Networking Functions for Wireless Sensor Network Applications: an SDN-based Approach

Melek Charfi, Alexandre Mouradian, Véronique Vèque

Networking Functions for Wireless Sensor Network Applications: an SDN-based Approach

Melek Charfi*, Alexandre Mouradian†, Véronique Vèque‡
Université Paris-Saclay, CNRS, CentraleSupélec,
Laboratoire des Signaux et Systèmes, 91190, Gif-sur-Yvette, France
*melek.charfi@u-psud.fr, †alexandre.mouradian@u-psud.fr, ‡veronique.veque@u-psud.fr

***Abstract*—Wireless Sensor Networks (WSN) are considered to be a key enabler towards the Internet of Things (IoT) concept and one of its main underlying technologies. Applications built on top of IoT infrastructures such as WSNs share a set of common functions that concern not only data management such as data collection, dissemination or aggregation, but also classical networking functions, such as routing or addressing. Even though application deployment is becoming more feasible, it can still be a complex process especially in the presence of vertical solutions. WSNs should, therefore, be more flexible in order to facilitate the development of efficient applications. A promising networking paradigm which can help achieve such a requirement is the Software-Defined Networking (SDN) concept which decouples the network data plane from the control plane. To allow multiple applications to share and reuse common networking functions, we propose to implement them as SDN reusable building blocks and to instantiate and customize them through SDN rules. On the other hand, we provide a rigorous definition of these functions and of their relationships in order to make them reusable to efficiently develop the applications. We consider the smart city as a typical WSN/IoT domain of applications. Our approach aims at facilitating the development, use and management of such functions in a WSN and thus of IoT applications.**

***Index Terms*—Internet of Things, Wireless Sensor Networks, Smart city, Software-Defined Networking, Networking functions**

I. Introduction

In recent years, the study and use of Wireless Sensor Networks (WSNs) have become increasingly popular as they are considered to be the main enabler towards the concept of Internet of Things (IoT) [1]. The IoT concept aims at creating an ecosystem of connected objects using various types of WSNs depending on the application requirements such as Low-Power Wide Area Networks (LPWAN), Wireless Personal Area Networks (WPAN) or Wireless Body Area Networks (WBAN) [2]. The ease of deployment and the low cost of these WSNs have made their use more and more popular. Today, we find WSNs in a wide variety of IoT applications ranging from the industrial and security domains such as the process automation and video surveillance applications, to applications of personal use like home automation and medical supervision. However, in the last few years, WSNs have become famously associated with the IoT major application domain of smart cities, and increasingly popular with applications such as transportation, environment monitoring, water and gas metering, smart parking, etc [3]. These applications have in common the data collection aspect because their main objective is to measure different physical phenomena and communicate these data to a collector sink. The early adoptions of WSNs were mostly application-specific deployments leading to various vertical systems tightly coupled on both the sensing and the communication planes of the network. Thus, the application is the only user of the WSN which is strongly coupled with the sensing and communication capabilities of the nodes. It is therefore necessary to deploy as many WSNs as there are applications even when they cover the same area and could share a single infrastructure. Much efforts have been put into loosening the vertical approach of deploying WSNs and allowing applications to share the infrastructure. The first step towards this goal was to deploy multiple and different types of sensors and propose integrated solutions where applications can use the infrastructures sensing capabilities to offer various services [4]. However, WSN nodes are still coupled with the vendor's communication plane which makes the network control and management difficult, and imposes the reprogramming of the nodes (using a vendor-specific proprietary process). This constraint imposed by vertical solutions (Fig. 1 (left)) makes the development and management of efficient applications a difficult task, especially when many IoT applications share the same underlying functions to perform their tasks regardless of the service they offer.

The wide variety of applications for the smart city raises the issue of interoperability as different applications have different requirements [5]. These applications, built on top of an IoT environment such as a WSN, often share a set of functions that are necessary and fundamental for their development and support. At the network level, these basic functions are essential to enable distributed applications over a WSN-based IoT network, hence we refer to them as "networking functions". Some of these functions are traditional functions encountered in classical IP networks such as routing, forwarding or addressing, and some others are IoT-specific. For instance, in WSN infrastructures, functions such as aggregation, dissemination, or clustering are used very often and should be provided in a middleware layer. Moreover a further key point to facilitate the task of building efficient applications consists in simplifying the implementation of these functions which results in a better control and management of the network.

