TinySDN: Enabling TinyOS to Software-Defined Wireless Sensor Networks

Bruno T. de Oliveira[1], **Cíntia B. Margi**[1]

[1]Escola Politécnica – Universidade de São Paulo
Departamento de Engenharia de Computação e Sistemas Digitais

{brunotrevizan,cintia}@usp.br

Abstract. *Software-Defined Networking (SDN) has been envisioned as a way to reduce the complexity of network configuration and management, enabling innovation in production networks. While the first SDN approaches focused on wired networks, there were proposals specifically for Wireless Sensor Networks, but none of them targeted the TinyOS environment. This paper presents the TinySDN tool, a TinyOS-based Software-Defined Networking implementation. It comprises two main components: the SDN-enabled sensor node, which has an SDN switch and an SDN end-user device, and the SDN controller node, where the control plane is programmed. The SDN features have been implemented and tested in a real mote, and will be shown in the demo proposed.*

1. Introduction

Wireless Sensor Networks (WSN) have been used to support several different applications, mainly related to monitoring and detection. Nodes in a WSN are typically battery-powered and resource constrained (i.e. limited amount of memory, processing and communication), and communicate through a multihop ad hoc network [Culler et al. 2004].

Software-Defined Networking (SDN) has been envisioned as a way to reduce the complexity of network configuration and management, initially focused on wired networks. The main approach to SDN is OpenFlow[1], which focus on wired networks.

SDN for WSN imposes different challenges and requirements, but provides several opportunities. Among the challenges we highlight limited resources of WSN nodes: energy, processing, memory, and communication. Requirements are related to the applications characteristics (e.g. data frequency and size), as well to the nodes behavior due to duty-cycles, operating systems and programming approach. On the other hand, opportunities provided by SDN include: to improve resource reuse, to implement node retasking, node and network management, as well as to enable experiments with new protocols, and to ease transition to standard protocols for deployed networks in WSN and Internet of Things context [de Oliveira et al. 2015].

The main proposals in the literature concerning SDN for WSN are: Flow-Sensor [Mahmud and Rahmani 2011], Sensor OpenFlow [Luo et al. 2012] and SDWN [Costanzo et al. 2012]. Flow-Sensor [Mahmud and Rahmani 2011] proposes sensor nodes with the main features of OpenFlow. Sensor OpenFlow and SDWN propose a clear separation between data plane and control plane, a centrally controlled communication protocol between these two planes and some data plane features.

[1]https://www.opennetworking.org/sdn-resources/openflow/

Gante et al. [Gante et al. 2014] propose a framework to apply SDN to WSN management. Besides the benefits highlighted by previous works, authors include accurate localization and topology discovery as advantages of SDN usage in WSN. The SDN controller should be implemented as part of the WSN sink.

TinySDN [de Oliveira et al. 2014] is a flow-ID-based approach that improves on previous work by addressing the use of multiple SDN controllers and by discussing common WSN characteristics, such as (i) in-band control, opposed to SDN implementation in wired networks that can leverage from out-band control; (ii) higher communication latency; (iii) smaller link layer frames; and (iv) limited energy supply. Furthermore, the paper describes the protocol for communication between controllers and WSN nodes, and discuss implementation issues and approaches for TinyOS [Hill et al. 2000].

SDN-WISE [Galluccio et al. 2015] defines simple mechanisms for the definition and handling of the Flow Table that make it stateful, pursuing two goals: (i) to reduce the amount of information exchanged between sensor nodes and the SDN network controller, and (ii) to make sensor nodes programmable as finite state machines, so enabling them to run operations that cannot be supported by stateless solutions.

Given the lack of available SDN implementations for TinyOS, a widely used WSN operating system, we present TinySDN tool. It enables the SDN paradigm, making testing new protocols easier and providing multiple controllers for software-defined WSN. Each sensor node is composed of an SDN switch and an end-user device, which we call *SDN-enabled sensor node*. The control plane is programmed through a central component, which we call *SDN controller node*, thus centralizing the control plane of the WSN.

2. The TinySDN Tool

This section presents the TinySDN architecture overview by describing its components and protocols.

