Watr.li – Developing a showcase application for the RIOT operating system

Lucas Jenß *lucasandreas.jenss@haw-hamburg.de*
Lotte Steenbrink *lotte.steenbrink@haw-hamburg.de*

1 Introduction

The Internet of Things (IoT) is a promising paradigm capable of changing the way we interact with the world– and the way it interacts with us. It describes the vision of small sensors and computers embedded into everyday objects, which autonomously communicate with each other, the Internet, and their surroundings. Since those embedded devices are typically very constrained in memory as well as computation and energy capacities, operating systems powering them must be lightweight, energy-efficient and possess a small memory footprint. RIOT [1], an Operating System (OS) for the Internet of Things, was designed with these criteria in mind. And while the development on the operating system is thriving, it is lacking on the application side. All existing demo applications focus on technical details, which are often hard to communicate and less eye-catching for the broader public. Additionally, experiences with operating a network of RIOT nodes in a long-term, real world scenario are sparse.

To patch these gaps, this paper presents a RIOT-based plant monitoring application which can be shown at fairs, used in workshops, extended and referenced for new projects. Using IoT hardware and sensors, the wellbeing of plants can be monitored and communicated to the user through a web interface. This is challenging in terms of networking, used standards, heterogeneity of hardware and covers a broad array of RIOT features. The diverse nature of the setup also shows domain experts how RIOT integrates with other IoT technologies such as IEEE 802.15.4, 6LoWPAN, and CoAP. Additionally, it makes RIOTs capabilities approachable and visible to observers of varying technical knowledge. To many people, the problem of keeping plants alive is relatable, and a table full of flowers and hardware can help spark the curiosity of passers by.

The remainder of this paper is structured as follows. An outline of the use case for the system presented in this paper is given in Section 2. Section 3 explores the literature pertaining to the use case, followed by an introduction of the main open standards used (Section 4). The system concept, beginning at the hardware layer and working its way to the application layer, is presented in Section 5. This is followed by a description of the implementation of the aforementioned concept (Section 6) as well as the issues found and insights gained during that phase (Section 7). Finally, the community building aspects of the project are outlined in Section 8, concluding the paper with an outlook and future work in Section 9.

2 Towards the Internet of Plants

Keeping plants alive in an office without regular hours is hard. Either everybody thinks their colleagues have already watered the plants or multiple people water the same plant, resulting in either drought or overhydration. This can be solved with technology: If each plant humidity status is displayed publicly, co-workers can take matters into their own hands without fear of interfering with an absent colleagues' plant-watering scheme. Over the course of this paper, such a system, dubbed "Watr.li" (pronounced "waterli") will be introduced.

To the end user, Watr.li consists of two different entities: one wireless sensor node per plant, which is plugged into its flower pot, and a web interface which displays the status of monitored plants. Because Watr.li should make life easier for the user, it should require minimal installation effort and thus

be self-configuring. This means the user will not have to re-program their nodes when they change plants, or register each new plant with the Watr.li network. Whenever a new plant is detected by Watr.li, the web interface will show a new, blank spot for this plant. Now, the humidity needs and a picture of the plant can be added by the user.

A plant has three statuses: *happy*, *okay* or *thirsty*. Whenever a member of the Watr.li network feels thirsty, it will send a tweet and/or E-Mail. After it has been watered, it will tweet and/or E-Mail again to prevent overhydration. Additionally, the status of all plants can be checked through the web interface. This way, users can see that they should water an "okay" plant before a long weekend even though its humidity status has not dropped to thirsty yet.

3 Related work

Over the course of less than ten years, availability of hardware targeted at Wireless Sensor Network (WSN) has increased while costs have decreased significantly. This has increased the range of opportunities for WSN, with plant monitoring being but one of the many opportunities. In large-scale agricultural irrigation, water usage is reduced on the basis of sensor data, which is collected and transmitted with the help of WSN [2, 3]. Large-scale pastures can be monitored in order to improve irrigation schemes with the use of a low-cost Tmote Sky based system, as presented by [4].

But with decreasing cost, applications in a smart home context are also becoming economically viable. For example, a WSN based system for garden watering built with TinyOS on TelosB motes (ZigBee compliant) has been built and evaluated, including plans to add remote control capabilities in future work [5].This system is able to sense plant humidity through a moisture sensor and directly control the irrigation of each plant using an electrovalve. A more recent example is the commercial product "Parrot Flower Power" [6] which does not allow for automatic irrigation of the plants but provides customers with a light-, humidity-, fertilizer- and moisture sensor. Based on this data transmitted to a corresponding smartphone app via Bluetooth, the user can accurately determine whether a plant should be watered, fertilizied or moved to another position based on temperature and light readings.

4 Technologies

The difference between Watr.li and existing solutions presented in Section 3 is that Watr.li relies exclusively on Open Standards and Open Source software to achieve its goals. The main technologies that were employed in the Watr.li project are introduced in this section.

IEEE 802.15.4 The IEEE 802.15.4 standard specifies the physical (PHY) and media access control (MAC) layers for low-rate wireless communication in personal area networks [7]. It was specifically designed for embedded devices: its hardware is cheap to produce, energy efficient and uses small packets – with a frame size of 128 bytes, only 81 bytes of payload are left to the upper layers. For comparison: the size of a IPv6 base header alone is 40 bytes.

6LoWPAN To adapt all IPv6 packets to the restrictions of an IEEE802.15.4 transceiver, a 6LoWPAN [8] adaption layer was used. This adaption is achieved through compression of IP headers down to 3 bytes and packet (de)fragmentation, if necessary.

RPL The Routing Protocol for Low Power and Lossy Networks (RPL) [9] is specifically designed for networks in which all traffic is directed towards one central, strong node, the root node. RPL achieves this by generating a Destination Oriented Directed Acyclic Graph (DODAG) topology which originates at the root node. Routes to the root node are then established via parent node, i.e. each node sends its data to a parent which is located closer to the root node in the tree topology. The data is then forwarded recursively until the root node is reached [10].

CoAP The Constrained Application Protocol (CoAP) [11] is an application layer transfer protocol designed to meet the requirements of highly resource-constrained devices and machine-to-machine (M2M) scenarios such as those encountered in Low-power and Lossy Networks (LLN) [12]. Its messages can also be statelessly translated to HTTP requests, thus allowing for a proxy between an LLN and the Internet. This is done to facilitate the integration of constrained nodes into Internet.

5 System concept

This section describes the underlying concept of the Watr.li system, which is comprised of two main components: several Plant Nodes (PNs) and a single Display Node (DN). Every monitored plant has its own PN which measures the soil humidity and transmits it wirelessly to the DN. If the DN is not directly reachable from the PN (i.e. it is not a one-hop neighbor), the PN will use intermediary PNs to send its data towards the DN. The DN then stores and evaluates the received sensor data, making it available for display to the user or further propagation into the Internet, for example to Twitter. Figure 1 illustrates a possible Watr.li setup.

The remainder of this section will detail the different layers of the Watr.li concept, beginning at the hardware level and working its way up to the User Interface (UI) concept.

5.1 Hardware

The PN is battery-powered since it is impractical to wire up every monitored plant with a power cord. To achieve long lifetime while on battery power, the PNs hardware must also be energy efficient. Therefore, a resource-constrained sensor node with only a few kilobytes of RAM/ROM and low computing power was chosen as suitable hardware: the Atmel SAM-R21. This board features an IEEE 802.15.4 transceiver allowing for energy efficient communication with the DN. Plant moisture is sensed using a DFROBOT SEN0114 humidity sensor [13]. An optional USB/UART converter can be attached for debugging.

The DN on the other hand is more focused on reliability. It has to manage all incoming traffic from the PNs as well as maintain a steady connection to the Internet. Since it is a single device per Watr.li set-up and thus can be connected to a power outlet, energy-efficiency is not a concern. The ARMv6-based Raspberry Pi (RasPi) was therefore chosen as supporting hardware due to its low cost and adequate performance. Since the RasPi is not equipped with an IEEE 802.15.4 transceiver, a Rosand Technologies R-IDGE 6LoWPAN USB "Router"[1] [14] was used as an external transceiver. This component is capable of receiving IEEE 802.15.4 transmissions and transforming the contained 6LoWPAN packets into regular IPv6 packets, thus acting as a gateway between the IEEE 802.15.4 network and the RasPi.

1. Despite its slightly misleading name, the R-IDGE device is mainly an IEEE 802.15.4 transceiver with optional 6LoWPAN and RPL functionality.

5.2 Operating Systems

The tasks of the DN are processing PN information, hosting a web server to display PN data to the user, and working as a border gateway between IEEE 802.15.4 and an Ethernet connection to the Internet. Since the DN is connected to a stable power source it is not required to be especially energy-efficient. Linux was chosen as the most suitable operating system for these tasks for two main reasons. First, it is well supported on the chosen hardware platform, the RasPi, which comes with an official distribution (Raspbian) specifically tailored to the platform. Second, drivers for the employed USB gateway were only available for Linux.

Battery powered embedded devices such as the Plant Nodes, however, lack the hardware resources to support common desktop/server operating systems such as Linux or Windows, which are aimed at supporting multi-tasking of dozens of applications and providing graphical user interfaces. The PNs do not need most of these features. Instead, they require an operating system which is memory efficient and supports low-power wireless connectivity, preferably based on open standards.

Operating Systems such as Contiki [15, 16], TinyOS [17, 18] and RIOT [19, 20] are designed to operate in the challenging environment of interconnected embedded devices, differing mainly in kernel architecture and the programming models that are employed for application development.

Event-based operating systems such as Contiki or TinyOS have adopted a programming model that differs from conventional C applications and only support a subset of the C language (Contiki) or provide a C dialect (TinyOS) for application development. RIOT, in contrast, offers a "traditional" threading and scheduling scheme, POSIX-compliance as well as full C and partial C++ language support. This significantly decreases the learning curve for novice IoT developers, as they do not need to learn new paradigms and can code using most of the techniques they already know.

Figure 1. An example of a Watr.li network. The blue antennas depict Plant Nodes and the monitor in the center is the Display Node. The connections between the nodes shows a possible tree topology formed towards the Display Node acting as root node.

Additionally, external libraries can be ported to RIOT with just a few patches, as shown by the example of the microcoap port [21].