Therefore, a more flexible solution should be considered to achieve a practical and powerful method for implementing IoT networking functions and making them more reusable and less

application-dependent. Software-Defined Networking (SDN) represents a promising networking paradigm that can address the above-stated issue by simplifying the network design and operation through vendor-neutral devices. The SDN paradigm is based on the physical separation of the network control plane from the data plane. The control plane represents the intelligent part of a networking device which is detached from its data plane and placed in a central entity that acts as the network brain [6]. Although the SDN concept was originally created for wired networks, we use its concept for managing the common networking functions used in WSN-based IoT applications. Recently, due to its increasing popularity, many works have been interested into applying the principles of SDN to the IoT and especially to WSNs [7]. The idea is to make the sensor devices SDN-capable by breaking the communication plane, which is classically coupled with the sensor, into a data plane (forwarding plane) and a logically centralized control plane as depicted in Fig. 1 (right).

We propose an SDN-based approach to implement common networking functions for WSN-based IoT applications through the smart city example. We propose to place these functions at the controller level to create a toolbox of reusable and interacting blocks with exposed interfaces, and convert their behavior into SDN rules and commands to be installed on the network nodes. We then test this approach by presenting a use case of some of these functions in a WSN-based data collecting application, and use a wireless SDN solution as an experimenting platform.

The rest of the paper is organized as follows: in Section II, we give some motivating examples on the use of the networking functions in WSN-based smart city applications and introduce our approach to implementing these functions using an SDN architecture. In Section III, we present, through two use cases of smart city applications, the implementation process and simulation of our networking functions toolbox. Finally, a conclusion is drawn in Section IV.

II. Networking functions for WSN

Many WSN-based IoT applications share a set of common networking functions regardless of the services they provide. Furthermore, these functions are dependent on each other when performing a certain task. To demonstrate the validity of our approach, we consider two smart city application scenarios which are the smart parking and the smart monitoring in order to better introduce the concept of the networking functions.

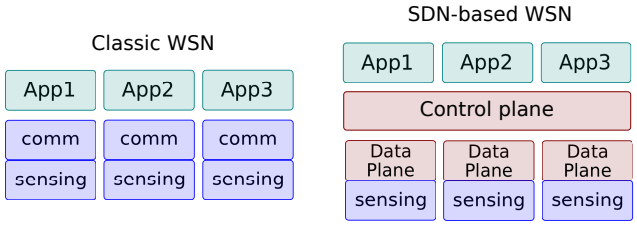

Fig. 1. Vertical versus SDN-based WSNs

A. Networking functions in smart city applications

The goal of a smart parking application is to retrieve information about parking occupancy and provide enough information to the users so they can obtain a parking space in a reasonable time (typically less than randomly looking for a spot), as illustrated in Fig. 2 (right part). Data collected by the sensors monitoring the parking spots is transmitted to a sink and then to a server where it is processed and made accessible to the users. Here, the main task of the WSN network is data gathering. Precisely, nodes collect the information about whether the parking spot is free or occupied. The most obvious function to perform such a task is routing the packets from the source node to the sink. For the application to be more useful, localization is needed to give an exact location of the free parking spaces. Another useful function to consider is the data aggregation which is the process of combining the data arriving from different sources of the network to eliminate redundancy, minimize the number of transmissions or reduce energy consumption. Data dissemination is another example of function used in this scenario. It represents the process of efficiently spreading data to a set of nodes in the network. For instance, the sink may disseminate data to nodes in order to change their status from free to reserved or to update some parameters.

The second application example is the smart environment monitoring depicted in Fig. 2 (left part). The goal of this application is to monitor environmental information such as temperature, humidity, pollution and noise levels, atmospheric pressure, etc. In this scenario, sensors are deployed in a geographic area in order to periodically collect the measured data. Here again, routing is the main function responsible for transmitting data from the sensor nodes to the network sink. In the case of temperature monitoring, it might be sufficient to collect the average temperature of a certain area, which can be done using an in-network aggregation function rather than re-transmitting every single measure by the sensors of that area. In this use case, localization can also be used to add relevance to the measured temperature value. In the case where the application needs to change parameters such as the collection period, a new value needs to be disseminated into one or more nodes of the network.

