average number of hops vs. varying communication range.

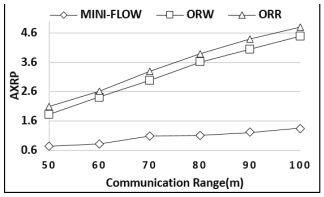

Fig. 13. Average number of redundant packets (AxRP) vs. varying communication range.

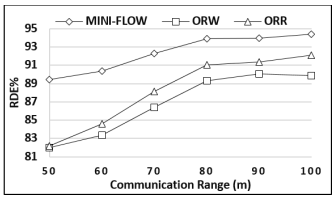

Fig. 14. Average Routing Distance Efficiency (RDE) vs. varying communication range.

waits until one of its potential forwarders wakes up and receives the packet at a given simulation time. **(5)***Network Lifetime***:** The time (in seconds) from the starting moment of the simulation to the moment that the first node runs out of energy.

1) Varying the Communication Ranges: This section evaluates the performance of the network varying the communication range. The communication range (CR) varies from 50m to 100m. The default parameters for this experiment are listed in Table IV(A).

a) Energy Consumption: The results of evaluating the energy consumption (EC) are depicted in Figure 11. Several observations are concluded. In the case when the communication range is lower than 80m, the energy consumption decreases as the range of communication increases. The reason behind that is the number of hops decreases as the range of communication gets larger (see Figure 12). However, greater communication range increases the number of candidates for each node, which in turn increases the number of redundant packets (AxRP) and maximizes the energy consumption (see Figure 13). In the case when the communication range is greater than 80m, the energy consumption increases as the range of communication increases. The reason behind that is the dramatic increment in the negative effect of the duplicate packets AxRP generated from multiple receivers. MINI-FLOW achieved better energy saving than ORR and ORW due to the following two reasons. First, the number of hops is controlled by the *H-distribution* such that the packets are mostly routed via the paths with the minimal number of hops. Second, the MINI-FLOW forwarders are well controlled by Eq. (16) and Eq. (17) such that few of the nodes are allowed to be selected as forwarders, which in turn reduces the negative effect of the duplicate packets generated from multiple receivers. Although ORR calculates the maximum number of forwarders, it incurs a big amount of energy consumption during the calculation. Moreover, ORR periodically updates the EDC metric to achieve energy balancing among the nodes. However, its EDC metric is recursively calculated in the sink. This means that in each update phase, each node sends its current energy state to the sink that re-computes the EDC metric and sends back the new EDC metric (*Expected Duty Cycled*) to each node. Obviously, this hinders the network's performance and depletes its restricted resources. ORW computes its EDC metric in the initializing process without considering the residual energy of the nodes. Both ORR and ORW suffer from the negative impact of duplicate packets.

b) The Number of Hops: The results of evaluating the number of hops are depicted in Figure 12, which indicate that the number of hops decreases as the range of communication increases since fewer numbers of hops means shorter *Routing Distance*, which in turn costs less energy. The average number of hops in ORW is comparable to ORR since both of them utilize the *Expected Duty Cycled* wakeups (EDC) as a global routing metric.

EDC utilizes the expected number of wakeups until the packet reaches its intended destination (through single hop or multiple hops). ORR is approximately similar to ORW with a minor difference. ORR removes the third term in ORW (i.e., removes the forwarding-cost term from EDC) and adds the residual energy parameter to the first term of EDC. In MINI-FLOW, the number of hops is controlled by the *H-distribution* such that the packets are regularly routed through minimum hops.

c) Redundant Packets: The results of evaluating the average number of redundant packets (AxRP) are depicted in Figure 13. AxRP measures the negative impact of duplicate packets generated from multiple receivers. We concluded that AxRP increases as the range of communication increases. Greater communication range inevitably imposes the node to select more candidates, which in turn increases the probability of multiple receivers. MINI-FLOW is designed to reduce the number of duplicate packets by Eq. (16) and Eq. (17) in each node such that the number of candidate nodes is controlled by the heuristic function of uplink flows and the heuristic function of downlink flows.