5.3 Network Stack

In order to establish communication, all nodes require a common set of protocols which both PNs and the DN need to be able to speak. Figure 2 illustrates how Watr.li's network stack accomplishes this: Every information exchange between PN and DN is wrapped in a CoAP packet and transmitted via UDP/IPv6 over IEEE 802.15.4. The appropriate header compression and packet fragmentation is performed by a 6LoWPAN adaption layer. All packets destined for the global Internet, such as webpages or notification tweets, are sent via HTTP and Websockets over traditional TCP/IP over Ethernet.

5.3.1 Routing Protocol

A direct connection to the Display Node can not be guaranteed for every Plant Node. Thus, a routing protocol is needed to connect all Plant Nodes into a multi-hop network and facilitate communication with the Display Node. At the time of implementation, RIOT was equipped with two different routing protocols: Ad Hoc On-demand Distance Vector Routing Version 2 (AODVv2) and RPL. The former is a reactive protocol made for sparse point-to-point communication, while the latter is a proactive protocol designed multipoint-to-point communication. In its current state, Watr.li's communication pattern can be described as multipoint-to-point, as all traffic is flowing from the Plant Node to the Display Node only. This, and the fact that currently no implementation of AODvv2 is available for Linux, made RPL the current routing protocol of choice for Watr.li. subsection 10.5 discusses circumstances under which this might change.

5.4 Communication model

To guarantee a smooth startup process and a correct flow of information, a communication model was created for the Watr.li system. The sequence diagram in Figure 3 depicts the messages that are sent during system operation after auto-configuration, which itself will be performed aided by RPL, the chosen routing protocol. RPL was chosen as the routing protocol for the Watr.li network since it is designed for LLN and an implementation is already available for RIOT. The auto-configuration itself will is explained in more detail in subsection 6.5.

Once auto-configuration has completed, the PN has knowledge of how to send messages to the DN,

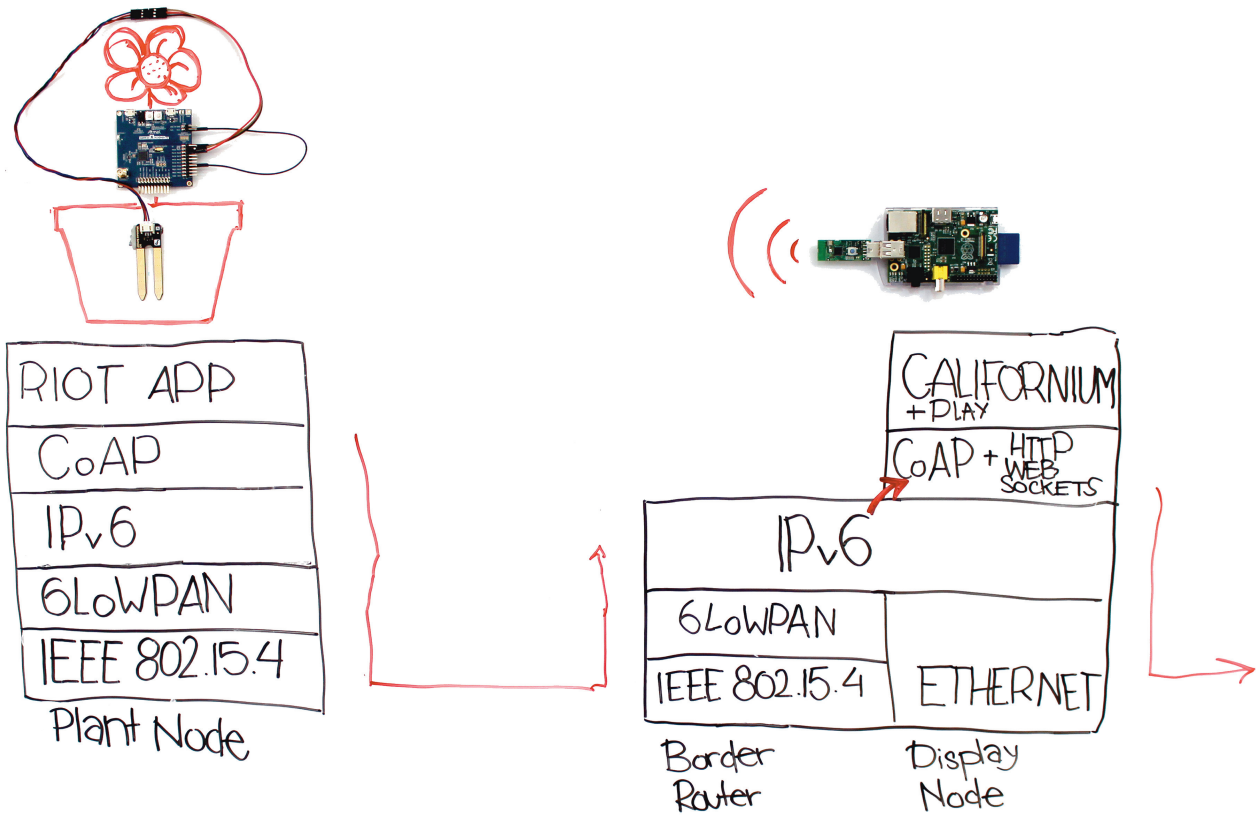

Figure 2. Employed network stack on both Plant Node (left) and Display Node (right). The center arrow shows the flow of a CoAP packet from the PN to the DN whereas the right arrow shows the flow of a HTTP/WebSocket packet from the DN towards the Internet.

thus sending a `Register` message containing a unique identifier to alert the DN of its existence. This prompts the DN to present the new plant node to the user (here called `Display`) so that it can be assigned a name, watering needs, etc (see subsection 5.6). After registering, the PN will start to send `Humidity status updates` to the DN whenever a change in soil humidity is detected. On receiving such an update, the DN will both present the updated information to the user as well as determine whether a notification must be sent to one or more notifications endpoints such as E-Mail or Twitter.

5.5 User interface (UI)

The UI is an important aspect of both a real-world application as well as a showcase. In the latter, it is often unnecessary to display the actual inner workings of a system such as Watr.li: not only are the routing protocol, sending packets from constrained nodes, etc. are difficult to visualize in real-time, they also contain little value to the end user. Thus, the primary goal of the Watr.li UI is not to present the actual inner workings of watr.li, but to communicate the resulting changes clearly.

To achieve this, the user-facing components of the Watr.li system were modeled as a set of three user stories. User stories are short feature descriptions from the user's point of view and are employed in Behavior Driven Development (BDD) to describe the desired features and behavior of an application or system [22]. The Watr.li user stories are the following:

1) As a user, I want to monitor the humidity of my plants tailored to their individual needs

2) As a user, I want to conveniently access the humidity status of my plant

3) As a user, I want to be notified when a plant needs to be watered

Story 1 is the most important one, since it outlines the main functionality of the system. It entails that,

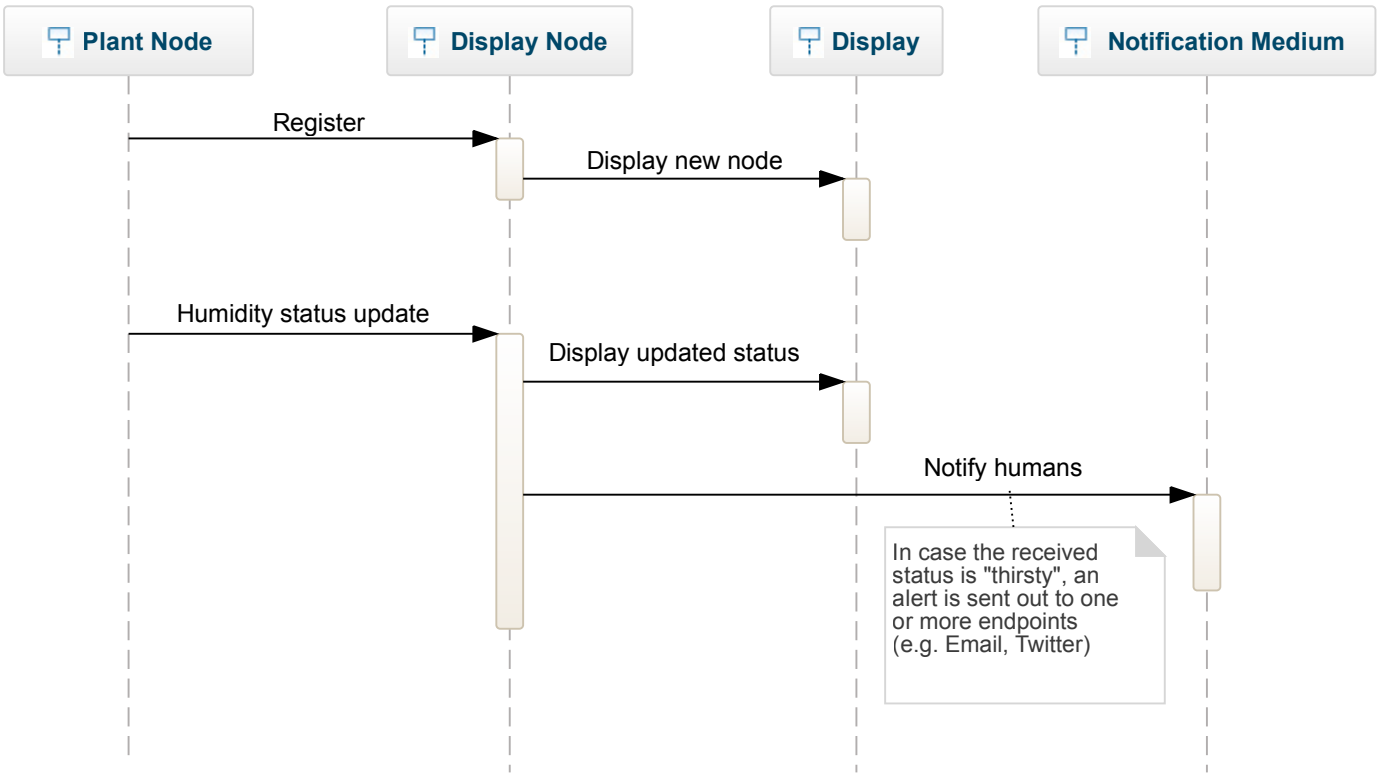

Figure 3. UML sequence diagram showing an abstract overview of the different messages sent by the entities in the Watr.li system.

when a new PN is added to the system, the user must be able to:

1) Assign a name, location and/or picture to the PN (and thus to the plant) to make it easy to distinguish multiple PNs when more than one is registered.

2) Calibrate the watering needs of the plant, i.e. which sensor readings correspond to the three plant states *happy*, *okay* and *thirsty*.

