Research Article

A Dynamically Reconfigurable Wireless Sensor Network Testbed for Multiple Routing Protocols

Wenxian Jiang,¹ Chenzhe Gu,¹ and Jingjing Wu^{2,3}

¹College of Computer Science and Technology, Huaqiao University, Xiamen 361021, China

²College of Mathematics and Computer Science, Quanzhou Normal University, Quanzhou 362000, China

³Fujian Provincial Key Laboratory of Data Intensive Computing, Quanzhou 362000, China

Correspondence should be addressed to Wenxian Jiang; jwx@hqu.edu.cn

Received 20 May 2017; Accepted 23 August 2017; Published 8 November 2017

Academic Editor: Javier Prieto

Copyright © 2017 Wenxian Jiang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Because wireless sensor networks (WSNs) are complex and difficult to deploy and manage, appropriate structures are required to make these networks more flexible. In this paper, a reconfigurable testbed is presented, which supports dynamic protocol switching by creating a novel architecture and experiments with several different protocols. The separation of the control and data planes in this testbed means that routing configuration and data transmission are independent. A programmable flow table provides the testbed with the ability to switch protocols dynamically. We experiment on various aspects of the testbed to analyze its functionality and performance. The results demonstrate that sensors in the testbed are easy to manage and can support multiple protocols. We then raise some important issues that should be investigated in future work concerning the testbed.

1. Introduction

The self-organization and multihop features of wireless sensor networks (WSNs) allow nodes to communicate with each other quickly. Nodes in WSNs, which are different from traditional network nodes, often carry out information collection, signal processing, and wireless communication simultaneously [1]. Meanwhile, new network structures and algorithms have been constructed and tested on simulators. Simulation allows WSNs research to be carried out quickly and easily but may provide inaccurate results because of the lack of a real environment. This has motivated the design and implementation of a number of WSN testbeds by researchers across the globe. However, most testbeds concentrate on specific applications or routing protocols, resulting in a lack of abstraction and reconfiguration capabilities, which makes policy changing and network management rigid.

Network reconfiguration allows an operator to change functions and update policies at any time. These dynamic characteristics make a network flexible and scalable. Meanwhile, owing to the resource limitations of WSNs, it is important to create an adaptive architecture to reduce the

complexity of deployment and execution. In general, there are three forms of reconfiguration for WSNs: software, data routing, and hardware [2]. Hardware reconfiguration provides flexibility on a sensor itself, such as activity or energy consumption. In terms of functionality, software and protocol reconfiguration can support various scenarios at runtime. For most networks, reconfiguration can be implemented by dynamic reprogramming [3, 4] or the loading of different components [5].

In this study, a dynamically reconfigurable WSN testbed is designed and it is easy to manage and reconfigure with different routing protocols. Rather than simply implementing a testbed, the target focus is on the architecture of software and protocol reconfiguration. The testbed touches data and routing control traffic (including forwarding policies and protocol reconfiguration). In the design and experiment, the separation of the data and control planes can help to reduce the complexity of the platform and allow the support of various routing protocols. Meanwhile, a programmable *flow table* of the data plane makes the implementation of policy changes fast and clear. Our work presents a concrete form of the method and concept for establishing the testbed and

demonstrates its flexibility and scalability by several protocols in the experiment.

The remainder of this paper is organized as follows. In Section 2, we describe the motivation for this work. An architecture overview of the testbed is presented on the framework and operating system in Section 3. The method of routing reconfiguration and data processing is described in Section 4. Section 5 describes the general method for implementing a protocol for the testbed. Section 6 presents some experiments based on different protocols, in order to evaluate the testbed. Conclusions and directions for future work are presented in Section 7.

2. Motivation

WSN testbeds usually consist of a set of sensor nodes, a backchannel for testbed management connectivity, a database for data logging, and a management or debugging software application deployed on a server. Previous research has focused on the scale, speed, mobility, network architectures, and so on [6]. For example, [7] presented one of the first fully developed WSN testbeds and provided an open source tool to users for creating their own experiments. Reference [8] can support different network architectures and topologies by reprogramming or suspending some motes. By deploying two different radios on one mote, [9] implemented a heterogeneous and complex testbed to support various speed rates in the network. For high performance hardware and the mass deployment of nodes, researchers have utilized different testbeds to complete various tasks, such as system design, protocol testing, and performance evaluation. Table 1 provides a lateral comparison of the main characteristics of some important testbeds.

