Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

FlockLab 2.0: Linux Platform

Semester Thesis

Dario Leuchtmann
ldario@student.ethz.ch

Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich

Supervisors:
Jan Beutel
Roman Trüb
Prof. Dr. Lothar Thiele

June 24, 2019

Acknowledgements

I thank to Prof. Dr. Thiele that he gave me the chance to do this semester thesis at his institute. Furthermore I thank Dr. Jan Beutel who has inspired me to do a semester thesis about FlockLab because of his course "Low Power System Design". Last but not least I thank Roman Trüb who advised me during the entire project.

Abstract

FlockLab [1] is a wireless sensor network (WSN) testbed for embedded systems and is located on the campus of ETH Zurich. The nodes are time synchronized in the backend which allows distributed and synchronized power measurement and GPIO tracing and actuation. Since its launch in 2012 FlockLab has developed a high utilization by dozens of universities world wide for multiple research papers. All this leads to the desire to further develop the FlockLab system.
The current hardware is several years old and has therefore neither free capabilities left to install new services nor improve existing ones. To encounter this problem the goal of this thesis is to port all the currently available services of the FlockLab system to a new platform and add more accurate power measurement capabilities.
With this thesis we are going to show that not only the power measurement and GPIO tracing frequency can be improved but also enough free CPU capacity is left to double the load from FlockLab services. To achieve this we have ported the observer to a Beaglebone Green.

Contents

Chapter 1

Introduction

1.1 Motivation

Since the launch of FlockLab in 2012 it has become more and more important. From 2012 with in total 156 tests the usage increased to a total of 12570 tests in 2018, which can be seen in Figure 1.1, and is currently used by 125 different institutions all over the world [1]. This clearly shows the importance of a WSN testbed such as FlockLab for research purposes. Not only the usage increased but also the available technologies in the market changed dramatically. The processing capabilities of low power wireless sensor nodes have increased over the past few years in such a way that also the requirements for a testbed environment have changed. While for example the Mica [2] platform from 2002 is only able to handle a few kilo bytes of code newer platforms such as STM32L4 [3] uses a powerful 80 MHz ARM Cortex M4 [4] SoC. The combination of a high usage of FlockLab and new requirements to such testbeds directly leads to new desires for FlockLab.

Unfortunately the current Gumstix [5] platform which is used for the FlockLab observers uses an old Gumstix verdex pro XL6P chip which is not very powerful compared to modern chips. When a test is running around 98 % of the CPU capacity is used. For this reason it is neither possible to add new services to the FlockLab testbed nor improve the existing ones, so there was the idea to build a new version of FlockLab, the so called FlockLab 2.0.

The goal of a FlockLab 2.0 is to have a more accurate power measurement and not only measuring the current but also the voltage. The power measurement should also have finer grained data points regarding the frequency, as well as the GPIO tracing should work with a high frequency of up to 10 MHz and another requirement for the future is to have live Debugging. Furthermore, the testbed should be added more nodes which cover indoor and outdor sites and should be distributed across the city of Zurich.

The goal of this semester thesis is to port the previous FlockLab observer services running on the Gumstix Platform to the more powerful Beaglebone Green [6] platform running with Debian 9 [7]. Furthermore, the power profiling should

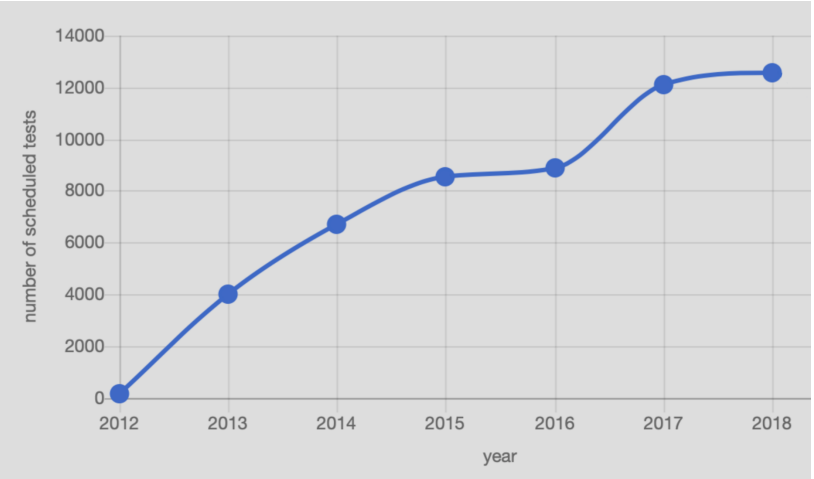

Figure 1.1: The total number of users by year of the FlockLab test bed environment [1].

be made more accurate by using the RocketLogger [8] to perform the power measurement.

1.2 Related Work

There are multiple wireless sensor network testbed environments available which can be used for free for research purposes. In this section we are going to give a glimpse of the available networks that are constructed in a similar way as FlockLab. Therefore, we have picked three of the most widely used testbeds that have been developed over the past 15 years.

- MoteLab [9] was a WSN testbed at Harvard University. When it started in 2005 It consisted of 26 Mica2 motes and was enlarged over time to more than 190 Tmote nodes spread over multiple floors.
- The Fit IoT-Lab [10] in France consists of over 1500 spread across six different sites with some of them mobile. Those mobile nodes allow tests with moving devices which will be a part of future IoT devices. The collected data consists of network related metrics such as throughput, delay, and overhead.
- Indriya [11] is another highly used testbed at the Singapore University. Currently they are developing Indriya2 consisting of 74 TelosB and 28 CC2650 nodes to allow heterogeneous network tests with different node

types at the same time. Furthermore they want to increase the data collection rate.

MoteLab was a very early WSN testbed that allowed, similar to FlockLab, to program tests via a web interface. A main difference between MoteLab and FlockLab is that in MoteLab the nodes were directly connected to the server without an observer in between, which makes it harder to replace a target with another one.
Whereas IoT-Lab focuses on the network capability by measuring metrics like throughput, delay and overhead, FlockLab also measures the power consumption. Indriya original software was derived from MoteLab and therefore had a lot of limitations Indriya2 is designed to address these limitations. Similar to FlockLab 2.0 Indriya2 is designed to support multiple targets and support higher data rates.
With the new hardware platform in FlockLab 2.0 it is possible to add new services and therefore to meet the future trends.

1.3 Naming Definitions

We are going to use some terms in this thesis which are not commonly used in this way. Therfore I introduce the naming definitions of this thesis to reduce the confusion with all the different hardware and software components.

- **previous FlockLab** refers to the old FlockLab with observers running on the Gumstix platform.
- **FlockLab 2.0** refers to the new FlockLab with observers running on the Beaglebone platform.
- **Gumstix** observer refers to the old observer used in the previous Flocklab.
- **observer** refers to a Beaglebone green running with Debian 9. The exact task of the observer will be explained in a later chapter.
- **RocketLogger** [8] refers to a measuring device for voltages and currents.
- **target** refers to the devices under test that build the WSN.

Chapter 2

Previous FlockLab

2.1 General

FlockLab is a testbed for Wireless Sensor Networks (WSNs) at ETH Zurich consisting of 27 nodes spread over one floor of the ETZ building. It is designed as a 3-Tier-Architecture each of which has a different purpose. The three different layers are:

- System layer
- Observation layer
- Server

These three layers are depicted in Figure 2.1 The communication between the system layer and the observation layer is via a serial connection which is either USB or UART and an additional I2C connection plus multiple GPIO pins.
For the communication between the observation layer and the server a TCP connection is used where the observation layer initializes the connection and the server listens and sends to it.

2.1.1 System layer

In the system layer are the actual nodes, the so called targets, that build the WSN to perform the tests. The targets are the devices under test (DUT) and can be individually programmed by the user. During the test the power consumption, the GPIO input and output and the Serial input and output is measured and traced. This data is then collected by the server.

