Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

LORA-based Data Collection on DPP

Semester Thesis

Michael Keller
kelmicha@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

August 3, 2018

Acknowledgements

I would like to thank the TIK institute for letting me write this thesis and be part of their research. A special thank goes to my supervisors Jan Beutel and Roman Trüb for their support during the thesis.

Abstract

Climate change affects our world. With the rise of temperature, permafrost regions get smaller and smaller and leave unstable grounds. In addition block glaciers begin to melt and move faster. Both phenomena increase the risk of rockfalls and rock waste avalanches. In some cases the rocks can reach populated areas and damage houses or infrastructure.

To predict such events ETH, as part of the Permasense consortium [1], is developing and installing sensor networks in mountain areas, where possible falling rocks could endanger people and infrastructure. These networks gather data as the ambient temperature, the movement of the rocks or vibration intensities.

As the nodes are installed in areas where it is not possible to connect them to a power grid, they run on batteries. These batteries should last as long as possible, to ensure continuous and low-maintenance operation. To achieve this the nodes need to be very energy efficient.

A large part of the energy is used for communication between the nodes. Therefore a communication protocol to gather the data from all nodes needs to be designed with a radio duty cicle that is as small as possible. This protocol should also adapt to network changes in case of node or link failures. Dozer is one possibility of such a protocol [2]. It has been implemented in TinyOS [3] and is currently used at different test sites in the alps.

The goal of this thesis was to port the existing Dozer implementation to a new platform. This new dual processor platform separates the application from the communication part. A communicaiton board featuring a radio capable of FSK and LoRa modulation has been developed in a concurrent master thesis [4]. The implemenation was done without any OS support to stay as close as possible to the hardware and get the best possible timings.

Contents

Acknowledgements **i**

Abstract **ii**

1 Introduction **1**

1.1 Climate Change and Natural Hazards 1

1.2 Permasense 1

1.3 Sensor Networks 2

1.3.1 Data Gathering 2

1.4 Goals 2

2 Background **3**

2.1 Related Work 3

2.2 Dual Processor Platform 3

2.3 Communication Board 4

2.3.1 LoRa Modulation 4

2.4 Evaluation Board 4

3 Dozer **5**

3.1 Overview 5

3.2 Bootstrapping Phase 6

3.3 Connection Phase 6

3.4 Normal Operation / Data Sending 8

3.5 Topology Control 8

4 Implementation **9**

4.1 Message Types 9

4.1.1 Dozer Message 9

4.1.2 Header 9

4.1.3 Payload 10
4.2 File Overview 11
4.3 Topology Control 12
4.3.1 Bootstrap 12
4.3.2 Connection 13
4.3.3 Data Sending 13
4.4 Radio Administration 15
4.5 Radio Specifics 16
4.5.1 Rx Callback 16
4.5.2 Tx Callback 16
4.5.3 Radio Timeouts 16
4.5.4 Addressing 17
4.5.5 Channel Activity Detection 17
4.6 Timers 17
4.6.1 Timer Queue 17

5 Evaluation **19**
5.1 Test Configuration 19
5.2 Evaluation Board 19
5.3 Flocklab 19

6 Conclusion **21**

7 Future Work **22**
7.1 Testing 22
7.2 Timing 22
7.3 LoRa Modulation 22
7.4 Channel Activity Detection 23

A How to use **1**
A.1 Serial Outputs 1

B Task Description **5**

Bibliography **9**

Chapter 1

Introduction

1.1 Climate Change and Natural Hazards

With climate change and the rise of temperature natural catastrophes begin to occur more frequently. In the mountains permafrost and glaciers disappear. The thawing grounds become unstable, rocks or ice masses may break off and rattle down the mountain sides to the valley floor. Rock glaciers, which are a mixture of rock and ice that move slowly down the mountain sides, also thaw and can move faster with rising temperatures. They push big masses of rock waste downhill, which could fall off if it gets steep enough and the ice does not hold them back anymore. Wherever they originate, rock waste avalanches can destroy everything in their way. They are a great danger for houses and infrastructure situated at mountain sides or narrow valley constrictions. These events are very hard to predict. To observe critical sites by measuring key quantities as movement, vibration or temperature sensor nodes have been deployed by the Permasense consortium.

1.2 Permasense

The Permasense consortium is a collaboration of different research institutes and companies, including ETH [1]. They develop, deploy and maintain sensor networks, that are able to gather data in challenging high mountain areas, as for example on rock glaciers in the Matter Valley above Herbriggen and Randa. The data collected by these sensor networks are used to investigate changes of the environment as the movement of rock glaciers or long term evolution of the temperature. Algorithms are designed to predict the trend of the movement and warn the surrounding areas in case of an imminent rock waste avalanche or similar dangerous events.

1.3 Sensor Networks

Sensor nodes in hardly accessible high mountain areas run only on batteries. It is also not possible or very costly to change them regularly. Some small amount of energy can be harvested for example with solar panels or thermoelectric devices. But for long term autonomous operation the nodes need to be very energy efficient. A tradeoff between the amount of data samples and energy consumption needs to be found. But the data acquisition and handling is not the only part where the power used is key. A crucial aspect of low-power operation is the communication part. Transmitting and receiving with a wireless radio needs quite some power.

1.3.1 Data Gathering

As the data samples are useless on a node somewhere in the mountains they need to be collected. Besides the small sensor nodes there is a bigger node, called the sink, which gathers all the data and forwards it to a radio station connected to the internet. This could be a station from a radio car or even a station on the valley floor. The nodes all try to send their data to the sink. If this is not possible because they are to far away or the connection is disturbed by objects in between, they send it to a neighboring node which then forwards it to the next node until the sink is reached. This is also referred to as multi-hop data sending. To prevent uncoordinated sending and receiving of data, which would result in many collisions and the need to resend the data, specific communication protocols have been developed. These protocols coordinate the data collection of the nodes and try to minimize the time the radio needs to be active to reduce energy consumption.

1.4 Goals

Since the first deployments of networks a new Dual Processor Platform (DPP) for the nodes has been developed. The DPP consists of two parts, separating the application processor from the communication processor. For this DPP a communication board, which features FSK and additionally the LoRa modulation [5] has been designed in a concurrent master thesis [4].

The main goal of this thesis was to port the existing Dozer implementation to the new communication board for the DPP. To stay as close as possible to the hardware and get exact timings no operating system was used. After the implementation the protocol had to be tested.

Chapter 2

Background

2.1 Related Work

Dozer was proposed as a quite simple and very energy efficient data gathering protocol. It achieves very low radio duty cicles, but has higher latency compared to other protocols. It has been deployed on various test sites by the Permasense consortium and has proven to perform under harsh circumstances in high mountain areas.