According to Table 1, WSN testbeds have been designed to achieve various functions, scales, and communication layers, but most of these do not have the ability to support multiple and dynamic protocol tests. Although there are some testbeds that can support different network architectures, the routing layer still needs to be reconfigured or even redeployed for every experiment. It is worth mentioning that IOT-Lab provides three layers of API: drivers, operating systems, and communication libraries, allowing users to define their own network policies. However, regarding the following demands, there are significant disparities and challenges present in existing testbeds.

Network Programmability. The support of multiple applications or protocols: this is achieved by programmable hardware or new design structures between the data and control planes [4, 13]. However, existing WSN testbeds always concentrate on a specific application or scenario, which leads to incompatible policy changes or function that are too simple.

Packet Processing. Data traffic and control traffic should be separated by different modules. In some situations, the data process consists of nodes performing flow-based packet forwarding, and the network contains one or more controllers (or sinks) managing all other nodes. The network will send

FIGURE 1: Testbed framework.

unknown packets or commands to the controller to avoid excess processing logic, which reduces the complexity of network.

Multiple Applications. A centralized network control provides an application-independent method of management, so that ordinary nodes do not need to care about specific processes. In traditional network devices, the function of a hardware component or software module is designed for known protocols or application scenarios. However, applications are transparent to a node, which means that multiple applications or protocols could be mixed without mutual interference.

Another challenge is that embedded operating systems for WSNs do not support dynamic reconfiguration effectively. Although there are some operating systems that can load applications dynamically, such as Contiki [14], these always lack protocol or algorithm reconfiguration. The task scheduling for WSNs is usually linear and single threaded, which does not distinguish traffic with different demands. This high degree of coupling makes it hard to change policies in real time [15].

The challenges described above are inherent to WSNs and deeply rooted in the architecture. A reconfigurable testbed should support runtime reconfigurations to the application and data routing. Although existing architectures support some kinds of well-designed applications and algorithms, they still lack good abstraction and make networks rigid and inelastic.

3. Architecture and Features of the Testbed

This section provides an overview of the implementation of the proposed testbed on real hardware. Various components of the testbed are described, which include the control workstation, the OpenFlow controller, the node programmer, the sensor nodes, and a group of computers that will collect the run data by wired connections. A schematic diagram of the testbed is presented in Figure 1.

3.1. Testbed Framework and Hardware. The network controller shown in Figure 1 is different from the sink in the original WSNs. It is part of the test area that manages the

TABLE 1: A comparison of WSN testbeds.

Testbed	Nodes and scales	Primary features	Multiprotocol support
MoteLab [7]	190 TmoteSky motes	Aiming at programming environment, communication protocols, system design, and applications	Single protocol
SignetLab [8]	48 EyesIFxv2 motes	Software to control motes and dynamically creating architectures	Single protocol with variable network architectures
Sensei-UU [10]	TelosB motes and mobile robots	Smart phone support and repeatable experiments are at different locations	Single protocol
NetEye [11]	130 TelosB motes	Web-interface to create and schedule experiments	Single protocol
EasiTest [9]	EZ271 and EZ521 motes	Heterogeneous multiradios to support various speed rates	Single protocol
IOT-Lab [12]	Over 2700 motes and mobile robots	Heterogeneous platform, mobility support, and different APIs	Single protocol with programmable routing layer

whole network, not only the communication interface. In this design, every sensor node can be configured as a sink, which allows the network to adapt to some routing protocols with one or more data centers, such as the directed diffusion (DD) protocol [16] and the low energy adaptive clustering hierarchy (LEACH) protocol [17]. One problem with the testbed concerns the data collection of sensor nodes. Most WSN testbeds transmit the testbed metadata (the testbed configuration commands, internal data of nodes, statistical information, etc.) by the wireless channel, which takes significant channel resources and memory space away from nodes and results in a higher latency or loss rate when a test is run. Therefore, we set a group of data terminals to collect sensor data instantaneously by a UART (universal asynchronous receiver/transmitter) or USB (universal serial bus) port. For a sensor node, UART/USB transmission requires fewer computing resources than radio frequency (RF).