Both applications stated above share a set of networking

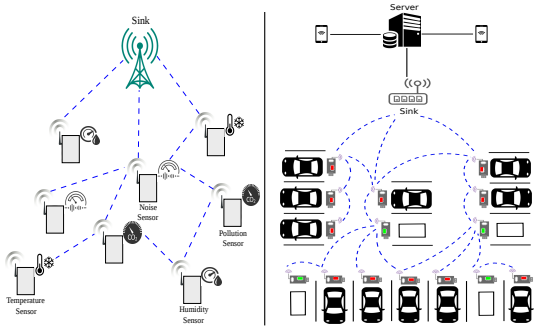

Fig. 2. Smart monitoring and smart parking applications

functions on which they base their services namely: routing, aggregation, dissemination, collection, and localization. They belong to a wider set of similar functions such as addressing, forwarding, clustering, topology discovery, etc. When running a smart city/IoT application on top of a WSN, these functions are seamlessly executed to ensure the final intended task of the application and sometimes can be dependent on each other.

B. Dependencies of networking functions

We consider the above-mentioned networking functions to be a crucial element when designing an IoT application based on a WSN. Therefore, it is important to facilitate the implementation of these functions as they represent the basic components to build efficient applications. Fig. 3 shows a basic Systems Modeling Language block diagram where the functions are represented as reusable building blocks interacting via dependency relationships. Here, a dependency relationship signifies that a block (a modular unit that describes the structure of a system or element) requires other blocks for its specification or implementation [8]. Abstracting these functions as reusable blocks means that each one has to define an input interface. When a function block is called by a user or by another block, the corresponding input parameters need to be provided in order to be properly executed. The implementation of this function block will then use the controller capabilities in order to generate the adequate rules and commands to be sent to the nodes. Each function block has its own input interface to be properly used by other blocks or directly called by the application developer, and therefore, abstracts the network lower levels. Table I provides the interfaces of the different function blocks. Here, the network discovery block takes as input all the control packets sent from the infrastructure nodes to collect network information (e.g. node's neighbors, battery levels, link states, location, etc), builds a global view on the network status, and provides it to other blocks. The path routing block depends on the network discovery block to acquire the network's global topology and provides a shortest path from a source to a destination. The data collection block is responsible for instructing the nodes on how and when to collect the data, and depends on the routing block to create the route paths from the data gathering nodes to reach the sink. This block can be called by passing inputs such as the list of collecting nodes and the data collection period. The Data dissemination block depends on the same network discovery block to generate the necessary rules for an efficient one-to-many (from the sink to the nodes) dissemination process, and takes as input the data to disseminate and the destination nodes. In a similar way, the data aggregation block depends on the network discovery block to determine some configuration parameters (e.g. the nodes that will aggregate the data), as well as the routing block to get information about the data flow in the network. Some aggregation parameters should be provided when this block is called such as the aggregation period, the aggregation technique or the aggregation nodes if needed. Once a function block is called with the right input parameters, it outputs the adequate SDN rules and commands to be installed on the network nodes. For this, the SDN rules and commands translation block are in charge to install SDN rules in the nodes using platform-specific packets.

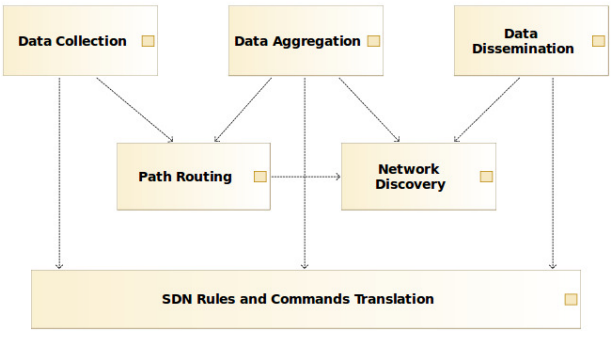

Fig. 3. Networking functions dependency.

