Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

LWB with Long-Range Modulation

Master 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

April 18, 2019

Acknowledgements

I want to express my great gratitude for the TIK institute and Prof. Dr. Lothar Thiele for giving me the chance to write this thesis. A special thanks goes to my supervisors Roman Trüb and Jan Beutel. They formulated a very interesting problem statement for this thesis and I could always ask them if I needed help.

Abstract

The Permasense [1] consortium is active in the field of low-power wireless sensor networks. They are deployed in high alpine areas to collect data such as temperature, vibration and also the displacement of rocks. One goal is to be able to predict rock waste avalanches and major rockfalls with this data.

To explore the possibilities of the LoRa modulation, a new radio communication platform was developed in a master thesis in 2018 [2]. The SX1262 radio [3] on the platform allows to switch between FSK and LoRa modulation on a single chip. During the same thesis a network flooding protocol called Gloria was implemented on the new platform. It is based on Glossy [4] and is one of the first flooding protocols with the LoRa modulation.

During this thesis the existing version of Gloria was extensively tested and improved. Finally Gloria is a highly configurable and independent flooding protocol which can be used to build higher layer communication protocols on top of it.

A simple version of such a protocol called LWB [5] was implemented with the help of Gloria. The main goal of this thesis however was not to get a very sophisticated version of LWB, but to explore the possibilities to extend it with the help of the LoRa modulation. The extension allows to include nodes into the LWB network, which are too far away to participate in the Gloria floods of the basic LWB version. This is done while preserving the stateless concept of LWB.

Contents

Chapter 1

Introduction

This thesis is part of the Permasense research. During a previous thesis [2] a new communication platform was developed. On top of this platform a network flooding protocol called Gloria was implemented. It is based on Glossy [4].

This chapter gives an overview of Permasense and some more details on the already existing hardware and software. Finally it states the goals for this thesis.

1.1 Permasense

The permasense consortium [1] develops, deploys and maintains wireless sensor networks in high alpine areas. The tasks range from the development of the hardware inside a node over the implementation of communication protocols to the evaluation of the data. The data collected can be temperature, vibration or even GPS. One example of such a network is placed on a rock glacier in the Matter valley. The goal of these sensors is to gather vibration and GPS data to predict rockfalls or rock waste avalanches. With this predictions precaution measures can be taken to protect people and infrastructures in the valley.

1.2 DPP LoRa

The new communication platform features the STM32L433CCU6 microcontroller from STMicroelectronics [6] and Semtech's SX1262 LoRa transceiver [3].

1.2.1 SX1262

The SX1262 transceiver offers the possibility to easily switch between FSK and LoRa modulations on a single chip. It has a receiver sensitivity of down to -125dBm for FSK and -148dBm for LoRa. The current consumption in receiving

mode is only 4.2 - 5.3mA. The transmit power can be chosen in the range of -9 - 22dBm, resulting in a current consumption of 23 - 120mA.

Whereas FSK offers higher data rates, LoRa can be used for communication over longer distances. LoRa also has the possibility to choose between different spreading factors (SF). Higher spreading factors allow for even further transmissions with the drawback of a longer time on air. More details on LoRa are given in [7]. A simple overview which shows the tradeoff between range and time on air can be seen in Fig. 1.1. For example the time on air for 8bit is 640µs for FSK with a data rate of 200 kbit/s, 13ms for LoRa with the lowest spreading factor (SF 5) and can go up to 1s for LoRa with the highest spreading factor (SF 12).

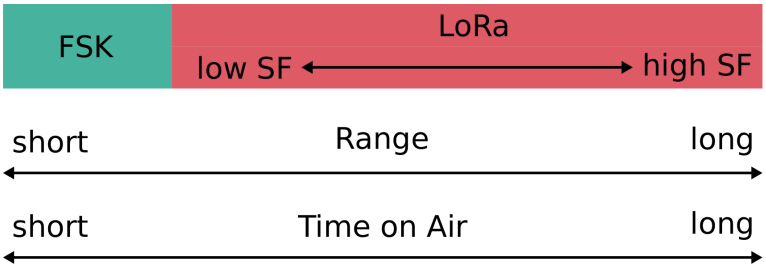


Figure 1.1: Time on air and range comparison of FSK and LoRa. The energy needed for a transmission is proportional to the time on air.

1.3 Flora

Flora was also developed in the previous thesis [2]. It is a collection of software written to simplify the use of the new platform. It for example offers drivers for the control of the radio and a hardware timer. Additionally it features a command line interface to communicate with the platform over a serial interface. This allows to start and control measurements externally.

The radio drivers interrupt routine was slightly adapted, to propagate more information to higher layers of the protocol stack. For example the `RxDone` interrupt was extended with a parameter stating if the received messages CRC code was correct or not. Also the timer module was extended and in all the drivers some bugs were detected and fixed. For more details see Section 5.2.

1.4 Gloria

A first version of Gloria was also already implemented before this thesis. However it has only been tested in a table top setup with a couple of nodes. During this thesis Gloria has been thoroughly tested on Flocklab [8], which offers the

possibility to use up to 27 nodes. In addition to the bugs that were fixed, the implementation was improved in terms of stability and performance and enhanced with some additional features. For more details on the concepts of Gloria see Chapter 3, for information regarding the implementation Section 5.3.

1.5 Goals

The goals of this thesis were to test Gloria on a larger scale and implement a simple version of LWB [5] with it. LWB should be extended to make use of the possibility to easily switch between the FSK and LoRa modulations. This extension should make it possible to connect nodes to the network which are too far away for the basic LWB protocol. These nodes should be included into the LWB schedule with as little overhead as possible. Also the nodes in the network should remain statless. Meaning they do not know anything about the network topology. More details on the given scenario and the implementation of LWB and the extension can be found in Chapter 4 and Section 5.4.

Chapter 2

Background

2.1 Flocklab

Flocklab is a testbed developed and maintained by ETH [8]. It features 27 nodes in close proximity to each other and since recently also a few nodes further away. The Flocklab web interface allows to schedule tests without having to be physically at the test facility. It is possible to track the serial output and measure the power consumption of the nodes. Furthermore 7 GPIO pins can be used to generated interrupts or to observe the activity of the nodes.

2.2 Glossy

Glossy [4] is a network flooding and time synchronization protocol. The goal of network flooding is to distribute a message to all nodes in a multi-hop network as fast as possible, without having to know anything about the network topology. The basic idea of Glossy is that all nodes which receive the message retransmit it at the exact same time. This is achieved by a deterministic software delay between the reception and the transmission of the message. Due to the capture effect the messages can be received correctly, even if multiple nodes send simultaneously. The reliability of Glossy can be increased by listening and retransmitting multiple times instead of going to sleep after the first received and relayed message.

Additionally to the fast and reliable network flooding, Glossy also allows the nodes to synchronize to the initiators time. The flood start can be reconstructed from the radio interrupts and the slot lengths. If the initiator includes his flood start time into the message the nodes can also synchronize absolute to the initiators time. This means that they can compensate the offset between their local time and the initiators time, so that all nodes have the same time reference.

Fig. 2.1 shows an example of a Glossy flood where each node retransmits the message three times.

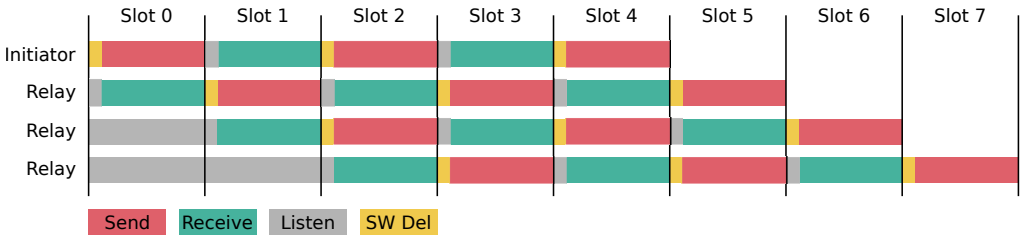


Figure 2.1: Example of a Glossy flood.

2.3 LWB

The Low-Power Wireless Bus (LWB) [5] is a protocol which exclusively uses Glossy floods to handle the communication between the nodes. The host node schedules the communication into different rounds. Each round consists of multiple slots. The time between the rounds is calculated based on the traffic demand in the network. The different slots per round can be seen in Fig. 2.2.

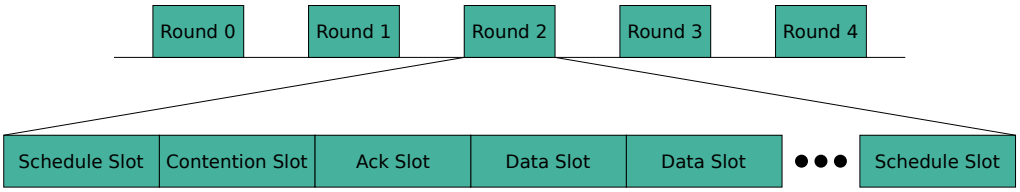


Figure 2.2: LWB round breakdown.

The schedule slot is used by the sink to send a schedule to all other nodes. This includes the round period and the information when a node is allowed to send. In the contention slot nodes can try to request a data stream from the sink. As multiple nodes may want a stream, collisions can occur. A stream is characterized by the amount of data per time a node wants to send. If a stream request reached the sink it gets acknowledged in the ack slot and the sink starts to allocate data slots for this node. Nodes that do not get an acknowledgment back off from the contention slot to reduce contention. A node that has an allocated data slot in a round, may send its data in this slot. The number of data slots and which nodes may send in which slot can vary from round to round and is defined by the schedule. The schedule slot at the end of the round can be used to adapt the round period according to new traffic demands requested in the current round.

2.4 Related Work

Since Glossy [4] showed that concurrent transmissions can be used for very efficient and fast network flooding, multiple protocols were developed based on concurrent transmissions. Chaos [9] uses the flooding mechanism of Glossy but each node adds its own data to the message. Syncast [10] is optimized in terms of energy efficiency and tries to make synchronous transmission of longer packets more reliable.

None of these protocols however made use of the LoRa modulation. But [11] shows that also LoRa is a possible alternative for network flooding protocols based on concurrent transmission. They also present an alternative which schedules the retransmissions with a slight offset to increase the reliability.

With all these protocols there are many opportunities for extensions. LWB [5] or Crystal [12] for example make use of Glossy floods to abstract the complexity of the multi-hop network. Baloo [13] introduces a middle layer to separate the flooding part of the protocols from the higher layer part which makes use of the flooding. This allows to implement multiple flooding mechanisms and switch between them on a higher layer for comparison or even usage in the same scenario.

Chapter 3

Gloria

Gloria is a flooding and time synchronization protocol based on Glossy (Section 2.2). All nodes that receive a message, broadcast it at the same time. This allows to achieve very fast and efficient coverage of a multi-hop network, without having to know anything about the topology. It also means that the intermediate nodes have no state. The concurrent transmissions can be correctly decoded due to the capture effect.

The basic concepts of Gloria have been developed and implemented in a previous thesis [2]. But the code has not been tested on a larger scale. During this thesis the existing code has been extensively tested on Flocklab (Section 2.1). The tests revealed a couple of major bugs only visible in larger multi-hop networks. For example, a wrong radio configuration after wakeup decreased the actual transmit power which lead to a reduced range. In addition to the bug fixes, the protocol was made more robust and highly configurable for the use in many different scenarios. At last Gloria has been decoupled from any other code except the radio drivers and the timer. This makes it an independent building block which can be used to build higher layer protocols as for example LWB on top of it.

