Project PtOWSN: Modeling OpenWSN MAC Layer using Ptolemy II

Antonio Iannopollo, Ben Zhang
EECS, UC Berkeley
{antonio, benzh}@eecs.berkeley.edu

Abstract

In this project, we model and implement the MAC layer of OpenWSN with Ptolemy II. Our focus is on the state machine implementation of IEEE802.15.4e standard that controls the physical radio and interface with upper network layers. With such granularity, our framework is suitable for studying many interesting features of the time-slotted channel hopping protocol. We present a preliminary study of the energy consumption and time synchronization in a simple multi-hop data transmission application.

1 Introduction

With the emerging *Internet of Things* paradigm, standardization progress has been made to unify the communication between low-cost, low-speed ubiquitous wireless devices. In particular, IEEE 802.15.4 [1] was defined to specify the physical (PHY) layer and media access control (MAC) for low-rate wireless personal area networks (LR-WPANs). Though not reliable and deterministic, the main MAC strategy is still carrier sense multiple access with collision avoidance (CSMA/CA). A new IEEE802.15.4e Time-Slotted Channel Hopping (TSCH) standard [2], which achieves high reliability through frequency agility (channel hopping) and low-power through time synchronization is under development. The OpenWSN [7] effort at Berkeley aims to provide an open-source implementation of the complete IEEE802.15.4e protocol stack on a variety of software and hardware platforms.

In the OpenWSN project, researchers have developed OpenSim and OpenVisualizer, which enable simulating and visualizing an OpenWSN network without the need for physical devices. Though this has greatly facilitated the development of OpenWSN-oriented applications, their simulator works by compiling the firmware and run it on a regular PC. Such a simulator[1] is tremendously useful for debugging the firmware code since you can "freeze" the execution and inspect variables. But it's not intended for large-scale simulations and network characteristic analysis. From the original paper [7], authors stated that the goal of OpenSim is to demonstrate that *"it is possible to build the OpenWSN stack and applications, and emulate a full network on Windows or Linux"*. From one of the discussions with the first author, we know that *"with 4 nodes, you are simulating at the speed of 1X [of the real-time]"*. Therefore, we need a more systematical way of modeling OpenWSN protocol for both understanding and assessing various characteristics of the network.

In this project, we model the OpenWSN protocol using Ptolemy II [3, 4]. Though many other platforms [5, 6] exist for network simulation, we pick Ptolemy because it has well-defined models of computation and many useful features (including multiform time, an existing wireless domain, etc.) for modeling such distributed systems. Also, there is a growing community that is using Ptolemy to build wireless sensor network applications. The Ptolemy OpenWSN (PtOWSN) implementation would serve as the platform for them.

2 Background on OpenWSN

In this section, we provide a brief overview of the OpenWSN protocol stack. Interested readers can read [7, 2] and visit OpenWSN website[2] for more information.

Slotframe: IEEE802.15.4e defines a *slotframe* structure. A slotframe is a group of temporal slots which is repeated over time. Current OpenWSN implementation uses 15ms slots. A *schedule* tells the node whether it should transmit, receive or sleep in a particular slot. The construction of such schedule if out of the scope of IEEE802.15.4e.

Time Synchronization: When a node joins a network, it aligns its slot boundary with other nodes (slot synchronization) and set its Absolute Slot Number (ASN) to the network ASN (ASN synchronization). In this way, all nodes will enter in a new slot at the same time (following their pre-defined schedule).

[1] "emulator" is a better term.

[2] https://openwsn.atlassian.net/wiki/

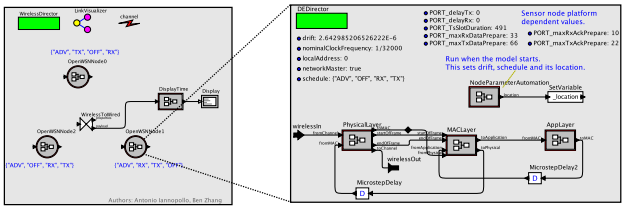


Figure 1. An example application of three nodes (*left*) and the simplified network stack (*right*).