Figure 1 depicts the two TinySDN types of nodes: *SDN-enabled sensor node* and *SDN controller node*. Each *SDN-enabled sensor node*, where data plane components are installed, connects through multi-hop wireless communication to an *SDN controller node*, where the control plane logic is executed, allowing the interaction between the two planes.

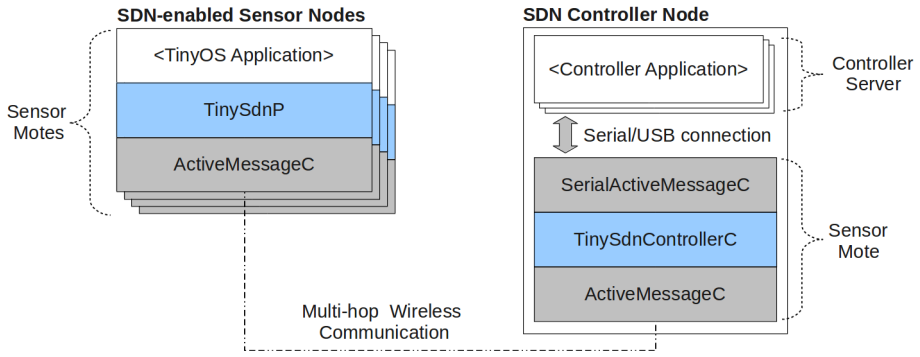

Figure 1. TinySDN Architecture Components [de Oliveira et al. 2014]

SDN-enabled sensor node is the component that runs on sensor motes, providing the forwading service to applications as a network protocol. As discussed

in [Luo et al. 2012], end devices are considered peripheral to SDN and hence out of the scope of OpenFlow, the main SDN project nowadays. On the other hand, sensor nodes behave like end devices by generating data packets to transmit sensed data, in addition to merely forwarding data as SDN switches do. Thus, *SDN-enabled sensor node* is a type of node that plays both roles: SDN switch and SDN end device.

SDN Controller Node is the node (or nodes in case of multiple controllers) that performs SDN controller tasks, i.e., executes the control plane: keep a network topology view, and apply definitions of SDN applications by creating and managing network flows using the specific protocol described in [de Oliveira et al. 2014]. Each *SDN-enabled sensor node* must find an *SDN controller node* to assign and then start to send network topology information and request flow specifications when necessary.

Next, we describe the specification of flows and actions protocol, the two types of nodes, as well the procedures to *SDN-enabled sensor nodes* find an *SDN controller node* and to establish communication with it, and to collect network topology information.

2.1. SDN-enabled Sensor Node

As shown in Figure 1 (left), *SDN-enabled sensor node* is composed of three parts: *ActiveMessageC*, *TinySdnP*, and *TinyOS Application*.

The *ActiveMessageC* is a TinyOS component that manages and provides programming interfaces to interact with the radio module of the sensor node, which comprehends the functions of link and physical layers. It is used by TinySDN to perform all tasks related to medium access and wireless communication, such as link quality estimation and control/data packet sending/receiving.

The *TinySdnP* is the main component of TinySDN, which is responsible for checking if a received packet matches a flow in the `flow table` and then perform the related action, or otherwise sends a flow request to an *SDN controller node*. Thus it is responsible for performing `flow table` update when receiving a flow setup response.

The *TinyOS Application* part is the equivalent to end-user device; it generates data packets and then places them on the network using the programming interface provided by the TinySDN component. It should be written by the network programmer, according to the WSN application, but we provide an example of this.

2.1.1. Finding a Controller and Establishing Communication with It

The first task of an *SDN-enabled Sensor Node* at network startup is to find an *SDN Controller Node* and establish communication with it in order to send control packets. For this, TinySDN employs the collection tree protocol (CTP) [Gnawali et al. 2009] as underlying protocol. In addition to being a widely used protocol in TinyOS-based applications, we selected CTP because of two features:

- **Hardware independence –** it adopts as metric the `ETX` value given by TinyOS Four-Bit Link Estimator Component [Fonseca et al. 2007], a software component that estimates the link quality between single-hop neighbors from the amount delivered or lost messages instead of using the traditional received signal strength indication (RSSI), a specific and hardware-dependent feature.