3.1 Gloria Terminology

Table 3.1 shows a list of terms used during this chapter. Which parameters can be set to configure a flood can be seen in Section 3.3.

Term	Description
Ack	Acknowledgment message. Contains no information except the destination for the ack.
Ack flood	A flood with the possibility to send acks. (see Section 3.2)
Ack/Data (sub)slot	A time slot that is used to send the ack/data message. It is part of the general slot described below.
Data message	The message containing the data that should be transmitted with the flood.
Destination	The node for which the message is intended. Does not relay the message.
Initiator	The node that starts the flood.
Normal flood	A flood without any acks. (see Section 3.2)
Maximum number of acks	The maximum number of acks is a configurable parameter and defines how many acks a node should send. It corresponds to the retransmissions for data messages.
Relay	A node that does not start the flood and is not the destination of the flood. It just relays the message.
Retransmissions	The number of retransmissions defines how many times a node should send out a received message. It can be set at flood start.
Slot	A flood is divided into equal parts that are called slots. They can consist of an ack and a data or just a data subslot.
Slot limit	The slot limit specifies how many slots are contained in a flood. It is used to set a duration limit to the flood.
Sync flood	A flood with the flood start in the initiator's time appended to the message payload. No separate flood but an extension to the normal or ack flood.

Table 3.1: Terminology of Gloria.

3.2 Concepts

Figure 3.1: Gloria slot layout.

Gloria has two different layouts. One for normal floods and one that allows to send an acknowledgment interleaved in the same flood. The latter is referred to as ack flood. The floods are separated into different slots as shown in Fig. 3.1. In contrast to normal floods, the slots in ack floods contain a data and an ack subslot.

Details on the different floods are described in the following sections.

3.2.1 Normal Flood

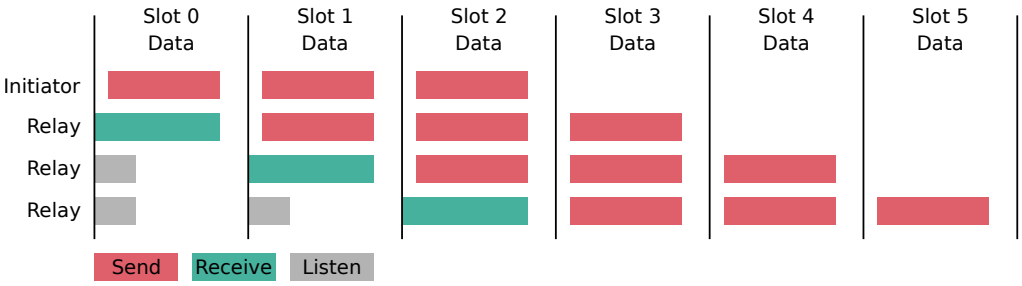


Figure 3.2: Broadcast Gloria flood with 3 retransmissions.

A normal Gloria flood is the one that is the most similar to Glossy. It is separated into slots which are only long enough to send the data message. A node that received the data message retransmits it starting in the next slot. In contrast to Glossy, the node does not listen for messages anymore once one was received. In Glossy the retransmissions are timed by an exact software delay after the reception. Thus Glossy needs to receive the message for every retransmission. For Gloria, the retransmissions are all scheduled with a timer relative to the first reception. This allows it to send in consecutive slots, which reduces the number of slots needed. As the delay between reception and transmission in Glossy is highly optimized, it is shorter compared to Gloria. This means that less slots do not necessarily translate to shorter flood lengths.

As an example, Fig. 3.2 shows a normal flood with three retransmissions. The initiator sends the data message three times and finishes the flood. All other

nodes listen for the message and retransmit it also three times. In this case the flood is a broadcast and all nodes should receive it. Therefore also all nodes retransmit it. If the flood is no broadcast, the destination node does not retransmit the message. The maximum flood duration can be configured by setting the slot limit.

3.2.2 Ack Floods

In ack floods the slots are divided into two subslots. One subslot for the data transmission and one for the transmission of acks. Because the acks are very short data subslots are usually longer. The length of the flood can also be limited by setting the slot limit. But as the slots are longer, the whole flood will be longer for the same slot limit compared to normal floods.

Ack floods are only useful if the message has a destination. For broadcasts they need more time and energy compared to the normal flood, as the slots are extended by the data subslots and the nodes listen for acks. But no ack will be sent as the flood has no destination.

There are two different ack modes. They have the same slot layout but follow a different scheme.

Ack mode 1

The goal of ack mode 1 is to save energy compared to the normal flood. As it is not unusual that in this mode the ack does not propagate back to the initiator, it cannot be used to acknowledge data messages reliably. The concepts of this mode are explained with the help of an example shown in Fig. 3.3. The maximum number of retransmissions is set to 3 and the maximum number of acks to 2. The slot limit is higher than the number of slots needed.

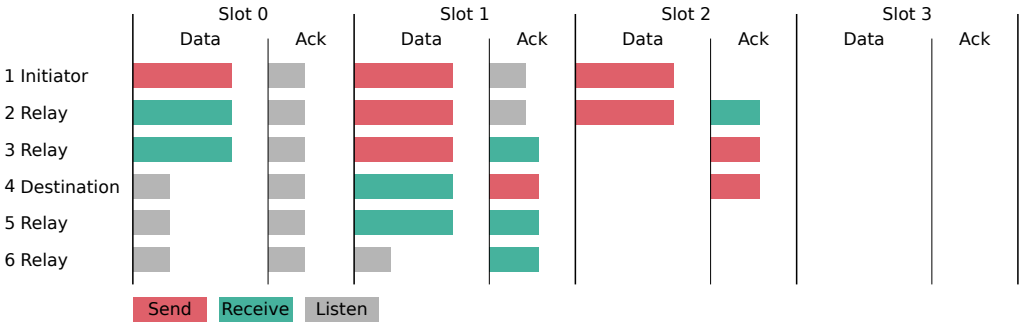


Figure 3.3: Flood with ack mode 1, 3 retransmissions and the maximum number of acks set to 2. All nodes are finished with the flood after slot 2.

When the destination (node 4) receives the message it does not retransmit it,

but starts sending acks in the next ack slot. Any node that has already sent the data message at least once and receives the ack, stops sending the data message and retransmits the ack (node 2 and 3). Nodes that have not received any data message or have not yet retransmitted it and receive an ack, stop participating in the flood and set the radio into sleep mode (node 4 and 5). As the acks are very short, a node needs less energy to send an ack than a data message. Also the nodes which do not send anything at all clearly save some energy compared to the normal flood, in which they would retransmit the data message.

The maximum number of acks that should be sent can be set independent of the maximum number of data retransmissions. It makes sense to set the maximum number of acks lower than the maximum of data retransmissions, because a node that sends too many acks has again an increased energy consumption.

To prevent nodes that have already sent almost all data messages from also sending a lot of acks and using more energy than in a normal flood, acks are only retransmitted in slots before the last active slot. The last active slot is the slot the last data message would be sent. It is calculated based on the slot the data message was received (`first_rx_index`) and the number of data retransmissions (`max_retransmissions`). The `first_rx_index` for the initiator is set to -1.

$$\text{last\_active\_slot} = \text{first\_rx\_index} + \text{max\_retransmissions} \tag{3.1}$$

The last active slot of node 2 is slot 3 (0 + 3) and the first slot it could send an ack is also slot 3. As acks are only retransmitted in slots before the last active slot, node 2 does not send any ack. The last active slot of the initiator is slot 2 (-1 + 3). That is why it does not listen for acks in slot 2 and 3.

The examples above show that although the maximum number of acks is a configurable parameter, the actual number a node can send is also limited by the number of data retransmissions as shown in Table 3.2.

destination	max_retransmissions
relay	max_retransmissions - 2
initiator	max_retransmissions - 2

Table 3.2: Upper bound on the number of acks a node can send based on the maximum number of data retransmissions.

The idea behind this was that all nodes that received the message before a certain node, are already finished with the flood before the node itself. To send an ack back after all data retransmissions are over would increase the energy consumption on the node and the other finished nodes would also not profit, as they already sent the data message the maximum number of times.

Ack mode 2

As opposed to mode 1 the idea of this mode is that the ack reaches the initiator of the flood. This means that a data message can be directly acknowledged in the same flood and there is no need for an extra flood to achieve this. As with ack mode 1 the concepts are explained with an example in Fig. 3.4. The maximum number of retransmissions and the maximum number of acks is set to 3.

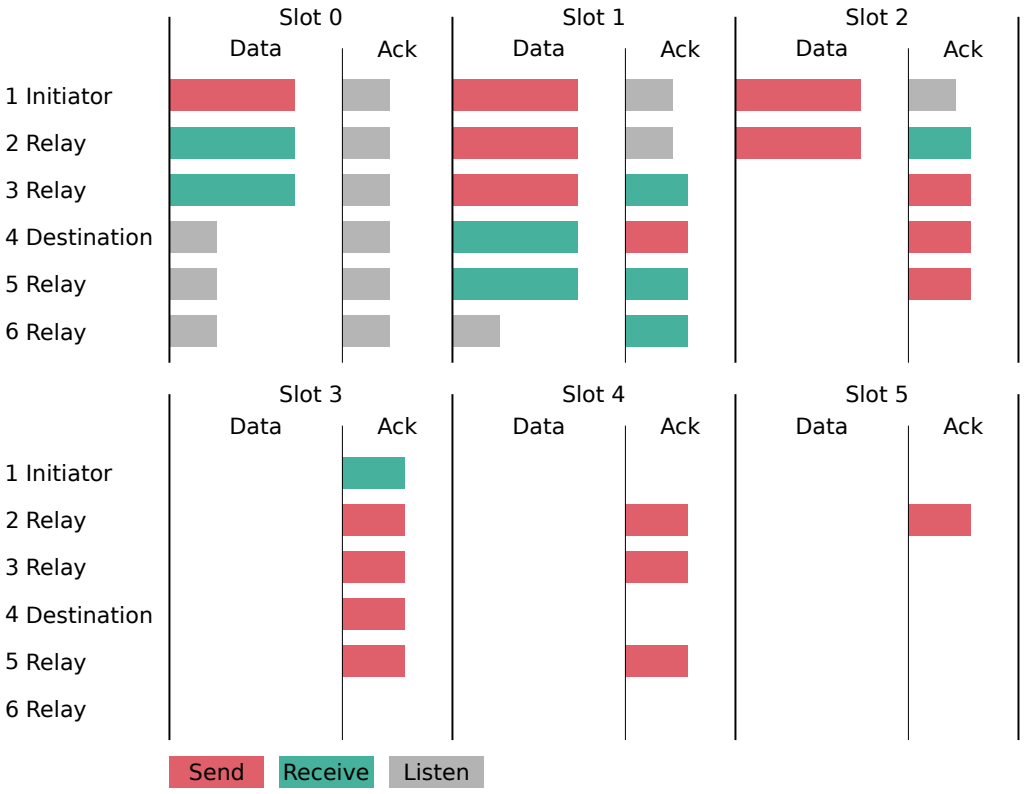


Figure 3.4: Flood with ack mode 2, 3 retransmissions and the maximum number of acks set to 3. All nodes are finished with the flood at the end of slot 5.