As the applications for wireless sensor networks demand increasing computational resources, there is a trend towards multi processor architectures, to divide application and communication. This means each processor can be chosen to suit best to its task and the interference between completely different tasks gets minimized in comparison to just one processor. The TIK institute at ETH has developed a way to connect those processors with a minimal overhead in communication. The communication is handled by BOLT, which is a processor that is only used to handle the communication between the other processors. This means that the application processors are really independent and can focus on their tasks. As Dozer is still an exceptional alternative if latency is not critical, the way to go is to adapt it to the new dual processor platform.

2.2 Dual Processor Platform

The idea of the dual processor platform (DPP) is to separate the application part of a sensor node platform from the communication part. Each part has its own processor dedicated to the respective task. The communication between the two processors is handled by BOLT [6]. Bolt is a stateful interconnect. This means its also a processor with some memory to store the messages to pass. So both application parts can independently communicate with BOLT and do not require to synchronize with each other. This allows for specialized and thus very energy efficient communication and application routines.

2.3 Communication Board

Important for this thesis is the communication part of the dual processor platform. In a concurrent master thesis a new communication board has been developed [4]. It features the 32 bit STM32L443CCU microcontroller from STMicroelectronics with a clock of up to 80MHz [7] and the Semtech LoRa transceiver SX1262 [8]. The transceiver also supports, additionally to the commonly used FSK modulation, Semtech's LoRa modulation.

2.3.1 LoRa Modulation

The LoRa modulation is a registered trademark of Semtech [5]. It is a form of chirp spread spectrum modulation. The data signal is modulated onto a chirp signal, that continuously varies in frequency. This results in a much broader bandwidth, which makes the signal robust to interference and allows the reception of the signal even if the signal to noise ratio is very small. LoRa modulation therefore allows for much farther transmission distances as for example FSK modulation at the same output power.

2.4 Evaluation Board

As the new communication board was not ready at the beginning of this thesis, most of the time a development kit from Semtech was used. It consisted of the NUCLEO-L476RG board from STMicroelectronics [9] and the SX1262 evaluation board from Semtech. The STM32L476RG microcontroller [10] on the Nucleo board is very similar to the one on the new communication board. This allowed to run the code on both platforms with only minimal changes.

Chapter 3

Dozer

Dozer is a low-power data gathering protocol developed at ETH [2]. The concepts described in this chapter are the ones from the original paper. They are also applied in the TinyOS implementation of Dozer and are reused for the implementation done in this thesis.

3.1 Overview

Dozer first builds a tree like network topology, where all data is sent to the root or sink. If there are node or link failures Dozer adapts the topology so no node in reach of another one is lost.

Each group consisting of one parent and its children has its own time synchronization. This means there is no global time synchronization, which would be more complex and costly to implement. The parent periodically sends a message called beacon, allowing the children to update their timings. A random jitter in the beacon send times is used to let the different schedules of the nodes drift. Otherwise it could happen that active phases of multiple schedules are very close to each other. If for example multiple nodes then send their beacon at the same time, the collision could make it impossible for other nodes to receive the beacons.

Each node wakes up periodically at the beginning of an interval to send its beacon. As there is no global time synchronization the interval start and end times are different for each node, but the interval length is the same. The interval starts with sending the beacon message. The rest is divided into slots of equal length. A parent advises each of its children a different slot, so the child knows when it can send data to the parent. The first slot after the beacon is sent, is used as contention window for new nodes to connect. The composition of the Dozer interval is also shown in Fig. 3.1.

Figure 3.1: Overview of the Dozer interval.

3.2 Bootstrapping Phase

A node that is not connected to the network, does not send a beacon, but starts listening for activity on the channel. If it receives no message in this time, it reduces the listening time and tries again at the beginning of the next interval. The listening time gets reduced until a minimum is reached. The node then listens for only a short time at the beginning of each interval. This is done to minimize the energy consumption of a disconnected node, while still preserving the chance of connecting to the network, if another node gets reachable.

If the node detects activity on the channel, it starts listening for the whole next interval. This time it specifically looks for beacon messages. Each beacon contains information about the potential parent that sent it, which is used to rate it. The rating criteria are the distance to the sink and the amount of children already connected to that node. All the potential parents get stored, to be able to quickly connect to another parent if the current one is lost.

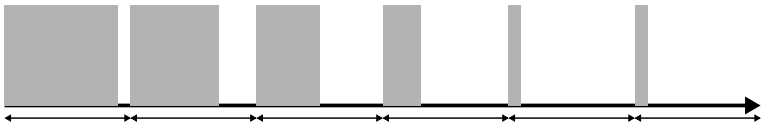


Figure 3.2: Dozer bootstrapping phase with decreasing listening times for the case that no channel activity was detected.

3.3 Connection Phase

The connection procedure is shown in Fig. 3.3. After listening for beacons the node tries to connect to the node with the best rating. It starts listening, when the next beacon of this node is meant to arrive. If the correct beacon is received, it sends an activation message. The activation message does not contain any information. It is just used to signal the potential parent, that a node wants to connect to it. If multiple nodes send an activation message at the same time, the potential parent possibly can not receive the messages correctly but it still detects that the channel is busy. The parent then starts listening for connection requests. After the activation message the child sends the connection request with a random delay. The random delay should prevent collisions if multiple

nodes want to connect. Upon reception of a connection request the parent stores the ID of that child and responds with a handshake message, containing the slot number for this child to send data. No handshake is sent if the parent has no more free child slots. The sink accepts multiple connections per interval and again starts listening for connection requests after a sent handshake. All other nodes only accept one new child per interval. Once a node is connected to a parent it starts sending its own beacon.

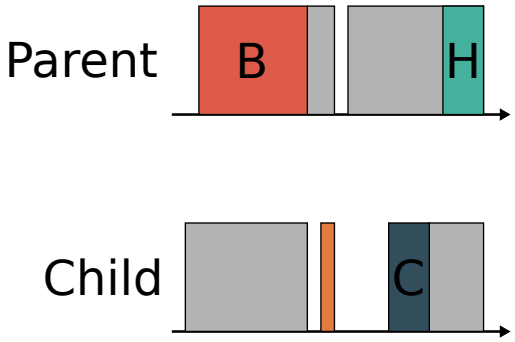


Figure 3.3: Dozer connection phase. The gray areas mean that the node is receiving.

3.4 Normal Operation / Data Sending

A node that is connected to a parent wakes up to receive its parents beacon, uses it to correct the relative timing to the parent and calculates the time to send data. Parallel to sending data to its parent it sends its own beacon and receives data from its children.

The parent node wakes up at the beginning of an occupied data slot and starts receiving. Data messages in contrast to the others are always acknowledged, so that no relevant data is lost. If the data transmission from a child is finished the parent waits until the next occupied slot starts, to receive data from the next child. An example of the data sending sequence can be seen in Fig. 3.4.