- **Multiple SDN controllers –** it uses a tree routing to deliver data through the network to a sink node, named root. Each node unconsciously adopts a root by sending data through the route with the lowest `ETX` sum. Thus, when multiple roots are announced each node adopts the root with best `ETX` without being aware that there is more than one of them, which builds a collection tree for each root.

2.1.2. Network Topology Information Collection

Network topology information collection is an important feature of TinySDN that enables the *SDN controller node* to build a view of the network by receiving information about sensor nodes and wireless links between them. It consists in each *SDN-enabled sensor node* recognizing its neighbors and measuring the quality of links with them, and then sending this information to the *SDN controller node* through the CTP route.

In order to recognize the neighborhood *SDN-enabled sensor nodes* uses the Four-Bit Link Estimator broadcast `beacon packet` and wait for responses. Upon receiving responses, each response sender is added to the `neighbor table` including the link quality metric (`ETX`). To allow us to obtain the neighbor table we needed to modify the Link Estimator interface and add a command as well.

Table 1 contains an example of `neighbor table`. The *Neighbor Id* column contains neighbors `TinyOS node id` (network node address) and the Link Quality column contains neighborhood measured link quality to correspondent neighbor. A lower `ETX` value means better quality, and infinity means that link was not sufficiently evaluated or the link is not available at the moment.

Table 1. Neighbor Table example.

Neighbor Id	*Link Quality (ETX)*
3	10
5	50
9	∞

2.1.3. Provided Programming Interfaces

The TinySdnP component provides to TinyOS application programmer two main interfaces:

- AMSend – it is an interface that enables sending data through the TinySDN tool protocol by calling the `send` command function, which receives three parameters: `data flow id` that identifies the flow to be used, a pointer to the `message packet` to be sent, and packet `length`. Furthermore, when instantiating this interface, it is mandatory to implement the `send done` event, which handles results (callbacks) of send command calls.
- Receive – it is an interface that enables receiving data through the TinySDN tool protocol by handling the packet reception event in a callback function implementation. This receive event implementation receives from the lower software lay-

ers the following parameters: `received packet`, a pointer to the packet's `payload`, and packet `length`.

2.2. SDN Controller Node

As observed in Figure 1 (right), it is composed of two different modules to be installed in different devices: sensor mote module and controller server module, described below.

- **Sensor mote module:** this module, which runs on a sensor mote, is responsible for communicating with *SDN-enabled sensor node* using *ActiveMessageC*. It uses *SerialActiveMessageC* to forward received messages from network to the *controller server module* and to forward messages from it to the network. The *TinySdnControllerC* part adapts messages and manages this communication.
- **Controller server module:** module that concentrates the control plane logic, i.e., keep a network topology view, hosts controller applications and apply definitions of it by creating and managing network flows. This module should run over a device with more resources that a sensor mote, such as a credit-card-sized computer running a Linux distribution, a laptop, a desktop or even a server. TinySDN defines the interfaces to be used in order to interact with the *sensor mote module*, and we provide a use case as a simple controller application, but more elaborated scenarios and applications could be defined by the network programmer.

2.3. Specification of Flows and Actions

A flow and its actions are a set of instructions used by the *SDN controller node* to program *SDN-enabled sensor nodes* behavior by sending flow setup packets that are translated into entries on `flow tables`. Thus, *SDN-enabled sensor nodes* are responsible for classifying packets in different flows and performing its corresponding actions.

TinySDN includes specification of two types of flows: control flow and data flow.

Data flows are used to forward data packet generated by applications through the sensor network. Three data flow actions are specified in TinySDN:

- Forward – classified data packet should be forwarded to a next hop sensor node specified in the action parameter field.
- Receive – classified data packet should be forwarded to application layer, i.e., delivered to the application.
- Drop – classified data packet should be dropped.

Data flows are specified as entries in the `data flow table`, where each entry is composed of four fields: *Flow ID*, field that identifies the flows and is used for packet classification; *Action*, which specifies the action to be performed; *Action Parameter*, field that contains a specific value related to action (e.g., in case of "Forward" action it specifies the next hop node id); and *Count*, field incremented when packets are matched to the flow entry, providing statistics. Table 2 (left) contains a `data flow table` example.