On the hardware side, the testbed consists of a CC2530 and CC2531 system on chip (SoC), developed by Texas Instruments. CC2530 is based on an enhanced 8051 core using the standard 8051 instruction set, which can operate at 32 KHz, 16 MHz, or 32 MHz to support different power modes (fully functional or ultralow-power modes). It has an 8 KB static random access memory (SRAM) and a programmable Flash (32/64/128/256 KB) to support an embedded operating system. CC2530 supports a variety of serial interfaces, including the serial peripheral interface (SPI) and UART (CC2531 supports USB control) and features an IEEE 802.15.4/ZigBee RF transceiver in the 2.4 GHz band. In the context of our testbed, the controller is built on CC2531 and the sensor nodes are built on CC2530 or CC2531. The nodes of both types can all be powered by USB interface. In addition, for the convenience of the testbed deployment, in this work CC2530 will be equipped with three AA batteries when required.

3.2. Testbed Operating System. All of the nodes run TinyOS [18], a specific embedded operating system for WSNs. TinyOS

FIGURE 2: Operating system component relations graph for the testbed.

is a component-based operating system, which provides a fast method of updating the system and reducing the coupling between modules. In this work, there are several components to separate the control and data planes and accomplish different functions such as data transmission, routing configuration, and user application. By overwriting the hardware interface, TinyOS is transplanted to CC2530. Figure 2 presents a component graph of the system. The system utilizes two important components, “SwitchPanelC” and “ControlPanelC,” to implement different network policies. The component “SwitchPanelC” provides a set of operations, which are stored in the component “ActionC,” to modify the flow table. The component “ControlPanelC” is a part of the control plane of the network (the system structure of the controller is similar to that of the sensor node, but its functionality is more complicated), which handles and stores information used in the network control. The component “BasicRoutingC” is maintained by “ControlPanelC,” and most routing policies are stored in this component. The component “RadioManagerC” provides a common interface

FIGURE 3: Platform framework based on OpenFlow.

for accessing the data transmission and reception directly. With this independent access control, routing information can be separated from general data, which makes dynamic policy changing possible.

4. Reconfiguration Kernel

On the software and protocol levels, software-defined networking (SDN) is developed to enhance the programmability and flexibility of wired networks [19]. As a realization of SDN, the starting point of OpenFlow is to allow researchers to develop and verify new network architectures and protocols conveniently on campus networks. OpenFlow provides a simple implementation for different applications, protocols, and network management systems by separating the control and data planes. Meanwhile, OpenFlow allows administrators to use the standardized interface to manage the network and change policies. For WSNs, SDN provides a method of reconfiguring the network dynamically, without reprogramming or redeploying [20].

In traditional networks, dynamic protocol reconfiguration is implemented by software and flexible protocol stacks. The processing of many network parameters is required, which conflicts to the demands of WSNs. Recent work has demonstrated the potential of SDN in sensor networks and has discussed basic considerations regarding implementation, including architecture and deployment [21, 22].

In comparison with general network platforms based on OpenFlow, a platform in WSNs exhibits some different characteristics, such as the networking mode, topology, transmission rate, and fault tolerance [23]. Therefore, a node in WSNs must be able to support several network layer functions. Therefore, the concept of flows is incorporated into WSN to simplify the architecture of the testbed. This should be constructed from several parts, as illustrated in Figure 3. In this paper, the control and data planes are separated, not only for unknown packets and the flow table configuration, but also for routing policy management. This separation allows

the extraction of routing information from data transmission and the reduction of the coupling of the two planes.

4.1. Basic Routing Control Plane. For most routing protocols, some defined routing policies require the cooperation of different nodes or control by a specific node (i.e., a sink or cluster head). However, this could result in high-coupling relations between routing information exchange and data forwarding. To separate routing operations and data processing, the control plane manages routing operations to configure the current network state, such as MAC learning, clustering, and signal-aware topology discovering. These operations are independent of forwarding policies in the data plane, even though they are always different for each node. In particular, the control plane marked by the dashed line in Figure 3 separates routing messages from data flows, which means that the control plane can handle not only unknown packets, but also routing policies.

4.2. Programmable Flow Table. In the data plane, all packets are handled as flows. The forwarding policy of flows is controlled by a *flow table*, which is defined by an OpenFlow controller. Each item or flow entry in the table contains three basic fields: a set of rules related to the flow, one or more actions that can be executed when a flow is matched to an entry, and some statistical information regarding the entry. The content of an entry depends on the running protocol in the network and can be updated to adopt changes to the protocol if necessary. In addition, because of the special character of WSNs, data generated by a node itself must also be reasonably processed, even if it is a routing node. Therefore, nodes and flow tables in a WSN must have the ability to handle locally generated data. This can be solved by adding an additional flow entry to the flow table.