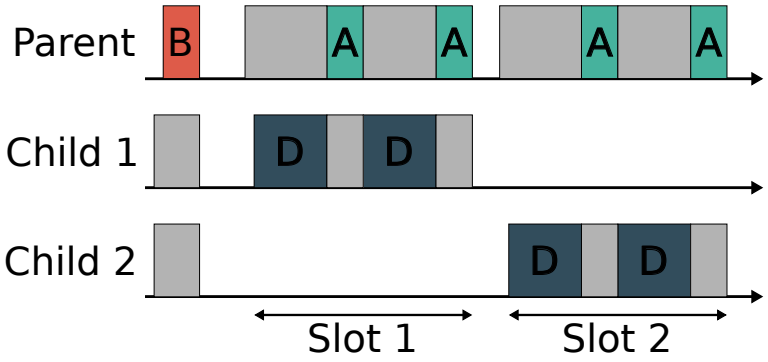


Figure 3.4: Dozer data sending phase. The gray areas mean that the node is receiving.

3.5 Topology Control

If some expected beacons from a parent do not arrive or the data messages do not get acknowledged, the parent gets marked as not reachable. In this case the node tries to reconnect to the next best rated potential parent. In case of no more known potential parents, the node goes back to the bootstrapping phase.

From time to time a node listens for beacons from stored potential parents to update their rating, or even for beacons of yet unknown nodes. This allows the node to reconnect without using to much energy, that would be needed for the bootstrapping.

Chapter 4

Implementation

This chapter describes the details of the implementation done in this thesis. It is assumed that the reader is already familiar with the concepts of Dozer as explained in Chapter 3.
The topology control and the radio administration details have been ported from the TinyOS implementation with only minor changes. The differences primarily lay in the radio handling and the usage of timers as described in Section 4.5 and Section 4.6.

4.1 Message Types

4.1.1 Dozer Message

The different message types and their composition is shown in Fig. 4.1. The main message type is the *dozer_message*. It consists of the header the payload and some metadata. Only the header and payload part are actually sent. The metadata part contains information that is added after the reception of a message. This is the receive time, the size of the received message and the measured RSSI and SNR.

4.1.2 Header

The source field contains the ID of the node that sent the message, the destination contains the address of the node that should receive it. The address can either be a node ID or the broadcast address, if the message is for all nodes.
The ack field is used to signal the receiver, if a message should be acknowledged or not. During the data sending phase the child can also use it to signal the parent, that this is the last message sent.
The type defines what kind of payload is contained in this message.

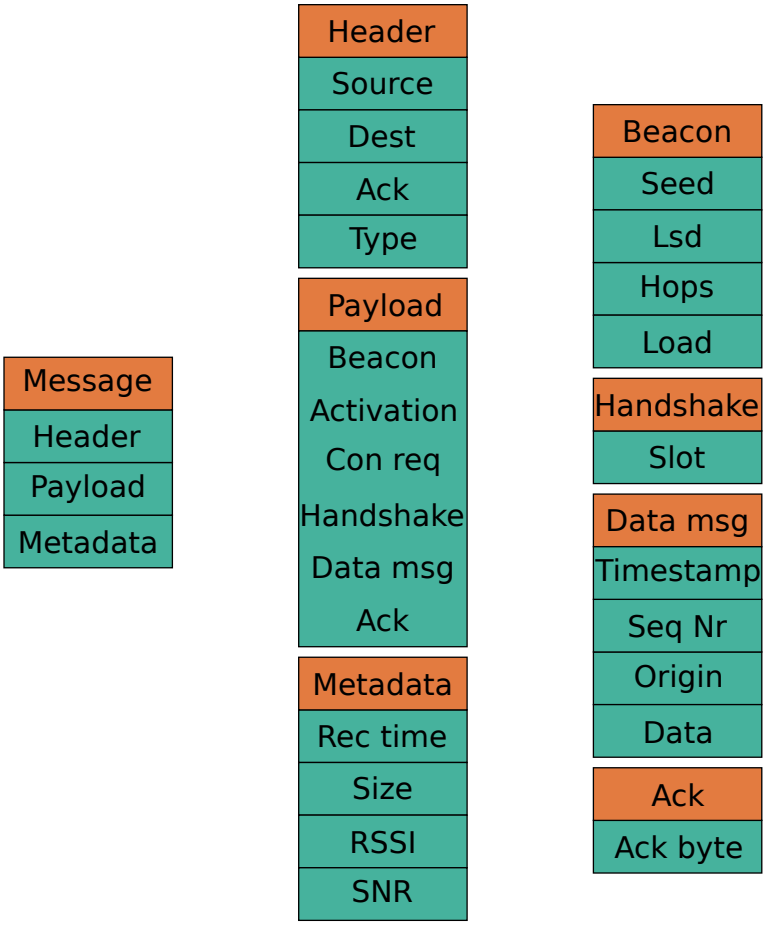


Figure 4.1: Message types.

4.1.3 Payload

The payload part of the message depends on the message type. It is implemented as a union, which means that a message has just one of the available payloads active at a time. The different possibilities are shown on the right side of Fig. 4.1. The activation and connection request messages are not listed separately as they have no payload.

Beacon Message

The hops field indicates the distance between the sender and the sink, the load how many children are already connected to the sender. The seed and lsd values are used to calculate the next beacon receive time.

Handshake Message

The handshake message contains only the slot number, that the parent allocated for this child.

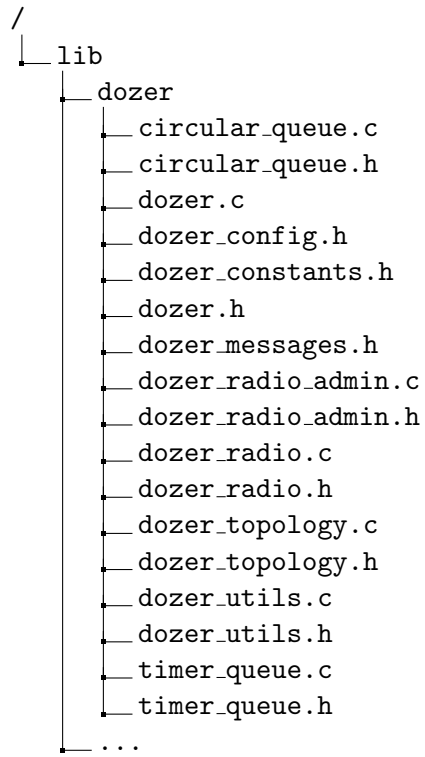
Data Message

All data messages get numerated to tell them apart. This information is stored in the sequence number field. Additionally the data message contains the ID of the originating node and the timestamp, when it was generated. The data field is an array of data bytes.

Acknowledgment

The acknowledgment message only contains one byte, which is used by the parent, to tell the child how many more data messages it can store in its data queue.

4.2 File Overview



The main part of the implementation is divided into two parts. The topology control in the *dozer_topology* files is the heart of the implementation. It handles the bootstrapping, manages all important information about potential parents and triggers the radio administration to send or receive packages at the right times. To get the timing as good as possible error estimation and drift corrections for beacon receiving are also done by the topology control.
The *radio_admin* files contain the radio administration part. It handles the message sending and receiving, chooses what to do after a successful or failed send or receive and signals the topology control if the receive or send task was successful.

