IT-SDN: Improved architecture for SDWSN

Renan C. A. Alves[1], **Doriedson A. G. Oliveira**[1], **Gustavo A. Núñez**[1], **Cintia B. Margi**[1]

[1]Universidade de São Paulo – São Paulo, Brazil

Abstract. *Wireless Sensor Networks (WSN) and the Internet of Things (IoT) are promising technologies for new applications. One of the main challenges in such environments is how to enable flexibility on infrastructure. Software Defined Wireless Sensor Networks (SDWSN) has been presented as an effective approach to address several issues in WSN and IoT. There are several approaches in the literature, but unfortunately there are drawbacks for these approaches. We present IT-SDN, an open SDWSN tool, that has a clear separation of the protocols used to achieve southbound communication, neighbor discovery and controller discovery, unlike other approaches on the literature. We also demonstrate how to evaluate the main performance metrics of an IT-SDN deployment.*

1. Introduction

Wireless Sensor Networks (WSN) and the Internet of Things (IoT) are composed of devices capable of sensing/actuation, communication and processing, thus enabling the development of applications in several areas, such as health, environmental, industrial, urban monitoring. One of the main challenges in WSN is how to enable flexibility on infrastructure, considering that all nodes behave both as forwarding devices and end nodes, there are different routing patterns, different applications being executed by the nodes, and yet it should be energy efficient and limit the control traffic (both by reducing the number of control messages and by limiting the frame size to the typical IEEE 802.15.4 frame).

Software-defined networking (SDN) is an approach to manage networks that separates the control plane from the data plane. The routing/forwarding decisions are made by the SDN controller based on network information received. Openflow [McKeown et al. 2008] was the first protocol proposed to establish the communication between forwarding nodes and the SDN controller.

Software Defined Wireless Sensor Networks (SDWSN) has been presented as an effective approach to address several of these issues in WSN. There are several approaches in the literature [Kobo et al. 2017]. Unfortunately there are drawbacks for these approaches. The ones based on OpenFlow have issues concerning frame sizes, introduced overhead and use of TCP as underlying communication protocol. SDN-Wise implementation is not completely available for download and use. TinySDN provides the code and related documentation, but it is highly dependable on TinyOS, thus limiting the platforms it could be deployed. Furthermore the use of the ActiveMessage function limits interoperability with other systems.

In this paper, we describe IT-SDN, an SDWSN tool that is completely open and available, unlike SDN-Wise. While its design is inspired by TinySDN [de Oliveira et al. 2014], we improved the architecture, protocols and implementation. The underlying architecture is independent of the operating system and its functions. IT-SDN has a clear separation of the protocols used to achieve southbound

communication (SB), neighbor discovery (ND) and controller discovery (CD), unlike SDN-Wise and TinySDN, which allows evaluation of different approaches to achieve neighbor and controller discovery. Furthermore, our implementation of IT-SDN supports configuring multiple nodes with a single packet, being up to the controller whether to use it or not. This feature was envisioned in previous work but has not been implemented [de Oliveira et al. 2014, Galluccio et al. 2015].

This paper is organized as follows. Section 2 presents IT-SDN architecture and specification, while Section 3 discusses the software architecture and implementation details. The proposed demonstration is described in Section 4, while Section 5 presents final remarks and future work.

2. Architecture

In this section, we discuss our view of the SDWSN architecture and our implementation approach, considering the target operating system. Our SDWSN architecture contains three components as shown in Figure 1, namely a southbound (SB) protocol, a neighbor discovery (ND) protocol and a controller discovery (CD) protocol.

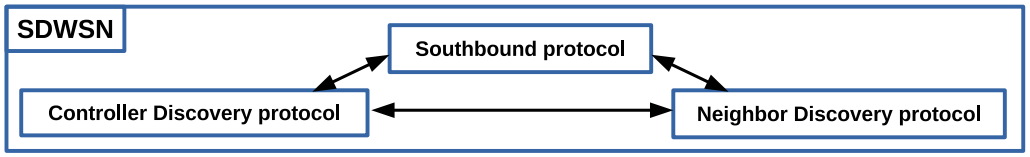


Figure 1. SDWSN architecture

The purpose of the SB protocol is to define the communication procedure between controller and SDN-enabled devices, including packet format definition, how to process these packets (Section 2.1) and the operation workflow (Section 2.2).