2.1.2 Observation layer

The observation layer consists of more powerful nodes, which are called observers and are based on the Gumstix platform. The observers perform the measure-

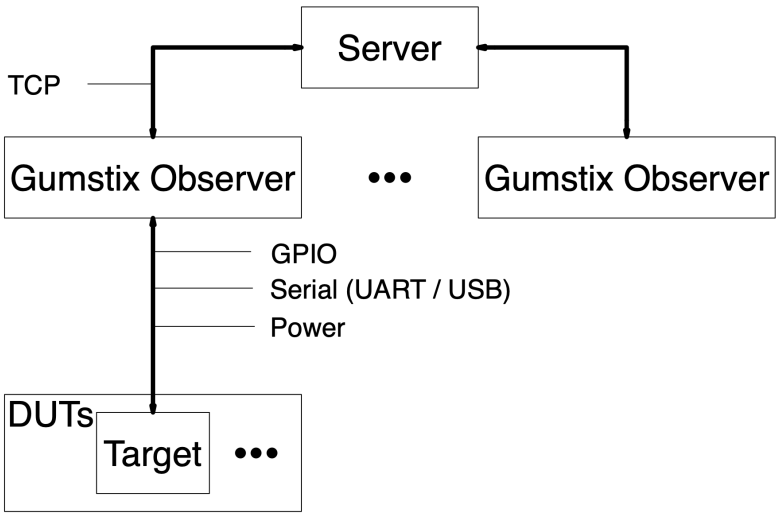

Figure 2.1: The 3-Tier-Architecture of the previous FlockLab [1].

ments and handle the entire test setup. All the observers are time synchronized such that the power measurement and GPIO actuation and tracing are comparable between the nodes. This time synchronization is implemented with a NTP Ethernet synchronization plus a Glossy [12] based wireless communication.
The results of the measurements and the serial output from the targets are both stored locally and also forwarded via a TCP connection to the server during the test. After the test the server collects all the test data and removes it from the observers.

2.1.3 Server

The server is the interface between the testbed and the users. It is required to store, schedule, and start all the tests, collect the data from the observers, and provide the results to the users.

2.2 Services

To perform the tests, the previous FlockLab provides multiple services [13]:

- GPIO tracing and GPIO actuation (Max rate 10 MHz)
- Power profiling and voltage control (Max rate 28 kSamples/s)
- Serial communication via UART

- target programming

These services are required to install, start and stop the tests and to read out and measure the test results. The GPIO tracing and actuation allows to set and get states of the DUTs for debugging purposes. The power profiling and voltage control is used to measure the power consumption of the targets which is a key feature of FlockLab. Serial communication is used to send the serial output from the targets to the Gumstix observer witch supports the user to debug its tested program. The target programming is required to reprogram the targets such that the users can individually program the targets.

2.3 Hardware

Each observer runs on a Gumstix Linux platform which is installed on the so called FlockBoard, a generic hardware interface between the observer and the targets. The FlockBoard can host up to four different targets at the same time. The key feature of the Gumstix computer are [5]:

- Gumstix verdex pro XL6P
- 600 MHz PXA270 processor
- 32 MB flash
- 128 MB RAM
- Attached 8GB microSD card for data storage

The key features of the FlockBoard are [1]:

- 4 generic hardware interface slots to attach any kind of wireless sensor node with a simple adapter board
- RTC with battery backup (keep date and time over power cycles)
- Voltage and current measurements on-board
- DAQ Board for accurate tracing (FPGA based) and Glossy Sync [12]

2.4 Software

The observers run on an openembedded Linux [14] OS with kernel version 2.6.33. For the scripts running on the observers is Python 2.7 [15] in use.

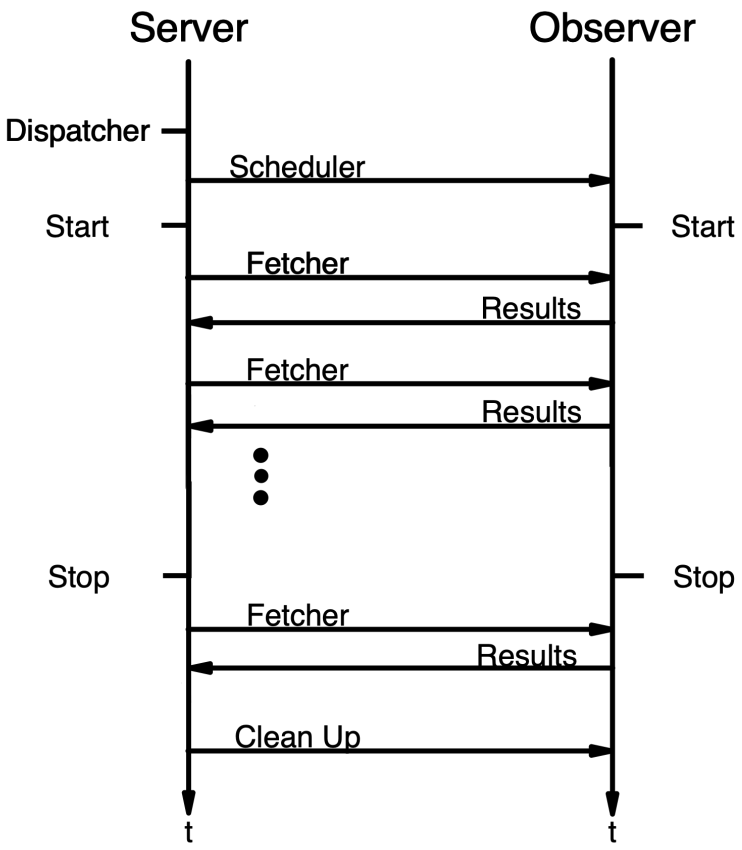

Figure 2.2: The interface protocol between server and Gumstix observer.

2.5 Interface

In Figure 2.2 the interface between the server and the Gumstix observer is displayed. The entire test is orchestrated from the server. Whenever a test is uploaded to the server via the webinterface the dispatcher analyses the XML on the server and generates a test in its database. To prepare the test on the Gumstix observer the server runs a scheduler which uploads the test and executes a start script on the Gumstix observer. During the test the fetcher collects all the data from the Gumstix observer that is ready to be collected. At the end of the test the fetcher collects all the remaining data and does a clean up on the Gumstix observer. The server does interpret the gathered data and provide the result files to the user.

2.6 Serial ID

Each target has a unique serial id which is stored in the FlockLab database on the server and is mapped to a specific type of target such as DPP or tmote. The server does not only have to know the Gumstix observers that are available but also which kind of targets are connected to those. Therefore the server checks from time to time the serial ids of the targets connected to the Gumstix observers. This is done via a W1 I2C connection. The request is simply done with the following command:

```
tg_serialid.py [--target=<int>] [--searchtime=<float>]
                [--maxretries=<int>]
```

where the three arguments, target, searchtime, and maxretries, are optional and are currently never used. The Python script which is called by running this command handles all the communication and can be found on the GitLab repository.

2.7 Test Setup

As it can be seen in Figure 2.3 the test is split up into three parts.

1. Test preparation: The image is programmed onto the target, takes 3 minutes for every test
2. Execution: The test is executed, duration depends on the test definition
3. Cleanup: The test data gets collected and the target is cleaned up, takes 3 minutes for every test

The communication between the server and the Gumstix observers is via TCP, which is a protocol with non-fixed latency. It is therefore possible that some Gumstix observers are communicating faster than others. This would cause different test start times at the different targets. To circumvent that, the time for the test preparation is fixed to 3 minutes, which is long enough to ensure that all the targets are correctly programmed. At the actual test starting time all Gumstix observers reset their target which ensures that all targets start at the same time.

2.8 Target Programming

The

```
target reprog
```

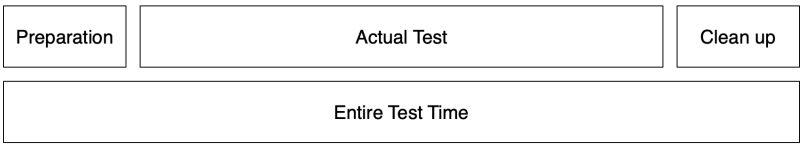

Figure 2.3: The three parts of a test.

is, as the name already implies, used to perform the reprogramming of the targets. The following command is called from the server once during the test preparation part and a second time at the end of the cleanup.

```
tg_rerprog.py --image=<path> --target=<string> [--port=<string>]
[--core=<int>] [--noreset]
```

At the end of the test the programmed image is a dummy image that does nothing but blink the LED to ensure that the node does not interfere with a later test. For this reprogramming the noreset flag is not set. The reset will be pulled at the actual start time of the test as it is explained in the section Test Setup.
During the test preparation the test image is programmed onto the target but the test does not start yet. Therefore the noreset flag is set to keep the target in the non-active mode. Each test has a start and a stop time which determine the time window of the actual test. All selected targets but maximal one per observer are reset at the beginning of the actual test to ensure that the program does not start in advance.