The main difference to mode 1 is that nodes wait for an ack, even if they are finished sending the data message (node 1). They listen during the ack slots until an ack is received or the slot limit is reached. The ack is retransmitted even if the node has not sent any data messages upon ack receive (node 5). Nodes that receive an ack without having received a data message stop participating in the flood and set the radio into sleep mode (node 6).

In this mode the maximum number of acks a node can send is not limited by the maximum number of data retransmissions. A node keeps sending acks until it has sent the maximum number of acks or has reached the slot limit.

3.3 Flood Configuration

This section describes the different parameters that can be set before starting a flood. For more details on how, the data types of the parameters and the possible values see Section 5.3.1.

3.3.1 General Gloria Parameters

This are the parameters that should be set for every flood, independent if the node is the initiator or not.

Ack Mode

Allows to select one of the two ack modes or the normal flood. More details on the different modes can be found in Section 3.2.2.

Marker

A timestamp used to define the flood start. If the marker is in the past the initiator does not send. All other nodes start to listen anyhow, when low power listening (see Section 3.3.3) is disabled.

Maximum Number of Acks

Similar to the maximum number of retransmissions (see Section 3.3.1) this is the maximum number of acks that should be sent. This parameter is only relevant for ack floods.

Maximum Number of Retransmissions

This parameter allows to set the number of retransmissions for the data message. It is an upper bound as further retransmissions can be prevented if an ack is received or the flood has reached the slot limit. More retransmissions mean more redundancy and increase the probability that a flood is successful.

Slot Limit

The number of slots for the corresponding flood can be limited by setting the slot limit. This parameter is used to control the maximum flood duration. More slots usually translate to a higher probability that a message reaches its destination.

But the probability also saturates for high slot counts as the flood dies out if the nodes have sent the message the predefined number of times.

Sync Flood

For sync floods the timestamp of the flood start in the initiator's time is appended to the payload. It can be used by all receiving nodes for absolute synchronization to the initiators time. For low power listening (see Section 3.3.3) also the receiving nodes need to know if the flood is a sync flood. For more details on the flood start see Section 3.5.1.

3.3.2 Gloria Tx Parameters

The important parameters for the initiator are explained in this section.

Destination

If the flood should not be a broadcast, the node ID of the destination can be defined. If it is zero the flood has no destination and is a broadcast.

Message Type

The message type parameter is not used during the flood run, but is meant to identify the payload on a higher layer protocol.

3.3.3 Gloria Rx Parameters

In this section the parameters only needed for the receiving nodes are explained.

Guard Time

This parameter allows to set a guard time for the reception. It will be appended to the beginning and the end of the listening period defined by the rx timeout (see Section 3.3.3). If low power listening (see Section 3.3.3) is enabled, the guard time is added at the beginning and the end of each listening slot. If the listening slots get too long, so that they overlap or the time between the slots is to small to go to standby and receive mode again, the node will listen continuously.

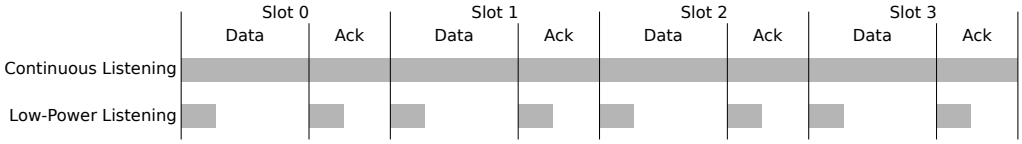


Figure 3.5: Comparison of low power listening and the normal continuous listening.

Low Power Listening

Low power listening allows a node to save energy by listening only a short interval for each slot instead of the whole time. This only works if the receiving node knows the exact length of the message. Else it could not calculate the slot lengths. With low power listening enabled the nodes do not listen for the predefined rx timeout (see Section 3.3.3). Instead they listen for a short period in every slot until the slot limit is reached. Fig. 3.5 shows the difference of the listening periods with and without low power listening.

Rx Timeout

The duration for the initial listening period is defined as rx timeout. If it is set to zero the radio listens until a package is received successfully. The radio continues listening if it receives a message with a CRC error and the timeout has not yet expired. The timeouts to listen for acks or for low power listening are calculated during the flood. This parameter is not used if low power listening is enabled.

Sync Timer

Gloria has a built in option to adapt the timer offset for absolute synchronization to the initiator's time. The offset is then calculated and adapted during the flood. The adaption is only possible for sync floods, as it needs the information from the appended timestamp. If the timer synchronization should include drift, the calculation and adaption of offset and drift need to be done on a higher layer protocol as for example in Section 3.5.2.

3.4 Finished Flood

This section describes which information is provided by Gloria, after a flood is finished. More details about the implementation can be found in Section 5.3.1.

Ack Destination

The ack destination tells the nodes for which one the ack was intended. The destination node of the data message adds the node ID of the initiator to the ack, before sending it. This can for example be important for contention periods, where multiple nodes start a flood with the same destination. The destination node then acknowledges the first received flood.

Source

The flood also contains the node ID of the initiator.

Received Marker

The received marker is the timestamp that was appended to the payload in a sync flood. It contains the flood start in the initiator's time (see Section 3.5.1).

Reconstructed Marker

As a message is received the actual start of the flood is reconstructed in the local time. This reconstruction is based on the header / Sync Word valid interrupts from the radio (see Section 3.5.1).

3.5 Gloria Timing

3.5.1 Flood Start

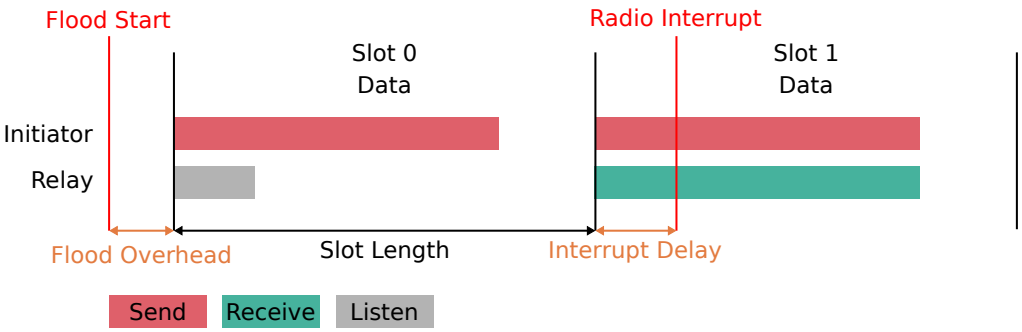


Figure 3.6: Flood timing.

Fig. 3.6 shows the timing values needed to reconstruct the flood start on message reception. The radio interrupt is triggered by a correct reception of the header for LoRa and the Sync Word for FSK. The interrupt delay has been measured

and found to only depend on the modulation setting. This means it is different for LoRa and FSK but also for different data rates (FSK) or spreading factors (LoRa). The slot length can be calculated from the message size as explained in Section 3.5.3. The flood overhead is a constant for each modulation setting. It is needed to make sure a node has enough time to initialize the flood before starting it. Because of this, the flood start is not equal to the first transmission. The timestamp of the flood start in the initiator's time is the one that gets transmitted in sync floods. It is saved as received marker. Based on the parameters explained above the flood start can be reconstructed in the local time on the other nodes. The reconstructed flood start is saved as reconstructed marker.

3.5.2 Synchronization

For absolute synchronization to the initiator, the reconstructed and the received marker can be used. Absolute synchronization means that the timers are set to the same value. The difference between the two markers is the offset the local timer has in comparison to the timer at the initiator. This is only possible if the flood is a sync flood, else no timestamp is appended to the flood. Absolute synchronization can be important if for example sensor data should be labeled with a timestamp. The timestamps of the data collection then are all in the same time reference.

If relative synchronization is enough, it is not needed to send sync floods. In contrast to the absolute synchronization timer values are not the same, but the nodes can still schedule the next communication relative to the last one. An approximation of the synchronization error can be calculated by taking the difference from the expected flood start and the reconstructed marker.

3.5.3 Flood Duration

As a flood is separated into different slots, the maximum duration of a flood is given by the length of the slots and the slot limit. The slot limit is a parameter that can be configured and therefore allows to set a limit on the flood duration.

The data subslot length is determined by the chosen modulation and the payload size. For sync floods the payload is increased by the timestamp size. Ack subslots are only dependent on the modulation, as the ack message is always of the same size. The total slot length is the addition of the ack and data subslot lengths. The flood length can then be calculated by the number of slots and an additional overhead at the beginning. The overhead is used to initialize the flood and make sure the nodes have enough time to go into rx or tx mode.

The calculations of the data and ack subslot and the total slot lengths is shown below.

$$\text{data\_slot\_length} = \text{data\_time\_on\_air} + \text{data\_slot\_overhead} \tag{3.2}$$

$$\text{ack\_slot\_length} = \text{ack\_time\_on\_air} + \text{ack\_slot\_overhead} \tag{3.3}$$

$$\text{slot\_length} = \text{data\_slot\_length} + \text{ack\_mode} * \text{ack\_slot\_length} \tag{3.4}$$

Where the `ack_mode` parameter represents a bool to differentiate between normal and ack floods.

The duration of the whole flood can then be calculated as follows.

$$\text{flood\_duration} = \text{flood\_init\_overhead} + \text{slot\_limit} * \text{slot\_length} \tag{3.5}$$

More details on the slot overheads and the initial flood overhead can be found in Section 5.3.3.

3.6 Ack Mode Comparison

This section compares the two ack modes described in Section 3.2.2. First three different examples are discussed and in the second part an estimate on the energy consumption is given.

3.6.1 Examples

Figs. 3.7 to 3.9 show the same setup with a slightly different evolution of the flood. The goal is to acknowledge the message reception. This means that for ack mode 1 two floods are needed, whereas for ack mode 2 it can be done in one flood. The dark gray part between the floods for ack mode 1 marks the time overhead needed to start the second flood. The initiator and the destination are separated by two hops. All floods are sent with two retransmissions, one ack for ack mode 1 and two acks for ack mode 2. The slot limit is set to three to have one slot more as hops.

The data slots of the second flood for ack mode 1 are shorter as the ones for ack mode 2, as they contain only the information for an acknowledgment. In contrast one less slot is needed for ack mode 2 than for the two floods in ack mode 1 and the overhead for starting the second flood is zero. This results in less total time needed with ack mode 2 for short floods. But as the floods get longer (more hops) the shorter data slots get more important and eventually ack mode 1 needs less time. The data message length also plays a role, as the difference between the data slot lengths depends on the difference between the data message and the acknowledgment message length, which is sent in the data slots of the second flood.

