Flexible Devices for Arctic Ecosystems Observations

Lukasz Sergiusz Michalik
Otto J. Anshus
John Markus Bjørndalen

November 2017

Abstract

Devices for observing the environment range from basic sensor systems, like step-counters, through wild-life cameras, with limited processing capabilities, to more capable devices with significant processing, memory and storage resources. Individual usage domains can benefit from a range of functionalities in these devices including flexibility in prototyping, on-device analytics, network roaming, reporting of data, and keeping the devices and services available in spite of failures and disconnections. The problem is that either the devices are too resource limited to support the range of functionalities, or they use too much energy.

An important usage domain is COAT – Climate-Ecological Observatory for Arctic Tundra. Presently, best practice includes deploying wild-life cameras in the Arctic Tundra, and visiting them to manually collect the recorded observations. This is a problem because such devices can only be rarely visited, and manual approaches to fetching data and storing it do not scale with regards to number of cameras, handling of human mistakes, and with freshness of observations.

We present a prototype for observing the environment composed of a general purpose computer, a Raspberry PI, in combination with an ARM-based microcontroller. The combination enables us to create a more energy efficient prototype while supporting the needed functionality.

The prototype improves on currently applied methods of observing the Arctic tundra. The prototype automatically observes the arctic tundra through camera, humidity and temperature sensors. It monitors itself for failures. The data is stored locally on the prototype until it can be automatically reports to a backend service over a wireless network.

We have conducted experiments that show that task scheduling can reduce power consumption, and we identify some additional points that need to be addressed before we can run the device for long periods on battery power.

This paper was presented at the NIK-2017 conference; see http://www.nik.no/.

1 Introduction

Determining the characteristics of a platform for observing the arctic tundra is challenging for several reasons. The arctic conditions pose by itself challenges of limited physical access to observation units, demanding weather conditions, and lack of infrastructure for energy and data networks. There are also challenges in defining a platform able to allow for tracking progress in technology and a never ending stream of new observational requirements and needs.

Platforms for observing the arctic tundra are based on devices ranging from microcontroller-based devices including basic single-purpose sensors and wild-life sensors with limited processing capabilities, to computer-based devices with significant processing, memory, network, and storage resources.

An interesting and important effort to do observations of the arctic tundra is the Climate-Ecological Observatory for Arctic Tundra (COAT) [1]. COAT makes use of wild-life cameras placed below and above snow at multiple locations in Northern Norway. Microcontroller-based systems like wild-life cameras typically have limited functionality for doing flexible and adaptive observations, no automated reporting of observed data through data networks, no way to (remotely) reconfigure the camera unit after deployment, and no on-camera processing of the observed data. The wild life cameras basically must be manually configured, they can only do observations through a camera, temperature, and humidity sensors, and data must be manually and physically fetched by physically visiting the camera unit. However, the energy efficiency is high and the total energy usage is very low enabling operation for 6-12 months on a single battery charge.

Several types of Internet of Things (IoT) [2] sensor systems currently exist, such as Vicotee Njord[3], SensorTag 2[4] or Waspmote[5]. All of these systems can monitor the environment with multiple sensors and communicate with remote services while being low-power-consumption devices. However, some of these systems are still in a development stage and they can not be tested yet (Vicotee Njord), others cannot be extended with specific sensor types because of a closed design (SensorTag2) or due to custom module interface (Waspmote).

On the other hand, there is a variety of single-board microcontrollers, like the Arduino, and single-board general purpose computers, like the Raspberry PI. These allows for more open, flexible and general purpose. They can also be extended with additional modules because they support common interfaces such a Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), and Camera Serial Interface (CSI). A variety of peripherals, a big community of users and open-source code and examples make them an efficient platform for prototyping a customized solution for Arctic Observatories.

2 Related Work

Internet of Things (IoT) based solutions became popular in projects like smart-cities [14], [19], safety [12], [13] and health-care [15], [17], [18]. However, the approaches that are used in those solutions cannot be directly applied for Arctic region observations. Nevertheless, the properties of IoT components, which allow for flexible, modular design, where an incentive for us to start the project.