Re-synchronization: Since the clock on the embedded platforms is not perfect (typical clock drift is around 10 parts-per-million (ppm)), a re-synchronization mechanism is needed. OpenWSN performs re-synchronization whenever a packet transmission happens. Each node picks as reference the neighbor that is closer to the root of the network as its time master, and the synchronization is achieved by aligning its next slot boundary to the master's. When there is no packet transmission, a *KeepAlive* packet is generated (every 30 seconds is a typical value).

OpenWSN State Machine: In each slot, given the schedule, the node behaves according to a state machine. This state machine defines how the synchronization is reached and maintained, as well as transmission and reception. We defer the discussion here because this will be the main focus of our modeling work described in the next section.

Packet format: the OpenWSN stack defines several types of packet. In our work, we consider three of them, related to the MAC layer: `ADV`, `DATA`, `ACK`. The advertisement packet `ADV` is used to broadcast the network information (such as ASN) such that new node can join the network. The `DATA` packet encapsulates the upper layer payload. The `ACK` packet is used to acknowledge a data packet. It also contains time correction information such that the other node can perform re-synchronization (if needed).

3 Modeling in Ptolemy

In this section, we present our work in Ptolemy that models the MAC layer of OpenWSN[3].

Though our focus is on the TSCH state machine, we have also modeled a simple physical radio and abstracted the higher network layers as a single application layer (see Figure 1). The interface to the Physical layer is mainly achieved with 1) `startOfFrame` and `endOfFrame` ports, which indicate the radio status; 2) `fromPhysical` and `toPhysical`ports, which convey the payload. The interface to the Application layer is through `fromMAC` and `toMAC` ports.

[3]Though the model in each figure is vector graphics, you can zoom in and read the details. We strongly suggest reader to open our Ptolemy model for details. These figures are mainly for illustration purpose.

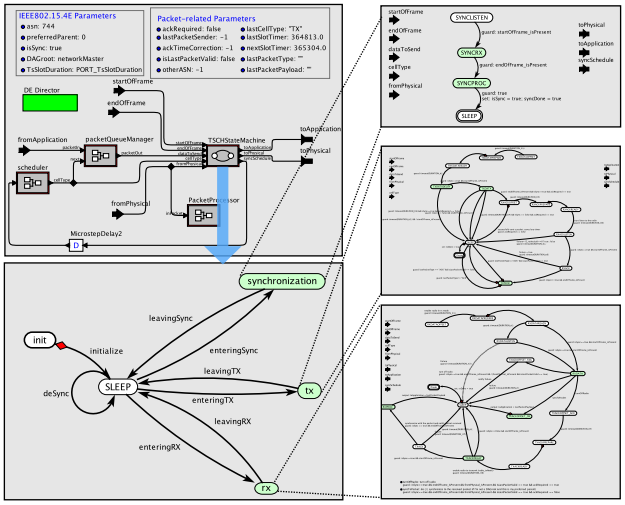


Figure 2. The state machine with refinements of TSCH protocol.

Inside the `MACLayer` composite actor, we have a few actors including `packetQueueManager`, `scheduler`, `PacketProcessor` and `TSCHStateMachine` (names are self explanatory). The most relevant part is the `TSCHStateMachine`, where we modeled the protocol behavior using Hierarchical Modal Models 2. It consists of five main states, which describe the node activity: `init`, `SLEEP`, `synchronization`, `tx` and `rx`.

During the initialization phase, we reset all protocol related parameters, When in the `synchronization` refinement, the node keeps listening for `ADV` packet and performs slot synchronization and ASN synchronization. Once the synchronization is done, it goes back to `SLEEP` state. The scheduler decides node actions:

- If the slot is `ADV` slot, then the node enters the `tx` state and sends an `ADV` packet.
- If it's `TX` or `RXTX` slot and the node has data to send (there are packets queued at `packetQueueManager`), it will enter `tx` state and send the data packet.
- If it's `RX` slot, or it's `RXTX` slot but the application doesn't have data to send, the node will enter `rx` state and listen for packets.