4.3. Actions and Miss Rule. Each flow entry has an *action field*, which contains one or more operations that could be

FIGURE 4: Relation between the control and data planes.

applied to the packet belonging to the flow. We argue that there are two kinds of action: *required actions* and *optional actions*. The flow tables must implement required actions, such as dropping, broadcasting, forwarding, and uploading the packet to the local network stack. Optional actions can enhance the functionality of the table, including modifying specific packet fields, sending requests to the controller, or performing other optimizing operations. The optional actions could be different for one node than for others for a specific network role or requirement (i.e., data collection or routing).

In traditional OpenFlow, if a packet cannot be matched to any flow entry, then it will be dropped by a *miss rule* or uploaded to the controller. In comparison with the flow entry in OpenFlow, the miss rule in our work contains one or more operations regarding coping with packets: such as dropping it, modifying (including creating) the flow entry, or uploading it to the controller. Like other flow entries, a miss rule can be updated independently for different protocol policies.

5. Protocol Implementation with Separation of Control and Data Planes

Figure 4 illustrates the relation between the control and data planes in one sensor node. As previously mentioned, a protocol will be configured in these two planes, so the design can be classified into two parts (the routing operation and flow table parts). In order to switch protocols dynamically, the flow table and basic routing control must be configured independently. Unlike traditional wireless network protocols, some protocols in WSNs have to accomplish other tasks before starting the routing algorithm, such as clustering, time synchronization, and energy-awareness. Therefore, we can

TABLE 2: Routing operation sequences.

Protocol	Routing operation sequence
Flooding	No routing operation
LEACH	(1) Time synchronization (2) Cluster head election (3) Cluster head broadcasting
Directed diffusion	Sink information broadcasting

specify a routing operation sequence in the control plane to implement these required tasks. These operations can be assembled together randomly (unless the control plane does not need any operations, such as flooding) to support different routing protocols. Table 2 presents some examples of different operation sequences.

5.1. Data Flows in the Network. Figure 5 illustrates the manner in which a packet is processed by the network. For the received packet, the treatment process can be considered as a function of action, which uses the packet itself and the matched flow entry as arguments. Thus, the following process formula can be applied when a packet is to be forwarded or received by the function:

$$\text{Act}_X(\text{Pkt}_m, \text{FE}_m) \longrightarrow \text{Node}_Y, \quad (1)$$

where the packet, the matched flow entry, and actions are represented by Pkt_m , FE_m , and the function Act_X . The result generated by Act_X will be attached to the next hop or local

FIGURE 5: Topology of the network and flow table processing a packet. Broken lines present optional methods of processing the packet.

node. Therefore, the final result can be obtained by a recursive application of formula (1):

$$\begin{aligned} \text{result} &= \text{Act}_{X+1} \left(\text{Act}_X (\text{Pkt}_m, \text{FE}_n) \right. \\ &\quad \left. \longrightarrow \text{Node}_{X+1}, \text{FE}_{(X+1)_n} \right) \longrightarrow \text{Node}_Y. \end{aligned} \quad (2)$$

Furthermore, a packet that does not belong to any flow entry will be processed by a miss rule as follows:

$$\text{Miss}_X (\text{Pkt}_m) \longrightarrow \text{Node}_{\text{Controller}}, \quad (3)$$

where $\text{Node}_{\text{Controller}}$ can also be the local node itself if the miss rule involves dropping the packet or other operations.

5.2. Procedure for Processing Packets. Forwarding policies are implemented as a set of flow entries in the flow table. Flow entries decide how packets will be treated, as well as other operations regarding the node itself. Flow entries have to be configured before a protocol is deployed. Rules in a flow entry are some conditions that data flow fields have to match [24]. Each field is given by two thresholds (the top and bottom thresholds) to maintain the matching strategy. The flow is considered as a match for a specific flow entry if the value of every field in the flow is located between these two thresholds. Table 3 presents some examples of different operation sequences. We extract the flow mask of every packet and use a resulting code to judge the match results of actions by Algorithm 1.