C. SDN approach

While in classical vertical network architectures each one of the networking functions is implemented in specific layers and thus used via vertical interfaces, we take advantage of the SDN concept and propose to implement these functions in the network controller. Our goal is to use the SDN principles to propose a generic implementation of these functions rather than an application-dependent implementation. SDN offers the necessary flexibility that helps the process of developing the networking functions, and facilitates their reusability to easily control and manage the network. Furthermore, these functions, mostly implemented in the application middleware, will be placed in the control plane to profit from the SDN programmability of nodes as shown in Fig. 4. In this architecture, the IoT devices (e.g. WSN nodes) represent the infrastructure layer (data plane) and are SDN-enabled in order to communicate with the controller. The controller plays the role of a middleware where networking functions are implemented as a toolbox of interacting blocks (as depicted in Fig. 3). These function blocks expose their input interfaces

TABLE I
NETWORKING FUNCTIONS INTERFACES

Function block	Input	Output
Network Discovery	- Control packets	Network information graph
Path Routing	- Network topology - Source node - Destination node	Network path
Data Dissemination	- Network discovery - Destination nodes - Data to disseminate	Dissemination and forwarding rules
Data Aggregation	- Network topology - Aggregation method - Routing function	Forwarding and execution rules
Data Collection	- Collecting nodes - Collection period - Routing function	Configuration and forwarding rules
SDN rules translation	- SDN rules and commands	platform-specific packets

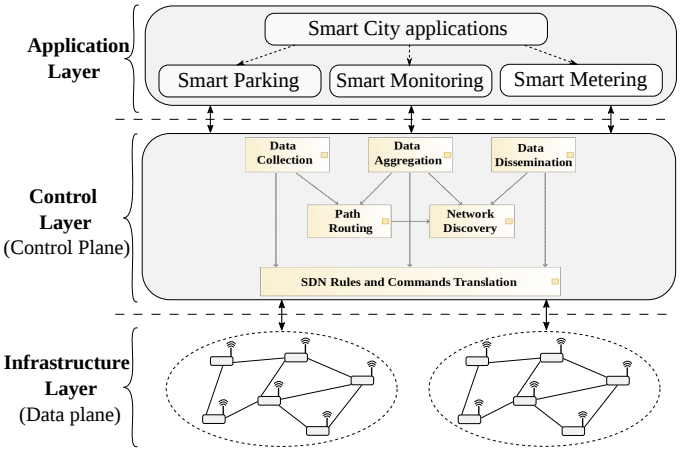

Fig. 4. SDN approach for networking functions.

for use by the application developers, and their outputs are converted into specific SDN rules and commands to be installed on the network nodes. At the top layer, different types of IoT applications can be developed based on the networking functions of the underlying layer.

III. Use case and Simulation

In order to further illustrate the benefits of our SDN-based approach for implementing networking functions, we consider two scenarios representing the smart parking and smart monitoring applications, and detail how they work and how some of the function blocks above-mentioned can be used at the controller.

A. Scenarios description

As described in Section II-A, the smart parking application aims at collecting information about the occupancy of parking spaces by sensing the presence of a car and relaying the information to a sink. The gathered information is then properly provided to the users who can reserve a parking spot beforehand or reduce the searching time otherwise. On the other hand, the smart monitoring scenario involves sensor nodes that will generate periodically the measured temperature and relay it to a sink. We consider the data collecting aspect of both applications in order to show how our toolbox can be reused to control and manage the network for each application requirements. Here, we consider a separate network for each scenario and we use our networking function blocks in order to execute the following tasks:

- Network discovery: the goal of this process, achieved by the Network discovery block, is to periodically discover the network status (e.g. topology, nodes battery levels, link states, etc) and provide it to other function blocks.
- Collecting data: both network devices periodically generate the data describing either the status of the parking space or the measured temperature. The data collection block will be used to program each network nodes to generate the data by passing to it the input parameters that specify the list of data generating nodes and the collection period.
- Routing data: the path routing block task finds the shortest path from each data generating node to the sink.
- Aggregating data: aggregating the data coming from the network nodes has the benefit of reducing the number of packets flowing in the network. An in-network aggregation of the sensed data is performed depending on the type of data, either lossless (e.g. data concatenation) or lossy (e.g. data average, maximum, minimum, etc). Therefore, some nodes will be programmed using this block with the input parameters specifying how and when data should be aggregated.
- Installing SDN rules: Once each function block is correctly called either by a user or by another block, a set of platform-specific packets are generated to install the generated rules and commands in the network nodes. This task is done using the SDN rules translation block.