The second user story is about retrieving the data of the monitored plant. Users should not have to be technical experts to retrieve the humidity status of the plants. Preferably, this should be possible remotely, i.e. over the Internet, thus making a website the ideal form of presentation.
Finally, the third story is about letting the user know when they have forgotten to water their plants. Ideally, multiple notification media such as E-Mail, Twitter and/or Facebook messages should be supported to adapt to the user's individual needs. The UI mockups derived from the user stories are shown in Figure 4.

5.6 Configuring watering needs of a plant

As can be seen in Figure 4, it should be possible to configure how much water a plant needs. This requirement is woven into Story 1 because different plants have vastly different watering requirements which must be taken into account. In Figure 4, a very simple and non-precise approach was taken: the watering needs are represented as a slider which the user can customize to indicate if a plant has low, medium or high water requirements. In reality however, this is not going to be sufficient to keep a wide range of plants with different necessities constantly happy. To achieve this, a plant database, similar to the one that "Parrot Flower" has developed [6], would be necessary. The user of the system could then select the correct plant or family of plants from a list, with the application automatically downloading the required parameters and adjusting notifications and humidity display accordingly.

6 Implementation

This section covers the implementation details of both the Display- and the Plant Nodes based on the

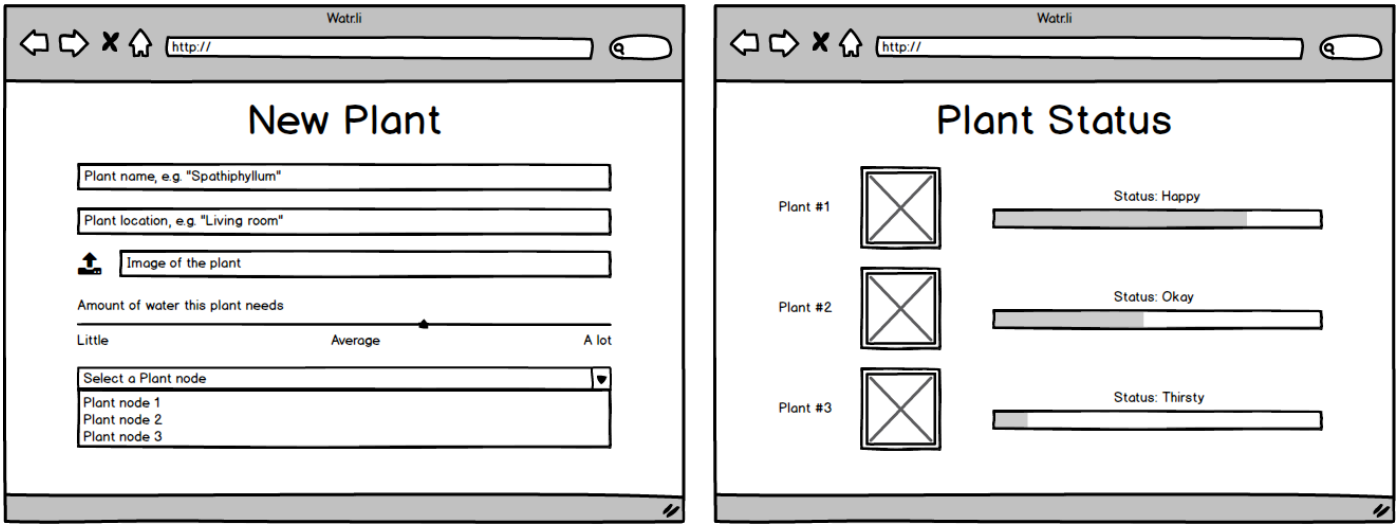

Figure 4. Conceptual mockups of the Watr.li UI. The plant creation screen (after a new PN has registered itself) can be seen on the left, the plant status overview to the right.

concept presented in Section 5. It aims to establish an unterstanding of the important mechanisms that make up the Watr.li system as well as the design decisions behind them.

6.1 CoAP communication

In subsection 5.4 the general communication model designed for Watr.li was introduced. This section gives an overview of the concepts behind CoAP and substantiates the abstract overview from subsection 5.4.

6.1.1 Representational State Transfer (REST)

CoAP aims to realize the Representational State Transfer (REST) architecture in a suitable form for resource constrained nodes, i.e. low processing power and limited RAM and ROM [11]. REST describes a set of design criteria initially introduced in [23, 24], which are outlined in this section alongside their implemention in CoAP.

- Initially envisioned for Web applications, many of the architectural critera align closely with how HTTP (over TCP) and the World-Wide Web (WWW) work. Unlike HTTP, CoAP employs an asynchronous datagram-oriented transport such as UDP, being a better fit for LLN.

- REST implies a **client-server architectural style**, separating data usage concerns from the data storage and processing concerns. However, M2M interactions typically result in a CoAP implementation acting in both client and server roles [11], although with disjoint interfaces for each role.

- All communication in a REST architecture is **stateless** in nature. That is, all requests from client to server must encompass all information necessary to understand, process and reply to the request.

- To improve network efficiency, responses must implicitly or explicitly be defined as cacheable or not. This enables in-network caching of responses and makes some request unecessary if the response is already cached at the entity issuing the request. In CoAP, this is implemented as the `Max-Age` option field which, if not present, defaults to 60 seconds. After this time, the information contained in the package becomes stale and must be requested anew.

The key information abstraction in REST is a resource. Any named information can be a resource, e.g., a plant, its humidity or today's weather in Hamburg. Individual resources, at which all CoAP requests are targeted, are identified by a Uniform Resource Locator (URL) in web-based as well as CoAP REST systems. In addition to the target URL, each CoAP request also contains one of four request methods which define the intention of the request:

- The **GET method** retrieves a representation for the information that currently corresponds to the resource identified by the request URL [11].

- The **POST method** requests that the representation enclosed in the request be processed. The actual function performed by the POST method is determined by the origin server and is dependent on the target resource. It usually results in a new resource being created or the target resource being updated [11].

- The **PUT method** requests that the resource identified by the request URL be updated or created with the enclosed representation [11].

- The **DELETE method** requests that the resource identified by the request URL be deleted [11].

Thus we can represent an example CoAP request, which would update the weather for Hamburg, as follows:

```
PUT coap://[::1]:5683/weather/hamburg/
Payload: "Sunny"
```

Note that this is a simplified representation of a CoAP request; all features not used in Watr.li are omitted.

6.1.2 Watr.li CoAP messages

As shown in Figure 3, all communication between PN and DN originates from the PN and is comprised of just two uniqe messages:

The registration of the PN with the DN is performed by the following message:

```
PUT coap://[DN IPv6 address]:5683/nodes
Payload: Unique node identifier
```

Humidity status updates are structured like this:

```
PUT coap://[DN IPv6]:5683/nodes/
    <unique node identifier>/humidity
Payload: Humidity value
```

For the format of humidity values see section 6.2.2. The unique identifier may be any kind of ASCII string as long as it does not contain any slashes.

6.2 Plant Node

The PN is conceptually less complex than the DN, since it has fewer responsibilities. As outlined in Section 5, it is responsible for registering with the DN and sending humidity status updates based on the moisture in a plant's soil.

The PN application is implemented on top of the RIOT operating system, which alreay includes some of the core components chosen for the Watr.li system: the routing protocol (RPL), a full IEEE 802.15.4/6LoWPAN network stack, drivers for the Atmel SAM R21 (SAMR) board as well as a minimal CoAP implementation called microcoap [25].

The implementation [26] consists of three elements. Sensing humidity data is implemented in `sensor.c/.h`, which includes configuration and initialization of the Analog to Digital Converter (ADC) as well as reading the ADC information on demand. Building the CoAP packets is implemented in `coap_ext.c/.h`. Said implementation is based on microcoap [25] which had to be extended in order to provide the needed features, as detailed in subsubsection 6.2.1. The entry point to the application is implemented in `main.c` and handles the initialization of the necessary RIOT subsystems: the transceiver, the network stack, the humidity sensor as well as RPL (and thus the autoconfiguration, see subsection 6.5). In addition it periodically queries the sensor for data and sends said data to the DN in case it has diverged more than a certain threshold from the previously sent values. The necessity of this mechanism is explained in subsubsection 6.2.2.

6.2.1 Extending Microcoap

To create and process CoAP requests, the microcoap library [25], which is provided by RIOT as an external package, was used. Microcoap has the advantage of being small and simple to use. However, like most CoAP implementations [27], microcoap has also been created with the assumption that IoT nodes are so constrained that they cannot decide autonomously when to send data, and thus need to be asked instead. Therefore, it offers only server functionality out of the box. An application using microcoap is able to receive a CoAP request, process it and respond with another CoAP message, but it is unable to create a CoAP message without having received an initial request. Unfortunately, according to the founder of the company which

created microcoap, there are no plans to extend its functionality [28].

This functionality is needed by Watr.li to autonomously send PUT requests with plant status updates, however. For that reason, an extension [29] was created which uses microcoap's internal functions to build such a PUT request, which can then dispatched by the Watr.li application. Due to time constraints, this extension was not designed to be modular, and functionality to generate any request type other than PUT is missing. It should be considered to extend microcoap to add this functionality in a more generic way.

While working with microcoap, two other hurdles were encountered. First, it was discovered that, by default, microcoap allows its users to create paths to endpoints with a maximum of two segments. Thus, an endpoint path of `/foo/bar` would be legal, while an endpoint path of `/foo/bar/baz` would exceed the segment limit. This posed a problem, since the endpoint path used by Watr.li, `/nodes/<node id>/humidity`, exceeded this limit, too. This could be fixed by increasing the `MAX_SEGMENTS` constant directly in the microcoap code. This approach turned out to be complicated to maintain, since a fork of microcoap had to be created on which `MAX_SEGMENTS` was changed. The Makefile for RIOT's microcoap packet also had to be adjusted to point to this dedicated fork instead of the main microcoap repostory, and this change had to be maintained in a dedicated RIOT branch. To eliminate this overhead, a patch was added to RIOT which allows to change `MAX_SEGMENTS` at compile time [30].

Second , while the micocoap codebase consists of only 702 Lines of Code (LOC), it was entirely undocumented, which complicated the development efforts. Thus, basic documentation Pull Requests (PRs) were submitted and accepted into the microcoap implementation [31].

6.2.2 Sensing humidity values

The ADC and the humidity sensor are the main components for sensing the moisture of a plant's soil. After calibrating the ADC (Section 12), the ADC returns values between 0 and $2^{12} - 1 = 4095$ since it is configured with a 12-bit sampling width.