In the first example in Fig. 3.7 the message always is successfully received at the next hop. The first two slots are almost identical for both modes. The initiator listens once more for ack mode 2. To propagate the ack back to the

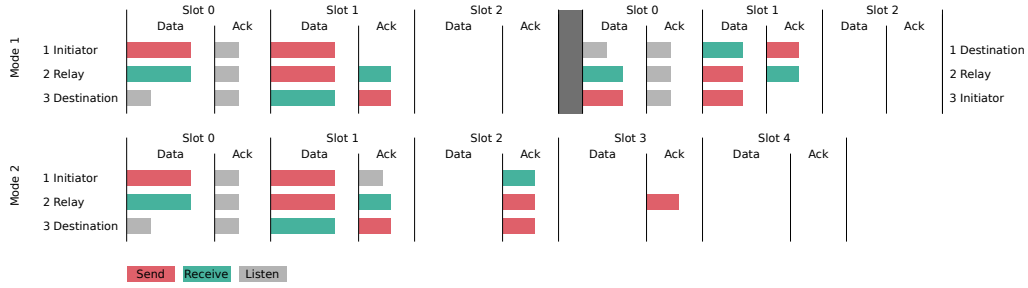


Figure 3.7: Ack mode comparison example 1.

initiator ack mode 2 needs three retransmissions of an ack. Ack mode 1 has four retransmissions from which 3 are longer than an ack. Also more time is spent on listening in the second flood for ack mode 1 than in slots 2, 3 for ack mode 2. In the end ack mode 2 needs less energy as ack mode 1. The total time needed is for this short example also less with ack mode 2.

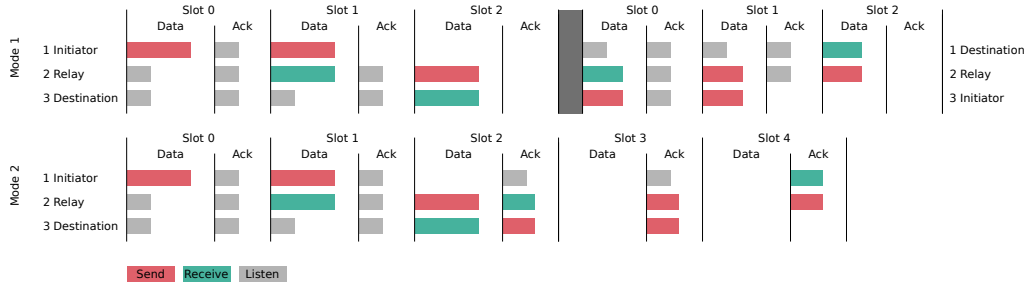


Figure 3.8: Ack mode comparison example 2.

Fig. 3.8 shows the same example but this time the data message is note correctly received at node 2 in the first slot. Also the ack gets lost once. Until the data subslot of slot 2 the two modes are again identical except for one listening period. For mode 1 the nodes do not listen or send in the last ack subslot. This time the number of retransmissions after the data arrived is the same. But they are longer for ack mode 1. Also ack mode 1 has more listening periods than ack mode 2. Again ack mode 2 is more energy efficient.

The third example in Fig. 3.9 shows the case, where the data message does not arrive at the destination in the first three slots and for ack mode 2 the ack does not arrive at all. For ack mode 1 the second flood is not initiated. Nodes 1 and 2 need some energy for listening, but nothing is transmitted. There are three more retransmissions for ack mode 2. As sending is far more expensive than receiving, ack mode 1 uses less energy in this case. And as the ack did not reach the initiator for mode 2, the whole process has to be repeated for both modes.

As long as the floods are expected to be successful ack mode 2 needs less energy. But it can take up more time, if the data message length and the number of hops

Figure 3.9: Ack mode comparison example 3.

increase. When there is a lot of interference and the floods are likely to fail, ack mode 2 can waste some energy for the partial back propagation of the ack.

3.6.2 Energy Comparison

To compare the energy consumption for the different ack modes a measurement was performed. The settings are shown in Table 3.3. One of the 26 nodes was selected as destination for all floods. The other 25 nodes alternating sent one flood with each ack mode. This resulted in 60 floods per node and ack mode.

Nodes	26
Floods	1500 per setting
Retransmissions	3
Acks	3
Powers	22dBm / 0dBm
Modulations	FSK 200kbit / LoRa SF5
Payload	12B

Table 3.3: Measurement setup parameters.

Figs. 3.10 and 3.11 show the average energy consumption per flood. Whereas the analysis for Fig. 3.10 was based on GPIO pin toggles, Fig. 3.11 shows the result of an analysis based on performance counters. Especially for the FSK modulation the two results differ significantly. The GPIO pins are set a little longer than the receptions or transmissions actually last. This leads to longer listening and sending times for the calculations with the GPIO pins as for the performance counters. But also the performance counters are not exact as the current consumption is assumed to be of perfect rectangular shape and the ramping is disregarded. This means these results are only a rough estimate. For LoRa these effects are not so relevant as the overhead for the ramping or the pin toggles is smaller compared to the time on air. The measurement shows that the energy used for ack mode 1 is definitely the lowest of the three modes. Compared to ack mode 0 it uses around 55% of the energy for LoRa and around 69% for FSK.

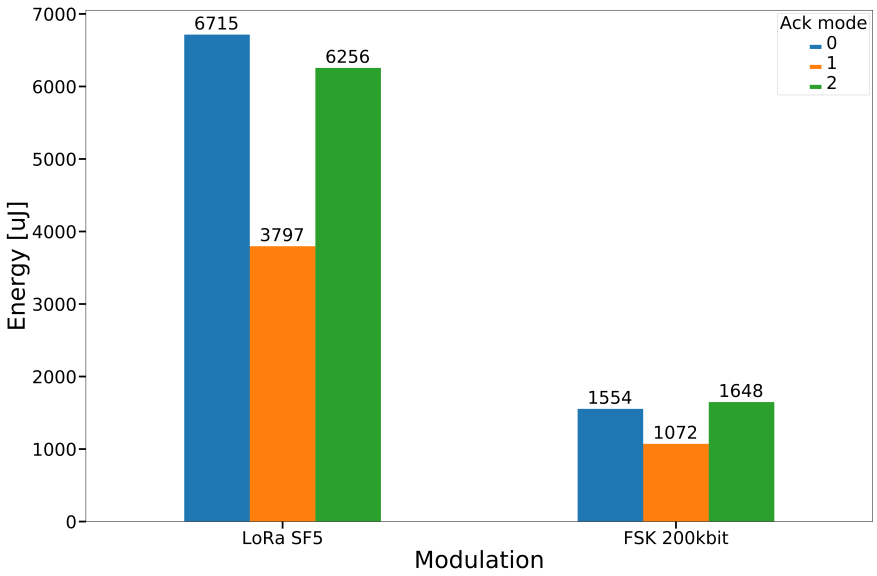


Figure 3.10: Average energy consumption per flood. Calculated based on the GPIO pins.

For this setup even the average energy for ack mode 2 is lower than without ack. Concerning that ack mode 2 already achieved that an acknowledgment reached the initiator this is even better. To get the same result with the other two modes two floods would be needed.

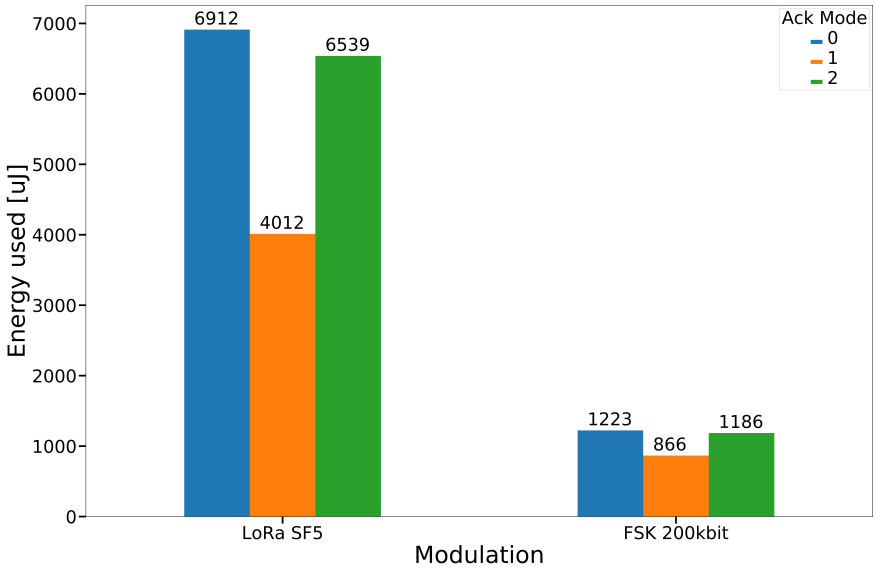


Figure 3.11: Average energy consumption per flood. Calculated based on the performance counters.

3.7 Characterization and Evaluation

This section describes the results of a comparison of FSK and LoRa. The first two subsections show the results of a larger measurement. The reliability is compared for different transmit powers in Section 3.7.1. The synchronization precision is shown in Section 3.7.2.

Section 3.7.3 gives a comparison to Glossy and the Section 3.7.4 shows some more measurements where a selected parameter was sweeped.

3.7.1 Reliability

The reliability is measured as the percentage of nodes which successfully received the flood.

Fig. 3.12 shows the nodes that participated in the measurement. The initiator was node 1 in the upper left corner. The parameters for the measurement are listed in Table 3.4. The 3000 floods are per power and modulation setting.

Fig. 3.13 shows the reliability of the floods for the different power settings. For LoRa the reliability does not significantly change as for 0dBm half of the nodes is already reached within one hop. Meaning that the network has at most two

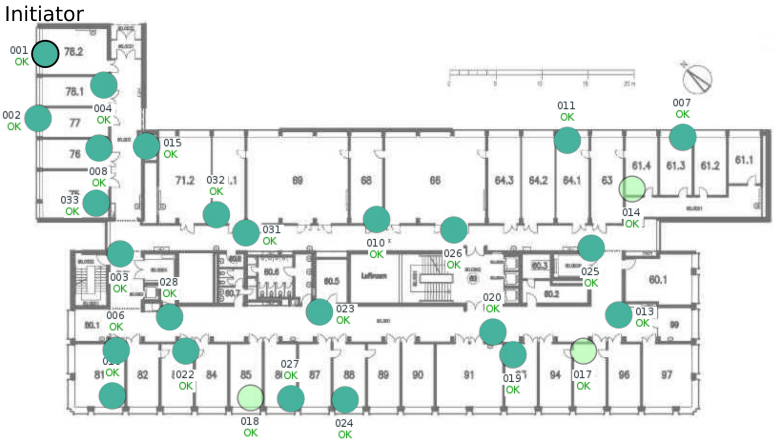


Figure 3.12: Measurement setup.

Nodes	24
Floods	3000 per setting
Retransmissions	3
Powers [dBm]	0, 10, 22
Modulations	FSK 200kbit, LoRa SF5

Table 3.4: Measurement setup parameters.

hops. For FSK the reliability is better with higher power settings. For lower power settings the link quality for the single hops degrades and the messages may not reach the next node, even with multiple retransmissions.

3.7.2 Synchronization Precision

The synchronization precision is extracted from the same measurement as described in Section 3.7.1. All nodes that received the flood toggled a GPIO pin after a predefined time interval. The GPIO event was logged by Flocklab together with the respective timestamp. The synchronization error was calculated as the difference between the pin toggle of a receiving node and the toggle of the initiator.

The resulting average error can be seen in Fig. 3.14. There is a variation between the nodes which is due to hardware differences on the nodes and the accuracy limit on Flocklab. But the average is below 1.5µs for all nodes.