The platform specific radio handling, as the configuration or the handling of the interrupts, is done in the *dozer_radio* files.

High level configurations for the protocol can be done in *dozer_config.h*. More detailed configurations are in *dozer_constants.h*. The different message types are defined in the *dozer_messages.h* file. The typedefs for the parents and children, the node configuration and the available states for the state machines are stored in *dozer.h*.

The *circular_queue* is used to store the data messages. The *timer_queue* handles the different timers.

4.3 Topology Control

The topology code builds around the beacon timer. The beacon timer fires periodically and initiates the bootstrap phase, the connection to a parent or the sending of a beacon to potential children. The sink does not need to connect to another node and just sends its beacon. All other nodes first check if they are in bootstrap mode. If this is the case they start the bootstrap overhearing phase.

4.3.1 Bootstrap

The bootstrap phase is built as a small state machine with three states, as shown in Fig. 4.2. The node begins the bootstrap phase in the *waiting* state and starts listening for activity. If nothing is received the listening time gets reduced until it reaches a minimum. The listening start time gets shifted by half the minimum each time the beacon timer fires. When it reaches the end of the interval it is reset to the interval start. This is done to listen at different times relative to the interval start, increasing the chance to receive something.

If something is received the node goes into the *activity detected* state. The next time the beacon timer fires, it goes into the bootstrap state *off* and starts listening for beacons for the whole next interval. All potential parents, from which a beacon was received during this time, are rated and stored. The most

important rating information is the distance to the sink, the second one is the number of children already connected to the potential parent. In case of a tie with those two rating informations the node ID of the potential parents is used as a last resort to separate them.

$$parent\_rating = distance\_to\_sink * MAX\_CHILDREN^2 + 1 + load * MAX\_CHILDREN + ID\%MAX\_CHILDREN$$

A node goes back to the bootstrapping state *waiting* if the connection to the parent is lost, or the connection attempts to all stored parents fail.

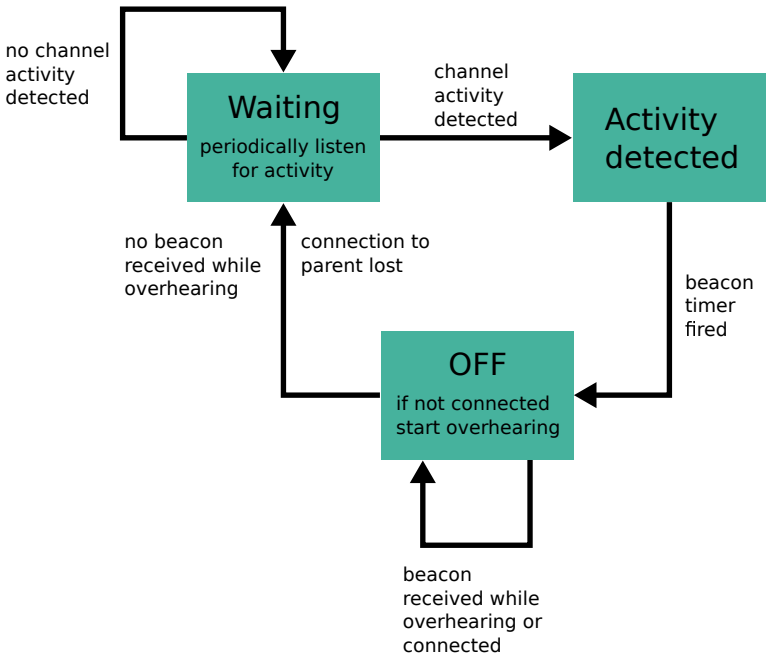


Figure 4.2: State diagram for the bootstrap phase.

4.3.2 Connection

After the sending of a beacon the node listens for channel activity. This is done by RSSI detection. If the value is above a certain threshold, it is assumed, that the channel is busy. The node then starts listening for connection requests.

4.3.3 Data Sending

Each child gets a slot allocated from the parent when it is allowed to send its data. After sending a data packet it waits for an acknowledgment. The parent includes

the information about how many more messages it can store in the message queue in the acknowledgment message. If this is zero, the last queue entry was filled with the last sent message. In this case the child stops sending data and tries again in the next interval. The parent does not send an acknowledgment at all if the data queue was already full upon data reception.

4.4 Radio Administration

The radio administration is built as a state machine as depicted Fig. 4.3. Most of the time it is in the *idle* state. It gets called by the topology control to send / receive a beacon or to send / receive data messages. In case of a receive, transmit or timeout interrupt from the radio it checks the current state, continues to the next state and signals the topology control what happened. It also checks if it is in the correct state before sending something.