We devised a custom SB protocol based on TinySDN [de Oliveira et al. 2014] in the sense that we also employ the "flow id" concept, i.e. packets are routed according to a predefined label. Some control packets are routed by address, as detailed in Section 2.1. Like TinySDN, we also use an underlying protocol to perform ND and CD, but we do not demand a predefined protocol nor use it to transmit control packets.

A ND protocol obtains and maintains node neighborhood information, while a CD protocol identifies a next hop candidate to reach the controller. In the case of static networks, the former may be executed just at network bootstrapping, conversely, it should run continuously in networks with mobility and changing links. The latter is used mostly at network bootstrapping, before the node gets routing information from the controller.

We designed both ND and CD protocols as separate entities due to the existence of many suitable alternatives to theses tasks [Chen and Bian 2016]. With this approach a researcher can easily change and compare the performance of different algorithms, or design and implement a new algorithm and integrate it with IT-SDN framework; also, a developer can choose the best alternative according to the application scenario.

To provide this flexibility, a ND protocol must conform to a predefined interface, which contains two functions (initialization procedure and packet reception) and generates events upon neighborhood changes. Similarly, there is an interface for CD protocols

that provides a function to get the current best neighbor candidate to reach the controller, initialization, packet reception and event generation (in case of candidate change).

2.1. Packet types

This section lists all packet types supported by IT-SDN, their content and how each of them should be processed.

Packet header: Every packet contains a type identifier, a "time-to-live" field, a sequence number and the source address. These values compose the packet header.

Flow request: These packets are always destined to the controller, therefore are routed according to the nodes flow id table. They contain a query about how to route a certain packet, in case of a flow table miss-match. This packet has two versions: one for solving unknown flow id-oriented flows and another for solving address-based routes.

Flow setup: The purpose of this packet is to insert or change a flow table entry in a certain network node. It can be sent in response to a flow request or in accordance to the controller routing policies (for instance, if a better route is discovered). This packet contains a destination address (hence routed by the address flow table), a route identifier, the action to execute upon receiving a packet with such identifier and an action parameter. The route identifier is a flow id or an address, while the action is “drop packet”, “receive packet” or “forward packet” and the action parameter is the next hop (if applicable). The framework could be further extended with other actions.

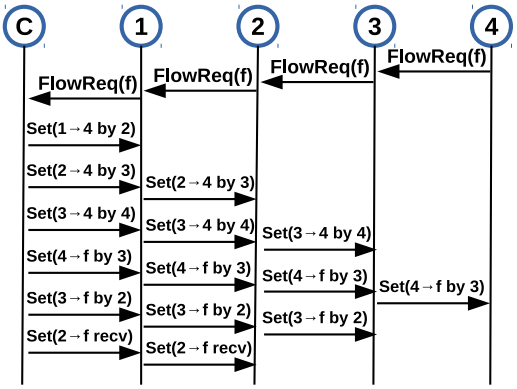
Source routed flow setup: These packets serve the exact same purpose of regular flow setups. However, flow setups are routed according to address flow tables, which have limited size. Therefore, nodes near the controller, which have to know how to reach a large number of nodes, could overflow the maximum number of entries in the address flow table in large networks. In order to avoid this undesired behavior, we added the possibility of source routing flow setup packets, bypassing the use of address flow tables. Since the controller already has a global view of the network, it is able to calculate the source route header information.

Multiple flow setup: This is a generalization of source routed flow concept, in the sense that it is able to set a route in many nodes with one single packet that is source routed (and therefore does not use address flow table). The packet is sent to the first node that should have its flow table altered, which in turn forwards the packet to the "next hop" field of the freshly configured flow table entry.

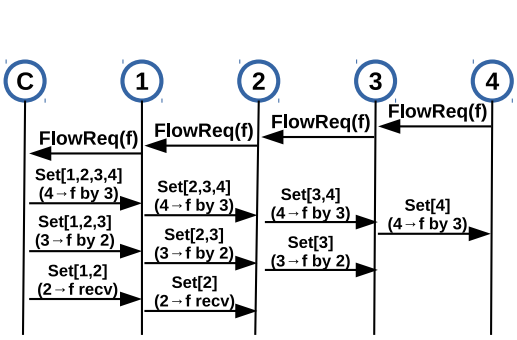
Figure 2 shows a comparison of the three flow setup possibilities, considering a 5-node linear topology (a controller and four nodes). In this example, node 4 asks how to route flow “f”, destined to node 2. Using regular flow setups, 15 control messages are transmitted or forwarded. Source routed and multiple flow setups use 9 and 6 messages, respectively.