Even though Internet of Things has been around for several decades it keeps posing similar problems for IoT developers. In a paper from 2006 [20] issues with

sensor network deployment, network coverage and energy consumption were pointed out. We have experienced similar concerns in our prototype. Those questions remain open even though the IoT technology has progressed, the available development's boards are more powerful, energy-efficient, and they have longer radio links. All that did not eliminate the issues with deployment of multiple devices in hardly accessible regions, occurrences of network partitioning or high energy demand.

We have grasped an idea of long-range radio communication for image sensor network from [16]. However, in [23] we concluded that sending the captured images or other type of heavy data via LoRa mechanism is not possible because of a limited data rate. Long range radio modules offer up to 20 kilometers of range with the costs of a small data payload (around 200 bytes) and long data delivery time (around 5 seconds). Such radio links do not permit to send anything else than raw text data, and thus they can not be used to deliver hundreds of images per day in an efficient way. However, LoRa can carry efficiently different types of data such as the device's internal status (called also keep-alive message), which includes information about remaining battery and storage capacity and occurred failures.

An interesting aspect related to the issues with growing sensor networks was described in [21]. The article points out that the heterogeneity of sensor devices in a single network introduces a challenge of finding the nodes capable of either processing or storing a data. We have developed a single device, along with a storage node, which in the nearest future might require to be extended with more types of sensors. If a significant number of different devices is deployed in the field, then It introduces a challenge of finding the data processing nodes in an efficient and energy-wise way. The article introduces tree types of algorithms (*request*, *traverse* and *mixed*) addressing this problem.

The idea of a customized, low-powered, long range device (called *SensorScope*) for environmental observations was introduced in [22]. The prototype is build on a Texas Instruments MSP430 microcontroller equipped with Semtech XE1205 radio transceiver which offers a range of up to 1200 meters and a decent power efficiency. This successful and already commercialized solution revealed challenges with programming and debugging the bare-microcontroller based system. A microcontroller alone could not be used to process the gathered data in-situ and would require all the data to be sent to backend-services. We decided to combine a general purpose computer (Raspberry PI 3) with a microcontroller in order to achieve some degree of data processing on site and to limit the system's power consumption. Our solution uses a newer type of transceiver with a theoretical range up to 20 kilometers. The system described in [22] was deployed on a glacier in Swiss Alps, which pose similar environmental conditions as the Arctic Tundra - the target deployment destination for our prototype. High energy consumption and instability issues in our solution cause, however, that in order to test it in the field in a similar way, we would have to first improve it significantly.

3 Architecture

We present both hardware and software architecture of our prototype which we called an *Observation Unit* (OU).

Hardware

To support flexible sensor prototypes and implementations, we need a system that lets us combine the power and programmability of a general purpose embedded system (Raspberry Pi class) with a microcontroller-style power efficient system. Sensor prototypes can start development and experiments running in the general purpose part, and optimize power efficiency by moving tasks over to the microcontroller. The architecture presented here reflects the two-board solution used in our current prototype, but the design could, in principle, be implemented with a single chip or single board system.

Observation Unit

As outlined before, our prototype is composed of a *main unit* and a *support unit* in the current stage. The units are connected via a communication bus as presented in Figure 1. To communicate with the *OU*, we introduced a simple back-end server which is referenced as a *gateway* in the remaining part of the paper.

The *main unit* is the heart of the system and its main purpose is to collect and send data such as sensor readings and internal state (health status) to the *backend services*.