Using the SDN-WISE platform [9], we develop a toolbox of functions at the controller level (Fig. 4) allowing a user to control the network by calling both the data collection and aggregation functions and passing specific inputs to achieve the described tasks.

B. Experimentation platform

The *SDN-WISE* [9], [10] platform provides a well-documented solution to implement SDN concepts for WSNs. It offers an extensible open-source solution as well as multiple deployment options and programming languages. *SDN-WISE* also includes several features that extend the basic principles of SDN and help carry out our approach. This software sets flexible rules specification to classify incoming packets and creating routing and forwarding strategies. A set of actions are applied once a match is detected. Some of these actions include forwarding a packet, dropping a packet, modifying a packet, turning off the node or executing a specific function in the node. The communication between the network nodes and the controller are established using a set of different packets that handle the control of the network and include: (1) *Request*/*Response* packets used when no match is found in the nodes flow table, (2) *Beacon*/*Report* packets which are control packets sent periodically by the nodes and help the controller maintain a global view of the network status, and (3) *Config* packets used for nodes programming like updating specific configuration parameters or remotely installing functions [11]. Moreover, *SDN-WISE* was proved to perform well compared with popular WSN communication technologies such as ZigBee and 6LoWPA,N and can out-perform them in some case [12].

We use Cooja [13] for simulating our scenarios based two WSNs. Cooja is an extensible and widely used simulation tool for WSNs. Although it was primarily designed for Contiki-OS sensors [14], it supports adding pure Java code nodes without any connection to Contiki, which is useful for networking level experimentations. This feature allows the integration of the *SDN-WISE* firmware to the Cooja nodes. The simulation

environment is depicted in Fig. 5. The Unit Disk Graph Medium (UDGM) is used as a radio medium where each node has a transmission range modeled as a disk and an interference range modeled as a bigger disk. No errors in packets reception or transmission are simulated in our experiments. Although Cooja's UDGM model does not accurately model a realistic medium, results confirm the expected functionality and feasibility of our networking functions implementation approach, which is the goal of our simulations.

C. Simulations setup

In the smart parking network, as depicted in Fig. 5 (left network), the sensor nodes play the role of detecting a car presence by periodically generating one of two possible values: "0" for a *free* space and "1" for an *occupied* space, along with a time-stamp of the measured value. We simulate a total of 25 sensors managed by one controller. Each node is uniquely identified (from 1 to 25) with the identifier 1 attributed to the sink which is directly connected to the controller. Similarly, the nodes of the smart monitoring network (right network in Fig. 5) generate periodically a temperature measure which we simulate here as a random value between 0 and 100. This network is also represented by 25 nodes (nodes 26 to 50) with the node 26 playing the role of the network sink. The *SDN-WISE* topology discovery protocol is executed at the start of each node in order to find a path to the sink. Each node then starts periodically sending control packets to the controller (i.e Network Discovery block) to build a consistent view of the network and gather its information [9]. The default *SDN-WISE* Dijkstra algorithm is used to find the shortest path between two nodes.

We have built a tool that provides a terminal where a function block can be called with the right input parameters, and directly interacts with our toolbox implemented at the controller. We demonstrate our work with an example of use of

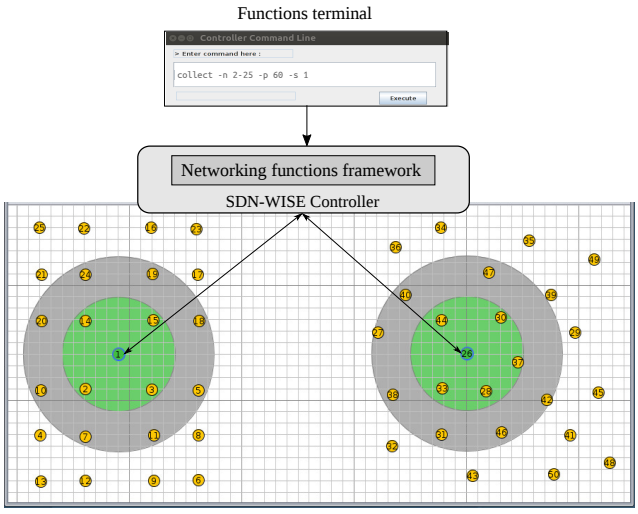

Fig. 5. The smart parking (left) and the smart monitoring (right) network topologies in Cooja using a 10 meter grid as a layout scale

two function blocks; the data collection and data aggregation blocks.