Chapter 3

Implementation of FlockLab 2.0

We have explained in the previous chapter that the observer hardware for the FlockLab has to be updated. The aim of this semester thesis is therefore to port the services running on the Gumstix observers on an openembedded Linux to a new platform. The goal is to have more free capacity on the new platform such that it will be possible to upgrade the current services and add new services in the future. In this chapter we are going to describe the new implementation and the design decisions that have been made.

3.1 Observer Platform and OS

The decision of the new observer platform was made beforehand by the team of Jan Beutel. They have chosen the Beaglebone Green [6] for the following reasons.

1. There exists a cape for the Beaglebone Green to do very accurate and fine grained power measurements, the so called RocketLogger [8]. Because this power measurement is already implemented and perfectly fits the requirements of high precision and high frequency measurements the team has decided to build up on this instead of developing a new system.
2. In the Debian image for the Beaglebone Green is the BeagleLogic tool implemented which allows to do fine grained GPIO tracing of up to 100 MHz.
3. The Beaglebone platform is widely used and therefore there exist a huge community which makes the development easier.

The Beaglebone platform supports multiple different Linux distributions. The decision has fallen to use Debian because both RocketLogger and BeagleLogic

is implemented on the Debian. For the final version the goal is to use Debian 10, but for this thesis we have used the latest stable version of Debian 9 because there was no stable version of Debian 10 at the beginning of my thesis.

3.2 Script Language

On the observer are multiple Python scripts running to enable the FlockLab services. On the previous FlockLab implementation is Python 2.7 used. The latest Python 2 version is Python 2.7.17 which was published in March 2019. In the middle of 2020 with Python 2.7.18 there will be the last Python 2 version [15] and there will be no support for Python 2 any more after this release. Since for the switch to a new platform all the scripts have to be looked at and slightly modified we have decided to update them also to Python 3 instead of keeping up an outdated version.
To configure the GPIO pins a Python library called Adafruit_BBIO.GPIO is used. This library still requires Python 2.7. For this reason the GPIO interactions are still implemented in Python 2.7.

3.3 Setup

To setup an observer for FlockLab 2.0 there is a shell script which can be executed in one single step and prepares a Beaglebone such that it fully works with the FlockLab. For the following setup there are some prerequisites that have to be met for the Beaglebone observer

- it runs Debian 9 (in the future Debian 10)
- it is reachable via SSH within the ETH network
- it accepts a public private key pair connection
- it allows to login with the root user via SSH

3.3.1 Setup Script

The shell script `setup_new_beaglebone_observer.sh` as it can be seen in Figure C.1 is located on the GitLab repository of FlockLab 2.0 in the observer folder. The execution of this file takes a couple of minutes and afterwards the Beaglebone is a perfectly working observer. The only thing that is left to do is to add the database entries on the FlockLab server such that the FlockLab server recognizes the observer. The exact instructions to use the setup script is described in Appendix B

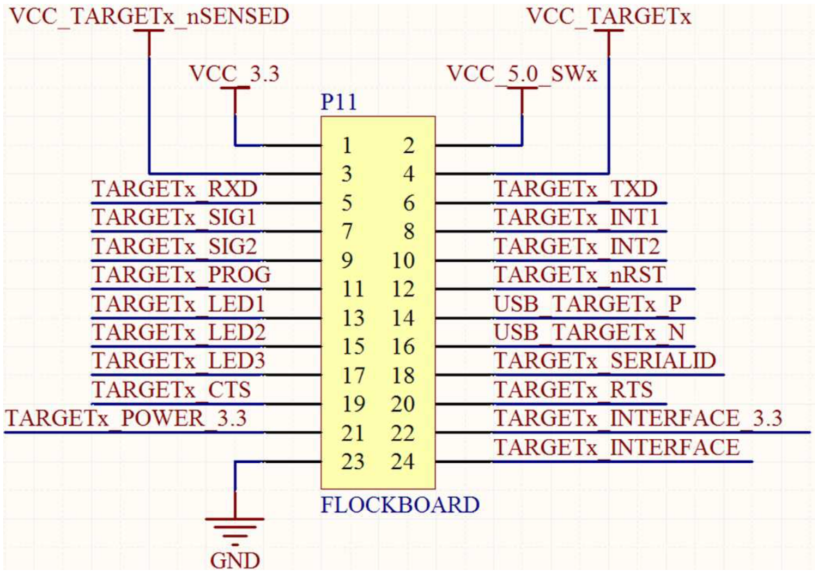

Figure 3.1: The pins of the target - observer interface.

3.4 Pin Mapping

In Figure 3.1 the pins of the interface between the observer and the targets are shown and in the Table B.1 the mapping to the Beaglebone observer is shown. Whereas in the mapping between the observer and the target the UART and I2C connections are fixed the GPIO pins can also be changed. The GPIO pins on the observer have been chosen because these pins are by default usable for GPIO connections without any further configuration. In the code the mapping is done by a mapper function in the FlockLab.py library file.

3.5 FlockLab Services

FlockLab as a testbed environment provides multiple different services to perform the tests. These services are crucial to get test results and therefore they build the core of FlockLab. Currently there are 5 different types of services implemented which are:

1. Serial logging
2. Serial forwarding
3. GPIO tracing
4. GPIO actuation
5. Power profiling

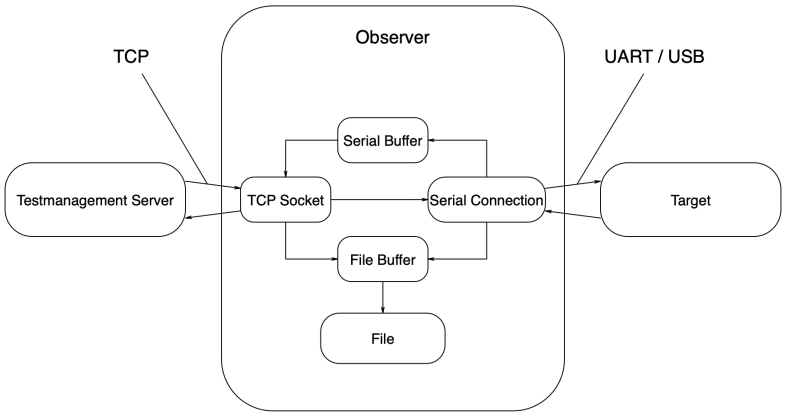

Figure 3.2: Handling of serial output data from the targets.

3.5.1 Serial Logging and Serial Forwarding

Serial logging and serial forwarding are two different services that implementation is combined together because they are closely related. Whereas the serial logging logs all the messages from the target and the server the forwarding part transmits all the messages from the target to the server. In Figure 3.2 the two services are depicted. On the right hand side there is a UART or USB connection to the target which allows to read the serial output from the targets as well as sending back some data. The serial output from the target will be stored in two different buffers, a serial buffer and a file buffer. Whereas the serial buffer has only one input line and forwards all the data directly to the TCP connection with the server, the file buffer has an additional second input line with all the data sent by the server to the target. The file buffer then writes the combined data from the server and the target to a file which will be sent to the server at the end of a test such that the log contains RX and TX messages of UART.

This is basically still the same as it was in the pervious FlockLab. There are some minor changes in the code regarding the compatibility of Python 3 and the Debian OS running on the observer. Some functions like the voltage setting or the target selection return a hard coded value because for this thesis the target is directly connected to the observer without the FlockBoard in between. All those functions are in the `flocklab.py` library file and are marked with a "TODO" and a brief comment.
To enable the UART serial connection a device tree overlay on the observer is required which enables the pins to be used for UART. Debian 9 already comes with the required overlay files which have to be enabled in the `/boot/uEnv.txt` file. Therefore the following two lines have to be in the `/boot/uEnv.txt` file.