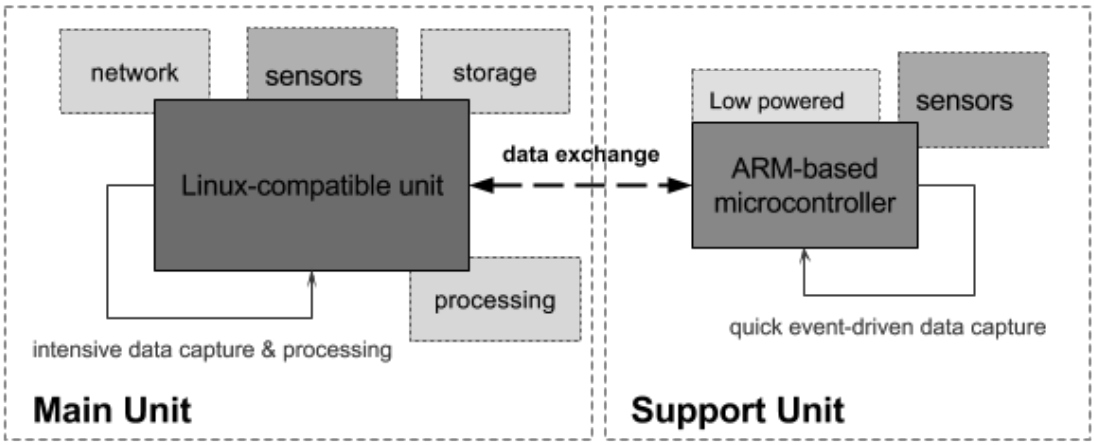

Figure 1: The Observation Unit hardware architecture.

The *support unit* is a proposed solution for limiting the energy consumption of the *main unit*. We have assumed that linux-compatible devices might demand too much energy to be operative during the whole period of observations. In this case, the *main unit* remains inactive (powered off) until the *support unit* determines that more powerful board should be turned on in order to execute its defined tasks. We were aware of the fact, that some devices are not equipped with a hardware clock, which prevents them of stamping the captured data with an accurate time, which might be important in order to analyze recorded information. Therefore some kind of time measuring module is required as well to be a part of the *support unit*.

Gateway

The *gateway's* hardware architecture consists of a computer unit with storage, network access and communication modules compatible with the networks used by the *OU*. The *gateway* architecture is presented in Figure 2.

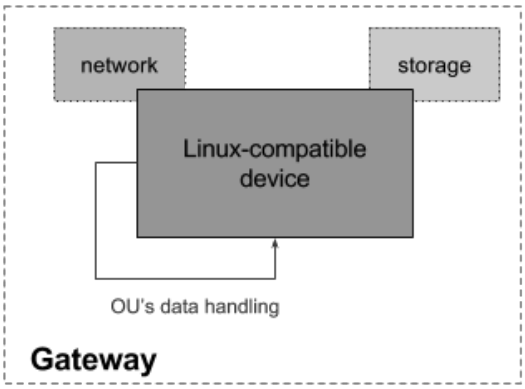

Figure 2: The Gateway hardware architecture.

Software

Observation Unit

The *main unit* executes batches of tasks based on a defined order. Single task should finish its execution as fast as possible to save device's power. When last task finish its running then device send information to the *support unit* with wake-up request on a specified time value. The *support unit* works as a node that executes only tasks requested by the *main unit*.

Gateway

Collecting data from the *OU* requires a type of *backend-services* running on the *gateway* in order to examine device's ability to communicate with an external server. Such server is responsible for receiving data and sending confirmation of such event back to the *OU*. The *gateway* software architecture is presented in Figure 3.

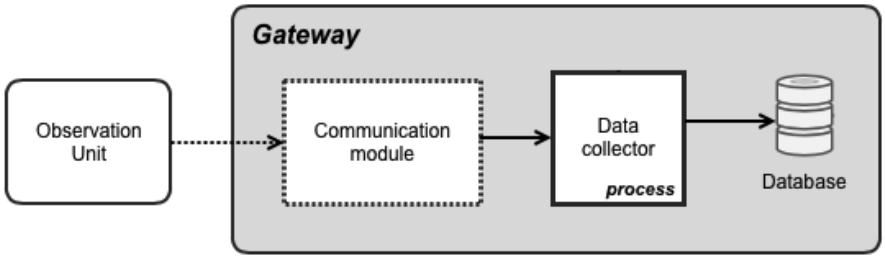

Figure 3: The Gateway software architecture.

4 Design

Observation Unit

A proposed design of the flexible device is composed of two development boards: a Raspberry-like based *main unit* and a Arduino-like microcontroller-board, the *support unit*, as shown in Figure 4. The first needs decent storage space for observations' data along with a camera-like sensor and some long range link to the *gateway*. It is also required for the unit to have an external way to communicate with the *support unit*. The latter, in order to keep the current time, should be

equipped with a hardware time clock like Real Time Clock (RTC) with a decent accuracy.