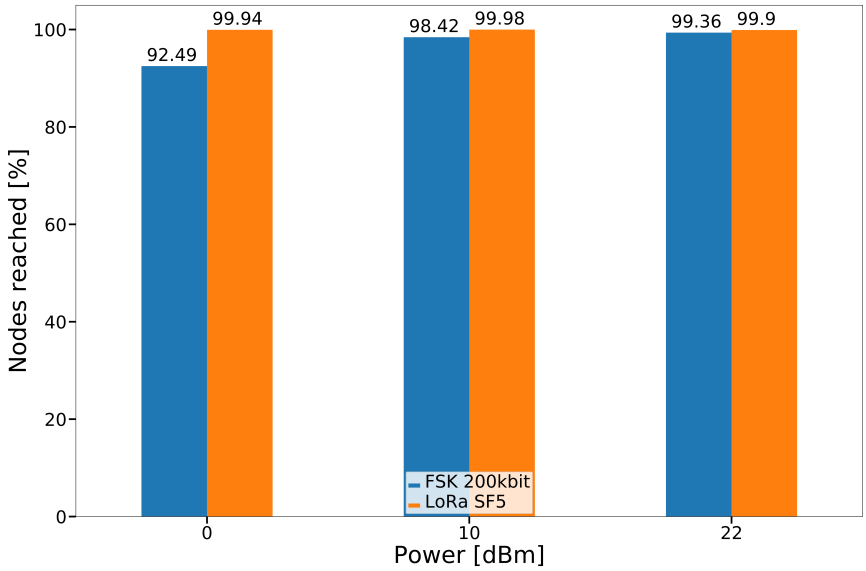


Figure 3.13: Comparison of the reliability for FSK and LoRa with different transmission powers.

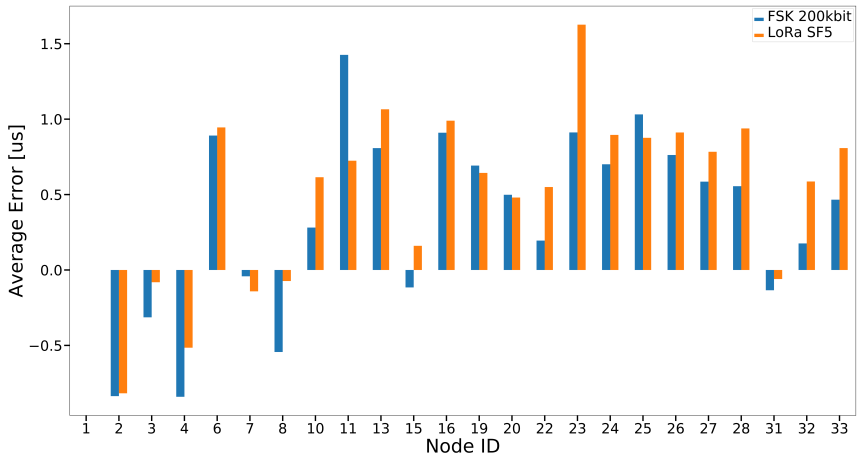


Figure 3.14: Average synchronization error.

The standard deviation (Fig. 3.15) is in the same order as the average error for FSK. For LoRa it is a lot higher. This is because of a broader distribution of the synchronization but also because of more spikes in the measurement. It is not

clear if the spikes are due to variations from the radio interrupt or due to measurement uncertainties on Flocklab. The higher deviation for node 11 and FSK however is due to synchronization problems on Flocklab. Spikes with a similar pattern were observed in other measurements independent of Gloria. For more information about the Flocklab synchronization problems see Appendix A.1.

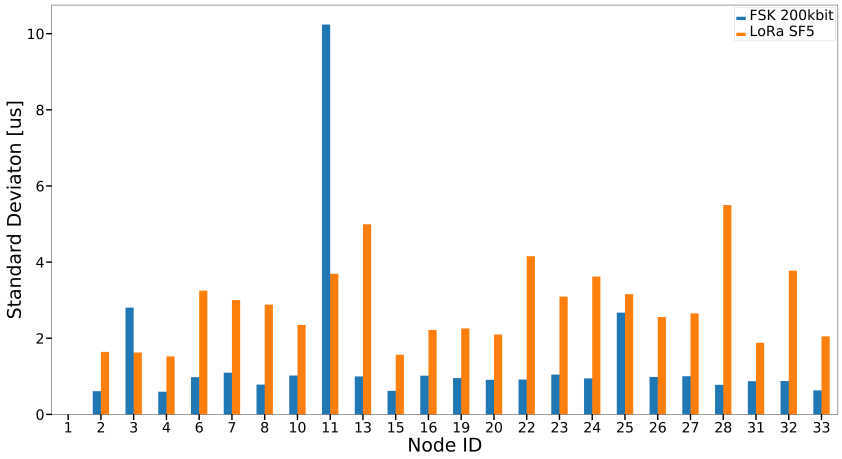


Figure 3.15: Standard deviation of the synchronization error.

Figs. 3.16 and 3.17 show the distribution of the synchronization error for the LoRa and FSK modulation respectively. The y-axis is plotted in a logarithmic scale. For FSK node 11 has been removed as it had synchronization problems introduced by Flocklab. The bins are 10µs wide. The plot for FSK contains 206021 samples, whereas the one for LoRa contains 222585. The difference is due to the missing samples from node 11 an due to the setting that only nodes which received the flood toggle the pin.

The bin with far the most occurrences is the one between -5 and 5 µs for both modulations. The plots show that the distribution is a bit broader for LoRa than FSK but also that there are more spikes.

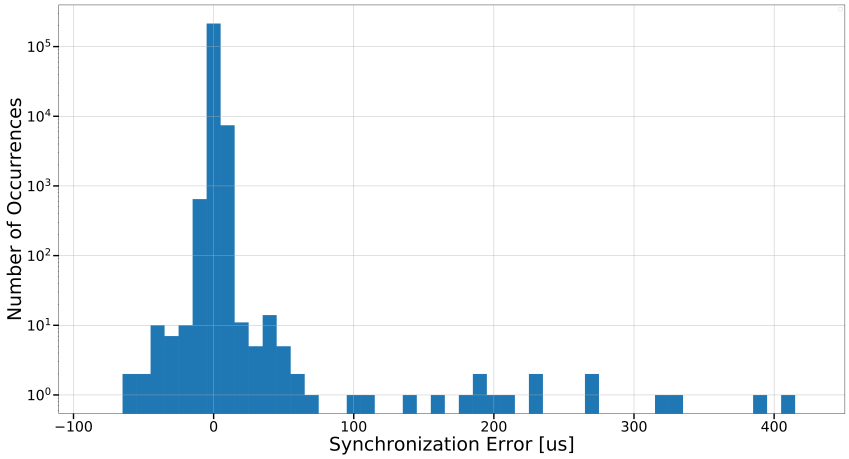


Figure 3.16: Distribution of the synchronization error for LoRa SF5.

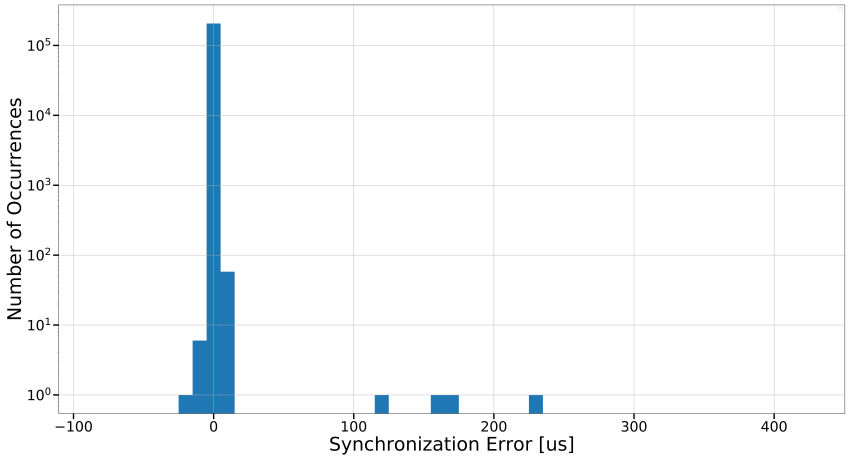


Figure 3.17: Distribution of the synchronization error for FSK 200kbit without node 11.

3.7.3 Comparison to Glossy

To compare the performance of Gloria to Glossy some measurements of Glossy on the CC430 radio were redone as for Gloria. In total six measurements with different retransmissions were performed. The measurement setup was chosen as close as possible to the Glossy one, but is not exactly the same. The nodes used can be seen in Fig. 3.18. The initiator was node 28. For the Glossy mea-

surement node 6 was not available and for Gloria node 14. Also there were 4 more nodes outside the building for Glossy. They have not been considered for the analysis. The parameters for the measurement can be seen in Table 3.5. For each measurement one retransmission was selected and 10000 floods were sent.

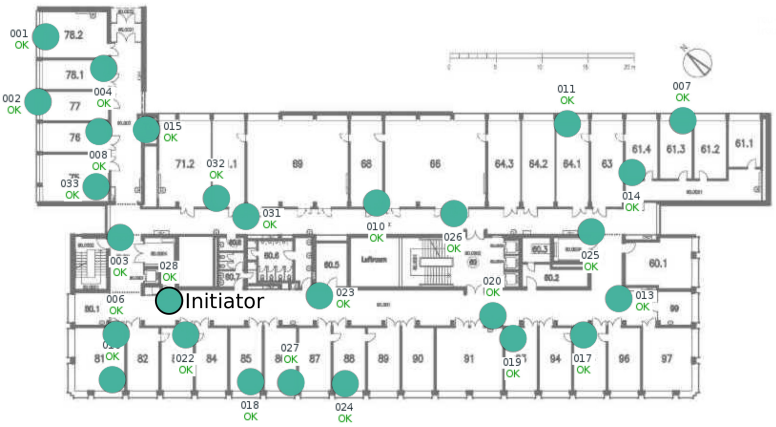


Figure 3.18: Measurement setup.

Nodes	26
Floods	10000 per measurement
Retransmissions	1 - 6
Power	12dBm
Modulation	FSK

Table 3.5: Measurement setup parameters.

Fig. 3.19 shows the average reliability for the different measurements. The reliability for Gloria is up to 1% below the one for Glossy. Especially for higher retransmissions. Also the reliability is lower for higher retransmissions for Gloria. This could be due to external interference, as the different measurements with the different retransmissions were performed after each other. It is not clear what makes the difference between Glossy and Gloria. More measurements with the exact same setup and measurements of the link reliability between the nodes need to be done, to get a better understanding.

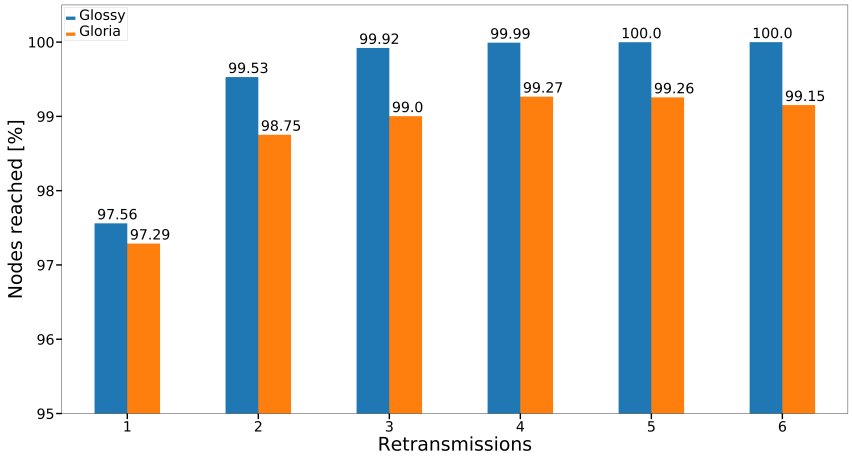


Figure 3.19: Reliability of Glossy and Gloria for different retransmissions.