Acknowledgement: Control packets may require delivery confirmation, what is provided by an acknowledgement packet, which relies on the sequence number.. For instance, it is used to ensure that the policy set by the controller has reached the nodes.

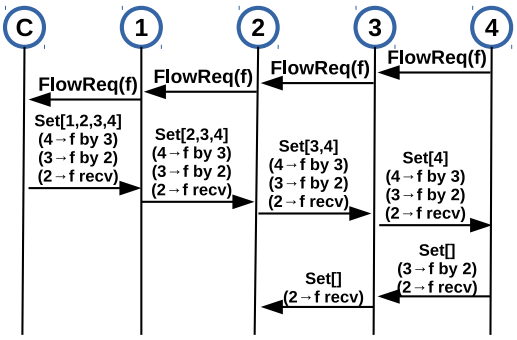
Neighbor discover / controller discovery: Packets used by the underlying ND/CD protocols. These packets are not processed by the SDN core, they are sent directly to the



(a) Regular flow setup: 15 transmissions



(b) Source routed flow setup: 9 transmissions



(c) Multiple flow setup: 7 transmissions

Figure 2. Comparison of flow setup types: setting flow "f" from node 4 to node 2

corresponding module.

Neighbor report: Nodes send this packet to update the controller network representation. It contains a list of the current node neighbors associated with a link metric (e.g. ETX, RSSI or LQI). The underlying ND protocol determines how often it is transmitted.

Data packet: Packets sent by the application layer, routed according to the flow id table.

2.2. Workflow

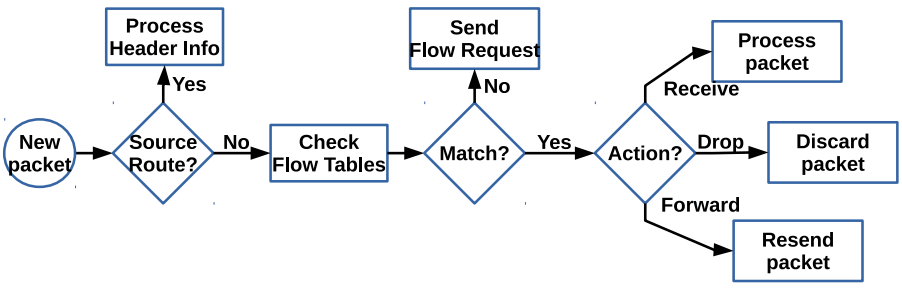


Figure 3. IT-SDN Workflow

An SDN-enabled node operates according to the workflow presented in Figure 3. Upon receiving a packet, the node checks whether it is routed by the flow tables or source routed. Source routed packets are always forwarded according to the header, while all other packets should match a routing rule in one of the flow tables. If the node does not

find a matching rule to process the packet, a flow request is sent to the controller. The node may store the packet until it gets a corresponding flow setup. If a matching rule is found, the corresponding action is executed, namely process the packet, forward it to a next hop or drop it. Other actions could be included.

On IT-SDN, the main sources of energy consumption are route requesting and route maintenance. Route requesting is based on the workflow above, and its overhead is decreased by the enhanced flow setup packets presented in Section 2.1. The larger the distance between node and controller, more energy is spent to configure a route. This can be alleviated by using multiple controllers, which could be closer to the nodes. Route maintenance is tightly connected to the underlying ND protocol, as it is responsible for detecting link status. Fine tunning ND message interval and link change threshold may greatly impact energy consumption.

3. Software Architecture

Considering the general architecture presented in Section 2, we instantiated a compliant software architecture targeting Contiki OS [Dunkels et al. 2004], an open source lightweight operating system for the Internet of Things that supports many devices (such as TelosB and SensorTag).

The software architecture components exhibited in Figure 4 maps to the three main components of the general SDWSN architecture: ND and CD protocols are clearly distinguished, while `process packet` and `core` cover the SB protocol.

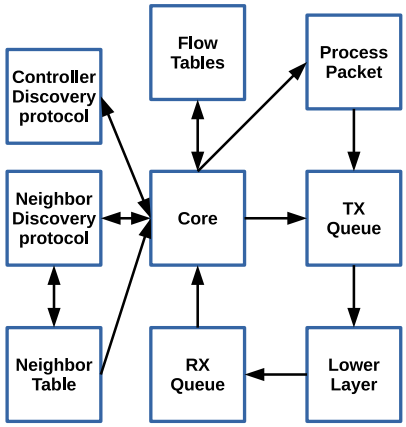


Figure 4. IT-SDN software architecture

Additionally there are four auxiliary components, namely the flow tables, a neighbor table and RX/TX queues. The flow tables store routing information set by the controller, divided into two tables: flow id table and address flow table. Their fields were already discussed in Section 2.1, with the addition of two columns, the number of times the entry has been used and the age of entry. These extra fields are used to decide which entry to overwrite in case the flow table overflows. An example is provided in Figure 5.