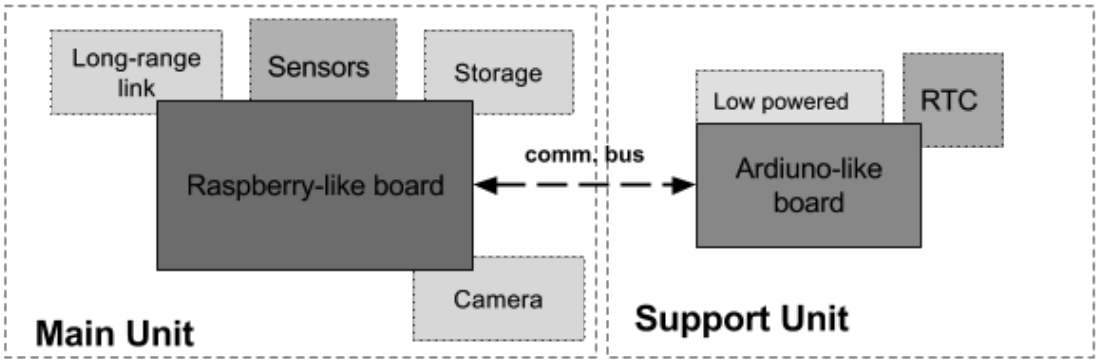

Figure 4: The Observation Unit design.

The software stack for the *main unit* should include a Linux compatible operating system along with popular programming languages in order to keep an ease of prototyping for the flexible device and make it open for further software modifications.

Gateway

The *gateway* might be based on a commodity Linux compatible device equipped with the same long range module as the *OU* in order to receive data. If we assume that one of the requirements for this unit is to have a decent long range coverage then it would be desired to use a small size hardware that might be easy placed in a high spot above the ground with a clean line of sights to the *OU*. Following this assumption, a Raspberry-like board seems to be perfect for the *gateway*.

5 Implementation

Observation Unit

The *observation unit* is based on Raspberry Pi 3B (the *main unit*) and Arduino Pro Mini (the *support unit*). Both operate on 5 volts of input voltage, which allows to use the same power source for them without additional logic-level converters. The prototype along with used modules are depicted in Figure 5.

To start experimenting with different classes of sensors, we have added two initial sensors: a camera and a temperature and humidity sensor.

The *communication module* selected for the prototype is LoRa module, specifically Dragino LoRa Bee,which is based on the Semtech SX1276 transceiver chip. The module is connected via SPI to the *main unit*.

The C++ application for sending data from the *OU* to the *gateway* is based on the *LowCostLoRaGw*[9] library made by Congduc Pham from the University of Pau in France. The code selects information from sqlite3 database stored on the *main unit*, where the *OU* saves all sensors' readings and its health status. The data is then sent to the *gateway*.

The LoRa module operates in 868 MHz band frequency and it is configured to use 125 kHz Bandwidth (BW), Spreading Factor (SF) 12 and Command Rate (CR)

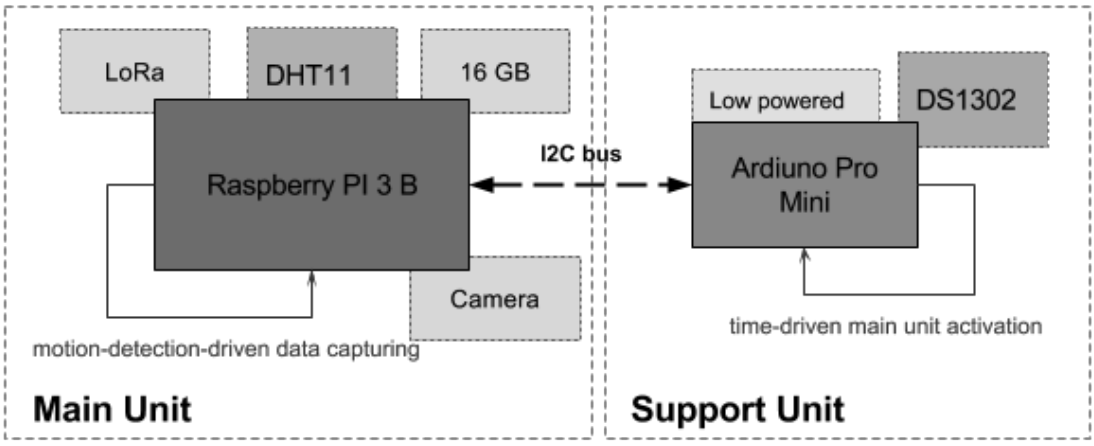

Figure 5: The Observation Unit prototype.