3.7.4 Parameter Sweeps

This section shows some more measurements where a parameter was sweeped. The measurements have less iterations than the ones above and are thus less meaningful. But they show the general behaviour if a parameter is increased or decreased.

Except for the parameter that is sweeped, the values set are those shown in Table 3.6.

Parameter	Default Value
Slot Limit	32
Retransmissions	3
Power LoRa SF5	0dBm
Power FSK 200kbit	22dBm
Number of Nodes	26
Iterations	100 per setting

Table 3.6: Paramter sweeps setup.

Retransmissions

Fig. 3.20 shows the reliability for different retransmissions. For 1 retransmission it is significantly lower for FSK. After that it still increases but not so fast.

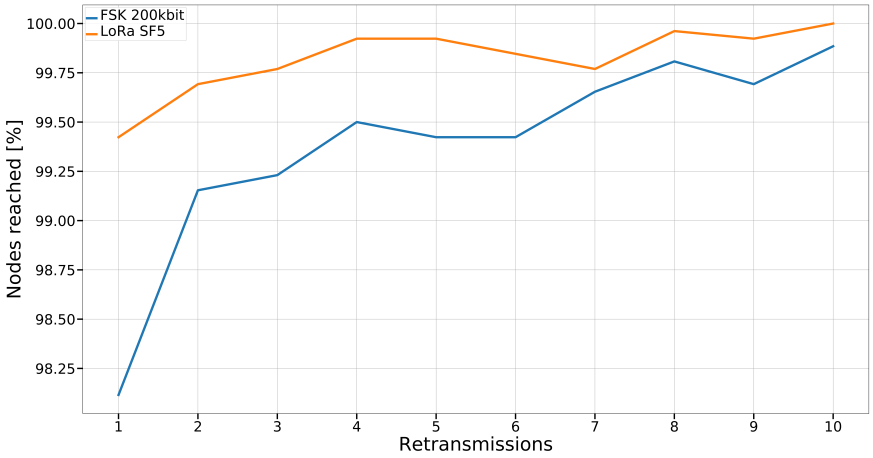


Figure 3.20: Retransmission sweep.

Slot Limit

The reliability saturates at around 8 slots for both modulations. After that it stays constantly high as shown in Fig. 3.21.

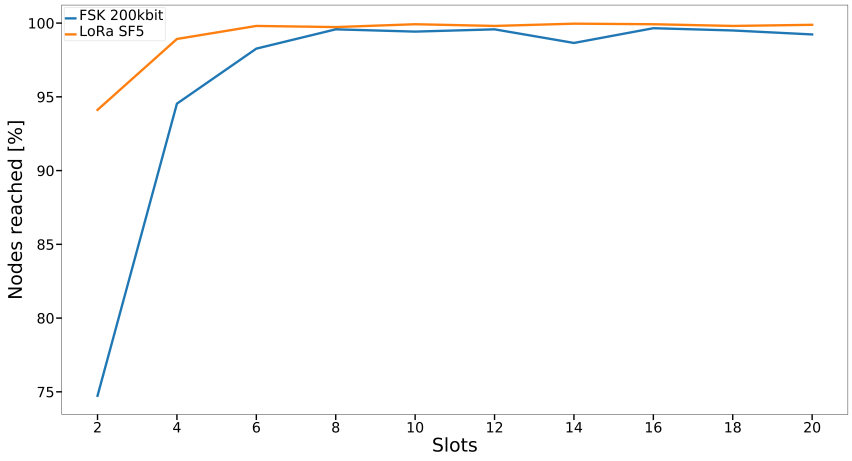


Figure 3.21: Slot sweep.

Powers

The reliability for the power sweep in Fig. 3.22 is worse than shown in Section 3.7.1. It is not clear why this is the case, but it could be due to more interference.

In general it can be seen that the reliability for FSK goes down for decreased transmit powers. For lower powers the link reliability between two nodes gets worse. This also decreases the general reliability. Also the decrease in range for lower powers means that there are not always multiple paths from the initiator to each node.

For LoRa the power does not really matter, as the nodes are close enough to have a good link reliability even for lower powers.

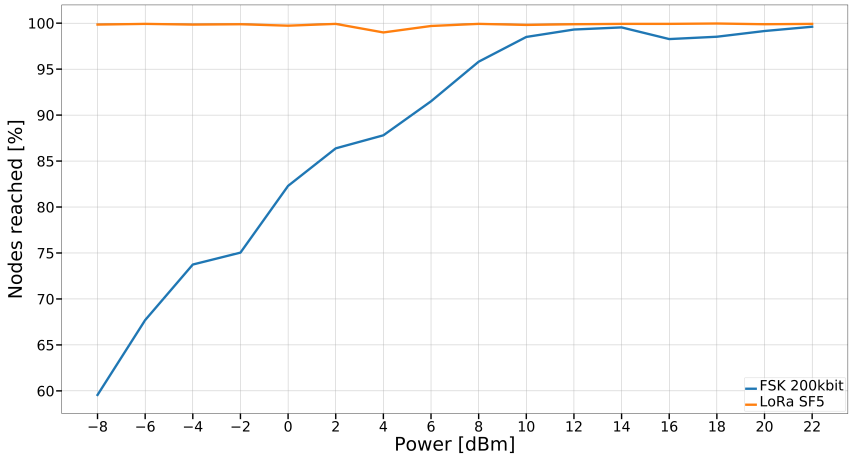


Figure 3.22: Power sweep.

Chapter 4

LWB

For this thesis a simple version of LWB was implemented on the new communication platform [2] with the help of Gloria. Additionally it was extended to make use of the radio's possibility to switch between different modulations with different range.

Section 4.1 describes the problem statement for the LWB implementation. The actual implementation of a simple LWB version together with the extension is explained in Sections 4.2 and 4.3.

4.1 Problem Statement

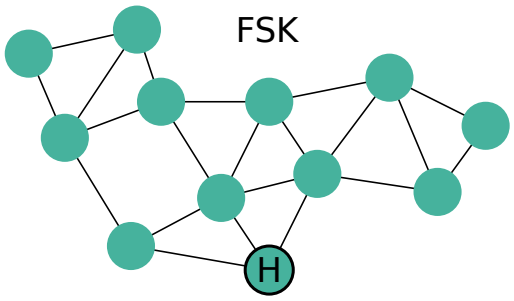


Figure 4.1: Simple LWB network.

For the simple LWB implementation done in this thesis and also the original LWB, all the nodes need to be able to participate in the network floods with the chosen modulation. An example of such a network can be seen in Fig. 4.1. The host node is marked with an "H". In this network all nodes are connected to each other with FSK modulation. It could be any other modulation but for this example it is assumed to be FSK. This means that a flood initiated by any node in the network should reach all other nodes.

The example network is now extended with more nodes that are out of range for FSK (Fig. 4.2). The only possibility to include those into LWB would be

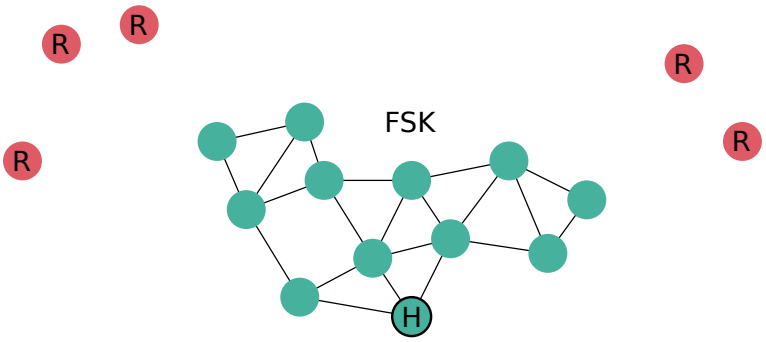


Figure 4.2: LWB network with remote nodes.

to flood the network with a LoRa modulation that can reach those nodes. But LoRa modulations are far more expensive in terms of time and energy than FSK.

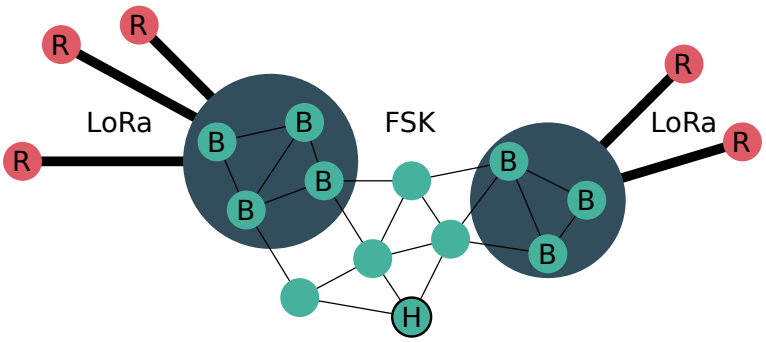


Figure 4.3: LWB network remote nodes and indicated bridge nodes.

The solution to this problem was to define clusters of nodes in the FSK network which can communicate to the remote nodes with a LoRa modulation. A version of such a clustering is shown in Fig. 4.3. These nodes are called bridge nodes and should handle the communication with the remote nodes. The idea was to include the communication from bridge to remote nodes into the LWB rounds in a similar manner as in the already existing LWB implementation. For this thesis it was assumed that the bootstrapping and clustering is already done. This means that the assignment of the nodes and the partitioning of the network into clusters is known at the host.

Actually the problem is not limited to FSK and LoRa modulations. It could also mean that the flooding network is connected by a LoRa modulation with a low spreading factor and the communication to the remote nodes is done with a higher spreading factor. But to keep it simple it is assumed that the flooding is done with FSK and the long range communication to the remote nodes with LoRa.

4.2 Simple LWB

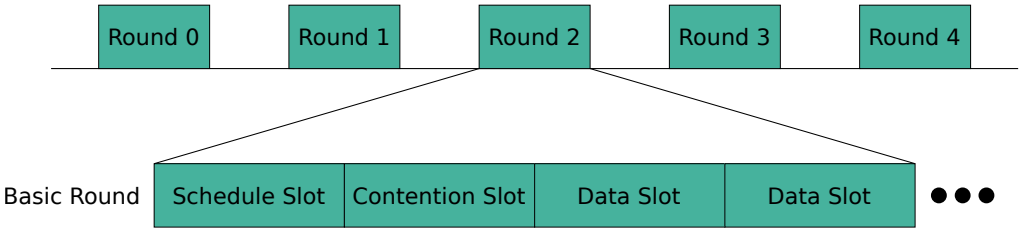


Figure 4.4: LWB overview.

LWB schedules its communication into different rounds. Each of these rounds is divided into slots as shown in Fig. 4.4. The schedule slot is used by the host to send a schedule and a synchronization timestamp to all other nodes. In the contention slot nodes can try to request a data slot. If they get a data slot assigned they can send their data to the host.

