EL024A Laboratory Exercise 2: Wireless Communication with Contiki-NG

Sebastian Bader

1 Background and Goals

This lab focuses on the wireless link between two nodes. During the lab we will look at the configuration of a wireless link and a network, as well as look at point-to-point data communication. We will use the nullnet networking stack, which means that packets are directly passed between the application and the lower communication layers. We will also have a first look at Cooja, which is Contiki's network simulator.

After this exercise you should be able to:

- Configure the frequency channel and PAN ID
- Broadcast simple data in a point-to-point fashion
- Unicast simple data in a point-to-point fashion
- Run simple network simulations in Cooja

1.1 Preparation

Read the following documentation pages. You can use the quiz on Moodle for some guidance what to focus on.

- Configuration
- Nullnet
- Cooja
- TSCH (For optional task)

1.2 Required Equipment

- a minimum of two Zolertia Firefly nodes
- a computer with the Contiki-NG development environment

1.3 Evaluation

This lab is evaluated through the demonstration of functional implementations.

Task 1: Running the nullnet examples

Nullnet is a way in Contiki-NG to get rid of higher communication layers. This can be useful if you want to develop your own network protocols, but will allow us to focus on point-to-point communication. You can read more about Nullnet here.

In this first task we will run the Nullnet examples provided by Contiki-NG. These examples cover both broadcast and unicast communication.

Task 1.1: Sending and receiving broadcasts

In broadcast communication, we do not provide a dedicated receiver. Instead, we transmit our data to anyone who can hear us. You can find an example for Nullnet broadcast communication in `examples/nullnet/nullnet-broadcast.c`.

Compile and upload this example to two of your nodes. Note that you can upload the same program to multiple connected nodes without providing any PORT value. Once the program has been uploaded to both nodes, login to one of the nodes (by providing a PORT address). You should see that the node periodically transmits a packet with a counter value, and informs you about every packet it receives. If others are running the program at the same time, you should be able to receive their packets as well.

Task 1.2: Sending and receiving unicasts

It is not always desirable to receive packets from every node around. Instead, most of the time, we want to transmit a packet to a specific node in the network. We can achieve this very similarly to the previous example, by using unicast communication. You can find an example for Nullnet unicast communication in `examples/nullnet/nullnet-unicast.c`.

In order to utilize unicast communication you need the address of the receiver. Contiki-NG uses 64-bit MAC addresses for this purpose. To obtain the MAC address of one of your nodes, compile and upload the unicast example and login to the node. The Nullnet examples include the shell module, which provides you with an interactive shell. Once you have logged in to the node, you can type `mac-addr`, to which the node will reply with its own MAC address. It should look something like this `Node MAC address: 0012.4b00.060d.6202`. Save this address, which is given in hexadecimal format.

Fill in your address in the `dest_addr` array to define the destination address of the unicast transmission. Same as in 1.1, compile and upload this program to two of your nodes (one of which being the node with the defined MAC address). When you login to the nodes, you should be able to see that one of them is transmitting periodically, but not receiving, whereas the other one is just receiving packets.

Task 2: Your own nullnet application

In order to better understand how these examples are working, lets create our own Nullnet applications by separating transmission and reception code. To make our lives a bit easier, we continue to work with broadcast communication.

Task 2.1: Adjusting communication parameters

Even if we are using broadcast communication, we can avoid listening to nodes that do not belong to our network. With the IEEE 802.15.4 protocol that our nodes are using, we mainly do so with two parameters, namely the frequency channel that we are communicating on, and the network ID of our network (PAN ID).

By default, Contiki applications utilize default parameters for network configurations. We can find out which parameters these are by using the `make viewconf` command in our application folder. From the resulting list you should be able to see that the channel is set to 26 and the PAN ID is 0xabcd.

In IEEE 802.15.4, we have 16 different channels to chose from (11-26) and the PAN ID is a 16-bit integer of our choice. We can change this configuration with the help of a `project-conf.h` file.

Create a new Contiki application by copying, for example, the nullnet-broadcast application (do not forget to copy or create the necessary makefile). Create a project-conf.h file in your new application folder and define in it new values for `IEEE802154_CONF_PANID` and `IEEE802154_CONF_DEFAULT_CHANNEL`. You do so by standard C precompiler defines. You can select a channel and PAN ID randomly, hoping that no other group utilizes the same combination. Finally, run the `make viewconf` command again to see whether the configuration has changed.

Task 2.2: Create your own transmitter and receiver

Utilizing the new communication parameters, we can create a broadcast link between two nodes without being affected by other communications. Create two programs for your two nodes, one for the transmitter and one for the receiver. The transmitter should periodically broadcast packets, whereas the receiver receives them.

After initialization, your receiver probably doesn't have anything to do. However, it is important that the receiver process is non-blocking. Otherwise it will prohibit reactions to incoming packets.

Task 3: Cooja simulations

Network simulations can be helpful to test and understand system behavior, particularly, when the network consists of many nodes. Contiki comes with its own network simulator called Cooja.

Task 3.1: Getting to know Cooja

In order to get started with Cooja, follow this tutorial. It will provide you with a first look at the different windows, and how to setup, start and stop simulations. It is recommended to use `Cooja motes` as the mote type in your simulations.

Task 3.2: Simulating nullnet applications

Now that you know how to use Cooja, you will simulate the Nullnet broadcast application that we are already familiar with.

Create a new mote with the Nullnet broadcast example and add two nodes to your simulation. You can move the nodes around with your mouse. When you click on a placed mote, its communication range is indicated with a green circle. Place the two nodes so that they are inside each others circle. Then start your simulation and check your results. What is the result when nodes are not in each others 'circle'?

Task 4: Time slotted communication with TSCH (Optional)

By default, Contiki nodes use a contention-based MAC protocol (CSMA). This basically means that nodes can send packets whenever they want, but risks for collisions and communication inefficiencies exist. As an alternative, Contiki support TSCH (Time slotted channel hopping), which implements a contention-free MAC protocol.

The Nullnet examples that we have used before are prepared for TSCH. You can enable TSCH by assigning `MAKE_MAC_TSCH` to the MAKE_MAC line in the makefile.

Running TSCH requires one node to take the role of the coordinator. You can achieve this by defining the coordinator in the code by assigning its MAC address to the `coordinator_addr` variable, or by entering `tsch-set-coordinator 1` in one of the nodes' shell.

Once operating, the application appears to operate the same way as before, though now using time synchronized, contention-free communication. This enables nodes, amongst others, to predict communications and thus to turn their transceivers off to conserve energy. If you run the application in Cooja or enable MAC layer information in the logging system, you can observe the enhanced beacons (EB) being sent out by the coordinator.