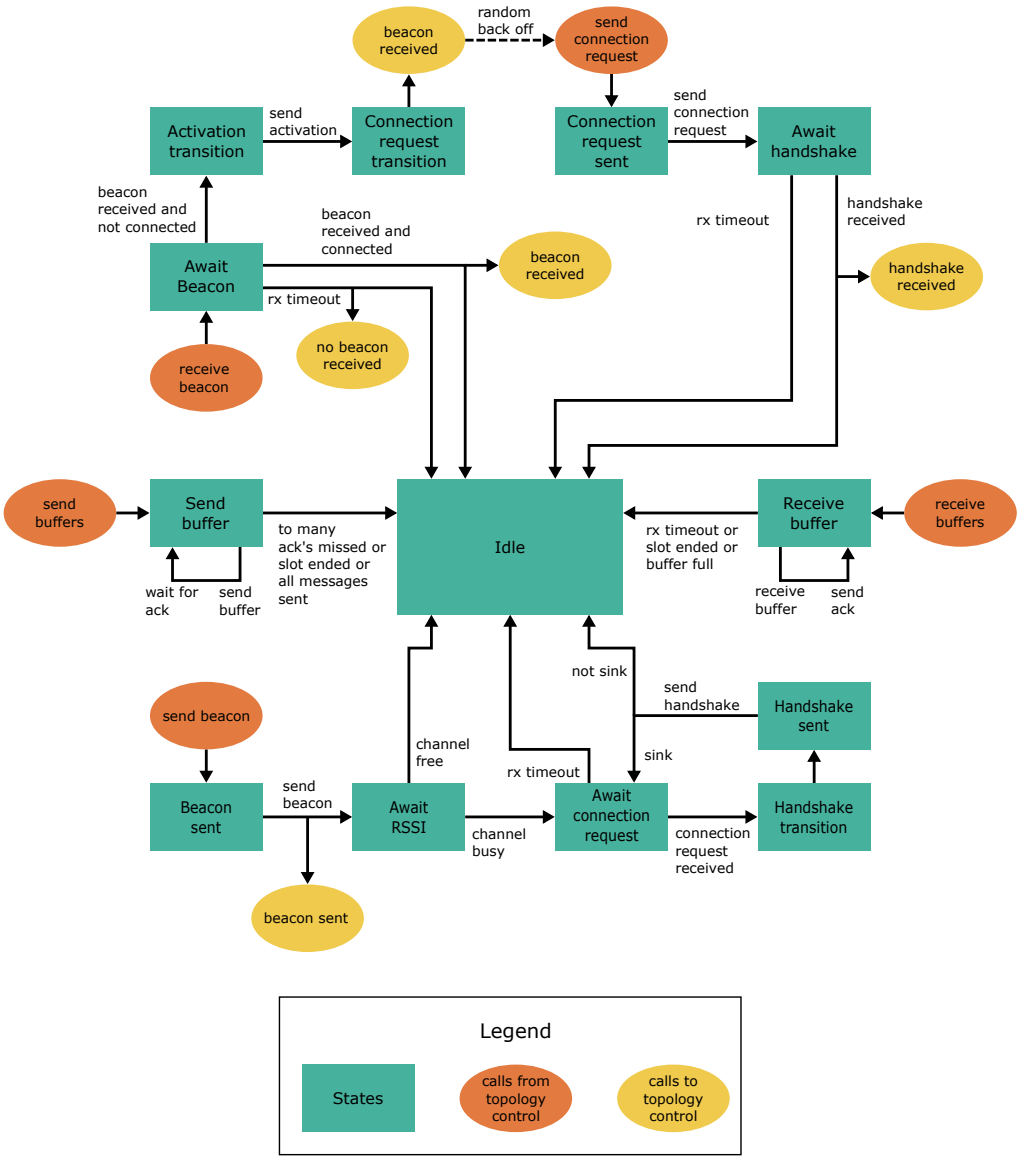


Figure 4.3: State diagram for the radio administration.

4.5 Radio Specifics

The *dozer_radio* files contains the platform specific radio commands. The *dozer_send()* and *dozer_receive()* functions first check if the radio is idle. If not they return an error. Else the radio is set in send or receive mode with the timeout specified. The *dozer_tx_callback()* and *dozer_rx_callback()* functions are triggered by the radio interrupts if a message is successfully sent or received. If the send or receive times out a timeout interrupt occurs and the *dozer_timeout()* function gets called.

4.5.1 Rx Callback

The *dozer_rx_callback()* function first checks the destination field in the message header. If the message is not for this node it continues listening.

Another field of the message header is used to request an acknowledgment for the message. This field is also checked and the acknowledgment is sent if requested. If an acknowledgment is received it gets passed to the *handle_ack()* function of the *dozer_radio_admin* file.

If the message is for this node and no acknowledgment is requested the message is forwarded to the *dozer_rx_done()* function of the *dozer_radio_admin* file.

4.5.2 Tx Callback

The *dozer_tx_callback* calls the *dozer_tx_done()* function of the *dozer_radio_admin* file, except when an acknowledgment was sent. In this case the *dozer_rx_done()* function gets called with the message that requested the acknowledgment.

4.5.3 Radio Timeouts

For most of the receive and all send commands a timeout is also given. The timeout is passed on to the radio, which then triggers an interrupt if the message could not be sent or no message was received before the timeout. The radio timeout timer runs with a 64 kHz clock leading to a resolution of 15.625 us. The current implementation however only uses millisecond precision for radio timeouts.

The receive timeout sometimes does not trigger an interrupt. If this happens the state machine can get stuck in one state. To prevent this an additional timer gets started at the beginning of a receive command. In case the interrupt does not occur the timer will fire and call the timeout callback.

4.5.4 Addressing

The radio offers to filter incoming messages by their address. To use this feature a configuration register needs to be set and the filtering address to be specified. Unfortunately this did not work, as the radio still passed on all received messages, even if the address was wrong, or the message was received with a CRC error. In the end an address field was added to the dozer message, that gets checked in the receive callback function.

4.5.5 Channel Activity Detection

The radio has the ability to get the RSSI value any time, when it is listening. Measuring the RSSI only once after the beacon sending, as it is done in TinyOS, was not enough to get a good result for the channel activity. The node now listens for some milliseconds and checks the RSSI again and again. If it once is larger then the predefined threshold, it is assumed that there is activity on the channel. Unfortunately this consumes more energy as the radio needs to be in receive mode, while checking the RSSI value. But as the RSSI values measured are not what was expected in terms of difference between a busy and a free channel, this was the method with a better success.

At the end of the thesis it was detected, that the unexpected behaviour may be caused by a too low transmit power. For higher transmit powers it should be possible to perform a single RSSI snif as in the TinyOS implementation.

4.6 Timers

The TinyOS implementation makes heavy use of timers. The microcontroller on the new board however did not have so many independent hardware compare registers. On the other hand some compare registers were only 16 bit wide. As a clock in the MHz regime should be used, to allow for very precise timings, this was not enough. It would have been possible to chain timers to build a 32 bit timer out of two 16 bit ones, but in the end it anyway was not possible to map all needed timers to a compare register. The solution to this problem was the implementation of a timer queue.

4.6.1 Timer Queue

The timer queue allows to have multiple timers running concurrently, while only using one hardware compare register. All the timestamps when a timer should fire are stored in a sorted queue. Additionally to the timestamp the queue entry also contains the timer name and a pointer to a callback function. The name

is used to be able to stop a timer without having to know the timestamp. The different possible timer names are declared in the *timer_queue.h* file. The callback function is triggered when the corresponding timestamp is reached. The current entry then is discarded and the compare register is reloaded with the next value from the queue.

If the next value has already passed, the callback nevertheless gets executed and the timer advances to the next timestamp. This could happen if a timer with an already passed timestamp gets inserted, or if two timestamps are too close to each other. The callback then is a bit too late but gets executed to prevent the protocol from getting stuck.

As the queue should be as simple as possible, to reduce time overhead after firing, there is no check if a timer has already been inserted into the queue. It would therefore be possible to have multiple timers with the same name in the queue. In the current Dozer implementation it has been taken care to only insert one timer with the same name at any time. If there would be multiple timers with the same name in the queue, that would be no problem, except if a timer should be stopped. In the case of a stop call, the queue gets traversed until the timer with the specified name is found. It then gets removed and the function returns. This means that only the one with the smallest timestamp gets removed.

Chapter 5

Evaluation

5.1 Test Configuration

For the test a function which generated data messages was included. Each message was marked with the node ID and a sequence number. The function was called each interval and generated ten messages each time. With an interval length of 10 s this gives an average of one message per second.

5.2 Evaluation Board

Most of the time during this thesis the code was tested on the evaluation board. The test setup contained three boards plugged in to the same PC and lying on the same desk. To simulate a multi-hop network topology the address control mechanism was slightly adjusted. The sink and one other node ignored messages from each other and continued receiving as if nothing happened. This resulted in a linear topology of the three nodes.

With only this three nodes the protocol performed very good. In the case that two nodes directly connected to the sink, all messages arrived at the sink and the topology was stable.
For the multihop configuration the topology was not perfectly stable, which was because of the fact that all nodes received all messages, leading to collisions and data misses.