4/5 for the maximum range. This configuration is optimal in the terms of long range communication, as specified in *Semtech LoRa Modem Design Guide*[6].

The *camera module* used in the prototype is *Raspberry Pi Camera Module v2*, which comprises an 8-megapixel sensor. We used a Python program to capture in a resolution of 1024 x 768 pixels, saving them to the device's storage space. We did not send the images using the LoRa module.

As an additional sensor, we used a *DHT11 humidity and temperature sensor* connected with a single bus (Single-Wire Two-Way) to the *main unit*, which means that a single communication line is used to request and receive 40-bits of data from the sensor.

We need to turn off the *main unit* in order to save the *OU*'s power. The issue with a Raspberry Pi is that it cannot wake itself up after sleeping for a while. Therefore, the *support unit* is responsible for waking up the Raspberry (after the Raspberry has shut down) using an external timer event. The prototype's boards communicate with each other using an I2C bus. However, since one of the Raspberry PI 3B I2C pins is also used to turn this device on, a signal switch (Keys SRD-05VDC-SL-C relay in this case) is required. It allows to share the same pin by two separate circuits and switch the pin to the circuit which is used at the moment.

An Arduino Pro Mini 5V was selected as the *support unit* for the prototype. It runs on only 12 mA of current and its firmware is written in C++. Therefore, every time the device's code is changed, it must be compiled on an external host and flashed to the board using an FTDI USB serial cable.

Since none of the boards used in the project have a build-in hardware clock, the *support unit* was equipped with an RTC module, specifically DS1302. In the current prototype the RTC module is used by the *support unit* to turn on the *main unit* at a specified time and to pass the current time for it. The first initialization is done by the *main unit*, using I2C communication with the *support unit* and *timezonedb.com* API.

A Raspberry PI 3B is used as the *gateway* as showed in Figure 6 and uses the same LoRa module as the *OU*. In order to make this device be able to receive LoRa messages, a modified *LowCostLoRaGw*[9] library is used. The application is executed as soon as the device's operating system is up and running and it remains in the background until the system's shutdown. All received messages are stored in

an sqlite3 database for further processing.

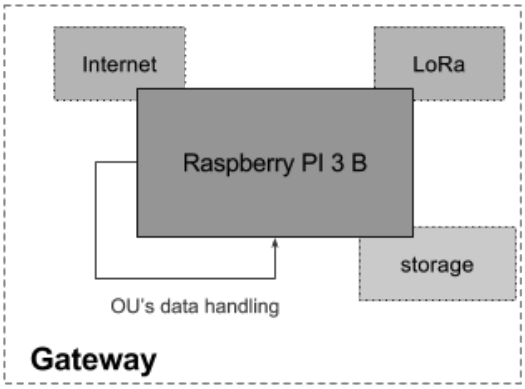

Figure 6: The Gateway prototype.

6 Experiments

In order to determine how well the *OU* performed in terms of power-consumption and task-execution times, we have designed and performed several experiments, two of which we report from here. See [23] for more details.

We used a *Mini USB Charger Doctor*[11] to measure power consumption. This device detects electric current consumption and voltage levels in the circuit connected to it via USB. Its specification states that it can measure a current ranging from 0 A to 3 A (with +- 1% of error and minimum resolution of 10 mA) and time (only when a device is draining power) from 0 to 99 hours.

Extreme load scenario

The main purpose of this experiment was to determinate how much energy the *OU* consumes when tasks are executed in predefined intervals. The device was monitored during 24 hours of testing for every interval case (the same for every task during the whole test) which gives 4 days of testing in total. To obtain the reference value for power consumption, the device was left in an idle state for another 24 hours. Each task was running independently in an endless loop with a sleeping time specified by the interval length. The list of tasks and intervals used are presented in Table 1.