The main goal of the smart parking application is to collect data from the sensors which is the outcome of executing the data collection block. Here, we simulate a use case where all the nodes randomly generate a parking space status and send it to the sink every minute. Using these inputs for the data collection function by an application developer, this translates in the controller terminal tool by calling the data collection block as follows: {*collect -s 1 -n 2-25 -p 60*}, where the option *-s* specifies the collection packets destination, the *-n* option specifies the collecting nodes (all the nodes except the sink) and *-p* specifies the collection period in seconds. The same process goes for the temperature monitoring application, where its network nodes randomly generate a temperature value every minute using the following command: {*collect -s 26 -n 27-50 -p 60*}. In both cases, this prompts the data collection block to perform the following operations: (1) calling the data routing block which will use the network discovery block to create the forwarding rules necessary to set up the shortest path from the sink to each data generating node, (2) creating the configuration packets with the specified collection parameters, (3) Calling the SDN rules translation block which generates and installs the SDN-WISE packets in the nodes. The structure of a data collection packet generated by each node is shown in Fig. 6.

The second networking function we consider is aggregating the packets generated from the nodes to the sink with the goal of reducing the number of packets transmitted in the network. In the smart parking network, we demonstrate with a scenario where the aggregating nodes are the closest nodes relaying packets to the sink (1 hop away from the sink) using a simple concatenation method. The aggregating node generates an aggregation packet from the *n* collection packets received in a specified aggregation period by concatenating the payloads of all received packets and their source node IDs (Fig. 6). The input parameters are executed as follows: {*aggregate -n 1hop -f aggConcat -p 120*}, where *-n* specifies the aggregating nodes (1 hop away from the sink), *-f* specifies the aggregation method and *-p* specifies the aggregation period in seconds (here, every 2 minutes). We do the same process for the temperature monitoring network, only in this case, the aggregation method we consider (here called *aggMaxMinAvrge*) generates a packet containing the maximum, minimum and average of the measures received in the specified aggregation period (Fig. 6). We also set the aggregating nodes those located 2 hops away from the sink. We set these parameters using the following command: {*aggregate -n 2hop -f aggMaxMinAvrge -p 300*} (the aggregation period is set of 5 minutes in this example). These configurations prompt the data aggregation block to perform the following operations: (1) creating the configuration packets necessary to install the aggregation technique in the aggregation nodes, (2) creating the rules instructing the aggregating nodes on how and when to aggregate incoming packets, (3) calling the SDN rules translation block to generate and install the SDN-WISE

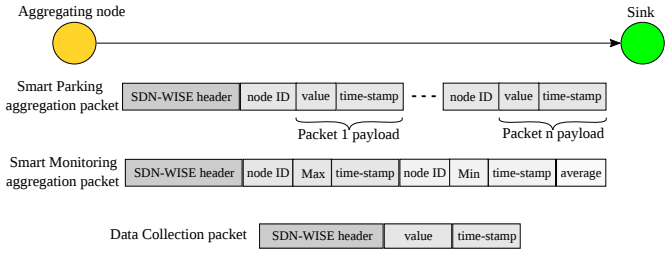

Fig. 6. Data collection and aggregation packets

packets in the nodes. The rule installed in an aggregating node and executed when a new collection packet arrives would be the following :

Condition:
if (packet_type = type_collection)
and (packet_destination = sink_address)
Action:
execute (aggregation_method)
drop (packet)