- `enable_uboot_cape_universal=1`

- `uboot_overlay_addr1=/lib/firmware/BB-UART2-00A0.dtbo`

The first line is to enable all kinds of overlays and the second one is a specific overlay to enable UART2 on the pins P9_21 (TXD) and P9_22 (RXD) on the Beaglebone. I have chosen UART2 because the pins for UART1 are by default already used. The `BB-UART2-00A0.dtbo` file does not only enable UART for the pins P9_21 and P9_22 but also maps it to the virtual console `/dev/ttyO2` where the data now can be written to and read from.
A complete uEnv.txt file including all the other changes I am going to describe below can be found in the GitLab repository for FlockLab 2.0 in the directory `/observer/config/`.

3.5.2 GPIO Tracing and GPIO Actuation

For the FlockLab 2.0, a totally new GPIO tracing and actuation will be implemented to enable finer grained tracing and actuation which is a demand in future WSNs because of more powerful low power devices. This new GPIO tracing is targeted to be based on the BeagleLogic [16] project. This decision has been made because of the following three points.

1. It enables a tracing of the pins with frequencies of up to 100 MHz, witch is 10 times more than the minimal defined requirements for FlockLab 2.0.
2. The tracing will be done with the PRU unit and therefore almost no processor capacity is required for the tracing witch leads to a lot of free capacity on the CPU.
3. The BeagleLogic suite is implemented by default into the Debian image for the Beaglebone green since the Kernel version 3.8 and therefore there is no setup required.
4. With one simple command (`sigrok-cli -d beaglelogic:logic_channels=8 -c samplerate=10M --samples 10M -o capture.sr`) up to 8 pins (`P8_39` to `P8_46`) can be continuously captured

Although those points are very promising there is one big drawback. The Kernel version we have chosen for this project at the beginning of the thesis is version 4.14. Unfortunately, the BeagleLogic tool does currently not run on this Kernel version and therefore we decided to postpone the GPIO tracing implementation with the BeagleLogic tool.
For the sake of completeness we have implemented a simple GPIO tracing based on the Adafruit.GPIO library. This implementation does not support high frequency tracing and is running on the CPU and furthermore it requires Python 2.7. The tracing is started together with the power profiling from the flocklab_scheduler.py script.

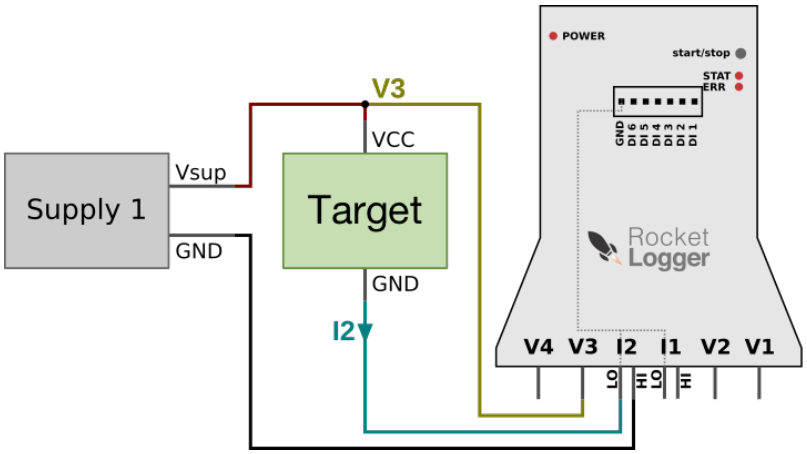

Figure 3.3: The recommended measurement setup for a single target.

3.5.3 Power Profiling

As described before in the previous FlockLab the power profiling does only measure the current but not the voltage. The voltage is estimated by using the supply voltage set by the observer as a constant value. Although this is theoretically true there will always be some small deviations to the set supply voltage. These deviations introduce errors to the power measurement which can be avoided by measuring not only the current but also the voltage.
In the new FlockLab 2.0 the power profiling is implemented in a completely new way with a much higher profiling frequency and also an additional profiling of the voltage. Both newly introduced features allow to measure with a higher accuracy and therefore it provides a high benefit to bring FlockLab to the next level.
The new power profiling is done with the external RocketLogger [8], which is a cape for the Beaglebone Green. At the moment an additional second Beaglebone is required to do the measurement. This is because the RocketLogger only works on Debian 7 but for FlockLab 2.0 Debian 9 (in the future Debian 10) is required, due to PTP time synchronization support. In the future, the goal is to implement the RocketLogger cape direrctly into the observer such that there is no additional hardware required any more. To do this the RocketLogger has to be updated which is out of the scope of this semester thesis and remains to be done in the future.
To start the external power profiling additional helper functions located in the FlockLab.py library, which we are going to explain later in this section, are required. In the future this will become even easier when the RocketLogger cape will be on the same Beaglebone and these helper functions can simply be replaced with a single start and stop command.
In Figure 3.3 the measurement setup is depicted. This is the setup as it is recom-

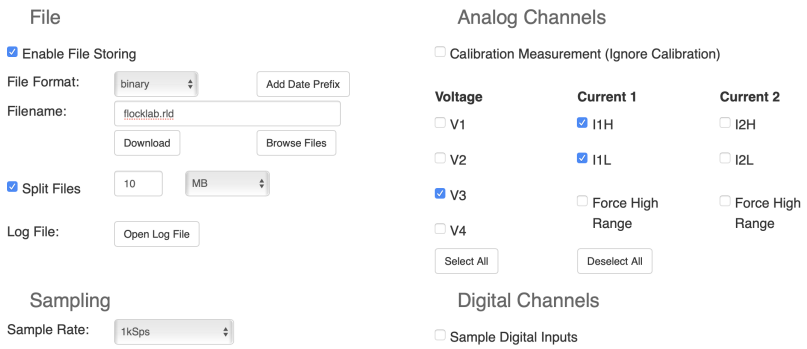

Figure 3.4: The measurement settings of the RocketLogger.

mended on the RocketLogger wiki [8] when there is only one target which power consumption has to be measured. The RocketLogger is connected in series for the current measurement and in parallel for the voltage measurement. The target is the device under test which performs the FlockLab tests. Currently the observer itself acts as a supply but in the future this will be the FlockBoard.
In Figure 3.4 the settings currently used for the power measurement is depicted. As it can be seen the sampling frequency is currently set to 1 kHz but it is possible to raise it up to 64 kHz. For the file format the binary file is chosen because it allows a better performance regarding the file size and data transfer. This settings can also be set via CLI commands, either as pre-settings or in form of arguments at the beginning of each measurement. The Interpretation of the binary data is not implemented so far and to be done on the sever where enough capacity is available. Furthermore there already exists a Python library to do this which is also part of the RocketLogger project. It remains to add this parsing to the server such that the results provided to the users are human readable.

Helper Functions

As I have mentioned above, for the power profiling there are three helper functions required to perform the measurement on the external RocketLogger.

1. `start_pwr_measurement()`
2. `stop_pwr_measurement()`
3. `collect_pwr_measuremen_data(test_id)`

The communication between the observer Beaglebone and the RocketLoggerr is done via a text file located on the observer Beaglebone. The start function simply writes ”start” into this file which is located at /home/debian/FlockLab/pwr_measurement.txt.

```
def start_pwr_measurement():
    file = open("/home/debian/flocklab/pwr_measurement.txt","w")
    file.write("start")
    file.close()
    syslog(LOG_INFO, "Started power measurement.")
```

Figure 3.5: Code of the helper function start_pwr_measurement().

```
def stop_pwr_measurement():
    file = open("/home/debian/flocklab/pwr_measurement.txt","w")
    file.write("stop")
    file.close()
    syslog(LOG_INFO, "Stopped power measurement.")
```

Figure 3.6: Code of the helper function stop_pwr_measurement().

The corresponding code is depicted in Figure 3.5. The stop function is almost the same except for the code word which is "stop" as it can be seen in Figure 3.6.

The third helper function, collect_pwr_measuremen_data(test_id), is depicted in Figure 3.7 and is called right after the stop function. It concatenates all the measurement files together to one big file renames it, adds a time stamp, and copies it into the directory where the server expects the file. It can also be called during a running test such that the data is sent in chunks to the observer. The fetcher running on the server does also collect the data during a running test which reduces the required data collection time at the end of a test.