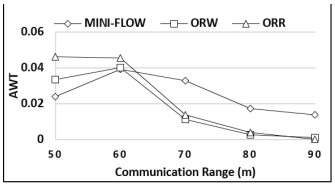

Fig. 15. Average waiting-time vs. varying communication range.

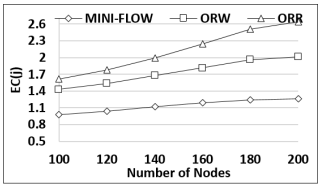

Fig. 16. Energy consumption vs. varying number of nodes.

d) Routing Distance Efficiency: The results of evaluating the average *Routing Distance Efficiency* (RDE) are depicted in Figure 14. We concluded that for a path, the RDE gets higher as the communication range increases since the packets travel through a smaller number of hops. MINI-FLOW shows higher RDE than ORR and ORW because the packets travel through the shortest path to the destination. ORR and ORW selected more forwarder nodes, which in some cases selected the active next hop node even if it is not towards the direction of the destination. This obviously increases the *Routing Distance* and consequently reduces the RDE.

e) Average Waiting Times: The results of evaluating the *Average Waiting Times (AWT)* are shown in Figure 15, from which we concluded that for each path, the AWT gets smaller as the communication range increases. Greater communication range drives the node to employ more candidate nodes, which in turn reduces the waiting time. In fact, increasing the number of candidate nodes has two different impacts, positive and negative. The negative impact is the increment in duplicate packets, which raises the consumption of energy. In contrast, the positive impact is the reduction in the number of waiting times such that the sender needs not to wait for a specific candidate to wake up and receive the packet. These two contradictory impacts are reflected by the nature of asynchronous duty-cycled WSN. From the network layer's point of view, WSN designers utilize multiple candidate nodes to reduce the waiting times. However, too many candidate nodes may simultaneously wake up, causing the duplication problem. MINI-FLOW is designed to counterbalance the two impacts by trading off energy and waiting time.

2) Varying the Number of Nodes: This section evaluates the performance of the network varying the number of nodes. The default parameters are listed in Table IV(B). The number of nodes varies from 100 to 200.

a) Energy Consumption: Figure 16 shows the results of evaluating the energy consumption by varying the number of nodes. In view of that, we concluded that the energy consumption increases as the number of nodes increases. Increasing the number of nodes either leads to a high density or to a large size of the network. High network density increases the number of candidates assigned for each node, which in turn generates more *Redundant Packets* (AxRP) and consumes more energy. On the other hand, the large size of the network increases the number of hops and forces the packet to travel through longer paths, which in turn consumes more energy. MINI-FLOW outperforms ORR and ORW for the same reasons explained in Subsection 6.2.1(a).

b) Number of Hops: The results are provided in the supplementary file.

c) Redundant Packets: The results are provided in the supplementary file.

d) Average Waiting Times: The results are provided in the supplementary file.

3) Varying the Wake-Up Intervals: This section evaluates the performance of the network varying the wake-up intervals. The default parameters are listed in Table IV(C). The active period varies from 1s to 5s. The results are provided in the supplementary file.

4) Network Lifetime: MINI-SDN showed longer lifetime than ORR and ORW. The simulation details are provided in the supplementary file.

5) Impact of System Parameters: The results are provided in the supplementary file.

6) Controller to End Nodes Overhead: This subsection evaluates the communication overhead of updating the flows varying network size. The results are provided in the supplementary file.

VII. Conclusion

This article investigates the software-defined wireless sensor networks and proposes an architecture and incorporated protocol to enhance the efficiency and scalability. The proposed architecture, MINI-SDN, is an attempt towards separating the control logic from the sensor nodes/actuators. In fact, complete separation of the two planes may degrade the performance of WSNs since such networks are intrinsically distributed and data centric. We proposed a data-centric protocol called *MINI-FLOW* that supports centralized and distributed routing mechanisms. *MINI-FLOW* presents three routing mechanisms, the uplink, downlink and intra-link. The uplink routing is based on a heuristic function that combines four values, the number of hops to the sink, the distance towards the sinks, the *Received Signal Strength* and the remaining energy. Regarding the downlink flows, we proposed two heuristic algorithms. The first algorithm, *Optimized Reverse Downlink*, considers the reverse paths of uplink flows. The second algorithm, *Location-based Downlink,* computes the paths based on a heuristic function that combines three values, distance to the end node, the remaining energy and RSS value. The experimental results show our architectureand protocol scale well with both network size and density.