D. Numerical results

After presenting the ease of development for both smart city scenarios, we now exhibit the performance of our tool. We focus on two performance criteria: execution time and loss ration. We present our numerical results by reporting some performance measures when executing the data collection and aggregation function block using the simulation parameters presented in Table II. In case of the temperature monitoring network, Fig. 7 shows the time required to generate the configuration packets which install the rules and set up the parameters of the corresponding nodes when the data collection and aggregation blocks are called. This time is measured when the nodes are located at a different number of hops from the sink. The collection function block takes an average time of 53 ms in the case of a node located 3 hops away from the sink, and 46 ms for 1 hop node. The measure includes the *Request* and *Response* packets generated for the routing process. When the data aggregation function block is called, it takes on average 513 ms for the aggregation process to be executed for the nodes neighboring the sink. This measure includes all the *Config* packets necessary for the aggregation method byte code, and also includes the *Request* and *Response* rules packets generated for routing purposes. This same process takes around 519 ms for nodes located 2 hops away from the sink, and 531ms when located 3 hops away. Fig. 7 also shows that, for function blocks, the number of hops to the sink does not have a big impact on the installation time. We also compute the measure loss ratio (*mlr*), for the smart parking scenario, after deploying the use case described in Section III-C where all the network nodes collect the available information of a parking spot and generate a measure packet every minute to the sink. Here, we consider the measure loss ratio instead of the packet loss rate because, in this case, the sink receives the parking space measures transmitted by the nodes in an aggregated packet containing different measures. This means

TABLE II
SIMULATION PARAMETERS

Simulation parameter	Value
N° of nodes	50 nodes
N° of sinks	2 sinks
Transmission range	30 meters
Interference range	50 meters
Transmission rate	250 kbps
collection period	1 and 2 minutes
Collection packet size	12 bytes

that a packet loss by one of the collecting nodes causes a missing measure at the sink. The value of *mlr* is calculated as follows: *mlr* = (1 - m_r/m_s) $\times$ 100, where m_r is the number of measures received at the sink and m_s is the number of measure packets sent by the collecting nodes.

We simulate the scenario with 3000 measure packets generated by the smart parking nodes (m_s = 3000) and aggregated (using the aggregation method described in Section III-C). Fig. 8 shows the measure loss ratio as a function of the aggregation period at the aggregating nodes. Measure loss is essentially due to the loss of either the collection packets sent by the collecting nodes or to the loss of aggregation packets sent from the aggregating nodes to the sink. This loss is caused by the interference at the UDGM radio medium between these data packets and the *SDN-WISE Beacon* and *Report* control packets (sent periodically every 10 and 20 seconds). *Beacon* packets are broadcast packets sent by each node to its neighbors and contain its distance from the sink and its battery level. *Report* packets are sent by each node to the sink and, in addition to the distance from the sink and the battery level, contains the list of the node neighbors and a quality indicator of each link. In our scenario, around 32400 control packets are generated. The high frequency of these control packets causes interference resulting in packet losses.

As presented in Fig. 8, when the *Beacon* and *Report* control packets periodicities are set respectively to 10 and 20 seconds, and for the different aggregation periods we obtain an average *mlr* that is less than 1%: going from 0.4% for a 2mn aggregation period up to 0.83% for a 5mn aggregation period. The

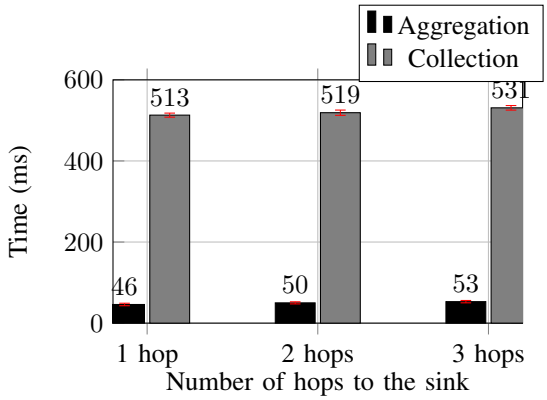

Fig. 7. Time for generating and installing configuration packets.