When the network is running, these operations will be executed at the appropriate time (i.e., the clustering operations should be executed before each running period). To avoid a situation where a flow is matched to several entries, we argue that every entry has a priority in the table. In addition, a field called the *flow mask* could decide which field should be added to the flow, thus reducing the length of a flow entry and rendering the checking of every field unnecessary. Moreover, protocols can be distinguished by their masks. When more than one protocol is run in the network, different flows could have the same mask, because

TABLE 3: Example of rules and actions of flow table.

Rule	Actions	Stats
$x \leq \text{SEQ} \leq \text{MAX}$	(1) Update the flow table	s_1
$\text{DST} = \text{Local Address}$	(2) Receive	
$\text{DST} \neq \text{Local Address}$	(1) Update the packet field (2) Forward	s_2
$\vdots$	$\vdots$	$\vdots$
$\text{TTL} = 0$	(1) Drop	s_n

of the requirements of protocols. Therefore, there will be another field to separate from these if necessary. The function *GetFlowMask* is used to achieve the masking of a flow and the function *GetMatchField* to locate the first match field. Each effective part of the mask represents a specific match field that will be checked by the function *CheckMatchField*. If all match fields conform to a flow entry, then the corresponding actions will be executed, and other entries will not be checked anymore. A miss rule will be applied when no entry matches the current flow.

6. Experiment and Evaluation

The principal feasibility of the testbed is evaluated on several features, such as the transmission efficiency, flow table processing capability, and energy consumption. By filling flow tables with different parameters, different routing protocols are tested to evaluate the effectiveness and correctness of the testbed.

6.1. Experimental Setup. The application scenario in our experiment consists of a group of sensor nodes and computers. Figure 6 illustrates a small section of the testbed that contains 16 sensors in a lab. In this scenario, sensor data and system information are collected by the terminal computer connected to a backbone Ethernet, which makes it convenient for the testbed to upload data and receive commands. A

```

Input: Payload; FlowTable[]; TableLength mask=GetFlowMask(Payload);
offset=GetMatchField(Payload);
for each i TableLength do for each bit in the mask do
    if bit = 1 then
        state=CheckMatchField(offset, FlowTable[i]); if state is true then
            update offset to next match field; end
        else state=false;
            break; end
    end end
if state is true then
    ExecuteAction(Payload, FlowTable[i]);
    break; end
end
if no action executed then
    ExecuteMiss(Payload); end

```

ALGORITHM 1: Packets process in switch plane.

FIGURE 6: Testbed scenario and setup.

wiring hub is attached to the controller to program sensor nodes as a whole. When the testbed is loaded, all nodes can be directly taken over by terminal computers.

6.2. Statistics of the Flow Table. In order to evaluate the validity in a common scenario, different protocols are transferred and applied to this platform and two indices are defined: the network transmission efficiency (NTE) and the average matching number (AMN). When a protocol is running, it can generate some redundant packets, such as beacons; thus NTE describes the proportion of the real data that is actually required by the network, reflecting the efficiency of the communication (including the two planes). AMN describes the average retrieval times of packets being processed by the flow table, which can be used to evaluate the processing speed and capability of a protocol. Figure 7 presents the results of our experiment for different protocols (flooding, LEACH, and directed diffusion). The expected manifestation of every protocol is given, and LEACH indicates the best performance (a high NTE and low AMN value). This can be reduced to 61.90% when the network recreates the route for each communication, although DD with the fixed path

achieves the highest NTE value (92.03%). This test proves that the testbed based on OpenFlow can support one or more protocols, and a horizontal comparison is made.

6.3. Energy Consumption. The energy consumption of a node, or even the whole network, can be effected by the protocol, topology, physical environment, and so on. From Figure 8, the influence of each protocol and the flow table content on the network energy consumption are presented. In the context of the same topology and communication frequency, because of the low-power consumption of WSNs, trends in energy among these protocols are implicit. However, the average energy consumption obviously varies. Meanwhile, because the configuration of a flow table determines how a flow is processed, some energy-intensive actions (such as forwarding or uploading a packet to the controller) can determine the energy consumption, which means there is a novel method for reducing the energy consumption by controlling and configuring the actions of a flow table.

6.4. Throughput of Control and Data Planes. In general, a control plane separates the network management flow from

FIGURE 7: NTE and AMN of protocols. We suppose that the topology of DD is fixed, and once all transmission paths are created they will not be modified further. On the contrary, the topology of DD* is dynamic, so it will recreate some paths for reliable communication.