Python Script on RocketLogger

On the RocketLogger runs a Python script all the time and looks on the observer whether there has been written start or stop into the specified file or not and

```
def collect_pwr_measurement_data(test_id):
    data_file_path = "/home/debian/flocklab/pwr/"

    read_files = glob.glob("%s*.rld" % data_file_path)
    while len(read_files) == 0:
        read_files = glob.glob("%s*.rld" % data_file_path)

    my_time = time.strftime("%Y%m%d%H%M%S", time.gmtime())
    with open("%spowerprofiling_%s.db" % (data_file_path, my_time), "wb") as outfile:
        for f in read_files:
            syslog(LOG_INFO, str(f))
            with open(f, "rb") as infile:
                outfile.write(infile.read())
                infile.close()
            os.remove(str(f))

    try:
        data_files = [f for f in listdir(data_file_path) if isfile(join(data_file_path, f))]
        for df in data_files:
            src = "%s%s" % (data_file_path, df)
            dst = "/home/debian/flocklab/db/%d/%s" %(int(test_id), df)
            copyfile(src, dst)
            os.remove(src)
    except (Exception) as e:
        syslog(LOG_INFO, traceback.format_exc())
```

Figure 3.7: Code of the helper function collect_pwr_measuremen_data(test_id).

accordingly to this signal starts or stops the power measurement. Furthermore it also checks from time to time if there are complete measurement files and if so copies them to the observer. This is done during the entire measurement such that in the end of a test not all the data has to be transferred at once which could take a lot of time but only the last part has to be transferred.
This solution introduces a lot of overhead for such a simple task as just start and stop a measurement but unfortunately it can not be done in a simpler way because of two reasons.

1. The RocketLoggerr cape blocks all the GPIO pins of the Beaglebone beneath it and therefore a simple single GPIO connection with 1 for running a test and 0 for stopping it can not easily be realized.
2. The command to start and stop the measurement responds with a blocking response that requires further inputs which introduces multiple hacks to circumvent this via a SSH connection.

For those two reasons we have decided to implement it in the way described above. The only drawback of the current solution is the overhead but because the RocketLogger will be included into the observer in a final version of FlockLab 2.0 it seemed to be the cleanest way because the required adaptions that have to be made are kept small.

3.6 Target Interactions

In the previous sections the interaction between the server and the observer and the services running on the observer are described. In this section we are going to explain the interactions between the observer and the target such that a test image can be installed on the target and the test can be executed. To enable this multiple script files on the observer are required which are called by the server.

3.6.1 Serial ID

The Serial ID request in FlockLab 2.0 is basically still the same as it was in the previous FlockLab except for some minor changes of the location where the W1 interface can be controlled in Debian 9 compared to openembedded Linux. This pin used for the W1 connection is the P8_11 pin on the observer. To enable it a device tree overlay is required.

Required Device Tree Overlays

To enable the W1 I2C interface a device tree overlay is required because the pin is by default only a simple GPIO pin without a serial bus. The required file,

`BB-W1-P8.11-00A0.dtbo`, is copied with the setup file to the `/lib/firmware directory`. To generate this `.dtbo` file a human readable `.dts` file is required which can also be found in the GitLab repository under `/observer/DTO/BB-W1-P8.11.dts` and figures as a template for future I2C connections if there are required additional ones or the current one should be used with a different pin. To generate the `.dtbo` file the following steps have to be done:

1. clone the repository from `https://github.com/beagleboard/bb.org-overlays.git`
2. put the `.dts` file into the `./src/arm` directory
3. execute the make file

3.6.2 Target Programming

The logic of the target reprogramming is still the same as it was in the previous FlockLab. Only the communication port for UART and the GPIO communication had to be updated. As described in Subsection *Serial logging and Serial Forwarding* and Subsection *GPIO tracing and GPIO actuation* we have used the `/dev/ttyO2` port for UART and use the Adafruit.GPIO Python library for GPIO.

3.7 Changes on FlockLab Server

To keep it simple it was targeted to reuse the interface between the server and the observer the same. In FlockLab 2.0 interface itself as it is depicted in Figure 2.2 is still the same, but the gathered data structure has changed because of changes of the power profiling and the GPIO tracing. As mentioned in Chapter *Previous FlockLab* this interpretation is done on the server side in `flocklab_fetcher.py`. We have added the two worker functions for the power profiling and GPIO tracing which now simply copy all the results together to one file and provide this to the user. It remains to the future to define the data format and to implement this into those two worker functions.

Chapter 4

Measurements

One of the reasons to replace the Gumstix observers of the previous FlockLab was that this platform is outdated and almost no free CPU capacity was left. Therefore a further development was not possible any more. For the observers in FlockLab 2.0 the goal is to have around 50 % free capacity left.

4.1 Measurement Setup

To perform the measurement additional tools are required. We have used the `iostat` Linux tool to do the CPU load measurement. This tool has a slight influence on the test results, because it will also require some CPU capacity. The `top` tool shows that its influence on the CPU load is between 1 and 5 percent. We therefore consider 5 percent inaccuracy as an upper bound for the error in our measurements.
During the measurements we have realized that the TCP connection from the external RocketLogger generates a huge load of up to 60 percent. This is because the RocketLogger constantly checks whether the power profiling has to start or stop. In the final FlockLab 2.0 the RocketLogger will be integrated into the observer and therefore this load will no longer exist. We have therefore performed the measurements with the power tracing turned off. A future integration of the RocketLogger into the observer is expected not to affect the CPU load significantly, because the RocketLogger uses the PRUs instead of the CPU to do the power tracing. According to a measurement with `top` on the RocketLogger its CPU is idle for 97 to 99 percent and the power measurement does require 0.3 to 1.0 percent of the CPU during a running test.
In the final FlockLab 2.0 neither the GPIO tracing nor the power profiling will have a large influence on the CPU load. We have therefore performed three different type of tests that affect the setup time, the clean up time and the CPU load during a test.

Test Duration [min]	Saved Time [%]
1	28.6
5	18.2
10	12.5
60	3.03

Table 4.1: Examples of different test durations and the corresponding saved time in percent if the test setup and cleanup time can be reduced from 3 to 2 minutes each.

- We have used different test scripts for the targets which affects the target programming time
- We have sent serial messages via the TCP socket at different rates
- We have sent serial messages via the TCP socket of different length

4.2 Measurement Results

4.2.1 Different Test Scripts

The goal of this measurement is to find an upper bound of the setup and cleanup time of the test. We have therefore used small (`flocklab-hello-world_dpp.xml`) and a large (`lwb-test.xml`) installation script that takes longer for the setup. In FlockLab different targets can be used which also have different programming times. To find the upper bound we have done the measurement with the DPP target which has three cores and therefore take up to three times longer since the cores have to be programmed sequentially.
We have performed in total 43 different test runs with the `flocklab-hello-world_dpp.xml` script. In this tests the setup time varied between 62 and 68 seconds with an average of 64.4 seconds and a variance of 2.09. In the five test runs with the `lwb-test.xml` script the test preparation time was up to 80 seconds.
In the previous FlockLab the setup time is fixed to 3 minutes. The results from our measurements show that it is possible to reduce this to maximum 2 minutes. The cleanup time did not vary at all between the different test scripts. It took the server constantly 72 seconds to fetch all the data from the observer and then an additional second to finish the test and prepare the data for the user.
During the setup and cleanup times FlockLab is busy without running a test. In Table 4.1 we have four examples how much the reduction from 3 to 2 minutes of both, the setup and cleanup time, can affect the total test time in percent. One can see that with FlockLab 2.0 up to 30 percent more tests could be performed compared to the previous FlockLab.

Waiting Time [ms]	f [Hz]	Avg Test Load [%]	Avg Total Load [%]
-	0	9.31	13.35
1000	0.97	8.34	35.93
100	9.74	8.76	35.58
80	11.82	15.43	36.34
70	13.61	19.72	37.17
60	15.76	21.15	37.37
50	18.91	18.41	36.93
40	22.83	24.30	37.65
30	29.52	27.56	38.11
20	41.57	27.52	37.60
15	57.46	39.44	39.87
10	87.01	53.51	41.10
5	162.61	78.53	44.17
0	881.40	95.70	46.17

Table 4.2: CPU load measurements with different serial message rates and there corresponding results.