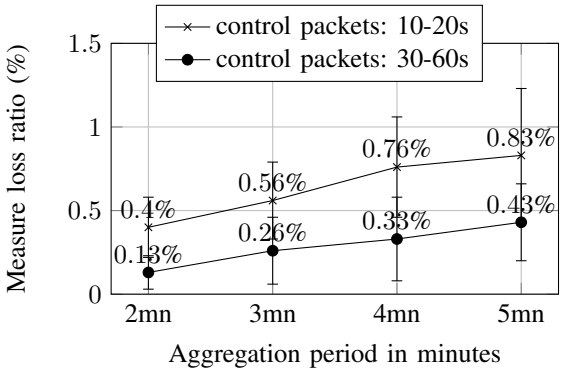

Fig. 8. Measure loss ratio as a function of the aggregation period for 3000 measure packets transmitted.

longer the aggregation period is, the bigger the aggregation packet gets. Consequently, when a bigger aggregation packet is lost, more measures are lost, and thus the *mlr* value becomes higher. We simulate the same scenario above-mentioned but modify the control packets periodicity, where *Beacon* packets are sent every 30 seconds instead of 10 seconds and *Report* packets are sent every minute instead of every 20 seconds. This reduces the total number of control packets to around 10800 packets. In Fig. 8, we can see the *mlr* dropping by 0.27% in the case of a 2 minutes aggregation period, and up to 0.4% when the aggregation period is set to 5 minutes. We point out that the average *mlr* values, for both cases, is less than 1%.

The approach presented in our work can be more useful with larger networks, as it provides an easy and flexible method to instantiate the networking functions in the network. We, thus, plan to target larger infrastructures for future experiments. We also aim at extending our work by covering more functions and further defining their interfaces and dependency relations. Our goal is to create a toolbox that supports a wide set of networking functions, in order to provide a flexible solution that is less dependent on the infrastructure.

IV. Conclusion

With the increasing popularity and complexity of WSN-based IoT applications, it becomes crucial to facilitate their development by facilitating the control and management of their underlying network infrastructure. Vertical deployments of WSNs make this process a difficult task and thus a more flexible solution should be adopted, especially when many of these applications share a set of common networking functions. In this paper, we introduced an SDN-based toolbox for using such functions as controller-level building blocks to facilitate the network control in a WSN infrastructure, and thus facilitating the task of building and deploying IoT applications. We motivated how some of the networking functions can be used in smart city applications, such as the smart parking and smart monitoring applications, and illustrated our method of implementing them using the *SDN-WISE* platform, a software-defined solution for WSNs.

The flexibility provided by the SDN principles facilitates the task of deploying and managing these functions by using the controller capabilities and abstracting them as blocks with well-defined interfaces. Our work offers a toolbox of functions that can be used for different types of WSN-based IoT applications that share the same networking functions. This provides re-usability and helps developers easily program the network infrastructure and create the desired service.

References

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[2] Jasper Tan and Simon G. M. Koo. A Survey of Technologies in Internet of Things. In *2014 IEEE International Conference on Distributed Computing in Sensor Systems*, pages 269–274, 2014.

[3] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of Things for Smart Cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.

[4] A. Dunkels, R. Gold, S.A. Marti, A. Pears, and M. Uddenfeldt. Janus: An Architecture for Flexible Access to Sensor Networks. In *Proceedings of the 1st ACM workshop on Dynamic interconnection of networks*, pages 48–52, 2005.

[5] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[6] D. Kreutz, F. MV Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[7] Habib Mostafaei and Michael Menth. Software-defined wireless sensor networks: a survey. *Journal of Network and Computer Applications*, 119:42–56, 2018.

[8] Object Management Group. OMG Systems Modeling Language, Version 1.5, May 2017.

[9] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo. SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 513–521. IEEE, 2015.

[10] Salvatore Costanzo, Laura Galluccio, Giacomo Morabito, and Sergio Palazzo. Software Defined Wireless Networks: Unbridling SDNs. In *2012 European Workshop on Software Defined Networking*, pages 1–6. IEEE, 2012.

[11] SDN-WISE The stateful Software Defined Networking solution for the Internet of Things. http://sdnwiselab.github.io/, accessed January 2019.

[12] C. Buratti, A. Stajkic, G. Gardasevic, Se. Milardo, M D. Abrignani, S. Mijovic, G. Morabito, and R. Verdone. Testing Protocols for the Internet of Things on the EuWIn Platform. *IEEE Internet of Things Journal*, 3(1):124–133, 2016.

[13] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with Cooja. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 641–648. IEEE, 2006.

[14] Contiki: The Open Source OS for the Internet of Things. http://www.contiki-os.org/, accessed February 2019.