Due to minor fluctuations in electric current flow, there are always tiny divergences in the measured values. With a spectrum of 0 (completely dry) to 4095 (submerged in water), sending a value change of 1 (i.e., 0.02%) is a waste of bandwidth and battery. This is why the PN application remembers the last value that was sent to the DN and only sends an update whenever the new value differs by a predefined threshold, currently set to `10`. This value is subject to change, should it prove to be impractical during long-term usage.

It should be noted that, for the Watr.li system, it was assumed that the relation between humidity of the soil and the value sampled by the ADC is linear. It was later discovered however, that the relationship between resistance and moisture potential is a non-linear function, which other researchers have determined through laboratory calibration [4]. For the purpose of this paper, however, assuming a linear relation proved to be sufficient for demonstration purpose.

6.3 Display Node

From the user perspective, the purpose of the DN is to collect plant data and make it available to the plant maintainers by visualizing the status of all plants as shown in the conceptual mockups of Figure 4. From the system perspective, it serves as a gateway between the wireless Watr.li network (wirelessly communicating via IEEE 802.15.4) and the Internet (to which it is connected via Ethernet). This gateway functionality is vital, since it translates the protocol stack used by Watr.li internally (mainly CoAP and IPv6+6LoWPAN) to the protocols used in the Internet (HTTP and IPv4/v6). In addition, the Display Node hosts the application which receives PN data, displays it to the user and posts notifications.

As has been established in subsection 6.2, the PN communicates with the DN via CoAP. While the PN employs microcoap to do so, on the DN the Californium CoAP framework was used instead. In contrast to microcoap, Californium is a feature-complete CoAP implementation written in Java, not targeted at resource constrained nodes.
To develop the web application the Play framework [32] was chosen, primarily because it also runs inside the JVM (as does Californium). This made it easy to relay the incoming CoAP messages to the web application. The communication between the two components is establishde via an Akka [33] Actor, which is the conventional solution for asynchronous communication within the Play

Framework.
Lastly, the user's browser communicates with the Play application server using HTTP as well as the WebSockets. The latter is used to provide near real-time updates without forcing the user to reload the website.

Figure 5 shows an overview of the DN architecture. In the bottom left, a resource-constrained node sends CoAP packets towards the DN. These packets are received by Californium and then relayed through Akka actors to the web application implemented in Play. Once received, the Play application stores the received data and updates the UI (Figure 6) in the user's browser (bottom center) accordingly via WebSockets. In case a notification needs to be sent (e.g. Figure 7), the Play application contacts one or more notification endpoints (bottom right) (e.g. an E-Mail server or the Twitter API) to dispatch the notification. In this last case, the protocol depends on the endpoint and is thus omitted.

Note that Figure 5 only describes the main application that was developed for the DN. Since RPL operates independently of said application, it is discussed separately in subsection 6.4.

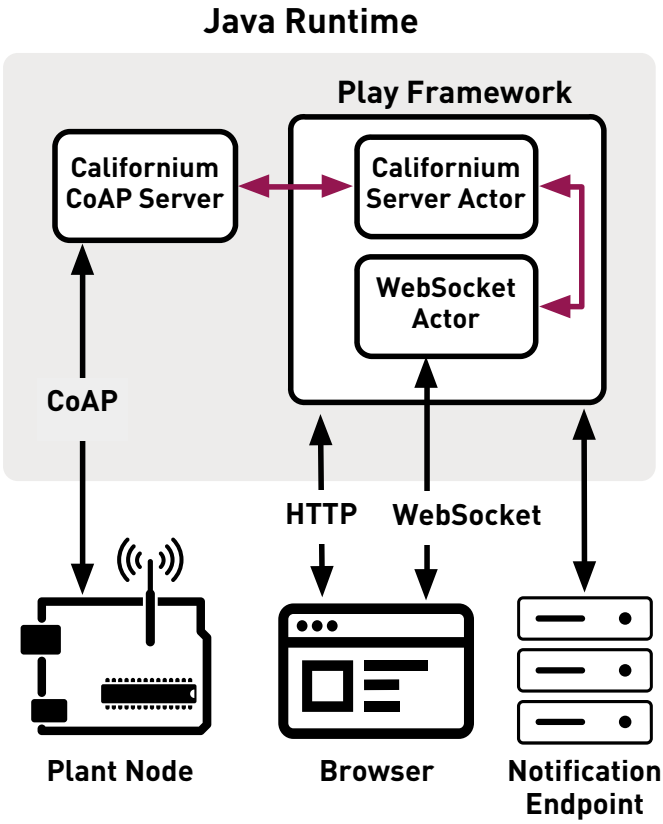

Figure 5. Overview of the Display Node's architecture and the involved communication protocols

6.4 State of RPL on Linux

Finding a working RPL implementation to use on the border gateway proved to be more challenging than anticipated, due to the state of IPv6 over Low power Wireless Personal Area Network (6LoWPAN) and RPL support on Linux.

At least three distinct RPL implementations currently exist for the Linux kernel [34]. In descending order of latest update to the source code, these are:

Unstrung [35] is a userspace implementation of RPL written in C++ and C. It was last updated on 2015–01–02. While it features a test suite, the contained tests were not passing at the time of access (see bibliography) and some of the tests resulted in a segmentation fault.

linux-rpl [36] is a kernelspace implementation of RPL written in C, comprised of several patches for the Linux kernel mainline versions 3.10 to 3.12 (current stable release is 3.19). The author has stated [37] that he intends to further work on this implementation but is currently unable to do so. A guide on how to use linux-rpl on the RasperryPi is available at [34]. Since

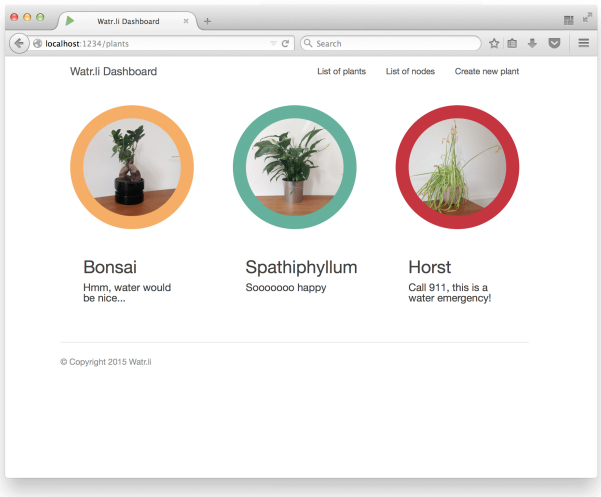

Figure 6. The dashboard web page as served by the Plant Node.

Figure 7. A tweet indicating that "Horst" has just been watered.

the RasperryPi does not come with an IEEE 802.15.4 transceiver though, additional hardware has to be purchased to follow the guide.

SimpleRPL [38] is a Python implementation of RPL running in userspace. It is stated in the project's README that no interoperability tests have been performed and that it is meant to be used only in a "secure environment" where there can be no malicious attacker on the network. The last update to this implementation was on 2013–06–09.

In summary, currently no feature-complete interoperable RPL implementation exists for Linux. In addition, the IEEE 802.15.4/6LoWPAN support of the Linux kernel is currently under development in a joint effort of the linux-wpan-next and linux-bluetooth-next kernel developers [39, 40].

However, Rosand Technologies distributes RPLd [41], a functioning, feature-complete RPL daemon. A look inside its source code reveals that this daemon is merely a wrapper around Contiki's native mode: RPLd spawns a Contiki instance, which operates as it would on any other embedded device. RPLs uses the RPL implementation running in this Contiki instance to execute the protocol operations. The packets generated by the Contiki instance are then passed on to the the network device, which sends them like it would send any other regular packet. Any received RPL messages are forwarded to the Contiki instance, which then handles them as usual. In case the Contiki instance performs a route update, this is communicated to the Linux routing table, too. This way, the Linux device running RPLd learns about all discovered routes. The version of RPLd available at the time of development was not compatible with the Linux kernel 3.18.

Due to the time constraints when developing this project, it was decided to update the RPLd provided by Rosand Tech to work with the Linux kernel 3.18 and use it on the Display Node.

6.5 Watr.li Auto-Configuration

To keep the maintenance effort as low as possible, the Watr.li network must be able to bootstrap itself as well as recognize and integrate new nodes autonomously.

To facilitate this auto-configuration of the PNs, the RPL DODAG-ID was chosen to be the IPv6 address of the DN, as mandated by the RPL standard[9].

Thus after joining the RPL tree topology, which includes receiving the DODAG-ID, the Plant Node automatically knows the address of the DN. After joining the DODAG, the PN sends a CoAP registration message (see subsubsection 6.1.2) to the DN such that the latter knows of its existence, allowing the user to create a new plant configuration based on the registered PN. The unique identifier contained in the CoAP registration message is stored on the DN alongside the IPv6 address of the PN.

The chosen approach of storing the IPv6 address as the DODAG-ID has the advantage that it is automatically distributed to all PNs in the system.

7 Testing

One goal of the project was to explore the process of writing a "real-world" RIOT application, and to document any bumps in the road so that they may be improved in the future. This documentation should not be limited to problems purely caused by RIOT. Information about problems which may be encountered in the environment a RIOT node operates in can be just as valuable when planning such a project in the future. This section documents the problems encountered and the resulting lessons learned while designing, implementing and testing the Watr.li's components.

7.1 SAMR specific issues

The SAMR evaluation board which was used for the Plant Nodes features 32Kb of RAM, which is close to the lower bound of memory into which RIOT can fit, if used with a full network stack and routing protocol. In addition, the SAMR is one of the platforms which have not been thoroughly tested at the time of this writing. In summary, the following problems were encountered during development:

- The remote debugger for the SAMR cannot be stopped manually. Trying to do so (e.g. by pressing Ctrl+C) results in the debugger connection being dropped. Thus the only way of stopping the debugger was through breakpoints or special assembly commands. This made unexpected code paths (such as the hard fault routine) difficult to debug, as the only way to see which path the code was taking was stepping through every line.

- The driver for the Analog to Digital Converter (ADC) for the SAMR board is currently under development [42]. While the current state is functional, additional changes [43] had to be performed to be able to receive debug output through the UART interface while the humidity sensor is attached. This is due to the fact that the `UART_0`, which is the default standard output interface, is wired to the same pin as the reference voltage for the ADC, which is needed for the humidity sensor. Thus the standard output had to be set to `UART_1`. This in turn required an additional UART to USB converter.

