Lattice Watering: Final Results

Christian Müller, Jonas Heinemann, Kaan Dönmez, Valentin Pickel

Software Project on Internet Communication
Summer Term 2022
Freie Universität Berlin
Institute for Computer Science

July 18, 2022

The Idea

Here we had a picture of us posing in front of a plant, but we removed it for data privacy.

Goals

- Develop a small but secure system for controlling several devices that can regularly water plants.
- Learn about techniques and challenges in developing IoT-based systems.
- Learn about RIOT.

Especially: With respect to the rough split we set at the beginning of the lecture, we worked in the world of the Internet of Things.

We will look at the hardware deployed, the way we designed the communication between devices and our development process.

Hardware Overview

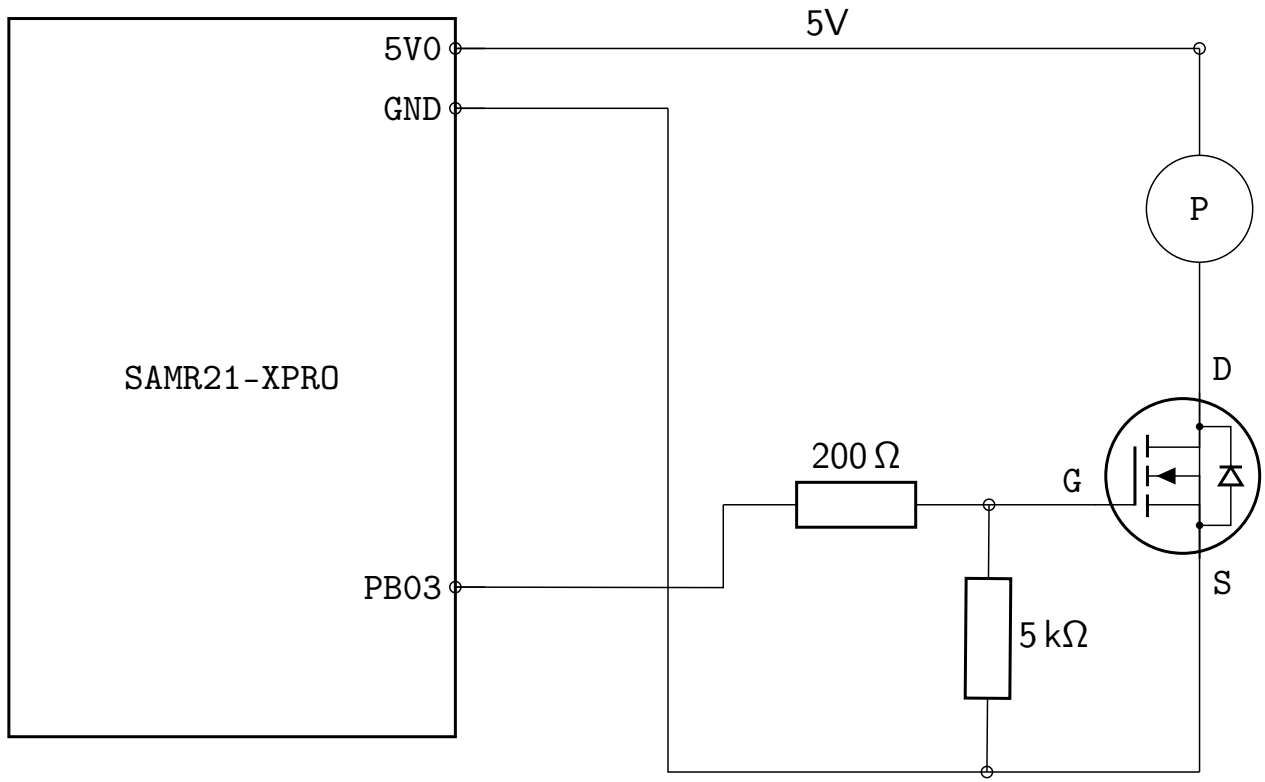
We differentiate between three types of devices: A *host*, a *border router* and *nodes*. The hardware used (for references see the file mentioned below):

1. One personal computer with the software set up.
2. Two SAMR21-XPRO boards (class 2 devices according to RFC 7228), one border router and one node.
3. Two micro USB cables, optionally a battery connector or powerbank.
4. One DRV8833 motor driver board.
5. One electronic pump. It should come with a long tube.
6. One capacitive moisture sensor.
7. Nine female jumper cables.
8. Five male jumper cables.