Task name	Idle-state Interval				System idle
	1s	10s	30s	60s	
1 x LoRa message sending	x	x	x	x	
1 x RTC value reading	x	x	x	x	
1 x DHT11 sensor's value reading	x	x	x	x	
1 x Camera image capture	x	x	x	x	
Test duration	24h	24h	24h	24h	24h

Table 1: List of tasks executed during extreme load scenario.

Medium load scenario

To futher conserve power, we experimented with powering down the Raspberry Pi in the *OU*. As soon as the device finished its booting sequence, it was executing the same tasks as described in section 6 and then it was turned off for the specified sleep time. This cycle was repeated in a loop of 24 hours. Four such experiments were performed – one for every interval. Tasks performed during this scenario and the lengths of power-off intervals are specified in Table 2.

Task name	Duration of shutdown state				System idle
	1s	60s	30m	60m	
1 x LoRa message sending	x	x	x	x	
1 x RTC value reading	x	x	x	x	
1 x DHT11 sensor's value reading	x	x	x	x	
1 x Camera image capture	x	x	x	x	
Test duration	24h	24h	24h	24h	24h

Table 2: List of tasks executed during medium load scenario.

7 Results

This section presents the results of the experiments conducted to measure the device's power consumption in the previously defined scenarios.

Extreme load scenario

The main purpose of this scenario is to determine how much energy the *OU* consumes when the *OU* alternates between executing tasks and idling. The results presented in Figure 7 show that the power consumption is proportional to the frequency of executing tasks. In the most intense case, the power consumption is 1,7 times higher than in the idle state. With about 1 minute between tasks, the power consumption was close to the idle state.

An interesting observation is that even in the idle state, the device consumes around 5 Ah a day. This would drain a decent smartphone battery, which has a capacity of around 2.5 Ah in half a day.

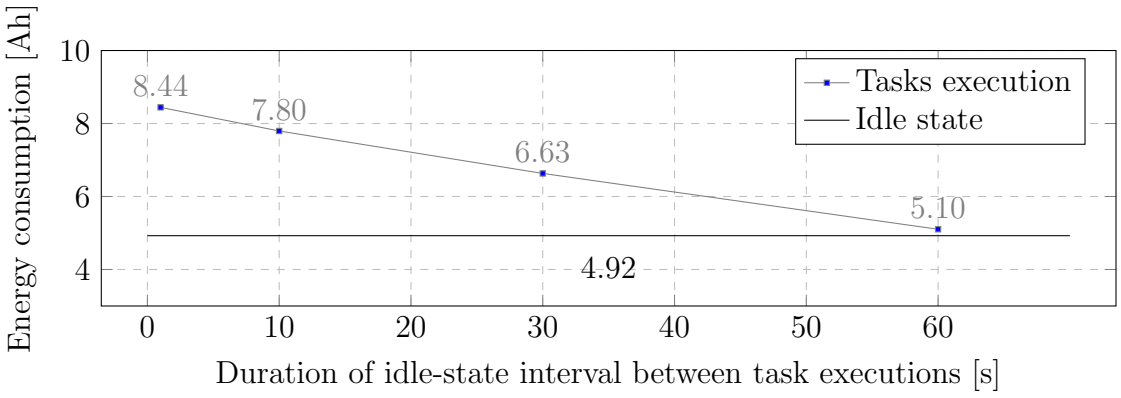

Figure 7: Energy consumption measured over 24h in extreme load scenario.

Medium load scenario

Figure 8 shows the energy consumption levels for 24-hour-long periods of task executing, interrupted with shutdown states of various lengths. The lengths of shutdown states are marked on the horizontal axis of the figure. The highest energy consumption measured in this scenario (5.43 Ah obtained for 1-second long shutdown intervals) is comparable to the lowest energy consumption measured in the extreme load scenario (5.10 Ah obtained for 60-seconds long idle intervals), as presented in Figure 7.