FIGURE 8: Energy consumption with different protocols.

the data transmission. Furthermore, in this testbed it can cope with routing operations such as the flow table configuration, clustering, and route requests. To illustrate the effect of the separation of the control and data planes in this testbed, the throughput of two planes with different protocols is evaluated. Figure 9 presents the statistics for each protocol in 500 s. Under the circumstance that the same data generation rate is maintained, the flooding protocol achieved a higher throughput than the other protocols on the data plane. This is because excessive redundant packets are present during the transmission. For the DD and DD* protocols with routing maintenance, the number of redundant packets is much lower than for flooding. On the control plane, the LEACH protocol achieves the highest throughput for the clustering operation but expends vast amounts of network resources. In fact, layered routing protocols (i.e., LEACH) consume more

FIGURE 9: Throughput of the two planes with different protocols.

FIGURE 10: Network traffic during protocol switching.

controlling flow on the transmission than planar routing protocols. In this experiment, the testbed clearly separated the two planes and reflected the respective characteristics of different protocols with OpenFlow.

6.5. Dynamic Reconfiguration with Different Protocols. Although most nodes in the network share common structures and functions, they can exhibit extremely different transmission policies when given different configurations in the flow table. All data packets sent (or received) by a node should be processed by the flow table, and this can affect the performance of the transmission instantaneously. A protocol switch can be completed by updating one or more flow tables with new contents. The overhead of deploying a protocol to one flow table is presented in Table 4 (the extra network traffic is not contained in the table). In Figure 10, the network traffic is tested when the running protocol is switched. The network load was recorded by the testbed during protocol configuration. The switch to the DD protocol consumes noticeable network traffic to update the flow tables. In addition, as the network grows in size, an inappropriate

TABLE 4: Net overhead of filling the flow table with a specific protocol.

Protocol	Overhead (byte)
Flooding	56
LEACH	23
DD (sensor node)	89
DD (sink)	31

protocol selection could lead to a network load increase. Therefore, the testbed based on OpenFlow can not only switch routing protocols dynamically without redeploying the whole network, but also choose an optimal protocol to reduce the network load.

7. Conclusions and Future Work

In this paper, a dynamically reconfigurable WSN testbed is established. To implement a software-based reconfiguration method in real time, a novel architecture using OpenFlow constructs the data plane in a protocol-independent manner. The experiments demonstrated that the testbed can reduce the complexity of network management and updating by separating the control and data planes. A user can manage the network easily by simply configuring different behaviors, without redeploying the whole network. The dynamic switching between protocols was implemented by a programmable flow table, which makes the network reconstruction faster and more convenient. Furthermore, it is obvious that large scale WSN will be applied to various scenarios which have different routing demands. So this ability of reconfiguration can support WSN virtualization when the scale is growing up and can be used in a multitenant architecture for cloud service.

However, there still remain issues that should be investigated. In OpenFlow, a controller should support extra functions for the network to process unknown flows or update flow tables. However, in some networks with fast topology changing, a controller cannot handle all messages instantly and exactly. Sometimes this could result in performance degradation, because of the uncertainty in deployment and network topology. Furthermore, it is difficult to deploy all protocols by one universal flow table configuration method. For example, the networking processes for a layered protocol and planar protocol differ significantly, which could result in a significant overhead for dynamic protocol switching.

There can be three sections to resolve these problems in future work. Firstly, a controller group can be applied to the network rather than only one controller, which can offload specified traffic or protocol to different controllers and enhance the functionality of control plane to optimize routing operations. Secondly, common features of transmission policies among different protocols will be extracted by analysis and experiment, so that flow table (including flow actions) can be simplified for packet process. Finally, there are some search algorithms and they will decrease the cost of flow table matching.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Science and Technology Plan Key Project of the Fujian Province of China under Grant no. 2014H0030, the Science and Technology Plan Key Project of Quanzhou City, Fujian Province of China, under Grant no. 2014Z102, the Natural Science Foundation of Fujian Province of China (no. 2017J01776), the Special Scientific Research Project of Provincial Colleges and Universities of Fujian Province of China under Grant no. JK2015037, and the Young Dr. Project of Quanzhou Normal University under Grant no. 2015QBKJ02.