The neighbor table provides a common data structure for the ND protocol to write discovered information, so the SDN core can read it and transmit neighbor report packets. The RX/TX queues are buffers to avoid dropping packets in case of network congestion or long delays on packets processing.

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

Figure 5. Example of flow table

The `core` component orchestrates all other modules on an event based fashion as shown in Figure 6. Neighbor report packets are sent to the controller upon an event from ND protocol. The CD protocol event is used to set the controller route for the first time. It uses a reserved flow id number to identify the controller flow. With this convention, nodes can send requests to the controller without knowing its actual address or the actual number of controllers.

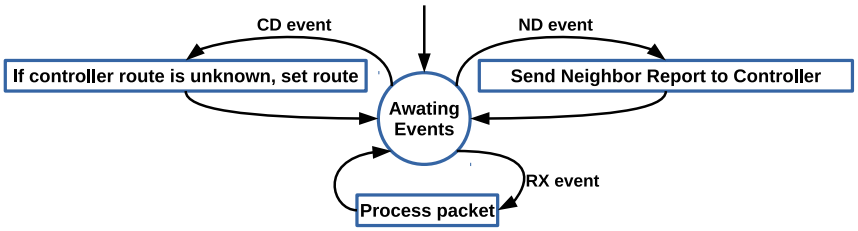


Figure 6. IT-SDN core operation

Received packets are processed as explained in Sections 2.1 and 2.2. It first checks the packets routing category (flow id, address or source routed) and then performs the associated action. This component also keeps a list of packets that are awaiting an appropriate flow table entry.

3.1. Implementation Details

We implemented the SDWSN protocol as a Contiki *network driver*, using a C language *struct* that abstracts layer 3 protocols. Our implementation could replace the existing layer 3 protocols available in the OS (IPv6 and RIME stack). In order to use Contiki's implementation of the Collect Protocol[1] as the SDN ND/CD protocols, we made our implementation compatible to the RIME stack by adding a compatibility byte to the SDN packet header (`github.com/contiki-os/contiki/blob/master/core/net/rime/collect.c`). Additionally, we use Contiki Link Layer Security (LLSEC) stack as a decision point to forward packets to SDN or RIME stack. We chose this approach to avoid modifying Contiki's code.

3.2. Controller

We provide a controller software along with the SDN-enabled node code. A Contiki node is used as a bridge between the wireless network and the controller software running at a PC. The Contiki node runs ND and CD related tasks, in addition to performing radio transmission and receptions

[1] `github.com/contiki-os/contiki/blob/master/core/net/rime/collect.c`

in behalf of the PC. The communication between PC and Contiki node is achieved through a serial (USB) connection (or TCP connection in case of COOJA simulations).

The PC software maintains a centralized global view of the network stored as a graph, according to the neighbor report packets received from the network nodes. This representation is used to calculate flow requests from the nodes, and to update flow tables according to Dijkstra shortest path algorithm. The flow table cache maintains a copy of routes previously configured on the nodes. Neighbor report messages may change the controller network view and trigger route recalculations. The flow table cache is used to avoid sending flow setup messages of unchanged routing rules. For instance, if a previous route $A \rightarrow B \rightarrow C \rightarrow D$ is recalculated to $A \rightarrow N \rightarrow C \rightarrow D$, node C flow table does not need to be updated.

3.3. IT-SDN website

IT-SDN is available at `http://www.larc.usp.br/~cbmargi/it-sdn`, including the source code and its license. The website also includes the installation and developer manuals. In order to run IT-SDN, it is necessary to use Contiki 3.0 and mspgcc LTS 20120406 (4.6.3), as detailed in the installation manual. The controller runs on Linux (we tested on Ubuntu 16.04 64bit) with Qt 5.8.

4. Demo description

This demonstration has tree main steps. First we will show how to program a WSN application using IT-SDN and run it on the COOJA [Osterlind et al. 2006] environment, emulating TelosB[2] mote. Next we will present an overview of IT-SDN main features, including CD, ND and SB message exchanges, which will create the necessary flows to deliver data from the previously created application. Lastly we will show how to evaluate the performance of an IT-SDN deployment, obtaining end-to-end delay, delivery rate, and control plane overhead.