Control flows are used to forward control packets from the *SDN controller node* to *SDN-enabled sensor nodes*, i.e., it is just used to establish downward routes to control packets, since upward routes (from controller to sensor nodes) establishment is performed by CTP. Control flow entries have a unique and default action: forward; it is because only the *SDN controller node* generates control packets and the control flow identification is the node id (or address). A control flow table example is presented in Table 2 (right), it includes two fields: *destination node id* and *next hop node id*.

Table 2. Data Flow Table and Control Flow Table examples.

Flow ID	*Action*	*Action Parameter*	*Count*
2	Forward	5	2
9	Receive	-	5
11	Drop	-	2
34	Forward	10	50

Destination Node Id	*Next Hop Node Id*
5	5
10	5
13	5
15	3

3. Demo Proposal

The proposed demo will depict a WSN application that senses and transmits such data to a sink periodically (every 2 seconds) using TinySDN as communication protocol. Its purpose is to provide an overview of the TinySDN main features, focusing on the flow creation process and the use of such flows to deliver data from source sensor nodes to destination nodes. Therefore, the demo is designed to enable the visualization of all network events related to those two tasks.

To achieve the goal we use COOJA [Osterlind et al. 2006]. COOJA is a cross-level sensor network simulator that adopts a hybrid approach being able to simulate the network level through the environment implemented in Java, and to emulate the operating system level by running native sensor node program integrating it with simulated network environment using the Java Native Interface. Since COOJA runs cross-compiled binaries for the TelosB[2] mote by employng the TI MSP430 emulator in its environment, it is enabled to emulate/simulate TinyOS applications. By adopting an emulation/simulation environment, the demonstration is not susceptible to uncontrolled interference. Furthermore, COOJA provide a graphical user interface and features to capture and visualize mote output and network messages, such as beacon, control, and data packets.

In the demonstration, we adopt two topologies: grid topology (or mesh), which is very common in WSN application and illustrates a network with medium to high density; and string topology to illustrate a low-density network. The flows setup are executed reactively for both types, data flows and control flows, as depicted in Figure 2.

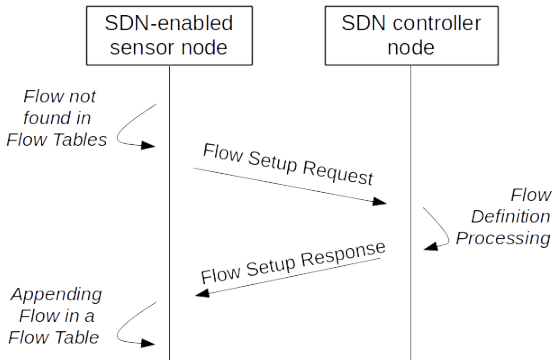

Figure 2. Reactive flow setup

[2]http://www.memsic.com/userfiles/files/Datasheets/WSN/6020-0094-02_B_TELOSB.pdf

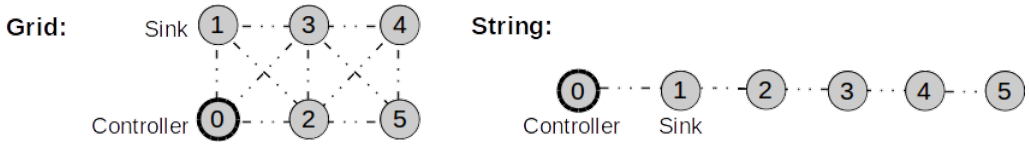

Figure 3. WSN topologies for demonstration

Once the control messages are exchanged, nodes will be able to send data to the sink. As depicted in Figure 3, the sensor nodes whose *node id* range from 1 to 5 are deployed with TinySDN, becoming *SDN-enabled sensor nodes*. In this demo they generate rules requests and perform the forwarding task according to the *Controller* instructions. Additionally, the nodes whose *node id* are 3, 4 or 5 transmit a counter (simulating data sensed). The node that *node id* $= 1$ is set as the data sink. A small number of nodes was selected in order to allow one to follow and understand the messages exchanged and displayed by COOJA. We could also increase the number of nodes to demonstrate TinySDN working with larger networks.