5.3 Flocklab

At the end of the thesis one bigger successful test over one hour was possible on Flocklab. Seven nodes were included, four of them connected directly to the sink, two had to connect to another node. This led to a maximum of one hop. More would have been preferable but the reach of the nodes was so good, that it was difficult to generate more hops.

Table 5.1 shows the number of messages that arrived at the sink / node 3. The nodes 20, 23, 25 and 32 were directly connected to the sink, nodes 7 and 11 over one additional hop. The main reason, that packages did not arrive at the sink, was that a node was not connected for a longer time and its data queue was full. So it had to throw away messages. The results are not very good but 1 hour is also not a very long time for dozer to stabilize its topology. Especially node 11 reconnected multiple times to different nodes. Nodes that were stable connected to the sink as 20 and 23 achieved a very high data arrival rate.

Origin	Number of messages
7	1950
11	108
20	3529
23	3428
25	2352
32	1896

Table 5.1: Number of messages per node that arrived at the sink.

Chapter 6

Conclusion

The Dozer data gathering protocol has proven to be very energy efficient. It is already in use in various test sites and it should be possible to also use it in the feature. To achieve this, Dozer has been implemented on a new dual processor platform during this thesis. The concepts have been ported from the previous TinyOS implemenation.

The version running on the new platform includes the connection, topology control and data gathering parts. Additional features as for example the ability to send commands to the nodes, or the gathering of status information can be built on top of the current version.

Unfortunately there was not enough time to repeatedly test the protocol on Flocklab. The testing that could be done in the end of the thesis showed that the implementation works, but it needs to be tuned to the new platform and some bugs may still exist. It is very important to extensively test the code on Flocklab, to be able to adapt all timing calculations to the platform and remove the last problems of the implementation.

Chapter 7

Future Work

7.1 Testing

Unfortunately the new communication board was only integrated into the Flocklab testbed [11] at the end of this thesis. This did allow to run the code in a larger topology, to fix some bugs, but time was to short to really test it. It would be very important to extensively test the implementation on Flocklab. It may be that the code works fine for smaller set-ups and shorter time, but fails after some time or with more nodes involved.

7.2 Timing

As the timers now run with a 8 MHz clock, the timings can be much more accurate. A next step would be to measure and exhaust the guard times used. Until now the timings have been adopted from the TinyOS implementation, which only uses 32 kHz timers. It should be possible to decrease the radio duty cicle by leveraging the faster timers.

7.3 LoRa Modulation

The communication board used for this thesis was also developed to look into the LoRa modulation. The switch from FSK to LoRa is not that complex. There are a few radio configuration registers, that need to be set correctly. The configuration functions are the same as currently used for the FSK modulation, only the parameters need to be changed.

The RSSI check done after sending the beacon could be a problem, as the LoRa modulation has a very low SNR. The radio features a channel activity detection mode specially for the LoRa modulation, which could be used instead.

To decide which modulation or even frequency to use at runtime, would be

another idea. This is a challenging task for which the possibilities need to be explored first.

7.4 Channel Activity Detection

In some last tests it was possible to do the activity detection with only a single RSSI sniff. The reason why it was not stable before is, that the transmit power was to low. As there was no time left to test it, the final version is still with a longer sniff time. To make the single sniff more reliable adding some payload to the activation frame would be the way to go.

Appendix A

How to use

The code can be found in [12]. To get it running on the new platform or the development kit please refer to [13].

A.1 Serial Outputs

There are several serial outputs that should help debugging. They have different priorities. The function *dozer_print()* in the file *dozer_utils* checks the priority and prints the string. The most important outputs, which are enabled at the end of this thesis, are all state changes in the bootstrap and radio administration state machines, the sending and receiving of data messages, the last sequence number of generated data messages and the outputs if failures occur. The sink does not generate data, it instead prints and deletes all the messages in its data queue. Fig. A.1 and Fig. A.3 show the connection and data sending between two nodes. The ubuf stands for 'used buffers' and if the sink has some messages in the queue it will print them afterwards, as in Fig. A.2.

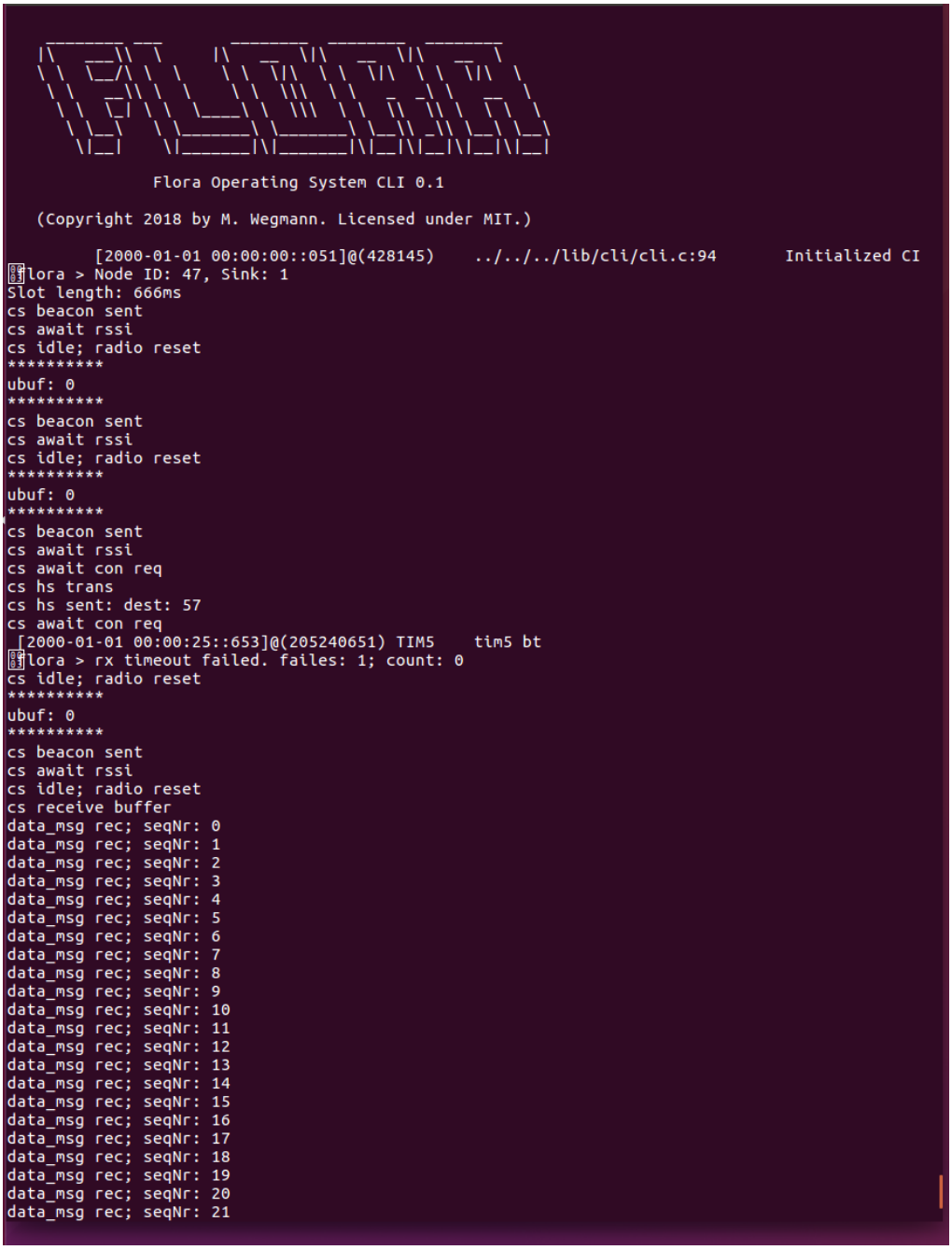


Figure A.1: Console output of the sink.