4.2.2 Serial Messages at Different Rates

In Table 4.2 the different measurement setups and their respective results are listed. The size of the message is 4 characters.

- The waiting time is the time between sending consecutive messages.
- The frequency describes the actual average frequency at which messages were sent based on the timestamps from the log file.
- The avg test load is the average load of the observers CPU between test start and test stop.
- The avg total load is the average load during the entire test including the setup and cleanup.

At a message rate of up to 10 messages per second there is no influence on the CPU load. Afterwards it slowly starts to kick in. At a rate of around 20 messages per second the CPU load is doubled up to roughly 20 percent. At a rate of almost 90 messages per second the 50 percent mark is reached.
These tests show that it is possible to send data with a rate of up to 90 messages per second while the CPU load is still around 50 percent.

Waiting Time [ms]	f [Hz]	Avg Test Load [%]	Avg Total Load [%]
140	6.93	12.28	36.16
120	8.03	14.44	36.25
100	9.74	16.39	36.46
80	11.77	19.14	37.15
60	15.72	19.94	36.85
40	22.48	22.34	37.27
20	43.52	29.28	38.09
10	88.75	54.45	41.10
5	43.52	29.28	38.09

Table 4.3: CPU load measurements with different serial message rates and a message size of 100.

4.2.3 Serial Messages of Different Length

For this test setup we have sent at a message with a length of 100 characters at different frequencies. This is 25 times longer than the message size in the previous test series. As expected the CPU load is more affected. It starts to kick at a frequency around 7 Hz instead of 10 Hz. But even these long messages can be transmitted at high frequencies. In fact the higher the frequency the closer are the results of the CPU load between the two message sizes. The full list of results can be seen in Table 4.3.

4.3 Conclusion

The test setup and cleanup time can be reduced to each 2 minutes such that the test time can be reduced by almost 30 percent. This allows to perform an significantly increased number of tests with FlockLab 2.0 compared to the previous FlockLab.
The serial logging can handle up to 90 messages per second and the CPU load is still below 50 percent during the test which allows to add more services to FlockLab 2.0.

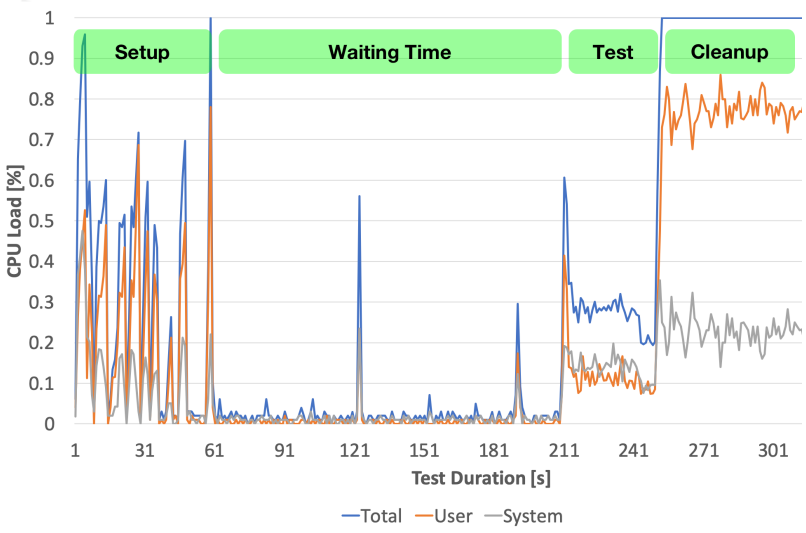

Figure 4.1: The CPU usage measures vs. the test duration measured with the iostat tool by user, by system and in total.

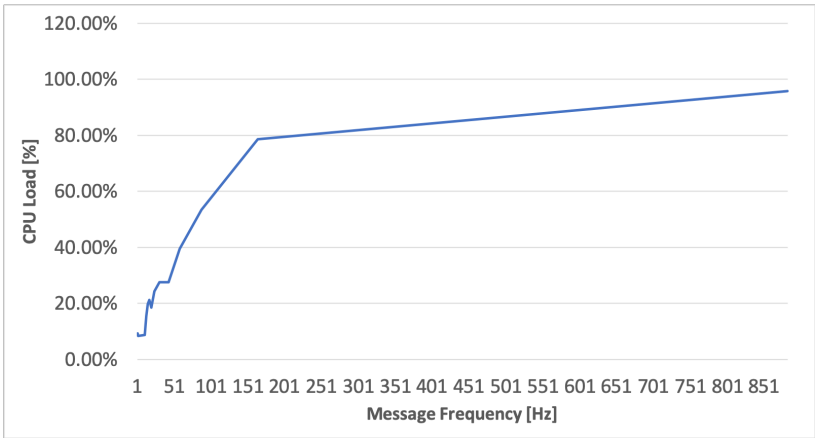

Figure 4.2: The CPU usage measures vs. the frequency of serial messages with 4 characters per message.

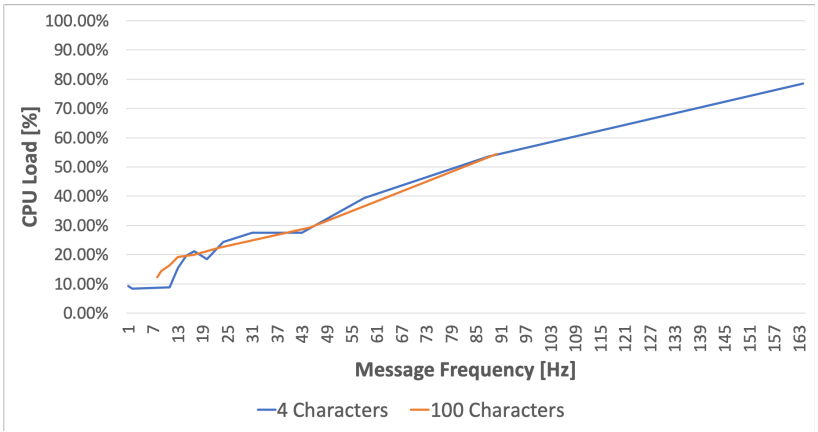

Figure 4.3: Comparison between the CPU usage measures vs. the frequency of serial messages with 4 characters and 100 characters per message.

Chapter 5

Future Work

The observer developed within this thesis is running on a Beaglebone green with Debian 9 and using an external RocketLogger for the power measurement and a coarse grained GPIO tracing. In the end the OS should be Debian 10, it is targeted to integrate the external RocketLogger in the observer and do the GPIO tracing BeagleLogic.

5.1 Debian 10

The first stable version of Debian 10 will probably be available in 2019 [7] but a fixed release date is not set yet. So the update from Debian 9 to Debian 10 can be done soon and it should not be a lot of work since there will not be major differences between those two versions.

5.2 RocketLogger

The RocketLogger was written for Debian 7, the last Debian version that used the SysVinit system [17] to start the essential boot processes. With Debian 8 the systemd daemon [18] was introduced. This causes major changes in the Kernel and it takes a great effort to port applications to the newer Debian versions. This will definitely be the heaviest part for the further development to make the RocketLogger ready for Debian 10.

5.3 Processing of Power Profiling Files

The files from the RocketLogger are not processed yet. On the server side these files have to be processed in a way that allows the user to read and interpret

them. The code to do this is part of the RocketLogger project and publicly available [1].

5.4 BeagleLogic

Since BeagleLogic is already integrated in the image of the Beaglebone Debian the switch to use it instead of the currently used coarse grained GPIO tracing will not take that big of an effort. The only drawback is that it does not run on the newest stable Kernel version and therefore this development depends on external developments.

[1] https://gitlab.ethz.ch/tec/public/rocketlogger/

Chapter 6

Conclusion

The goal of this thesis was to port the services running on the Gumstix platform to a new observer based on the Beaglebone Green. Furthermore the power measurement has to be done with the RocketLogger to enable high frequency power profiling.
With this thesis we have developed this new observer and shown that it possible to use the Beaglebone Green as an observer by adapting and writing all the therefore required code and a simple setup script that prepares a Beaglebone. Furthermore, we have shown that there is still a lot of free capacity available to add new services or further improve existing ones. Additionally the setup and cleanup time can be reduced by at least one third such that in FlockLab 2.0 up to 30 percent more tests can take place.