The `HWSETUP.md` file describes how to put a node together.
The biggest problem we encountered was the following: The pump needs more current than any other component, about 200 mA. But a GPIO pin on our board can only handle 92 mA! We may destroy the board by directly connecting everything.

Hardware Overview

Initial attempts at fixing the problem with the current. involved this design, sadly the transistor given by Hauke did not work, so he got us some predesigned motor boards for assistence.



Software Architecture

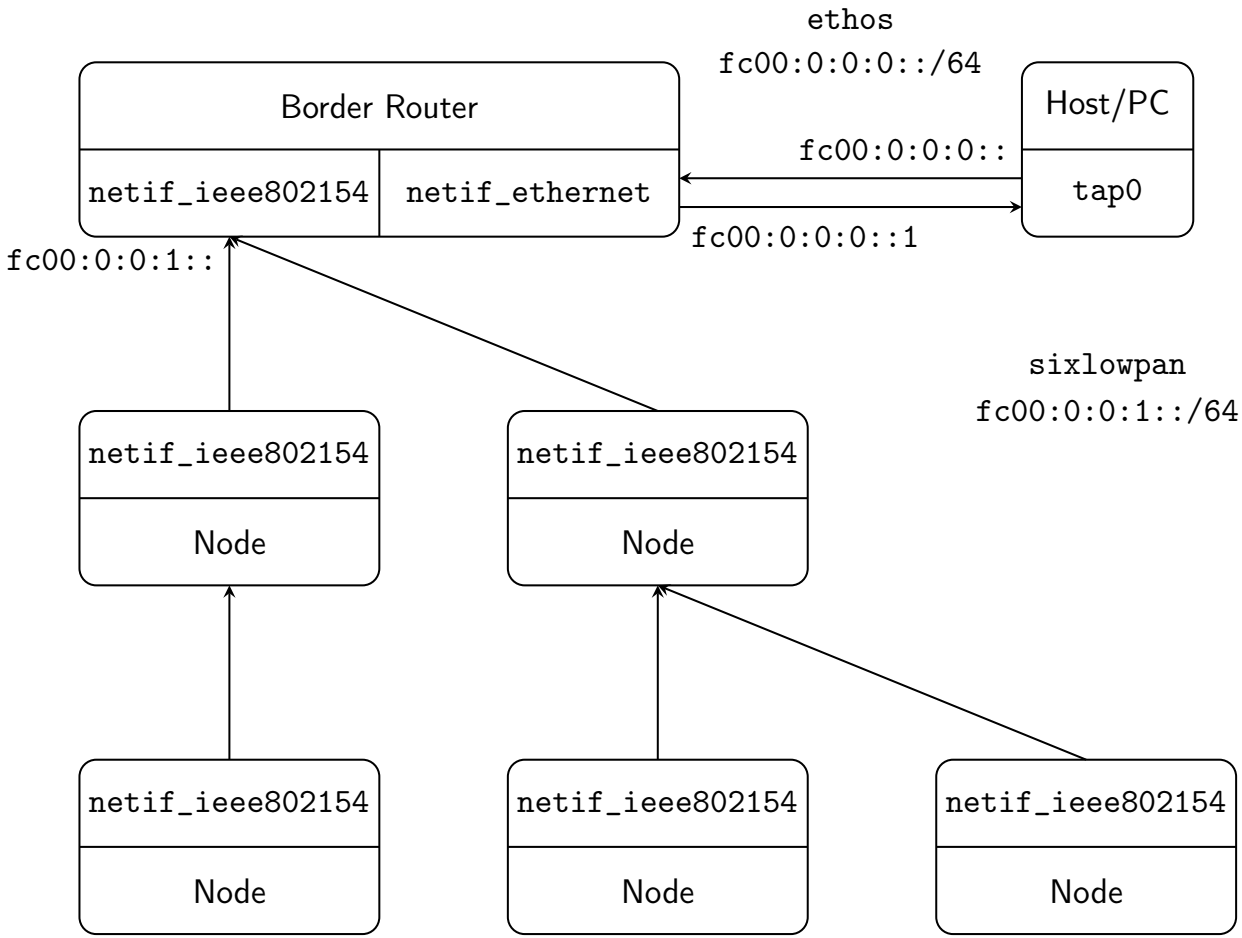
We developed two firmwares and two softwares in total. We list them with the programming languages used:

- `fw` (C, Makefile):
  This firmware runs on every board associated with a plant. These boards are connected to an humidity sensor and a pump. They use SixLoWPAN and CoAP with DTLS to send the sensor data to the host.
- `br` (C, Makefile):
  A border router based on RIOTOS. It translates the SixLoWPAN traffic and routes it to the computer over a USB cable. It also uses `ethos` (Ethernet over Serial), a small technology developed by the RIOTOS project.
- `proxy` (Rust):
  Translates DTLS traffic to UDP traffic. Thus, we have transport security in the SixLoWPAN network.

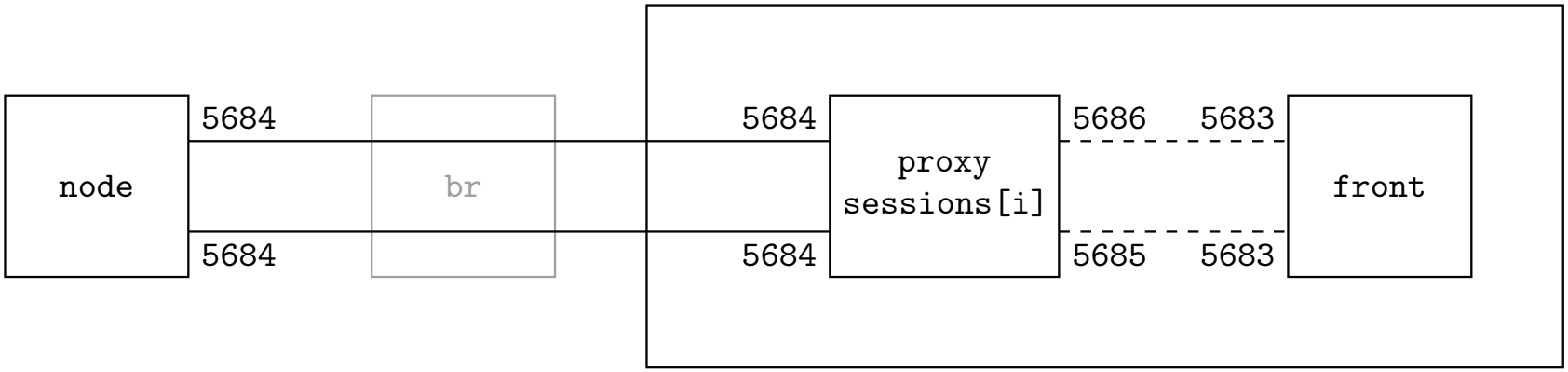
Software Architecture

- `front` (JS, HTML, CSS, SQL):
  The frontend has two parts: A node.js backend and a website.
  The backend stores the sensor data in a SQLite database and shares the data with clients connected to the website.
  The website allows the user to manually start pumps, look at the current humidity as well as data over time, calibrate sensory and more.