- The Cortex-M0 CPU on the SAMR board does not permit unaligned memory access. Due to an issue [44] in the network stack's IPv6 subsystem, where a pointer was cast to a `uint16_t` causing the compiler to emit a "load halfword" instruction on a non-halfword memory address, a hard fault was triggered. This only occurred when sending an IPv6 packet to a link-local address. This problem highlights the importance of automated testing of the example applications for the supported hardware platforms.

- Hard faults and other Interrupt Service Routines (ISRs) that stop the execution of RIOT do not emit any information. Due to the debugger issue mentioned above, it was not easily identifiable where the application had stalled. Emitting some information, for example a stack trace, when hitting an ISR such as a hard fault would aid debuggability of RIOT applications.

7.2 Issues due to memory constraints

Due to the low memory available on the SAMR board, the number of RPL routing entries that are stored at runtime had to be reduced from 128 (the default value in RIOT) to only four. Each entry occupies 32 bytes of RAM, resulting in 4KB additional RAM usage with the default number of entries. While the reduction was particularly drastic on the SAMR board, routing entry size would be an issue with slightly less constrained boards also. It should be noted that, in general, one would use RPLs non-storing mode in such a resource-constrained scenario. In non-storing mode, only the DODAG root maintains all the paths and a Source Routing Header (SRH) is attached to all point-to-point packets at the DODAG root [45]. However, non-storing mode was not available in the RPL implementation of RIOT at the time Watr.li was built.

Another problem that ocurred in this regard was that crashes of the PN application ocurred seemingly at random. In addition, these crashes were hard to debug due to the SAMR specific debugger issues described earlier. It was found that the default thread stacksize for the SAMR board was fairly low at only 1024 bytes. Doubling this stacksize still allowed the application with enabled networking to fit in RAM when including the RPL optimization previously discussed.

7.3 Neighbor Discovery

The Neighbor Discovery Protocol (NDP) [46] is used by IPv6 nodes (hosts and routers) to discover the link-layer addresses for neighbors on the same link. While RIOT features an NDP implementation, it was not functional when Watr.li was implemented. The main issues were:

- Only partially sending or answering neighbor solicitations

- Only partially acting upon router advertisements

As a result, sending to IPv6 nodes in the network never succeeded, since the link-layer address of the target node could not be found in RIOT's neighbor cache.

Since fixing the Neighbor Discovery (ND) was not a viable option both due to the time constraints of the project and because the RIOT network stack is currently being re-implemented, a workaround for the ND issue had to be found. This workaround consisted of sending all outgoing packets from RIOT

to the link layer broadcast address, but leaving the correct target IPv6 address intact. This resulted in the receiving nodes always accepting the packets on the link layer, but only delivering them to the application if the IPv6 address matched.

This approach had one unintended side effect: when trying to reply to the RIOT nodes from Linux, the responses would never arrive. In addition, after having tried to send a reply to a RIOT node, all subsequent traffic from that node would be dropped by the Linux network stack. The reason for this behavior is that, when responding, Linux tried to get the link layer address of the target RIOT node from its own neighbor cache. When failing to find it (because the RIOT node had never sent out a neighbor advertisement), it sent out up to three neighbor solicitations, to which it would also not receive a response. Upon failing to find a link layer address for the target node, the IPv6 address was marked as unreachable both for incoming and outgoing traffic. This issue was avoided by never sending any traffic from the Display Node to a target node, effectively removing bidirectional connectivity.

An alternative solution that was initially considered was that of manually adding the PNs to Linux's neighbor cache, thus forcing Linux to deliver the replies and avoiding NDs being sent. This approach had the downside of having to figure out the link layer address of the PN as well as having to add it to Linux manually, thus adding two additional steps in the setup of the Watr.li system. It was thus decided that, since two-way connectivity was not needed for the current use case, the previously explained approach was taken instead.

The issues described above were fixed with the introduction of the `gnrc` network stack in RIOT release 2015.09 [47].

7.4 Heterogeneous networking

The creation of Watr.li was also an interoperability test: Prior to Watr.li, RIOT had participated in several scheduled interoperability events, but these events are rare and it is difficult to catch all bugs in one day. However, because the Watr.li network consists of both RIOT and Linux nodes: different protocol implementations have to interoperate with each other.
One of the interoperating software components was RPL, with the implementation running on the DN being one that has undergone interoperability tests with other implemenentations [48] as well as performance and correctness tests [49]. This way, it was discovered that RIOTs RPL implementation used the wrong byteorder [50] in some places. After this problem had been fixed, both RPL implementations interoperated correctly, apart from the limitations imposed by NDP as explained in subsection 7.3.
Both IEEE 802.15.4 and 6LoWPAN on RIOT interoperated without problems with the implementation on the R-IDGE 6LoWPAN USB Router [sic].

7.5 Timers

The instability of the "vtimer" subsytem in RIOT [51] made prolonged operation of the Watr.li system impossible, since it would either crash or periodically scheduled events would never happen (i.e. humidity sensing), effectively freezing the system. As a result, all information gathered regarding the reliability and stability of RIOT are constrained to running the system for under an hour at a time. While a new timer subsystem called "xtimer" has been merged into the RIOT source tree at the end of August 2015 [52], it has not been evaluated in the context of Watr.li so far.

7.6 Hard faults on the SAMR

Even though some of the hard faults were fixed during development of the Watr.li application, instability during prolonged operation on hardware still occurred. Due to the fact that current RIOT hard fault handlers are simple `nop` loops, it is hard to notice and impossible to debug a hard fault without a debugger attached to the hardware. In combination with the timer problems referred to in the previous section, this made unattended operation of any kind unfeasible.

In summary it can be said that the fact that an application compiles, can be flashed and runs reliably for a few minutes currently does not allow any conclusions on whether unattended long-running operation is possible on the RIOT platform. To improve this situation, implementing more sophisticated fault handlers should be a priority. If these fault handlers would be able to print or even store a stack trace, debugging capabilities would be greatly improved when no debugger is attached.

8 Community Building

Apart from the internal evaluation of RIOT in the context of a "real-world" application, Watr.li was and is also used as a means for community building. To increase visibility for Watr.li as a demo application, steps in multiple directions were taken, which are detailed in the following.

8.1 The Eclipse Open-IoT Challenge

The Eclipse IoT working group [53] aims to create an "open Internet of Things", that is one built on Open Source and Open Standards. Among other things, the Working Group organized the annual Eclipse Open Iot Challenge to encourage developers to build IoT applications based on these standards and software curated by the Eclipse IoT WG. Watr.li, along with its blog and a presentation video [54] was submitted to the contest, and made third place.

Participation in the Eclipse Open-IoT Challenge had several benefits: seeing the other participating projects evolve served as technological inspiration while the challenge requirement to record the progress of the project resulted in a steady stream of documentation, most of which is more widely applicable than the project itself.

Furthermore, making third place in the Open-IoT challenge provided additional hardware and funds to purchase components to further evolve the project.

8.2 Curation of a Blog

During the participation in the Eclipse Open-Iot Challenge, tutorials in the form of blog posts were posted regularly on the Watr.li website [55], which in June 2015 had about 1125 visitors. The tutorials aim to help people who are new to RIOT make their first steps, and illustrate what can be achieved with RIOT. The hope is that "hands-on" documentation like these appeal to prospective RIOT users and contributors with a less academic or corporate background, such as hackers, makers and hobbyists. Informal feedback has been positive so far, as several people have thanked the authors for the helpful tutorials.

8.3 Planning and realization of a workshop

Members of the RIOT project were invited to host a workshop at the Eclipse IoT Days in Grenoble.

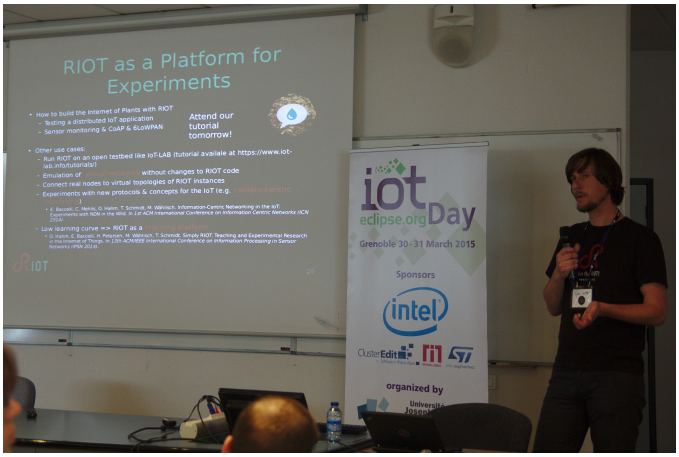

Figure 8. Oleg Hahm presenting RIOT at the Eclipse IoT Days

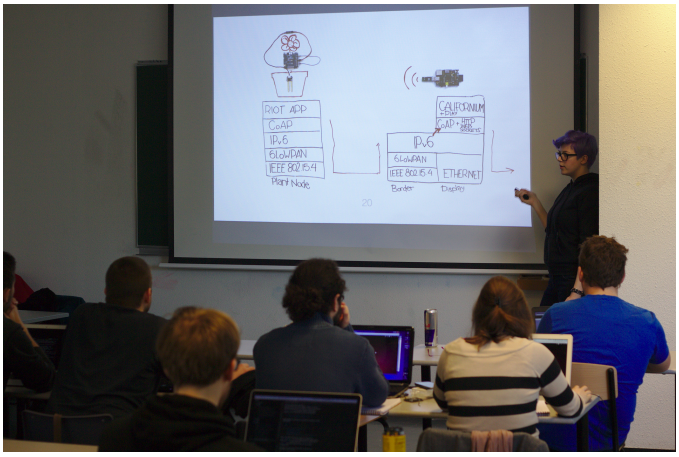

Figure 9. Lotte Steenbrink explaining watr.li to the workshop attendants

For this workshop, Watr.li was used as a demo application.

The workshop goal was to build an application employing the same protocols used for Watr.li to familiarize people with RIOT as well as core IoT technologies. To prepare all participants, a short introduction to RIOT [56] was given, and Watr.li was presented as an outlook showing the development possibilities that come with a system architecture comprised of Open Standards and Open Source.

Then, participants were each given a SAMR development board and encouraged to build their own distributed chat application using RIOT and CoAP step by step. With each step, they would learn something new about RIOT and networking. Additionally, the nature of the application could encourage communication and collaboration.