Bibliography

[1] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems," in *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*, ser. IPSN '13. New York, NY, USA: ACM, 2013, pp. 153–166. [Online]. Available: http://doi.acm.org/10.1145/2461381.2461402

[2] J. L. Hill and D. E. Culler, "Mica: a wireless platform for deeply embedded networks," *IEEE Micro*, vol. 22, no. 6, pp. 12–24, Nov 2002.

[3] *STM32L4 Series*, 2019. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32l4-series.html

[4] *Cortex-M4 Technical Reference Manual*, 2010. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439b/DDI0439B_cortex_m4_r0p0_trm.pdf

[5] *Verdex Pro Series*, 2019. [Online]. Available: https://www.gumstix.com/support/hardware/verdex-pro/

[6] *SeeedStudio BeagleBone Green*, 2019. [Online]. Available: https://beagleboard.org/green/

[7] *Debian Releases*, 2019. [Online]. Available: https://www.debian.org/releases/

[8] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele, "Measurement and validation of energy harvesting iot devices," in *Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, Lausanne, Switzerland, Mar 2017.

[9] G. Werner-Allen, P. Swieskowski, and M. Welsh, "Motelab: a wireless sensor network testbed," in *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, April 2005, pp. 483–488.

[10] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noël, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "Fit iot-lab: A large scale open experimental iot testbed," 12 2015.

[11] P. Appavoo, E. K. William, M. C. Chan, and M. Mohammad, "Indriya2: A heterogeneous wireless sensor network (wsn) testbed," in *Testbeds and Research Infrastructures for the Development of Networks and Communities*, H. Gao, Y. Yin, X. Yang, and H. Miao, Eds. Cham: Springer International Publishing, 2019, pp. 3–19.

[12] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," 05 2011, pp. 73 – 84.

[13] *FlockLab Services Specification*, 2019. [Online]. Available: https://gitlab.ethz.ch/tec/public/flocklab/wikis/Spec/Specifications

[14] *Welcome to OpenEmbeddedn*, 2017. [Online]. Available: http://www.openembedded.org/wiki/Main_Page

[15] *PEP 373 – Python 2.7 Release Schedule*, 2019. [Online]. Available: https://www.python.org/dev/peps/pep-0373/

[16] *Welcome to BeagleLogic!*, 2019. [Online]. Available: https://beaglelogic.readthedocs.io/en/latest/

[17] *System V style init programs - Summary*, 2018. [Online]. Available: http://savannah.nongnu.org/projects/sysvinit

[18] *systemd System and Service Manager*, 2019. [Online]. Available: https://freedesktop.org/wiki/Software/systemd/

Appendices

Appendix A

GitLab Directory Structure

In Apendix A the GitLab repository as a directory tree is listed. All the files we have presented in this thesis are listed here. The structure is still the same as it was in the previous FlockLab. The two additional folders `/workspace/observer/DTO` and `/workspace/observer/config` have been placed accordingly to the already existing structure.

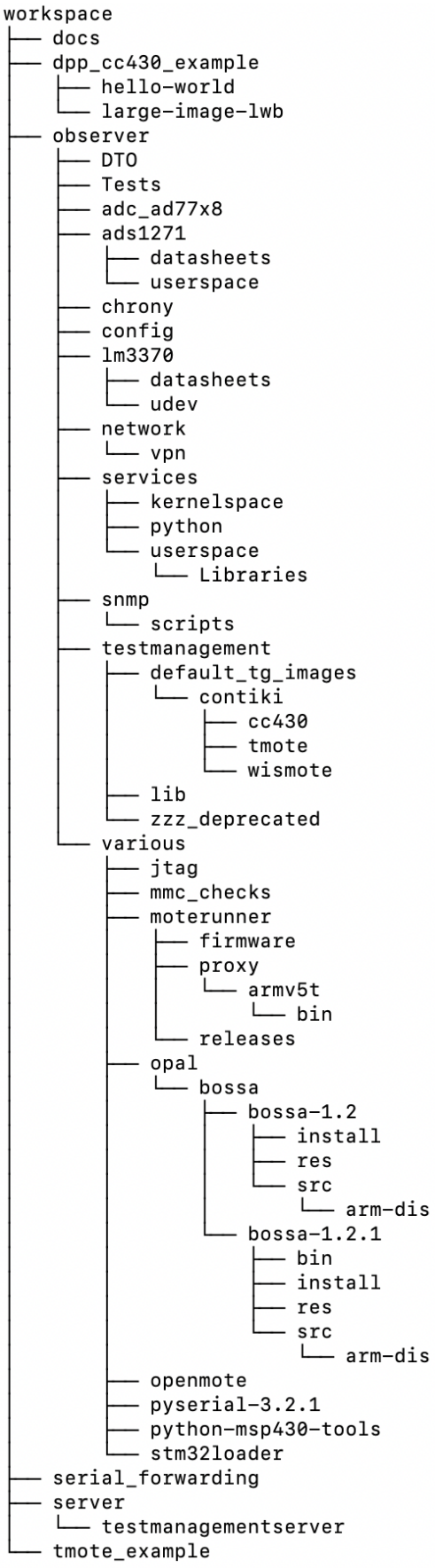

Figure A.1: The directory tree of the GitLab repository.

Appendix B

How To

The following steps have to be done to do the setup:

1. Download the observer folder from git lab to the local computer
2. Prepare in the .ssh folder a key file called FlockLab_dev
3. Set the variables "DATAPATH" and "KEYFILE" at the beginning of the script file accordingly to your needs
4. execute the shell script with "sh setup_new_beaglebone_observer.sh <host-name>" where <host-name>is the host name of the Beaglebone running Debian 9 that has to be set up

target Name	**observer Name**	**observer Pin**
RXD	UATR2_TXD	P9_21
TXD	UART2_RXD	P9_22
GND	GROUND	P9_1
GND	GROUND	P9_2
VCC 3.3	3V3	P9_3
GPS	GPIO_60	P9_12
GND	GROUND	P8_2
nRST	GPIO_69	P8_9
SERIALID	GPIO_	P8_11
PROG	GPIO_88	P8_28
INT1	GPIO_78	P8_37
INT2	GPIO_79	P8_38
RTS	GPIO_76	P8_39
CTS	GPIO_77	P8_40
LED3	GPIO_74	P8_41
POWER_3_3	GPIO_75	P8_42
INTERFACE_3_3	GPIO_72	P8_43
INTERFACE	GPIO_73	P8_44
SIG1	GPIO_70	P8_45
SIG2	GPIO_71	P8_46

Table B.1: Pin mapping between observer and the target.

Appendix C

Time Schedule

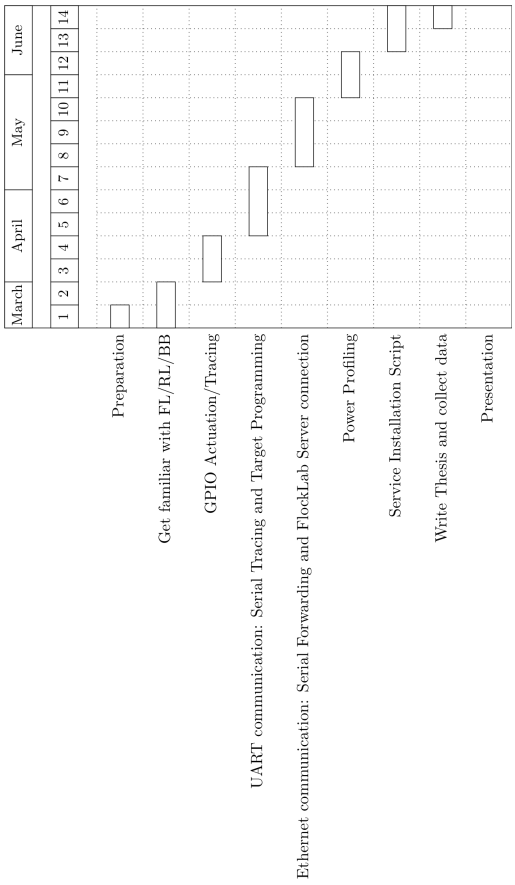

Figure C.1: The Time Schedule of this Semester Thesis.