Network Structure



Proxy


node
5684
5684
br
5684
5684
proxy
sessions[i]
5686
5683
5685
5683
front

Communication Protocols Involved

- All communication is based on IPv6 in the lower parts. We do not use any IPv4 addresses.
- The nodes, together with the border router, form a DODAG structure due to the RPL protocol. They also use SixLoWPAN to compress and fragment their IPv6 packets, such that if one places many around a huge house, they can route in a lossy network.
- The nodes indirectly talk with the DTLS proxy running on the hosts computer, which decrypts their traffic to forward the packets to the frontend.
- Notice that the finer details show up when looking closely: For the devices to find each others local address, IPv6 uses neighbor and router solicitation messages, and the DTLS protocol must establish a secure session before sending encrypted packets. So snooped traffic looks more complicated than expected.

Demo

Evaluation

We present notes on production-usability and some empirical test results. All tests were done asynchronously, with each testing person using a recent RIOT-OS version on their boards. These are highly non-scientific.

- *Production and Marketability* The system is working, but not very well tested. For production use, we require more usability tests, readily-set-up node boards etc.. Possibly ones with multiple pumps. (Like the other group did.)
- *Storage Usage* The table with the most entries will most likely be the `plant_humidities` table. Given one node, one entry in the table could be made up from a single byte for the node id (for the `plant_nodes` table), up to eight bytes for the timestamp and one 16 bit humidity value. So in a day, given that we send one value every five seconds, we might store up to $(1 + 8 + 2) \cdot (24 \cdot 60 \cdot 60)/5 = 19008$ bytes, so about 5702400 bytes ($\approx$ 6MB) in a 30-day-month.

Evaluation

- *Scalability and Availability Measures* For more nodes, one can easily allow more DTLS sessions to be added to the proxy. In the frontend we took care of some edge cases which could result in a slow service, like not requesting all data from the database when we want to build the humidity graph. Given a strong enough host, we claim that up to a hundred devices can safely be handled by the system. (Which may seem ridiculous, but it might yield an interesting discussion.)

Evaluation

- *Network Usage* We ran one node for an hour. We toggled the pumps twice and calibrated the sensory once, the result includes DTLS connection buildup. Relevant factors that may influence our result: There may be many more nodes and other devices in the network, additional RPL or ICMP or other protocol packets, packet loss may be higher. We only ran one test aswell, instead of running multiple and using a reliable averaging method.

`rx_bytes`	14886
`rx_count`	192
`tx_b ytes`	92537
`tx_unicast_count`	795
`tx_mcast_count`	71
`tx_success`	866
`tx_failed`	0

Evaluation

Energy Usage We set up a node board with a battery consisting of 4x AA batteries. The battery lasted 120 hours (5 days).
Ideas to increase battery lifetime:

- Reduce the interval for sending data packets
- Reduce the interval for extracting humidity data from the soil
- Possibility of deep sleep
- Rechargable batteries
- Small solar panels since plants are near sunlight anyway

Software Development Practices

We deployed:

- Agile development with Kanban
- Versioning and Branching with git
- Communication with Discord
- Splitting up in several groups and performing integration tests
- Regularly writing minimal documentation
- Code Conventions
- Static Code Analysis

And some more techniques.

Reflecting on our Work

The development was rather rough.

- Lack of documentation for specific parts, such as the humidity sensor. We are no electric engineers, despite our best efforts, and do not know all the small tricks.
- Initial hardware was simply failing. In our report, we outline how we tried to build a small circuit for connecting the pump, but a faulty transistor deminished our efforts.
- Despite our best efforts of communicating changes on different parts and how they effect others, this still lead to some systems not working. E.g. when the DTLS proxy was developed, a false commit broke communication on another branch as DTLS there was enabled, but proxy was not yet functional.

Despite that, the system is working and mainly employs secure and standardized, open technologies.

Possible Future Enhancements (*Some* Ideas, there are Lots)

- `hw`: A case for the controllers and their circuitry.
- `front`: TypeScript instead of JavaScript, use HTTP/2.0 or even 3.0 over QUIC.
- `proxy`: Better RNG for `tinydtls` and multiple possible improvements to the proxy, see `README.md`

And more, see the repository.