4.2.1 Slots

This section describes the different slots and the according messages that are transmitted in more detail. The content of the messages that are sent in the different slots can be seen in Fig. 4.5.

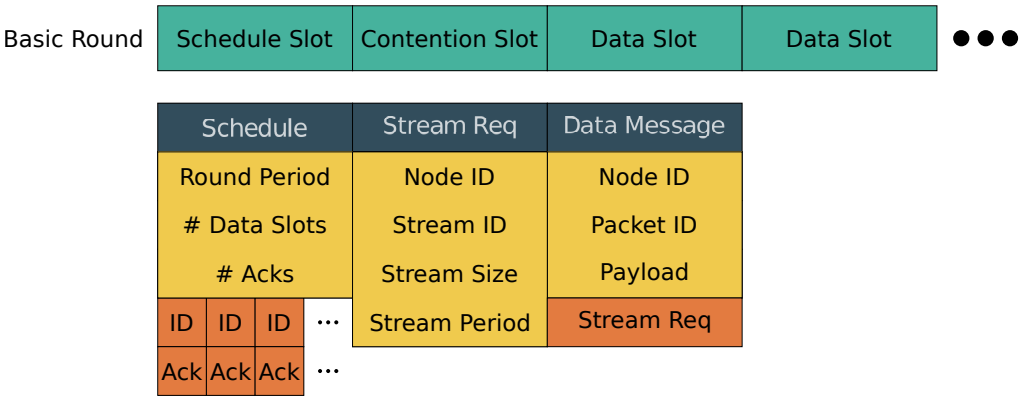


Figure 4.5: Basic round with message layout.

Schedule Slot

The schedule slot is used by the host to broadcast a schedule message to all nodes. The first element in this message is the round period. It is given in seconds and allows the nodes to calculate the start of the next round by adding it to the

start of the current round. The number of data slots indicates how many data slots are scheduled in this round. For each data slot the host appends a node's ID to the schedule to indicate which node is allowed to send in this slot. The last parameter is the number of acks appended to the schedule message. These acks are used to acknowledge stream requests piggy-backed to a data message as explained in Section 4.2.1.

The schedule is sent as a broadcast sync flood to allow all other nodes to synchronize to the host's time.

Contention Slot

In the contention slot the host node is listening for stream requests by the other nodes. In this slot multiple nodes may send at the same time. If a node has no stream request to send, it listens for other floods and retransmits the first received one. To be able to tell a node that its stream has arrived at the host the contention slot is built as an ack flood of mode 2 (see Section 3.2.2). If the stream request is not acknowledged the node backs off for a random amount of rounds to reduce contention. The maximum number of rounds to back off is a parameter of the protocol.

Each stream request is identified by its node and stream ID. The stream size is the number of bytes of one packet and the stream period is the time between two packet generations. To revoke an already assigned stream, a node can send a stream request with the same node and stream ID but with the period set to zero.

Data Slot

If a stream request reached the host, the host may allocate a data slot for this node. The nodes then can send a data message in the allocated slot. The data message consist of the node ID and a sequence number called packet ID. Each packet is identified by those IDs. The payload contains the actual data.

If a node wants to request a stream and already has at least one allocated data slot, it can piggy-back a stream request to a data message. As the data transmissions are not acknowledged, this stream request gets acknowledged by the host with the next schedule. Moving the stream requests to the data slots, decreases the contention in the contention slot and increases the probability that the stream request arrives, as there is no contention in the data slots.

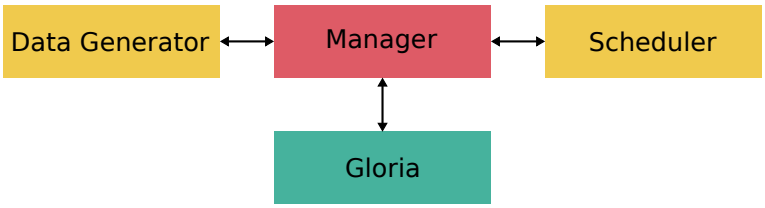


Figure 4.6: The different building blocks of LWB.

4.2.2 Building Blocks

As can be seen in Fig. 4.6, LWB consists of three building blocks and Gloria underneath it. The scheduler is only active on the host node. Only the data queue of the data generator is used on the host node to store the received data. On all other nodes the scheduler is disabled but the data generator periodically generates data. The manager is responsible to keep track of the time, initiate the Gloria floods and handle the finished floods.

In the rest of this section the functionality of the different blocks is explained in more detail.

Data Generator

The data generator replaces an actual application that would for example read out and process sensor data. It generates the first data packet and one stream request at the start of the protocol. The data period of the stream request is a random number between the round period and five times the round period. It is for example between 5 and 25s if the round period is set to 5s. The payload of the data messages is fixed to 16 bytes. The next packet generations are scheduled with a timer each data period. The packet contains a sequence number and the node ID for identification. The payload is constructed from those two and contains no relevant information. The packets are stored in a FIFO order linked list (see Section 5.4.1) called data queue.

Additionally to the data generation, the data generator also decides how many streams should be requested from the host. If the number of messages in the data queue exceeds a certain limit, a second stream is generated to reduce the backlog. The period of the second stream is set to the round period. When the queue size is back at a lower limit the second stream gets canceled. The limits for requesting a second stream or canceling it again are predefined constants. The number of streams a node requests is limited to the one for the periodic data generation and one for reducing the backlog.

The data generator also keeps track of the backoffs for the stream requests. They are set if the stream request was not acknowledged and are reduced at the end

of each round.

The host uses the data generator to store all the received messages during a round. At the end of a round they are printed via the serial interface and removed from the queue. From the serial output file the data packets that arrived at the host can then be reconstructed at the end of a test.

Scheduler

The scheduler is responsible to keep track of all the stream requests that arrived at the host. It checks if a received request has already been received before. In this case the request gets updated. If it has not been received before it gets stored in a stream queue (see Section 5.4.1). Each stream has a backoff value assigned to it, which indicates when the next data slot for this stream should be scheduled. The backoff gets initialized with the stream period, when the stream is first received. At the end of each round the scheduler iterates through all stream requests and removes those with a period of zero. For all others the backoff is reduced by the round period.

The round period is fixed and set at the start of the protocol. Based on the round period the scheduler can calculate the amount of data slots that fit into a round. The maximum amount is also limited by a constant to make sure the schedule message does not get too long.

To determine the schedule, the scheduler iterates through all received stream requests and allocates a data slot for each stream with a backoff of zero or below. The backoff for scheduled streams gets increased by the stream's period. This mechanism ensures that each node gets enough but not too many data slots assigned. If so much data is generated that it is timewise not possible to schedule enough data slots, the streams are handled in FIFO order. This also means that a node whose stream request is at the end of the stream queue may never get a data slot allocated.

While iterating through the streams the scheduler also checks if one of them has not yet been acknowledged. This happens if the stream request was piggybacked to a data message. A stream ack then gets appended to the schedule. The number of acks that can be appended to the schedule is limited by a predefined constant.

Manager

The manager handles the different slots in a round. It calculates the start times of the corresponding floods and makes sure the flood is correctly configured. Despite the fact that some messages can have different lengths, the slot lengths are equal for every round. This allows to calculate the start time of the next

slots by summing over the times of the previous slots.

On the host node, the manager gets the schedule from the scheduler, copies it to the flood message and starts the round by sending the schedule message. All other nodes start to listen for the schedule. If it is successfully received it gets stored and is used to schedule the next slots in this round. Else a node can not participate in this round and listens for the schedule again at the beginning of the next round.

For contention slots the manager checks with the data generator if there is an unacknowledged stream request that should be sent. The same applies to data slots with data messages if the node is the intended sender. If a node has nothing to send it starts listening and participates in the floods initiated by others. The host node saves the information from received stream requests and data messages. All other nodes do not save any information about other nodes.

4.2.3 Time Synchronization

With each sync flood in a schedule slot the nodes receive information about the absolute time synchronization to the host from Gloria (see Section 3.5). The received and the reconstructed markers are saved. At the end of each round the nodes try to adapt their timer to synchronize as good as possible to the host's time. If the number of saved markers is below a threshold, only an offset correction calculated from the latest markers is performed. Once there are enough markers available, the timer's offset and drift are compensated with linear regression as explained in Section 5.4.2. Section 5.2.2 gives more details on the timer configuration.

4.3 Long Range Extension

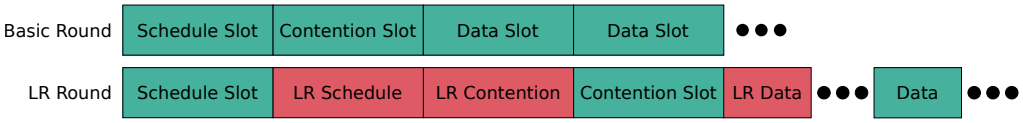


Figure 4.7: Basic LWB schedule compared to the long range extension.

The basic idea for the long range extension is to handle the remote nodes the same way as all others. As they can not participate in the normal floods, their communication needs to be scheduled in separate slots. Fig. 4.7 shows the additional slots that were inserted into the basic LWB round. They are similar to the ones that are already present in the basic schedule.

4.3.1 Long Range Rounds

The remote nodes have a different round period than the basic ones. This long range period has to be a multiple of the basic round period. It is fixed at protocol start by setting a constant which defines the multiplication factor.

Only the remote nodes which are connected to the same cluster can be active in the same round, as the scheduler selects one bridge node per round. The bridge nodes handle the communication with the remote nodes. They are alternated to distribute the energy overhead for long range communication between them. The communication between the bridge and remote nodes is done with Gloria floods consisting of only one slot and thus only one transmission. Those floods can easily be extended to consist of more slots and retransmissions for more redundancy or to acknowledge the messages between the remote and bridge nodes.

Figure 4.8: Example of a division of the rounds into basic and long range rounds.

If for example the network has two clusters as in Fig. 4.3 and the long range period is three times the normal period, the rounds would be distributed as in Fig. 4.8. The first round is a basic round without any long range slots or long range communication. In the second round there are allocated long range slots and one of the bridge nodes in the first cluster takes care of the long range communication. In the third round then a node from the second cluster acts as bridge. The fourth round is identical to the first one. For the fifth and sixth round the bridge nodes are changed.

In the current implementation it is not possible that multiple bridge nodes are active in the same round. This could lead to unwanted interference if they were active in the same slots or to very long rounds if each had its own long range slots. This means that the long range period must be larger than the normal period as soon as there are at least two clusters. This approach keeps the overhead for long range schedule and contention slots small compared to basic rounds. As a drawback the remote nodes have a higher latency.