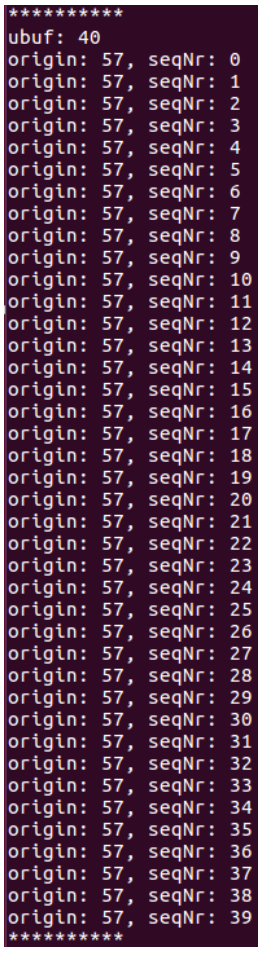


Figure A.2: Sink print off the messages in the queue.

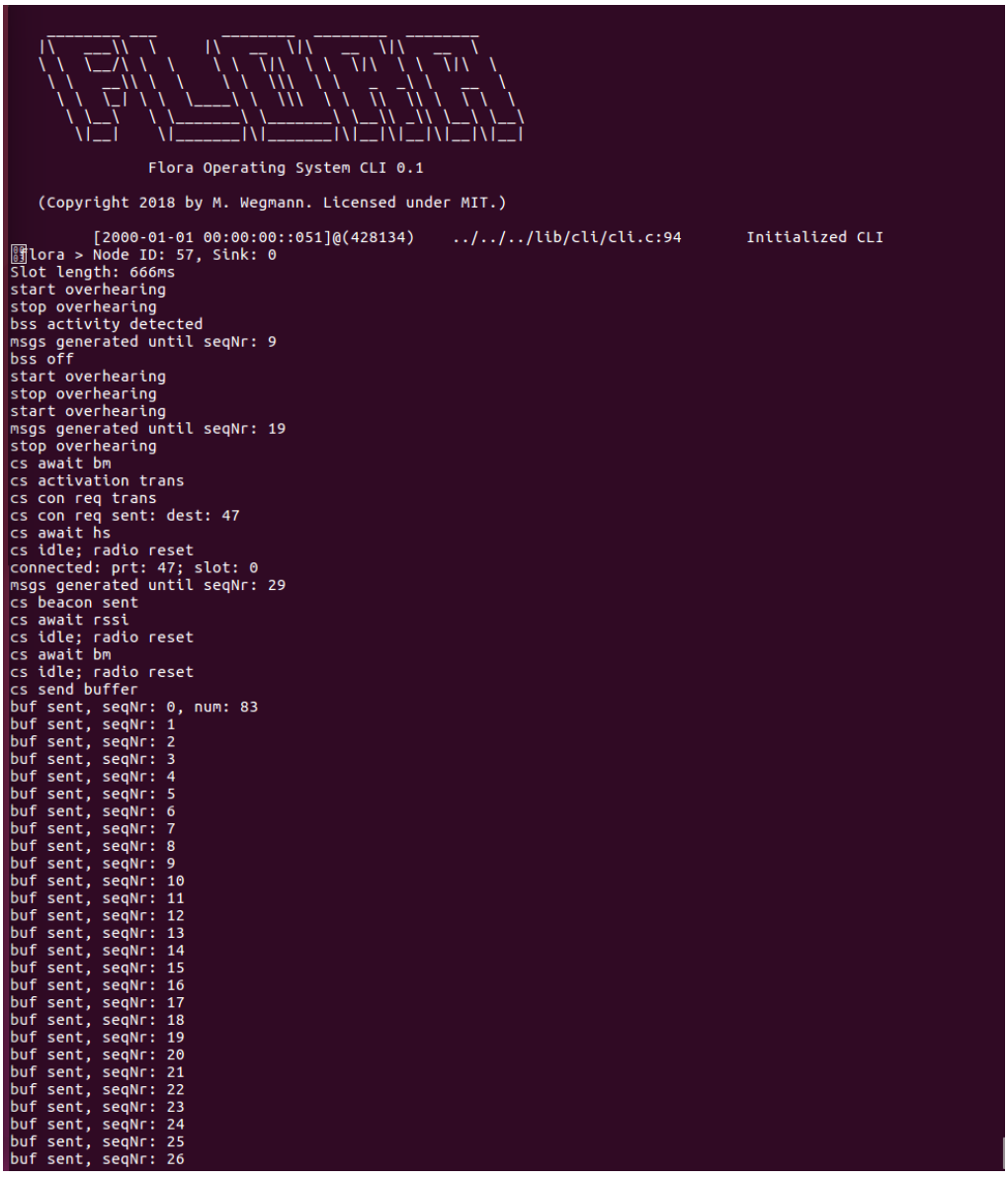


Figure A.3: Console output of a node connected to the sink.

Appendix B

Task Description

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Engineering and Networks Lab

ETZ G 75
CH-8092 Zurich

Dr. Jan Beutel
Gloriastr. 35
+41-44-6327032
+41-44-6321035
beutel@tik.ee.ethz.ch
www.tik.ee.ethz.ch/~beutel

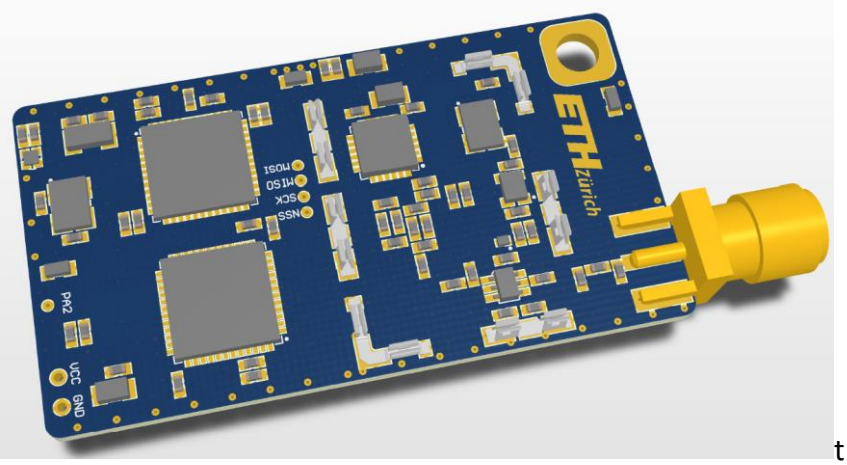
Semester Thesis Project
For
Michael Keller