We have observed that the overhead for shutdown (around 8 seconds) and booting the Raspberry (ranging from 25 to 45 seconds) is large enough that the effective interval between tasks executing is longer than the shutdown intervals presented in the figure. For a 1 second interval, the effective downtime is around 34-54 seconds, which means that the 1 second interval is comparable to the 1 minute idle state in Figure 7.

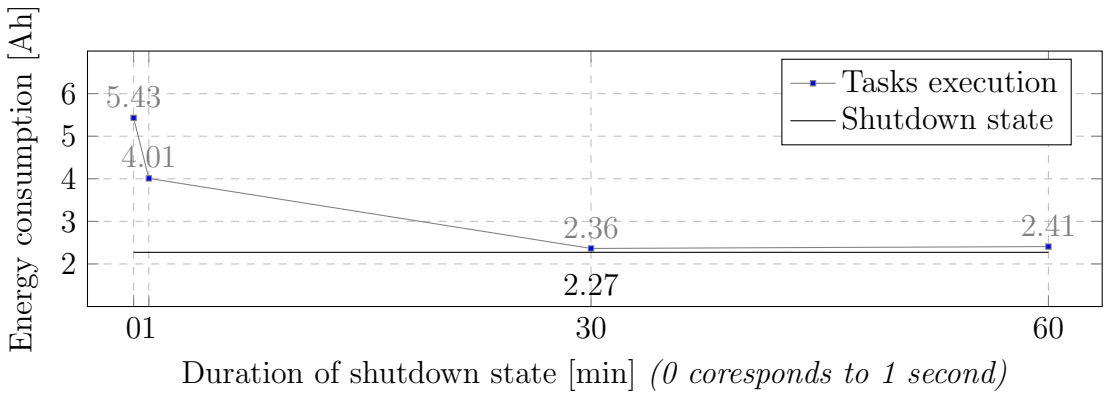

Figure 8: Energy consumption measured over 24h in medium load scenario.

An important observation is that the unit is still drawing conderable power even in the shutdown state. The *support unit*'s power consumption was measured to around 20 mA, which means that a powered-off Raspberry PI 3 consumes around 75 mA. To address this, we have used an Arduino-based Sleepy Pi 2 as a replacement *support unit*, and initial experiments show that it can cut the power to the Raspberry completely when it's shut down. To reduce power consumption further, we need to investigate reduction of boot and shutdown overhead in the Raspberry.

8 Future Work

Right now the *observation unit* consumes too much energy to be placed in the Arctic environment and survive on batteries for several months. The Raspberry PI 3 used as the *main unit* drains power even in the shutdown state, which requires an external power circuit controlled by the *support unit* to cut off such *power-leakage*.

Another approach to limit the *OU*'s power demand would be to replace Raspberry PI 3 with a much more power-efficient Raspberry PI Zero (a less powerful version of Raspberry PI family boards with a single core chip). Another option is to experiment with letting the *support unit* power down the Raspberry to conserve power. Initial experiments with a Sleepy Pi 2 shows that this can be a promising approach.

9 Summary

The goals of this project were to develop a system capable of observing the Arctic environment, detecting motion in front of the installed camera, reporting collected data and the state of system's sensors and to design a reference system for exploring power issues and corresponding solutions. Such goals were achieved in a form of proposed and examined in this paper prototype.

The prototype is a first attempt towards monitoring the wildlife in a more robust way than just through a simple photo camera capturing images when the motion is detected. The research brought much positive outcome. The *observation unit* provides a way of delivering information on a long distance using a low-powered communication module.

On the other hand, the *OU* consumes too much power to be placed in the field in its current form. The increased functionality of the prototype definitely did not come without a price, but with more extensive examinations and the addition of several missing parts it could be polished to the state when it proves useful in the real environment.

Since the *main unit* is under the control of a linux operating system, it allows to use much more existing software solutions and programming languages to deliver the desired functionality, which could not be provided by the embedded systems only. It opens a door for introducing further improvements without the necessity of being familiar with IoT development as a prerequisite.

The implementation of the *observation unit* prototype showed how many different aspects need to be taken into account beyond software implementation. The system's stability, for example, does not depend only on the quality of the software part, but also on the selected hardware components, which sometimes are less reliable than others. Testing several models of the same type of peripheral is highly advisable when choosing a candidate for the final solution to be deployed in the field.