In `tx` and `rx` states, there is a refinement which captures the complicated state transition of the radio. For example, when the node is sending a packet, it first wait a certain time (in `TXDATAOFFSET` state), then prepares the data for radio to transmit (`TXDATAPREPARE`), and after a few other states that are used to check the radio status, etc., it finally

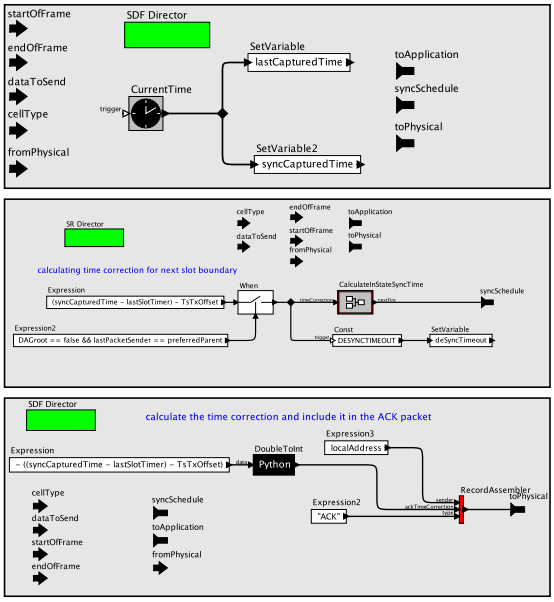


Figure 3. Modeling the the re-synchronization.

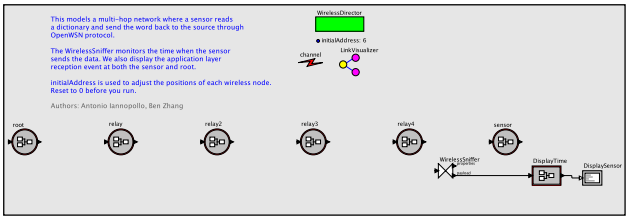
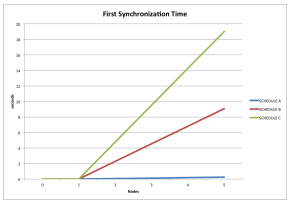
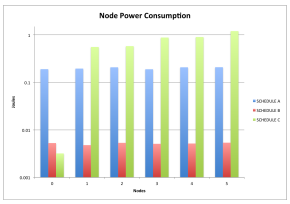


Figure 4. A multi-hop OpenWSN network.



(a) For each node, the time it first get synchronized (the time it joins the network).



(b) For each node, the measured power consumption after then network is on for 20 seconds.

Figure 5. Performance evaluation for different schedules in a multi-hop network. Schedule A, B, C are corresponding to $k = 0, 144, 306$.

enters `TXDATA` state and transmits the packet. Similar complexity exists for managing acknowledgment and receiving a packet. Given our space constraints, we refer the reader to our Ptolemy model for more details on state transition behavior.

The `synchronization` state is only entered when the node is not synchronized to the rest of the network anymore. Additionally, for every received packet, the node will capture the reception time (Figure 3 *up*). If the packet is from a time master, it will calculate the time discrepancy and use it to adjust its synchronization (Figure 3 *middle*). If the packet is from a time slave, then the node will still perform a calculation and send the time correction through `ACK` packet (Figure 3 *down*).

4 Case Study

We consider a multi-hop network which has a chain of OpenWSN nodes (see Figure 4), where each node has a different *NodeId* (assigned as an integer), increasing from left to right. Node schedules are constructed following the schema[4]:

NodeId	Schedules				
$3i+0$	`ADV`	`TX`	`RX`	`OFF`	$k \times$ `OFF`
$3i+1$	`ADV`	`RX`	`OFF`	`TX`	$k \times$ `OFF`
$3i+2$	`ADV`	`OFF`	`TX`	`RX`	$k \times$ `OFF`

where $i \in \mathbb{N}$ and k is specified to control the duty cycle of each node (higher k indicates more `OFF` in the schedule period and thus a lower duty cycle). Intuitively, when k is small, nodes can receive `ADV` packet in a relatively timely fashion. This helps especially in case of lost synchronization. However, in case of high duty cycle schedules, the price a node needs to pay is to spend more time in `TX` and `RX` states, which potentially increases the power consumption. On the other hand, if k is large, nodes that have lost synchronization will have to wait longer, leading to an increased energy consumption caused by turning the radio on and listening for long periods. Schedule duty cycle tuning is a critical activity in order achieve performance while minimizing energy consumption. In this work, we leave a formal study as future work, and only focus on cases when $k = 0, 144, 306$.