The workshop was structured such that everyone could perform the tasks at their own pace, aided by the attending members of the RIOT project. A set of instructions [57] in the form of a website was provided to all attendees. These instructions contained the following set of tasks:

The main task was to create a "chat application" over IEEE802.15.4 low-power wireless. Each feature of the chat application was designed as a single subtask which attendees could implement at their own pace. The tasks were as follows:

1) Building and flashing a RIOT application and then sending basic null-delimited strings over the wireless network with the built-in shell commands. This enabled the participants to familiarize themselves with RIOT.

2) Getting to know the RIOT shell by implementing a custom command to set a chat nickname which is prepended to every chat message.

3) Using third-party libraries not specifically written for RIOT by wrapping chat messages into CoAP packages with microcoap.

4) Learning about the basic networking capabilities of RIOT by sending and receiving the built CoAP packages over the wireless network.

5) Using additional networking APIs by implementing a shell command to change the channel by manipulating the CoAP resource endpoint, `/chat/default` or `/chat/riot`.

The tasks were designed such that they would provide an understanding of the RIOT application development workflow, showing that it is similar to writing a C application for other non-embedded operating systems. A reference implementation of the chat application can be found on GitHub [58].

8.4 Bayer Digital Summit invitation

The organizers of the Berlin branch of Hacking Health [59] discovered Watr.li through the Blog. They invited the project to showcase the prototype at the Bayer Digital Summit event on "Rising Technologies" in Düsseldorf on May 11th of 2015. Figure 10 shows watr.li's table at the summit. The on-premises feedback of attendees was positive and the authors learned about several similar technologies being developed across the different Research and Development (R&D) branches of the Bayer AG.

Figure 10. Lucas Jenß preparing watr.li's table at the Bayer Digital Summit

9 Conclusion

A real-world application has shown that RIOT is in need of further testing and demo applications, especially those interacting with hardware not running RIOT (such as watr.li's RasPi), in order to further the stability and interoperability of the system. The task force currently working on the re-designed network stack is a step in the right direction. A similar task force for the timer problems is desirable.

Watr.li has enormous expansion potential which could be used to test RIOT in more detail and learn about how smart, autonomously interacting "things" can form a new level of information detail which hasn't been accessible before. To get a fully working Watr.li network running reliably however, some more work lies ahead in terms of building upon the solid foundation and flexibility that RIOT provides in some, but not yet all, of its subsystems.

10 Outlook

Once Timers and Network Stack run in a stable and interoperable fashion, Watr.li should be used to collect valuable experience of running RIOT in a long-term, real-life deployment. Additionally, Watr.li's functionality may be expanded. This would not only benefit any office plants involved, but would also unlock new application scenarios for which the suitability of RIOT can be tested and, in consequence, improved. Such experiments and expansions may be, but are not limited to, the following.

10.1 Battery life experience

During the development process, all Plant Nodes have been powered by cable or spare battery packs made for smartphones. The next step will be to connect them to lithium-ion batteries which are smaller in size (and energy capacity) and experiment with their lifetime.

10.2 Custom Plant Node cases

Up until now, prototype Plant Nodes have been lying around close to their plant's pots. This is not feasible in the long term, as it is ugly, dangerous for the electronics involved, and annoying to move and clean. Designing a case which holds the battery and the SAM R21 board safely above the soil and lets the user plug the sensor firmly into the ground would be a major improvement. One way to create such a case could be with the help of a 3D printer.

10.3 Improved Plant Node identification

In the currently implemented system, PNs have to be matched to a plant by knowing which PN has which IPv6 address. This is not very user-friendly, since IPv6 addresses are hard to remember and generally not something that non-technical users should come into contact with. An alternative approach to displaying raw IPv6 addresses would be to generate a string that can be more easily recalled by the user using a dictionary approach. As an example, we will consider the following IPv6 address:

`2001:0db8:85a3:0000:0000:8a2e:0370:7334`

First, the segments of the IPv6 address could be mapped into a phrase-like structure:

> Adjective:Noun:Adverb:Verb:Noun:
> Noun:Verb:Adverb

Given a dictionary of words in any language, this could then be transformed into a phrase such as this one ("and" inserted for readility):

> boastful panda wildly jumping fences
> (and) yoghurt looking wisely

The latter, being much more easy to recall as the IPv6 address above, could be printed onto the custom PN cases (subsection 10.2) and then be displayed in the UI, making it easy to quickly identify any PN.

One problem with this approach is that each IPv6 address segments has $2^{16} - 1 = 65535$ possible values, whereas there are only about 10000 verbs, 4000 adverbs etc. To circumvent this, the modulo with the available number of words can be used. While the modulo causes the relation between IP addresses and identification clauses to loose its injectivity, in practice these collissions are not a problem because they would have to happen for all IPv6 address segments, which is very improbable.

10.4 Extended sensing

Apart from water, plants also have other needs. The PNs abilities should be extended to sense a multitude of other environment variables, such as oxygen, carbon dioxide, temperature and/or light. This, in turn, would require a more complex plant configuration mechanism compared to the one presented in subsection 5.6.

Another possiblity of extended sensing is of visual nature: attaching a camera to the PN would allow it to identify pests affecting the plant through image recognition.

10.5 Plant-to-plant coordination

In the current network, Plant Nodes only communicate with the Display Node, forming a tree-like traffic pattern with the Display Node at its root, acting as a so-called "sink node". This is perfectly common for many IoT environments. In fact, the Routing Protocol for Low power and Lossy Networks (RPL) [9], which Watr.li currently uses to establish connections throughout the network, is optimized for exactly this scenario.

However, the IoT is not just about conversations between things and humans. Machine to machine (or rather, thing to thing) communication is a central aspect of the IoT, and it should be part of Watr.li, too. In the future of Watr.li, plants may be enabled to talk amongst each other in addition to talking to the display node. This could enable plants to coordinate their findings amongst each other, report new or more detailed data, and even provide instructions on how to change the Watr.li ecosystem. Suppose, for example, all Plant Nodes are equipped with a light sensor. In case of under- or overexposure, the plants could coordinate amongst themselves who should switch places, and provide their humans with the resulting instructions.

Enabling plants to talk amongst each other would alter the traffic pattern of the IoP significantly. Since RPL, the underlying routing protocol currently in use, is optimized for multipoint-to-point traffic and routes all traffic through the DN, even if the communicating nodes are right next to each other, this may negatively impact the energy resources of nodes close to the root node in a multi-hop network. While an extension exists that introduces point-to-point communication to RPL [60], this does not change RPL's default mode of operation. Therefore, it may be useful to switch to a routing protocol specifically designed to work with multipoint-to-multipoint traffic when this becomes the most prominent traffic pattern. This requirement is met by the alternative routing protocols provided by RIOT, namely AODVv2 [61] and OLSRv2 [62] (in progress).

10.6 Plant-to-thing coordination

The Internet of Things bears more power than just connecting sensors to humans: Through autonomous machine-to-machine communication, things can cooperate amongst each other to shape their environment in a way that wouldn't be possible otherwise.

In the context of watr.li, some examples for this could be: plants communicating amongst each other if their needs are being met and consequently improving the situation on their own. To achieve this, the Internet of Plants could cooperate with other appliances, such as automatic watering systems (during holiday season), electronic blinds (to move when there's too much or too little light), or thermostats.

However, since home wireless automation over IEEE 802.15.4 is a comparatively new field with lots of incompatible, proprietary solutions, this vision may be a bit out of scope.

11 Appendix: Work distribution in this paper

When the work on this paper started, the system architecture design as well as the hardware were discussed and chosen by both authors. Once the general direction was clear, the authors allocated some tasks between themselves.

Lucas Jenß

- Assumed responsibility for the Front-End
- Installed Linux on the DN, including configuration of the R-IDGE USB Router
- Implemented the DN web application (subsection 6.3), the Eclipse Californium integration and the Watr.li UI
- Set up the blog software used for http://watr.li and maintained the blog layout
- Debugged issues on the SAMR board, including networking problems (subsection 7.1)
- Circumvented ND such that it was possible to send messages from RIOT to Linux with RIOT answering neighbor solicitations.
- Connected the humidity sensor to the SAMR (aided by Peter Kietzmann)

Lotte Steenbrink

- Assumed responsibility for the "Back-End"
- Wrote the RIOT application running on the PN
- Extended microcoap
- "Routing consultant"
- Designed the communication model (subsection 5.4)
- Implemented Twitter notification support on the DN

- Took care of administrative tasks regarding the Eclipse Open IoT registration etc.
- Performed the mockups for the UI design
- Updated the Contiki-based RPLd that comes with the R-IDGE USB Router to work with a current Linux kernel (subsection 6.4) (aided by Martin Landsmann)

All forms of outreach were done in teamwork. The blog was filled by both authors, and the workshops were prepared and lead with combined efforts. The promotional video was scripted by both authors. Lucas Jenß then took care of the technical realization. He provided camera equipment as well as a location. Lotte Steenbrink contributed storyboards and props.

12 Appendix: Calibration of the ADC

The employed humidity sensor basically consists of two electrodes which pass a current. These are tucked into the soil whose humidity we want to measure. The soil becomes more electrically conductive the more humid it is, increasing the current flow. On the sensor, a common collector circuit then translates the change in current flow to a change in voltage at its output, which is the sensor's output pin.

This value is sampled by the ADC on the SAMR. It is used in an operation mode that compares the input signal to a reference voltage and then quantizes the signal. The maximum sampled value is reached when the input has the same magnitude as the reference voltage, 3.3V for the Watr.li scenario. Ignoring small voltage drops caused by the transistor circuit, the sensor should roughly reach this maximum value when submerged in water. The ADC is used with the maximum sampling width of 12-bit, leading to a maximum value of 2^{12}–1 = 4095.

Ideally, the correlation between sampled voltage and value is as shown in Figure 11, i.e. the slope's gain is 1 and it's offset is 0.

A common issue with cheap ADC components is, however, that the sampled value never drops below a certain minimum and/or never reaches the maximum value. As a result, vendors include hardware capabilities to compensate these inaccuracies, as is the case on the SAMR. The calibration process is explained in this section.