References

[1] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, Feb. 2013.

[2] D. Zeng, T. Miyazaki, S. Guo, T. Tsukahara, J. Kitamichi, and T. Hayashi, "Evolution of software-defined sensor networks," in *Proc. IEEE 9th Int. Conf. Mobile Ad-hoc Sensor Netw.*, Dec. 2013, pp. 410–413.

[3] 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 Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 513–521.
[4] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling SDNs," in *Proc. Eur. Workshop Softw. Defined Netw.*, Oct. 2012, pp. 1–6.
[5] I. T. Haque and N. Abu-Ghazaleh, "Wireless software defined networking: A survey and taxonomy," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2713–2737, 3rd Quart., 2016.
[6] M. Ndiaye, G. Hancke, and A. Abu-Mahfouz, "Software defined networking for improved wireless sensor network management: A survey," *Sensors*, vol. 17, no. 5, p. 1031, May 2017.
[7] K. M. Modieginyane, B. B. Letswamotse, R. Malekian, and A. M. Abu-Mahfouz, "Software defined wireless sensor networks application opportunities for efficient network management: A survey," *Comput. Electr. Eng.*, vol. 66, pp. 274–287, Feb. 2018.
[8] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A survey on software-defined wireless sensor networks: Challenges and design requirements," *IEEE Access*, vol. 5, pp. 1872–1899, 2017.
[9] T. Luo, H.-P. Tan, and T. Q. S. Quek, "Sensor OpenFlow: Enabling software-defined wireless sensor networks," *IEEE Commun. Lett.*, vol. 16, no. 11, pp. 1896–1899, Nov. 2012.
[10] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in *Proc. 27th Biennial Symp. Commun. (QBSC)*, Jun. 2014, pp. 71–75.
[11] D. Moss and P. Levis, "BoX-MACs: Exploiting physical and link layer boundaries in low-power networking," *Comput. Syst. Lab. Stanford Univ.*, vol. 64, no. 6, pp. 116–119, 2008.
[12] Y. Chapre, P. Mohapatra, S. Jha, and A. Seneviratne, "Received signal strength indicator and its analysis in a typical WLAN system (short paper)," in *Proc. 38th Annu. IEEE Conf. Local Comput. Netw.*, Oct. 2013, pp. 304–307.
[13] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.
[14] A. Hawbani *et al.*, "GLT: Grouping based location tracking for object tracking sensor networks," *Wireless Commun. Mobile Comput.*, vol. 2017, pp. 1–19, Oct. 2017.
[15] M. Botta and M. Simek, "Adaptive distance estimation based on RSSI in 802.15. 4 network," *Radioengineering*, vol. 4, pp. 1162–1168, Oct. 2013.
[16] T. Miyazaki *et al.*, "A software defined wireless sensor network," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, 2014, pp. 847–852.
[17] J. Kang, D. Kim, and Y. Kim, "RSS self-calibration protocol for WSN localization," in *Proc. 2nd Int. Symp. Wireless Pervas. Comput.*, 2007, pp. 1–5.
[18] B. Pfaff *et al.*, "OpenFlow switch specification," in *Proc. ACM SIGCOMM*, Feb. 2011, pp. 1–56.
[19] V. Nascimento *et al.*, "Filling the gap between software defined networking and wireless mesh networks," in *Proc. 10th Int. Conf. Netw. Service Manage. (CNSM) Workshop*, Nov. 2014, pp. 451–454.
[20] A. Hawbani, X. Wang, S. Karmoshi, L. Wang, and N. Husaini, "Sensors grouping hierarchy structure for wireless sensor network," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 8, Aug. 2015, Art. no. 650519.
[21] J. So and H. Byun, "Load-balanced opportunistic routing for duty-cycled wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 7, pp. 1940–1955, Jul. 2017.
[22] E. Ghadimi, O. Landsiedel, P. Soldati, S. Duquennoy, and M. Johansson, "Opportunistic routing in low duty-cycle wireless sensor networks," *ACM Trans. Sensor Netw.*, vol. 10, no. 4, pp. 1–39, Jun. 2014.
[23] A. Boukerche and A. Darehshoorzadeh, "Opportunistic routing in wireless networks: Models, algorithms, and classifications," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 1–36, Jan. 2015.
[24] P. Santi, "Topology control in wireless ad hoc and sensor networks," *ACM Comput. Surv.*, vol. 37, no. 2, pp. 164–194, 2005.
[25] W. Xiang, N. Wang, and Y. Zhou, "An energy-efficient routing algorithm for software-defined wireless sensor networks," *IEEE Sensors J.*, vol. 16, no. 20, pp. 7393–7400, Oct. 2016.
[26] D. Zeng, P. Li, S. Guo, T. Miyazaki, J. Hu, and Y. Xiang, "Energy minimization in multi-task software-defined sensor networks," *IEEE Trans. Comput. 64, no.*, vol. 11, no. 2015, pp. 3128–3139.
[27] G. Li, S. Guo, Y. Yang, and Y. Yang, "Traffic load minimization in software defined wireless sensor networks," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1370–1378, Jun. 2018.
[28] A. Hawbani *et al.*, "Zone probabilistic routing for wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 18, no. 3, pp. 728–741, Mar. 2019.
[29] A. Hawbani, X. Wang, Y. Sharabi, A. Ghannami, H. Kuhlani, and S. Karmoshi, "LORA: Load-balanced opportunistic routing for asynchronous duty-cycled WSN," *IEEE Trans. Mobile Comput.*, vol. 18, no. 7, pp. 1601–1615, Jul. 2019.