Appendix D

Original Project Assignment

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Semester Thesis at the
Department of Information Technology and
Electrical Engineering

for

Dario Leuchtmann

FlockLab 2.0: Linux Platform

Advisors: Jan Beutel
Roman Trüb
Reto Da Forno
Tonio Gsell

Professor: Prof. Dr. Lothar Thiele

Handout Date: 11. 03. 2019
Official Start Date: 18. 03. 2019
Due Date: 24. 06. 2019

Initial Presentation (tentative): 04. 04. 2019
Final Presentation (tentative): TBD

1 Project Description

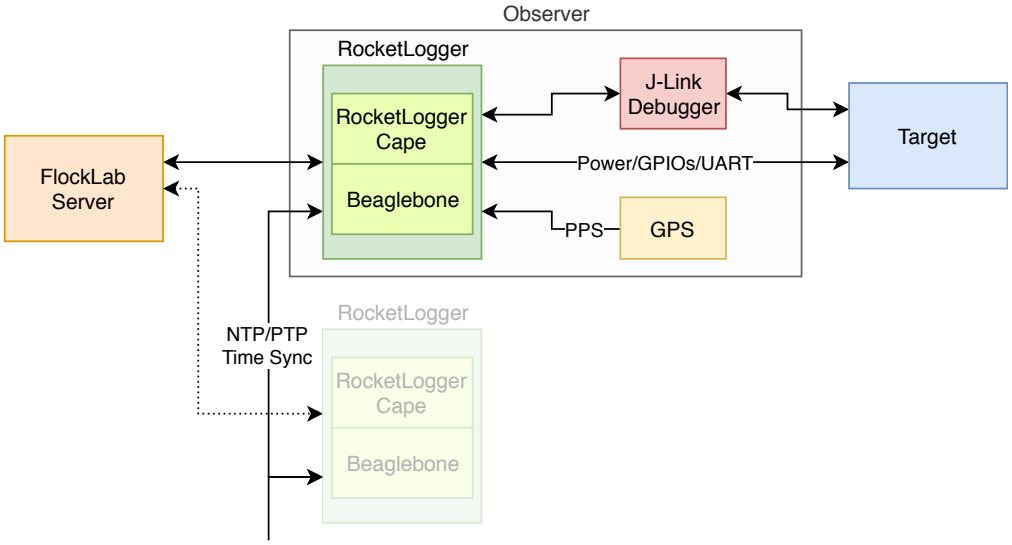

Figure 1: FlockLab 2.0 Architecture.

Since 2012, the Computer Engineering Group (TEC) operates the FlockLab testbed [1] for developing and evaluating wireless sensor network protocols. A testbed helps to reduce the effort of repeatedly deploying test networks when developing protocols for wireless sensor networks. Furthermore, such a testbed improves the reproducibility of experiments and allows to share infrastructure.

Currently, we are in the process of extending the existing short baseline distances in FlockLab by adding additional nodes on rooftop locations and with significantly larger spacing. The existing implementation of FlockLab is based on hardware components which are no longer in production and consists of a few patches which makes replication of the existing design impractical. Furthermore, there are new trends from industry, such as extended debugging and measurement capabilities, for which there is only limited support by the existing FlockLab platform.

Therefore, we target to implement the next generation of the FlockLab testbed, FlockLab 2.0, by replacing the existing FlockLab observer platform with a new one (see Figure 1). Among other improvements, we plan to incorporate a Linux platform with more performance, more precise power tracing based on the RocketLogger [2], as well as state-of-the-art debugging support.

The overall project is divided into the following four subprojects:

WP1: Linux Platform

The main focus of this subproject is the replacement of the Linux platform. The goal is to reuse the BeagleBone Linux platform from the RocketLogger [2]. The Linux Platform needs to implement and support all services of the existing FlockLab and to

interface with the existing FlockLab backend server. The services consist of starting and stopping tests, programming of the targets, generating GPIO events, collection and buffering of trace data (GPIO and power), UART interaction logging, forwarding of serial communication data streams, and transferring the collected data to the FlockLab server. An optional part is updating the PRU related implementation of the RocketLogger to work with the Debian 9 release upgrade (required by the PTP time sychronization of WP3).

WP2: Target Interconnection/Debugging

The goal of the second subproject is defining the connection of the targets, devices under test (DUTs), to the Linux observer platform. Different aspects of the interconnection shall be investigated: Power delivery to the target, options for programming the targets, serial communication over UART, and the debugging with the J-Link debugger. The different opportunities of the various debugging features (RTT, VCOM, Flash breakpoints, monitor mode debugging) shall be investigated. An optional task is to explore the simultaneous debugging of targets on multiple observers.

WP3: Time Synchronization of Observers

Since the testbed is a system of distributed observers which are used to develop and debug a network wide wireless protocol, the time synchronization of the obtained measurement data is very important. The FlockLab 2.0 is supposed to support large baseline distances which makes the use of the existing custom synchronization based on short-range communication infeasible. The goal of this subproject is to investigate two alternative time synchronization options which have the potential to provide sub-microsecond accuracy: A GPS based Pulse-per-second (PPS) signal and the Precision Time Protocol (PTP) over Ethernet. Interesting questions are, how the two solutions can be integrated with the Linux platform and what accuracy they can provide in the given setup.

WP4: Hardware Design

This fourth subproject takes care of defining the interfaces between the observer and the target nodes in collaboration of WP2. Furthermore the options of placement / interconnection of the various components/units is explored. An important aspect is the proper isolation of the target from the debugger in order to achieve precise power measurements. The calibration of the modified RocketLogger unit is part of this subproject, too.

This semester thesis will mainly focus on *WP1*.

2 Project Tasks

WP1 Linux Platform

- Formulate a time schedule and milestones for the project. Discuss and approve this time schedule with your supervisors.
- Get familiar with the observer implementation and interfaces of the existing FlockLab platform.
- Get familiar with the RocketLogger platform.
- Generate a list of all necessary components which need to be supported by the new Linux platform in order for the Linux platform to work as a FlockLab observer which can run tests.
- Implement and test the observer.
- Collect characteristic values of the implementation (e.g. remaining CPU load of the Linux platform while running a test, time required for setting up / stopping a test, etc.)
- Document your project with a written report. As a guideline, your documentation should be as thorough to allow a follow-up project to build upon your work, understand your design decisions taken as well as recreate the experimental results.

3 Project Organization

Deliverables

- Time schedule (at the end of first 2 weeks)
- Initial Presentation (3 min)
- Final Presentation (15 min)
- Code of implementation including documentation
- Written report which includes: Introduction, Analysis of related work, Documentation of decisions, Evaluation, Description and HowTo guide of the developed software.

Offers

- The supervisors offer the student the opportunity to do a rehearsal of the initial and the final presentation. The supervisors offer to give feedback how to improve the presentations.

- The supervisors offer to proof-read a draft of the final report. The draft is not required to be complete. The draft should be handed in no later than 1 week before the deadline of the thesis.

General Requirements

- The project progress shall be regularly monitored using the time schedule. Unforeseen problems may require adjustments to the planned schedule. Discuss such issues openly and timely with your supervisor.
- Use the work environment and IT infrastructure provided with care. The general rules of ETH Zurich (BOT) apply. In case of problems, contact your supervisor.
- Discuss your work progress regularly with your supervisor. In addition to such meetings, a short weekly status email to your supervisors is required containing your current progress, problems encountered and next steps.

Handing In

- Hand in a single PDF file of your project report. In addition, hand in the signed declaration of originality on paper.
- Clean up your digital data in a clear and documented structure using the provided GitLab repository. In the end, all digital data should be contained in the student's GitLab repository for the thesis. This includes: developed software, measurements, presentations, final report, etc. An exception is large amounts of measurement data which is stored separately (ask your supervisors!).

References

[1] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*, IPSN '13, pages 153–166, New York, NY, USA, 2013. ACM.

[2] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele. Measurement and validation of energy harvesting iot devices. In *Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, Lausanne, Switzerland, Mar 2017.