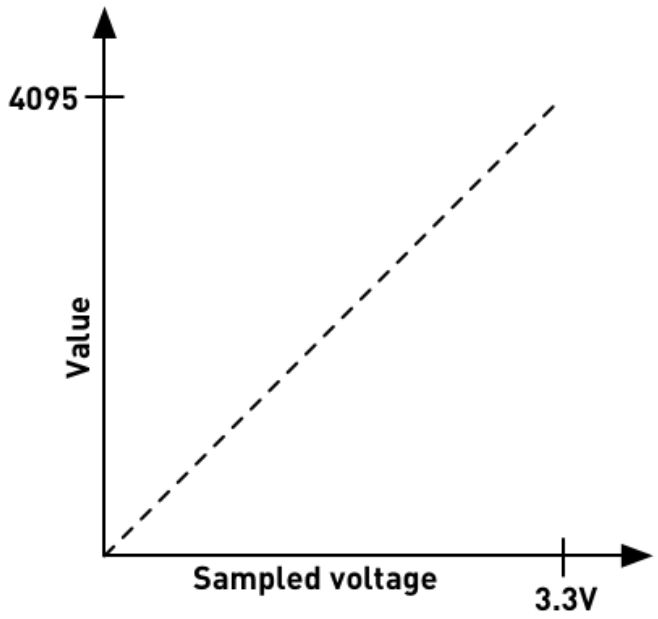

Figure 11. Correlation between voltage sampled voltage and value provided by the ADC

Some changes had to be made to the RIOT source tree in order to perform calibration. The configuration relevant to the ADC is stored in the `boards/samd21-xpro/include/periph_conf.h` file. The values which need to be tweaked for the calibration are `SAMPLE_0_V_OFFSET` (the offset) and `SAMPLE_REF_V` (the gain). To determine these values, an ADC test application [63] needs to be flashed and executed on the SAMR board. Additionally, the ADC conflicts with the default STDOUT device (`UART_0`) on the SAMR. To be able to see the debug output from the RIOT application nonetheless, `STDOUT` has to be moved to `UART_1`. Since, in contrast to `UART_0`, `UART_1` does not have a USB interface, an additional USB/UART converter had to be connected as illustrated in Figure 12.

To determine the ADC parameters, two tests need to be executed. The first one will identify the offset, i.e. the minimum value the ADC can return. For this, `SAMPLE_REF_V` needs to be initialized to `0` and `ADC_0_CORRECTION_EN` has to be disabled, i.e. set to `0` too. A GND pin has to be connected to the ADC pin `PA06` and a 3.3V pin (the reference voltage) to the `PA04` pin (compare Figure 13). Running the test application mentioned previously will now print values larger than 0,

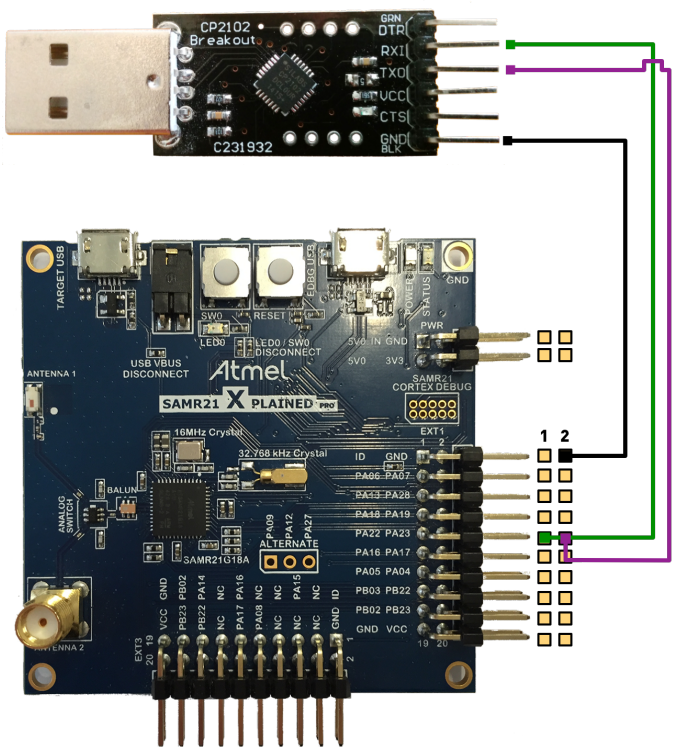

Figure 12. Wiring diagram connecting a USB/UART converter to the SAMR board.

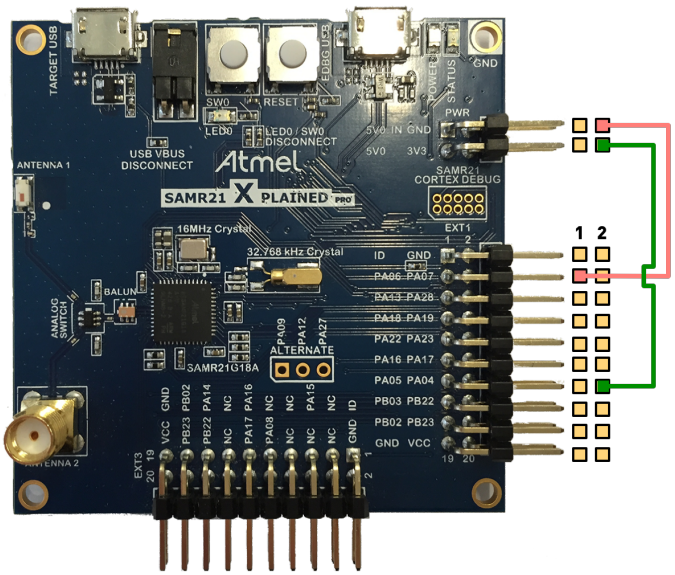

Figure 13. Wiring diagram for offset calibration of the ADC

which is the offset value. This is the value to which `SAMPLE_0_V_OFFSET` has to be set.

The second test will identify the gain correction. For this, the value that is measured by the ADC when the reference voltage is connected has to be retrieved. First, `SAMPLE_0_V_OFFSET` needs to be set to the offset value discovered previously. Second, `SAMPLE_REF_V` has to be set to 2048. Now, the 3.3V pin has to be connected to the ADC pin `PA06` as well as the reference voltage pin `PA04`(compare Figure 14). Running the test application will now yield values smaller than `4095`, which is the "measured maximum".

The general equation for the ADC hardware correction is

$$
\begin{aligned}
e &= \text{expected value} \\
m &= \text{measured maximum} \\
o &= \text{offset value} \\
g &= \text{gain value}
\end{aligned}
$$

$$
e = (m - o) * g
$$

For the `expected_value` we'll use 4095 since that

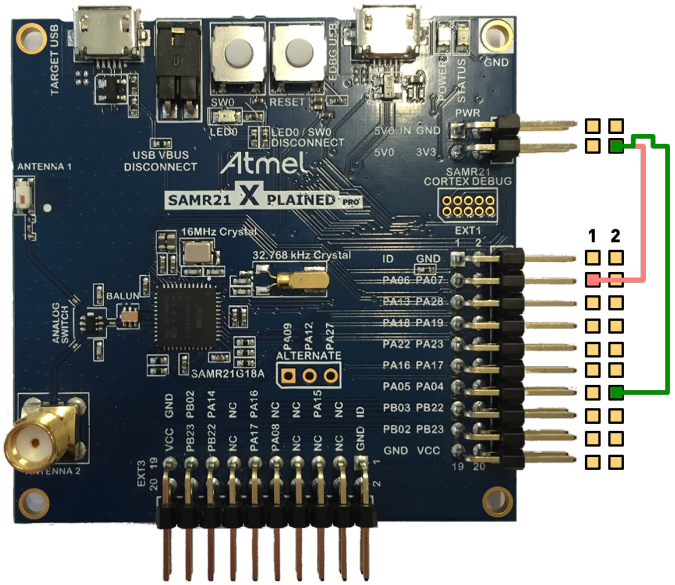

Figure 14. Wiring diagram for gain calibration of the ADC

is the maximum value the ADC can return with 12-bit resolution. The offset value and measured maximum were obtained previously by running the test application. Solving the equation for "gain value" and substituting the known variables will yield the needed value:

$$g = \frac{e}{m - o}$$

For the remainder of this section $e = 4095$, $o = 90$ and $m = 3700$ will be assumed, which yields $g \approx 1.13$. To set the microcontroller to the correct 12-bit gain value, a truncated binary fixed-point representation [64, 65] is needed. Finally, the decimal point is removed from the truncated binary representation and the resulting binary number is converted back to the decimal system:

$$\begin{aligned} 1.13_{10} &= 1.0010000101000111101\ldots_2 \\ 1.13_{10} &= 1.00100001010_2 \text{ (truncated)} \\ 100100001010_2 &= 2314_{10} \end{aligned}$$

The `SAMPLE_REF_V` parameter of the `periph_conf.h` has to be set to the result of this calculation. `SAMPLE_0_V_OFFSET` must remain set to the previously calculated offset value.

REFERENCES

[1] O. Hahm, E. Baccelli, H. Petersen, M. Wählisch, and T. C. Schmidt, "Demonstration Abstract: Simply RIOT – Teaching and Experimental Research in the Internet of Things," in *Proc. of 13th ACM/IEEE Conference on Information Processing in Sensor Networks Demo Session (IPSN)*, (Piscataway, NJ, USA), IEEE Press, April 2014.

[2] J. Balendonck, J. Hemming, B. Van Tuijl, L. Incrocci, A. Pardossi, P. Marzialetti, *et al.*, "Sensors and wireless sensor networks for irrigation management under deficit conditions (flow-aid)," in *Proceedings of the International Conference on Agricultural Engineering (AgEng 2008)*, vol. 3, pp. 452–2, 2008.

[3] M. Dursun and S. Ozden, "A wireless application of drip irrigation automation supported by soil moisture sensors," *Scientific Research and Essays*, vol. 6, no. 7, pp. 1573–1582, 2011.

[4] J. McCulloch, P. McCarthy, S. M. Guru, W. Peng, D. Hugo, and A. Terhorst, "Wireless Sensor Network Deployment for Water Use Efficiency in Irrigation," in *Proceedings of the Workshop on Real-world Wireless Sensor Networks*, REALWSN '08, (New York, NY, USA), pp. 46–50, ACM, 2008.

[5] C. M. Angelopoulos, S. Nikoletseas, and G. C. Theofanopoulos, "A smart system for garden watering using wireless sensor networks," in *Proceedings of the 9th ACM International Symposium on Mobility Management and Wireless Access*, MobiWac '11, (New York, NY, USA), pp. 167–170, ACM, 2011.

[6] Parrot SA, "Parrot - flower power - intelligent wireless sensor for your plants." `http://www.parrot.com/usa/products/flower-power/`, 2015. [accessed 2015-09-09].

[7] "IEEE standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans)," *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pp. 1–314, Sept 2011.

[8] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," RFC 6282, IETF, September 2011.

[9] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, IETF, March 2012.