Supervisor: Jan Beutel, Roman Trueb, Reto Da Forno

Start Date: June 18, 2018
Initial Presentation Date: June 18, 2018
Final Presentation Date (tentative): July 25, 2018
End Date: August 3, 2018

LORA-based Data Collection on DPP

The PermaSense project develops, deploys and operates wireless sensing systems customized for long-term autonomous operation in high-mountain environments. Around this central *element*, we develop concepts, methods and tools to investigate and to quantify the connection between climate, cryosphere (permafrost, glaciers, snow) and geomorphodynamics.



t

In this thesis project we aim at implementing a port of the Dozer [7] data collection protocol for the 3rd generation Dual Processor Platform (FLORA DPP) currently being developed. This platform is based on a recently released long-range, low-power, sub-GHz RF transceiver by Semtech (SX1262) and an ARM processor. The SX1262 transceiver incorporate different modulation modes (LoRa, (G)FSK, OOK) and an extremely high link budget at a very low power consumption (~4mA RX mode). In this project we first want to develop a basic data collection functionality based on the existing TinyNode/TinyOS implementation of Dozer using a statically set modulation mode. In a further step it is the possible to explore other modulation options and/or modifications to Dozer such as adaptive slot lengths, bidirectional data transfers etc.

Semester Thesis Project LORA-based Data Collection on DPP

The first part of this work will be based on the existing TinyOS/TinyNode implementation of Dozer and leverage the basic software support for the FORA DPP platform developed in a parallel thesis project by Markus Wegmann. For this second step it will be necessary to profile the network wide system behavior using our FlockLab testbed. Additionally, an ongoing thesis project by Jonathan Candel, comparing Glossy and Dozer performance based on models and experimental evaluation on the FlockLab testbed may be a valuable resource.

The driving application scenario is drawn from our ongoing work in environmental sensing using standard constant data rate sensors, GPS sensors and specifically the geophone sensing platform developed in a project by Akos Pasztor.

Thesis Project Assignment

- Formulate a time schedule and milestones for the project (duration 7 weeks). Discuss and approve this time schedule with your supervisor.
- Familiarize yourself with relevant past works within the project and the significant literature in the field. Search for new approaches and examine which concepts and new components could be of use here. Position your work within this context.
- Develop and implement a basic data collection system based on the current TinyOS/TinyNode implementation of Dozer on the FLORA DPP platform based on a static parameter configuration.
- Characterize the performance of this prototype in a suitable test setup.
- Perform extensive performance testing using our in-house FlockLab testbed.
- Optional: Explore different dynamic extensions such as adaptive slot length, bidirectional data transfer etc.
- Document your project with a written report, a short initial presentation, a final presentation and if applicable a demonstration of the prototype. 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.

Project Organization

General Requirements

- The project progress shall be regularly monitored using your time schedule and milestones. Unforeseen problems may require adjustments to the planned schedule and milestones. 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 excess 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 two paper copies as well as a single PDF file of your project report including the signed plagiarism statement.
- Clean up your digital data in a clear and documented structure using the GitLab repository provided.

References:

[1] https://www.tec.ee.ethz.ch/education/student-theses/general-information.html

[2] https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/declaration-originality.pdf

[4] Jan Beutel, Stephan Gruber, Andreas Hasler, Roman Lim, Andreas Meier, Christian Plessl, Igor Talzi, Lothar Thiele, Christian Tschudin, Matthias Woehrle and Mustafa Yuecel: PermaDAQ: A Scientific Instrument for Precision Sensing and Data Recovery under Extreme Conditions. Proceedings of the 8th International Conference on Information Processing in Sensor Networks (IPSN), p. 265-276, April 2009.

[5] Felix Sutton, Marco Zimmerling, Reto Da Forno, Roman Lim, Tonio Gsell, Georgia Giannopoulou, Federico Ferrari, Jan Beutel and Lothar Thiele: Bolt: A Stateful Processor Interconnect. Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys 2015), Seoul, South Korea, p. 267-280, November 2015.

[6] R. Lim et al: FlockLab: A Testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems. Proc. IPSN/SPOTS 2013.

[7] Nicolas Burri, Pascal von Rickenbach and Roger Wattenhofer: Dozer: Ultra-Low Power Data Gathering in Sensor Networks. International Conference on Information Processing in Sensor Networks (IPSN), Cambridge, Massachusetts, USA, April 2007.

Bibliography

[1] "Permasense," accessed July, 2018. [Online]. Available: www.permasense.ch

[2] "Dozer: ultra-low power data gathering in sensor networks," accessed July, 2018. [Online]. Available: https://dl.acm.org/citation.cfm?id=1236417

[3] "TinyOS," accessed July, 2018. [Online]. Available: www.tinyos.net

[4] "Master thesis markus wegmann," accessed July, 2018. [Online]. Available: https://gitlab.ethz.ch/tec/students/projects/2018/ma_mwegmann

[5] Semtech. (2015, May) LoRa^{TM} Modulation Basics. Document Version 2. [Online]. Available: https://www.semtech.com/uploads/documents/an1200.22.pdf

[6] "Bolt: A Stateful Processor Interconnect," accessed July, 2018. [Online]. Available: https://dl.acm.org/citation.cfm?id=2809706

[7] "Stm32l443ccu microcontroller," accessed July, 2018. [Online]. Available: https://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32-ultra-low-power-mcus/stm32l4-series/stm32l4x3/stm32l443cc.html

[8] "Semtech sx1262 transceiver," accessed July, 2018. [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1262

[9] "Nucleo-l476rg board," accessed July, 2018. [Online]. Available: https://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-nucleo/nucleo-l476rg.html#quickview-scroll

[10] "Stm32l476rg microcontroller," accessed July, 2018. [Online]. Available: https://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32-ultra-low-power-mcus/stm32l4-series/stm32l4x6/stm32l476rg.html

[11] "Flocklab," accessed July, 2018. [Online]. Available: https://gitlab.ethz.ch/tec/public/flocklab/

[12] "Code repository," accessed July, 2018. [Online]. Available: https://gitlab.ethz.ch/tec/research/dpp/com_boards/dpp2_lora

[13] "Dpp2 lora: Getting started," accessed July, 2018. [Online]. Available: https://gitlab.ethz.ch/tec/research/dpp/com_boards/dpp2_lora/wikis/home