We present the results of a network that has 6 chained nodes[5] under the three different schedules in Figure 5; and use two metrics to evaluate each schedule. The first metric is the time of first synchronization for each node in the

[4]Interestingly, we are inspired by Gustove function to came up with such a schedule so that at any given time, a node is communicating only with one of its neighbors such that there will not be hidden terminal problem if time synchronization is achieved.

[5]The first synchronization figure was obtained with a simulation of 20 nodes. Given that the linear relationship is clear, we present the graph with 6 nodes for consistency with the power consumption results.

network. This metric gives us an idea of how *reactive* the network is. The second is the power consumption of each node in the network for a given period of time. This helps to evaluate the overall lifetime of network devices, which correlates to the network *durability*.

Keeping looking at Figure 5, on the left, we see that the first synchronization time is proportional to the relative distance of each node from the root (as the reader intuition suggests). In case of higher duty cycle, when k is smaller (such as with Schedule A), each node waits for a shorter time to reach synchronization.

From the power consumption side (figure on the *right*), we have found that if the duty cycle is high (k is small), then most nodes are busy in `TX` and `RX` states, leading to similar energy consumption values. When the duty cycle is low, then nodes that are farther from the root tend to stay longer in the `synchronization` state (because of the less frequent `ADV` packets), resulting in longer times with the radio in listening mode. Interestingly, schedules with a moderate duty cycle (see Schedule B), where transmissions are less frequent but not in a way to cause node de-synchronization, lead to an overall smaller amount of energy consumption.

5 Conclusion

In this project, we modeled, using Ptolemy, part of the OpenWSN protocol stack, with an emphasis on the MAC Layer TSCH protocol. Though there are still many features we haven't fully implement (such as packet retransmission), our framework has already enabled some preliminary studies in particular network cases. We showed the time synchronization performance and power consumption estimation in our case study of a multi-hop network application. For future work, we are planning for further extensions of our model, including routing layer and advanced scheduling policies, as well as Internet of Things application development.

References

[1] 802.15.4-2006. IEEE standard for information technology, local and metropolitan area networks, wireless medium access control (MAC) and physical layer (PHY) specifications for low rate wireless personal area networks (WPANs). 2006.

[2] 802.15.4e 2012. IEEE standard for local and metropolitan area networks-part 15.4: Low-rate wireless personal area networks (LR-WPANs) amendment 1: MAC sublayer. 2012.

[3] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous concurrent modeling and design in Java. Technical Report Technical Memorandum UCB/ERL M04/27, University of California, July 29 2004.

[4] J. Davis II, M. Goel, C. Hylands, B. Kienhuis, E. A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie, et al. Overview of the Ptolemy project. Technical report, ERL Technical Report UCB/ERL, 1999.

[5] S. McCanne and S. Floyd. NS network simulator, 1995.

[6] A. Varga et al. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM2001)*, volume 9, page 185. sn, 2001.

[7] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister. Openwsn: a standards-based low-power wireless development environment. *Transactions on Emerging Telecommunications Technologies*, 23(5):480–493, 2012.

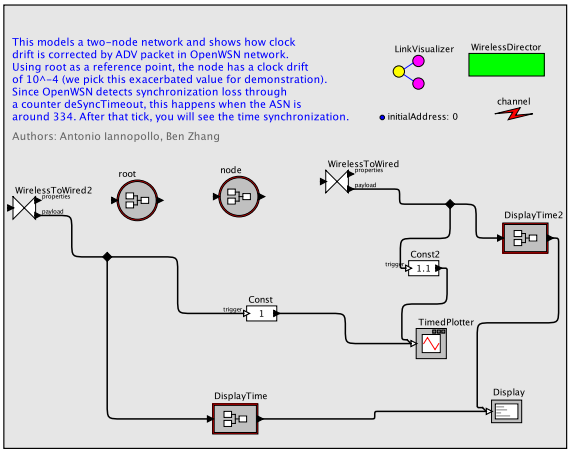


Figure 6. Ptolemy demo that shows the action of time synchronization.

A Demos

We have constructed two demos that can serve the purpose of understanding the time synchronization and multihop data transmission in OpenWSN protocol. They are named `demo_multihop_dict_nopower.xml`(see Figure 4) and `demo_sync.xml`(see Figure 6) in the accompanying folder with this report. Readers are welcome to open it and run it with Ptolemy II.