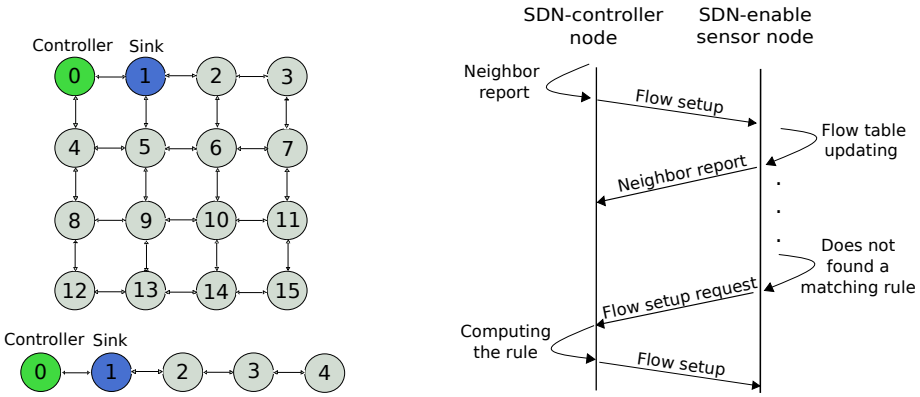
The application designed for the demonstration senses the temperature every 10 seconds and then sends its value to the sink. The demonstration of IT-SDN message exchange will occur in two phases. First we will use COOJA. For this setup we use a 4x4 grid topology, which includes one controller, one sink, and fourteen sensing nodes, as shown in Figure 7(a). Figure 7(b) depicts the message exchange, which can be visualized in the COOJA mote output. Once the controller knows its neighbors (event identified by a Neighbor Report), it sends flow setup messages, that cause the nodes to update their flow table and reply with a neighbor report. Next, if a node receives a packet for which it does not have a matching rule, it sends a flow setup request to the controller, which in turn processes the request and replies with a flow setup.

Lastly, we will execute the application developed in a small chain topology composed of 3 sensing nodes, 1 sink and 1 controller, as shown in Figure 7(a). We use the LEDs to indicate the node current state. Sensing nodes have 4 states: (1) all LEDs are off when the node does not have a route to controller; (2) red LED on indicates the node has a route to controller; (3) green LED on indicates the node has the data flow route; and (4) blue LED indicates the node is sending a Neighbor Report to the controller. For the sink, each LED is associated with a sensing node, i.e., it toggles the corresponding LED when the sink receives a data packet.

5. Final remarks

This paper presented IT-SDN, an SDN-based WSN framework. Its main features include OS-independent specification, clearly defined boundaries of underlying neighbor discovery protocol,

[2] `www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf`



(a) Topologies used in COOJA (top) and testbed (bottom)

(b) Messages exchange diagram

Figure 7. Topologies and messages exchange diagram for demonstration

support to source routed and multiple flow setups and statistics gathering. Future work includes integrating security mechanisms, including more complex flow table actions and enhancing controller decision mechanisms.

Acknowledgements

Authors would like to thank Filipe Calasans for helping with serial communication.
C. B. Margi is supported by CNPq research fellowship #307304/2015-9. R. C. A. Alves is supported by CNPq PhD fellowship #155372/2016-5. Gutavo N. Segura is supported by CAPES-PROEX fellowship #0212083 and the University of Costa Rica.

References

Chen, L. and Bian, K. (2016). Neighbor discovery in mobile sensing applications: A comprehensive survey. *Ad Hoc Networks*, 48:38–52.

de Oliveira, B. T., Margi, C. B., and Gabriel, L. B. (2014). TinySDN: Enabling multiple controllers for software-defined wireless sensor networks. In *IEEE LATINCOM*.

Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. In *IEEE LCN*, pages 455–462.

Galluccio, L., Milardo, S., Morabito, G., and Palazzo, S. (2015). SDN-WISE: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks. In *IEEE INFOCOM*, pages 513–521.

Kobo, H. I., Mahfouz, A. M., and Hancke, G. P. (2017). A survey on software-defined wireless sensor networks: Challenges and design requirements. *IEEE Access*.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., and Voigt, T. (2006). Cross-level sensor network simulation with cooja. In *IEEE LCN*, pages 641–648.