4. Used Resources, Source Code and Documentation Availability

TinySDN was implemented based on TinyOS[3] and its components, which we highlight: the collection tree protocol and the 4-bit Link Estimator. Thus, the TinyOS toolchain, including the build system, is a requirement to compile our tool.

We used TelosB mote as a platform during the development process and for the validation tests. In theory, TinySDN can run on any TinyOS-supported target hardware; however, additional tests should be performed for a reliable deployment based on other platforms. Also, we used COOJA to perform network environment tests.

TinySDN tool source code and the related documentation are available at `http://www.larc.usp.br/~cbmargi/TinySDN/`.

5. Considerations and Future Work

We introduced TinySDN, a tool designed to enable the TinyOS compatible platforms to Software-Defined Networking and allows the use of multiple controllers. Among the main benefits of using SDN paradigm in WSN scenarios, we highlight easier network protocol development and its hypothesis testing, as well as wireless sensor network management.

Currently, we are developing a controller with more functionalities in order to replace the current standalone controller. We also expect contributions from the community interested in improving the tool. Another future work topic is to extend matching rules by adding flexible bit masking capability, which enables interoperability with other protocols, but it implies a higher cost of communication and memory usage.

Acknowledgment

This work is funded by São Paulo Research Foundation (FAPESP) under grants #2013/15417-4 and #2014/06479-9. Cíntia Borges Margi is supported by CNPq research fellowship #307304/2015-9. The authors would like to thank the Laboratory of Computer Networks and Architecture of USP for providing the necessary infrastructure.

[3]Our implementation is based on the most recent released version of TinyOS (2.1.2), released on August 20, 2012. More information can be found at `http://www.tinyos.net`.

References

[Costanzo et al. 2012] Costanzo, S., Galluccio, L., Morabito, G., and Palazzo, S. (2012). Software defined wireless networks: Unbridling sdns. In *Proceedings of the 2012 European Workshop on Software Defined Networking*, EWSDN '12, pages 1–6, Washington, DC, USA. IEEE Computer Society.

[Culler et al. 2004] Culler, D., Estrin, D., and Srivastava, M. (2004). Overview of sensor networks. *Computer Magazine*, 37(8):41–49.

[de Oliveira et al. 2015] de Oliveira, B. T., Alves, R. C. A., and Margi, C. B. (2015). Software-defined wireless sensor networks and internet of things standardization synergism. In *Standards for Communications and Networking (CSCN), 2015 IEEE Conference on*, pages 60–65.

[de Oliveira et al. 2014] de Oliveira, B. T., Margi, C. B., and Gabriel, L. B. (2014). TinySDN: Enabling multiple controllers for software-defined wireless sensor networks. In *Communications (LATINCOM), 2014 IEEE Latin-America Conference on*, pages 1–6.

[Fonseca et al. 2007] Fonseca, R., Gnawali, O., Jamieson, K., and Levis, P. (2007). Four bit wireless link estimation. In *Proceedings of the Sixth Workshop on Hot Topics in Networks (HotNets VI)*.

[Galluccio et al. 2015] 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 *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 513–521.

[Gante et al. 2014] Gante, A. D., Aslan, M., and Matrawy, A. (2014). Smart wireless sensor network management based on software-defined networking. In *Communications (QBSC), 2014 27th Biennial Symposium on*, pages 71–75.

[Gnawali et al. 2009] Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., and Levis, P. (2009). Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 1–14. ACM.

[Hill et al. 2000] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K. (2000). System architecture directions for networked sensors. *SIGPLAN Notices*, 35(11):93–104.

[Luo et al. 2012] Luo, T., Tan, H.-P., and Quek, T. Q. S. (2012). Sensor openflow: Enabling software-defined wireless sensor networks. *IEEE Communications Letters*, 16(11):1896–1899.

[Mahmud and Rahmani 2011] Mahmud, A. and Rahmani, R. (2011). Exploitation of openflow in wireless sensor networks. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, volume 1, pages 594–600.

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