References

[1] "COAT", `http://www.coat.no/`. Accessed 28.08.2017.

[2] "Internet of Things", `https://goo.gl/gJ3L6d`. Accessed 28.08.2017.

[3] "Vicotee Njord", `http://www.vicotee.com/`. Accessed: 18.03.2017.

[4] "SensorTag 2", `http://www.ti.com/tool/cc2650stk`. Accessed: 18.03.2017.

[5] "Waspmote", `https://goo.gl/MSKObw`. Accessed: 18.03.2017.

[6] "Semtech LoRa Modem Design Guide", `https://goo.gl/U1ZMvK`. Accessed 17.04.2017.

[7] "pi-timolo", `https://goo.gl/KWNgn6`. Accessed 16.04.2017.

[8] "picamera", `https://goo.gl/johwTp`. Accessed 16.04.2017.

[9] "Low-cost LoRa IoT", `https://goo.gl/aEwHT3`. Accessed 17.04.2017.

[10] "Adafruit Python DHT Sensor Library", `https://goo.gl/8sRbul`. Accessed 18.04.2017.

[11] "USB Charger Doctor", `https://goo.gl/LA8Zyf`. Accessed 1.05.2017.

[12] S. Imran, V. Sirivastava, I. Hwang, Y.-B. Ko. *Poster: CarSafe – Feasibility Study of a Life Saving System in a Car*. MobiSys '16 Companion Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion, page 34. Jun 2016.

[13] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, D. Wagner. *Smart Locks: Lessons for Securing Commodity Internet of Things Devices*. ASIA CCS '16 Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, pages 461-472. May-June 2016.

[14] J. Moore, G. Kortuem, A. Smith, N. Chowdhury, J. Cavero, D.l Gooch. *DevOps for the Urban IoT*. Urb-IoT '16 Proceedings of the Second International Conference on IoT in Urban Space, pages 78-81. May 2016.

[15] A. S. Yeole, D. R. Kalbande. *Use of Internet of Things (IoT) in Healthcare: A Survey*. WIR '16 Proceedings of the ACM Symposium on Women in Research 2016, pages 71-76. March 2016.

[16] C. Pham, V. Lecuire. *Building low-cost wireless image sensor networks: from single camera to multi-camera system*. ICDSC '15 Proceedings of the 9th International Conference on Distributed Smart Cameras, pages 158-163. Sept 2015.

[17] V. Tan , S. A. Varghese. *IoT-Enabled Health Promotion*. IoT of Health '16 Proceedings of the First Workshop on IoT-enabled Healthcare and Wellness Technologies and Systems, pages 17-18. Jun 2016.

[18] D. T. Lai. *Keynote Talk: Harnessing Health IOT for Smart Healthcare*. IoT of Health '16 Proceedings of the First Workshop on IoT-enabled Healthcare and Wellness Technologies and Systems, page 1. Jun 2016.

[19] J.-E. Kim, M. Bessho, N. Koshizuka, K. Sakamura. *Enhancing public transit accessibility for the visually impaired using IoT and open data infrastructures*. URB-IOT '14 Proceedings of the First International Conference on IoT in Urban Space, pages 80-86. Oct 2014.

[20] A.S. Tanenbaum, C. Gamage, B. Crispo. *Taking Sensor Networks from the Lab to the Jungle*. Computer 39 (8), pages 98-100. Aug 2006.

[21] R. Kolcun, D. Boyle, J. A. McCann. *Efficient In-Network Processing for a Hardware-Heterogeneous IoT*. IoT'16 Proceedings of the 6th International Conference on the Internet of Things, pages 93-101. Nov 2016.

[22] F. Ingelrest, G. Barrenetxea, G. Schaefer, M Vetterli, O. Couach, M. Parlange. *SensorScope: Application-Specific Sensor Network for Environmental Monitoring*. ACM Transactions on Sensor Networks (TOSN), 6(2), Article No. 17. Feb 2010.

[23] L. S. Michalik. *EC3 - Edge Command-Control-Communication System for Arctic Observatories*. Master Thesis, UiT - The Arctic University of Norway. May 2017.