References

- [1] P. Rawat, K. D. Singh, H. Chaouchi, and J. M. Bonnin, "Wireless sensor networks: a survey on recent developments and potential synergies," *The Journal of Supercomputing*, vol. 68, no. 1, pp. 1–48, 2014.
- [2] H. Grichi, O. Mosbahi, M. Khalgui, and Z. Li, "RWiN: New methodology for the development of reconfigurable WSN," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 1, pp. 109–125, 2017.
- [3] P. Latha and M. A. Bhagyaveni, "Reconfigurable FPGA based architecture for surveillance systems in WSN," in *Proceedings of the 2010 International Conference on Wireless Communication and Sensor Computing, ICWCSC 2010*, January 2010.
- [4] P. Bosshart, G. Gibb, H.-S. Kim et al., "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM SIGCOMM 2013*, pp. 99–110, August 2013.
- [5] S. Bagchi, "Nano-kernel: A Dynamically Reconfigurable Kernel for WSN," in *Proceedings of the 1st International ICST Conference on Mobile Wireless Middleware, Operating Systems and Applications*, Innsbruck, Austria, February 2008.
- [6] L. P. Steyn and G. P. Hancke, "A survey of wireless sensor network testbeds," in *Proceedings of the IEEE Africon'11*, zmb, September 2011.
- [7] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: a wireless sensor network testbed," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN '05)*, pp. 483–488, April 2005.
- [8] R. Crepaldi, S. Friso, A. Harris et al., "The design, deployment, and analysis of signetlab: A sensor network testbed and interactive management tool," in *Proceedings of the 3rd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, TridentCom 2007*, May 2007.
- [9] Z. Zhao, G.-H. Yang, Q. Liu, V. O. K. Li, and L. Cui, "Eas-iTest: a multi-radio testbed for heterogeneous wireless sensor networks," in *Proceedings of the IET International Conference on Wireless Sensor Network (IET-WSN '10)*, pp. 104–108, November 2010.

- [10] O. Rensfelt, F. Hermans, P. Gunningberg, L.-Å. Larzon, and E. Björnemo, "Repeatable experiments with mobile nodes in a relocatable WSN testbed," *The Computer Journal*, vol. 54, no. 12, pp. 1973–1986, 2011.
- [11] X. Ju, H. Zhang, and D. Sakamuri, "NetEye: a user-centered wireless sensor network testbed for high-fidelity, robust experimentation," *International Journal of Communication Systems*, vol. 25, no. 9, pp. 1213–1229, 2012.
- [12] C. Adjih, E. Baccelli, E. Fleury et al., "FIT IoT-LAB: A large scale open experimental IoT testbed," in *Proceedings of the 2nd IEEE World Forum on Internet of Things, WF-IoT 2015*, pp. 459–464, ita, December 2015.
- [13] L. De Carli, Y. Pan, A. Kumar, C. Estan, and K. Sankaralingam, "PLUG," in *Proceedings of the the ACM SIGCOMM 2009 conference*, p. 207, Barcelona, Spain, August 2009.
- [14] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki—a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the 29th IEEE Annual International Conference on Local Computer Networks (LCN '04)*, pp. 455–462, November 2004.
- [15] M. R. Senouci, A. Mellouk, L. Oukhellou, and A. Aissani, "WSNs deployment framework based on the theory of belief functions," *Computer Networks*, vol. 88, pp. 12–26, 2015.
- [16] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 2–16, 2003.
- [17] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, 2002.
- [18] Tinyos home page, 2013, <http://www.tinyos.net/>.
- [19] O. N. Foundation, *Software-Defined Networking: The New Norm for Networks*, White Paper, 2012.
- [20] T. Luo, H.-P. Tan, and T. Q. S. Quek, "Sensor openflow: enabling software-defined wireless sensor networks," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1896–1899, 2012.
- [21] A. Mahmud, R. Rahmani, and T. Kanter, "Deployment of Flow-Sensors in Internet of Things' Virtualization via OpenFlow," in *Proceedings of the 2012 3rd FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing, MUSIC 2012*, pp. 195–200, June 2012.
- [22] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: from concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.
- [23] J. Horneber and A. Hergenroder, "A survey on testbeds and experimentation environments for wireless sensor networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1820–1838, 2014.
- [24] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in *Proceedings of the 27th Biennial Symposium on Communications (QBSC '14)*, pp. 71–75, Kingston, Canada, June 2014.