[10] M. Landsmann, H. Perrey, O. Ugus, M. Wählisch, and T. C. Schmidt, "Topology Authentication in RPL," in *Proc. of the 32nd IEEE INFOCOM. Poster*, (Piscataway, NJ, USA), IEEE Press, 2013.

[11] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, IETF, June 2014.

[12] M. Kovatsch, "Coap for the web of things: From tiny resource-constrained devices to the web browser," in *Proceedings of the 2013 ACM*

Conference on Pervasive and Ubiquitous Computing Adjunct Publication, UbiComp '13 Adjunct, (New York, NY, USA), pp. 1495–1504, ACM, 2013.

[13] DFROBOT, "DFROBOT SEN0114 humidity sensor." http://www.dfrobot.com/index.php?route=product/product&product_id=599, 2015. [accessed 2015-09-09].

[14] ROSAND Technologies, "R-Idge." http://rosand-tech.com/products/r-idge/prod.html, 2015. [accessed 2015-09-10].

[15] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *29th Annual IEEE International Conference on Local Computer Networks*, pp. 455–462, IEEE (Comput. Soc.), 2004.

[16] Contiki Project, "Contiki: The Open Source Operating System for the Internet of Things." http://www.contiki-os.org/, 2015. [accessed 2015-09-10].

[17] D. C. Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, "TinyOS: An operating system for sensor networks," *Ambient Intelligence*, 2004.

[18] TinyOS Project, "TinyOS Home Page." http://tinyos.net/, 2015. [accessed 2015-09-10].

[19] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *Proc. of the 32nd IEEE INFOCOM. Poster*, (Piscataway, NJ, USA), IEEE Press, 2013.

[20] RIOT, "RIOT - The friendly Operating System for the Internet of Things." http://riot-os.org/, 2015. [accessed 2015-09-10].

[21] M. Lenders, "microcoap: initial import by authmillenon · Pull Request #2383 · RIOT-OS/RIOT." https://github.com/RIOT-OS/RIOT/pull/2383/files, 2015. [last update 2015-02-11; accessed 2015-09-10].

[22] M. Landhäusser and A. Genaid, "Connecting User Stories and Code for Test Development," in *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, RSSE '12, (Piscataway, NJ, USA), pp. 33–37, IEEE Press, 2012.

[23] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," in *Proceedings of the 22nd international conference on Software engineering - ICSE '00*, (New York, New York, USA), pp. 407–416, ACM Press, June 2000.

[24] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[25] T. Jaffey, "1248/microcoap · GitHub." https://github.com/1248/microcoap, 2015. [accessed 2015-09-10].

[26] L. Steenbrink and L. Jenss, "nodes/plant_node at master · watr-li/nodes." https://github.com/watr-li/nodes/tree/master/plant_node, 2015. [accessed 2015-09-14].

[27] M. Kovatsch, O. Bergmann, and C. Bormann, "CoAP Implementation Guidance," Internet-Draft – work in progress 03, IETF, July 2015.

[28] T. Jaffey, "Toby Jaffey auf Twitter: "@watr_li We're using it in internal projects, where it's doing fine as is. Accepting pull requests though :)"." https://twitter.com/tobyjaffey/status/568425570574446592, 2015. [accessed 2015-09-10].

[29] L. Steenbrink, "nodes/coap_ext.c at master · watr-li/nodes · GitHub." https://github.com/watr-li/nodes/blob/master/plant_node/coap_ext.c, 2015. [accessed 2015-09-14].

[30] L. Steenbrink, "microcoap: add patch to easily increase MAX_SEGMENTS by Lotter-

leben · Pull Request 2733 · RIOT-OS/RIOT · GitHub." https://github.com/RIOT-OS/RIOT/pull/2733, 2015. [accessed 2015-09-10].

[31] github.com, "Commits · 1248/microcoap · GitHub." https://github.com/1248/microcoap/commits?author=Lotterleben, 2015. [accessed 2015-09-10].

[32] Play Framework, "Play Framework - Build Modern and Scalable Web Apps with Java and Scala." https://playframework.com/, 2015. [accessed 2015-09-10].

[33] akka.io, "Akka." http://akka.io/, 2015. [accessed 2015-09-10].

[34] M. Wasilak, "Linux RPL router." http://sixpinetrees.blogspot.de/2014/11/linux-rpl-router.html. [written 2014-11-16; accessed 2015-04-21].

[35] M. Richardson, "unstrung - an IETF roll - RPL (ripple) implementation for Linux." https://github.com/mcr/unstrung. [last update 2015-01-02; accessed 2015-04-21].

[36] J. P. Taveira, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks for Linux." https://github.com/joaopedrotaveira/linux-rpl. [last update 2014-01-29; accessed 2015-04-21].

[37] J. P. Taveira, "Re: [Roll] Looking for Linux implementation of RPL for interop testing." http://www.ietf.org/mail-archive/web/roll/current/msg09258.html. [written 2015-04-17; accessed 2015-04-21].

[38] T. Cheneau, "SimpleRPL." https://github.com/tcheneau/simpleRPL. [last update 2013-06-09; accessed 2015-04-21].

[39] M. Kleine-Budde, "Re: [Linux-zigbee-devel] Can I get the patches for 3.10 kernel." http://www.spinics.net/lists/linux-wpan/msg01496.html. [written 2015-03-04; accessed 2015-04-21].

[40] A. Aring, "Re: [Roll] Looking for Linux implementation of RPL for interop testing]." http://www.spinics.net/lists/linux-wpan/msg01692.html. [written 2015-04-17; accessed 2015-04-21].

[41] ROSAND Technologies, "RPLd." http://rosand-tech.com/downloads/index.html, 2015. [accessed 2015-09-20].

[42] Wiredsource, "Samr21: ADC implementation by wiredsource · Pull Request 2063 · RIOT-OS/RIOT · GitHub." https://github.com/RIOT-OS/RIOT/pull/2063, 2015. [accessed 2015-09-10].

[43] P. Kietzmann, "cpu/samd21: Add samd21 UART_1 implementation by PeterKietzmann · Pull Request 2457 · RIOT-OS/RIOT · GitHub." https://github.com/RIOT-OS/RIOT/pull/2457, 2015. [accessed 2015-09-10].

[44] L. Jenss, "Fix unaligned access on Samr21/Cortex-M0 by x3ro · Pull Request 2727 · RIOT-OS/RIOT · GitHub." https://github.com/RIOT-OS/RIOT/pull/2727, 2015. [accessed 2015-09-10].

[45] J. Ko, J. Jeong, J. Park, J. A. Jun, and N. Kim, "Towards full rpl interoperability: Addressing the case with downwards routing interoperability," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, (New York, NY, USA), pp. 353–354, ACM, 2012.

[46] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861, IETF, September 2007.

[47] O. Hahm, "RIOT-2015.09 - Release Notes." https://github.com/RIOT-OS/RIOT/blob/2015.09-branch/release-notes.txt, 2015. [accessed 2016-01-08].

[48] L. Guan, K. Kuladinithi, T. Potsch, and C. Goerg, "A deeper understanding of interoperability between tinyrpl and contikirpl," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pp. 1–6, April 2014.

[49] E. Ancillotti, R. Bruno, and M. Conti, "Reliable data delivery with the ietf routing protocol for low-power and lossy networks," *Industrial Informatics, IEEE Transactions on*, vol. 10, pp. 1864–1877, Aug 2014.

[50] BytesGalore, "sys/net/routing/rpl: apply correct byte order for RPL messages by BytesGalore · Pull Request 2431 · RIOT-OS/RIOT · GitHub." https://github.com/RIOT-OS/RIOT/pull/2431, 2015. [accessed 2015-09-10].

[51] O. Hahm, "vtimer: vtimer_msg test crashes after 49'20" · Issue 1753 · RIOT-OS/RIOT · GitHub." https://github.com/RIOT-OS/RIOT/issues/1753, 2015. [accessed 2015-09-18].

[52] K. Schleiser, "sys: add new timer subsystem by kaspar030 · Pull Request 2926 · RIOT-OS/RIOT · GitHub." https://github.com/RIOT-OS/RIOT/pull/2926, 2015. [accessed 2015-09-18].

[53] Eclipse IoT Working Group, "Eclipse IoT Working Group." http://www.eclipse.org/org/workinggroups/m2miwg_charter.php, 2015. [accessed 2015-09-10].

[54] watr.li, "Watr.li - Eclipse Open IoT Challenge 2015 on Vimeo." https://vimeo.com/122985560, 2015. [accessed 2015-09-10].

[55] watr.li, "watr.li — Building the Internet of Plants." http://watr.li, 2015. [accessed 2015-09-10].

[56] O. Hahm, "Joining the RIOT." http://watr.li/downloads/riot-grenoble.pdf, 2015. [accessed 2015-09-10].

[57] watr.li, "Workshop Guide — watr.li." http://watr.li/workshop-guide.html, 2015. [accessed 2015-09-10].

[58] watr.li, "applications/chat at workshop · watr-li/applications · GitHub." https://github.com/watr-li/applications/tree/workshop/chat, 2015. [accessed 2015-09-10].

[59] Hacking Health, "HackingHealth — Bringing innovation to healthcare." http://www.hackinghealth.ca, 2015. [accessed 2015-09-10].

[60] E. Baccelli, M. Philipp, and M. Goyal, "The p2p-rpl routing protocol for ipv6 sensor networks: Testbed experiments," in *Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on*, pp. 1–6, Sept 2011.

[61] C. Perkins, S. Ratliff, J. Dowdell, L. Steenbrink, and V. Mercieca, "Dynamic MANET On-demand (AODVv2) Routing," Internet-Draft – work in progress 12, IETF, October 2015.

[62] O. Hahm, "routing: add OLSRv2 by OlegHahm · Pull Request 2294 · RIOT-OS/RIOT · GitHub." https://github.com/RIOT-OS/RIOT/pull/2294, 2015. [accessed 2015-09-10].

[63] L. Steenbrink and L. Jenss, "nodes/adc_test at master · watr-li/nodes · GitHub." https://github.com/watr-li/nodes/tree/master/adc_test, 2015. [accessed 2015-09-14].

[64] University of Wisconsin, "Binary Fixed Point." http://www.cs.uwm.edu/~cs151/Bacon/Lecture/HTML/ch03s07.html, 2015. [accessed 2015-09-14].

[65] Exploring Binary, "Decimal/Binary Converter - Exploring Binary." http://www.exploringbinary.com/binary-converter/, 2015. [accessed 2015-09-14].