Ammar Hawbani received the B.S., M.S., and Ph.D. degrees in computer software and theory from the University of Science and Technology of China (USTC), Hefei, China, in 2009, 2012, and 2016, respectively. From 2016 to 2019, he worked as a Post-Doctoral Researcher with the School of Computer Science and Technology, USTC. He is currently an Associate Professor of networking and communication algorithms with the School of Computer Science and Technology, USTC. His research interests include the Internet of Things (IoT), WSNs, WBANs, WMNs, VANETs, and SDN.

Xingfu Wang (Member, IEEE) received the B.S. degree in electronic and information engineering from Beijing Normal University, China, in 1988, and the M.S. degree in computer science from the University of Science and Technology of China in 1997. He is currently an Associate Professor with the School of Compute Science and Technology, University of Science and Technology of China. His current research interests include information security, data management, and WSN.

Liang Zhao received the Ph.D. degree from the School of Computing, Edinburgh Napier University, in 2011. He is currently a Lecturer with Shenyang Aerospace University, China. Before joining Shenyang Aerospace University, he worked as an Associate Senior Researcher at Hitachi (China) Research and Development Corporation from 2012 to 2014. His research interests include WMNs, VANETs, and SDN.

Ahmed Al-Dubai (Senior Member, IEEE) received the Ph.D. degree in computing from the University of Glasgow in 2004. He is currently a Professor of networking and communication algorithms with the School of Computing, Edinburgh Napier University, U.K. His research interests include communication algorithms, mobile communication, the Internet of Things, and future Internet.

Geyong Min (Member, IEEE) received the Ph.D. degree in computing science from the University of Glasgow, Glasgow, U.K., in 2003. He is currently a Professor of high performance computing and networking with the Department of Mathematics and Computer Science, University of Exeter, Exeter, U.K. His current research interests include future Internet, wireless communications, multimedia systems, information security, high-performance computing, ubiquitous computing, modeling, and performance engineering.

Omar Busaileh received the B.S. degree in electronic and information engineering from the Hefei University of Technology. He is currently pursuing the master's degree with the School of Computer Science and Technology, USTC. His research interests mainly include WSNs, WBANs, and SDN.