4.3.2 Long Range Slots

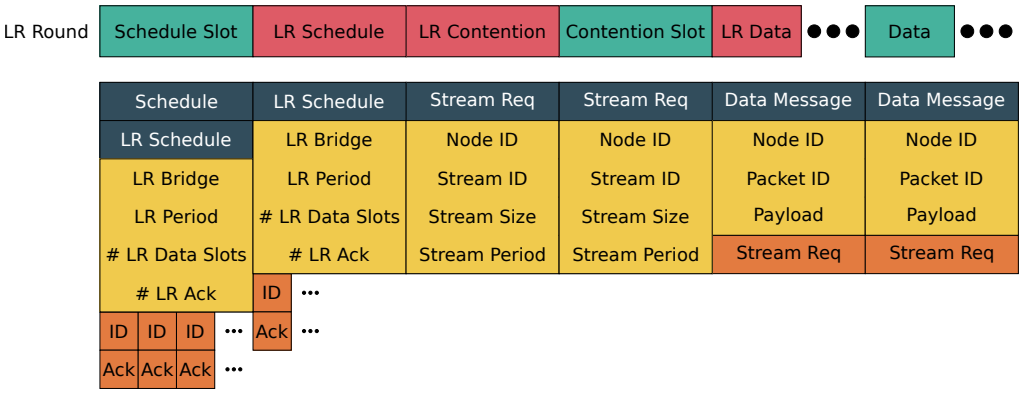


Figure 4.9: Long range round with message layout.

Fig. 4.9 shows the message layout for the different slots in a long range round. The messages are equal to the basic ones for the contention and data slots.

Schedule Slot

The schedule message contains the same general schedule information as in basic rounds. It is extended by a long range schedule the node ID's and the acknowledgments for the remote nodes. The long range schedule contains the same information as the general one adapted to the remote nodes. For example the period is longer. There is one additional parameter which defines the bridge node for this round.

Long Range Schedule

The bridge nodes takes the schedule message and reduces it to the important information for the remote nodes. They do not need to know the general schedule or anything about the allocation of data slots and the acknowledgments for normal nodes. In the end the bridge node sends out the long range schedule, the node ID's for the allocated long range data slots and the acknowledgments for

the remote nodes. As the long range data slots are scheduled before the normal data slots the remote nodes do not need to know how many normal data slots were allocated in this round.

Long Range Contention and Data

The long range stream requests and data messages are identical to the normal ones. When a bridge nodes gets a data message or a stream request from a remote node it stores it until the next normal contention or data slot and then relays it to the host. To be able to do this in the same round the long range slots are scheduled before their normal counterparts. The bridge nodes remove any messages from remote nodes at the end of the round.

4.3.3 Long Range Building Blocks

The basic building blocks are the same for the long range extension as for the basic implementation. This chapter describes the adaptions made to them.

Data Generator

The data generator works the same way on the remote nodes as on all other nodes. An adaption made for the long range case is the handling of data messages from remote nodes at the bridge nodes. These are added to the head of the data queue and not as the others to the tail. This assures that they are relayed in the same round as a bridge node always gets at least as many data slots allocated as it should receive data messages from remote nodes.

Scheduler

The scheduler is extended with queues for stream request from the remote nodes. It has one queue for each cluster in the network to be able to find the relevant stream requests for a cluster faster.

The algorithm to calculate the round schedule first of all checks if this round should be a long range round or not. It goes through all clusters in the network in round robin order. If not clusters are left the rounds are normal ones until it starts again with the first cluster. The number of normal rounds between the long range ones is given by the multiplication factor for the long range round period (`SLWB_LR_ROUND_MULT`) and the number of clusters in the network.

$$\text{normal\_rounds} = \text{SLWB\_LR\_ROUND\_MULT} - \text{number\_of\_clusters} \tag{4.1}$$

For long range rounds the scheduler iterates through all streams from remote nodes the same way as described for normal streams in Section 4.2.2. For each long range data slot assigned, also a normal data slot for the bridge node is allocated. This allows the bridge node to relay the data from the remote node in the same round. After the scheduler is finished with the remote streams it iterates through the normal ones. The time left for normal data slots is reduced by the already scheduled long range data slots and the corresponding data slots to relay the data.

Manager

The manager is extended to handle also the long range slots. Nodes that are not remote nodes nor the assigned bridge node for this round, skip the long range slots. For the long range schedule slot the manager makes sure that only the relevant information from the schedule is relayed to the remote nodes.

4.4 Characterization and Evaluation

This chapter shows the results of a 30min measurement of LWB. The measurement setup can be seen in Fig. 4.10. Table 4.1 shows the parameter settings for the measurement. For the normal flooding, FSK was used with a transmit power of 22dBm and for the long range communication, LoRa with 0dBm. The data period was fixed to 9s for the remote nodes and set to a random value between 3 and 18s for all others.

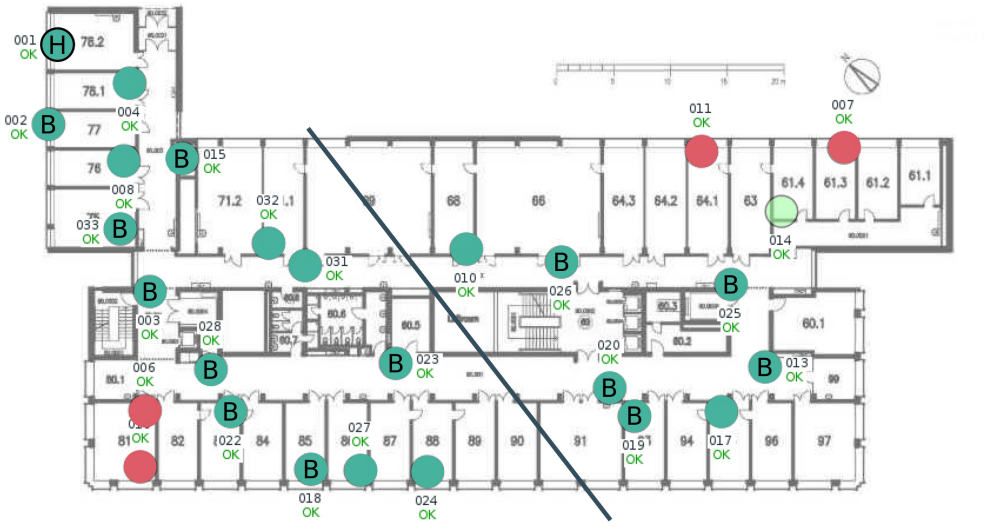


Figure 4.10: Measurement setup for LWB.

Normal Nodes	22
Remote Nodes	4
Clusters	2
Round Period	3s
LR Period	9s
Data Period	3 - 18s
LR Data Period	9s
Powers	22dBm / 0dBm
Modulations	FSK 200kbit / LoRa SF5

Table 4.1: Measurement setup parameters.

Four nodes where chosen to act as the remote nodes. They would have been able to participate in the normal LWB floods, but were set to only listen for LoRa transmissions, to simulate the behavior of nodes that are further away. To

handle the communication with the remote nodes the network was divided into two parts indicated by the blue line in Fig. 4.10. The nodes marked with a “B” acted as the bridge nodes for the left or right cluster respectively.

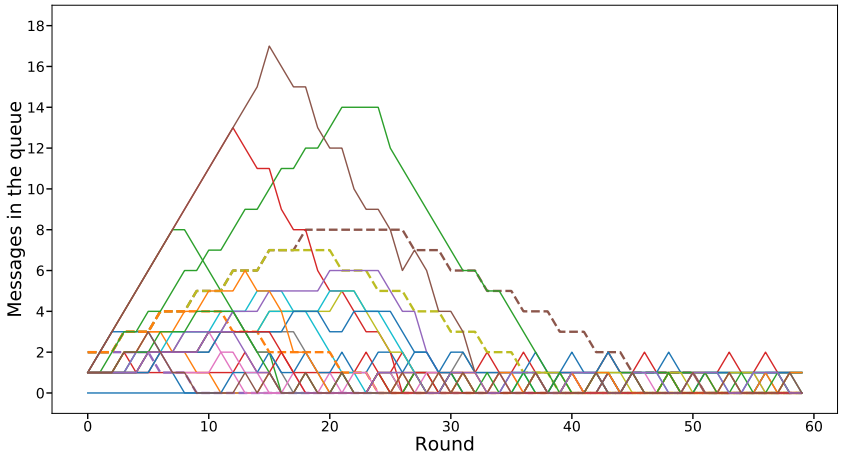


Figure 4.11: Number of messages in the queue. The dashed lines mark the remote nodes.

Fig. 4.11 shows the number of data messages in the queues on the different nodes for each round. The remote nodes are indicated by dashed lines. At the beginning all nodes start to generate data and try to request a stream. If they get a stream the queue size stays stable. The backlog can be reduced by requesting a second stream. When it is below 2 the second stream is canceled by the nodes. After around 25s all nodes have at least one stream. After 45s all nodes could reduce the number of messages in the queue to one or two. If a node missed a schedule and had more than 2 messages in the queue it requested again a second stream. The figure also shows that remote nodes are not discriminated.

The percentage of messages that were correctly received at the host is shown in Fig. 4.12. It can be seen that nodes 7 and 11 are not that good connected to the network as the rest. The remote nodes may have worse reliability as a message first needs to arrive at the bridge node and then via flooding at the host. On the other side the reliability for the nodes 6 and 16 is in the same range as for the normal nodes. This means that the bridge nodes have to be chosen carefully and if the links between bridge and remote nodes are not perfect, an acknowledgment mechanism has to be introduced to increase the reliability. In the current implementation the floods between the bridge and remote nodes only have one slot and no acks. They could easily be extend to make use of ack mode 2 to make sure that the data at least reached the bridge.

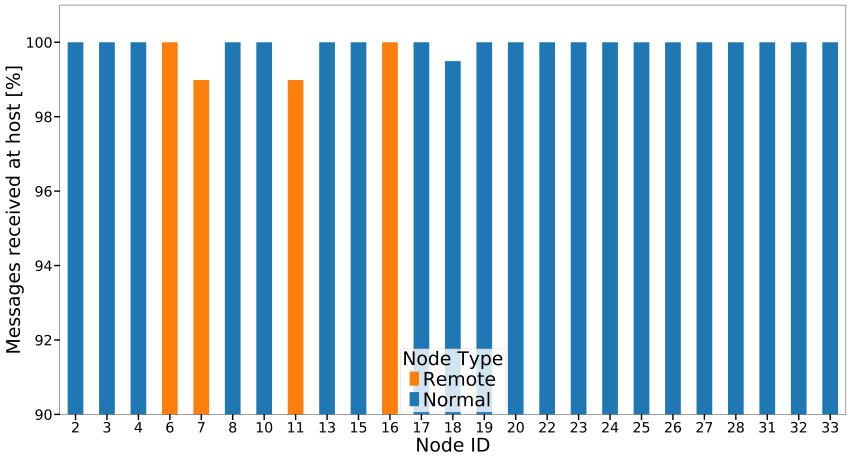


Figure 4.12: Percentage of successfully received packets at the host.

Chapter 5

Implementation Details

5.1 File Overview

This is an overview of the files used for the implementation of Gloria and LWB and other files referenced in this chapter. The top folder (flora) is the git repository [14].

```
flora/
└── lib/
    ├── arch/
    │   └── stm32hal/
    │       └── hs_timer.c
    ├── flocklab/
    │   └── flocklab.c/h
    ├── protocol/
    │   ├── gloria/
    │   │   ├── gloria_commands.c/h
    │   │   ├── gloria_constants.c/h
    │   │   ├── gloria_helpers.c/h
    │   │   ├── gloria_radio.c/h
    │   │   ├── gloria_structures.h
    │   │   ├── gloria_time.c/h
    │   │   └── gloria.c/h
    │   └── simple_lwb/
    │       ├── helpers.c/h
    │       ├── linked_list.c/h
    │       ├── slwb_commands.c/h
    │       ├── slwb_constants.c/h
    │       ├── slwb_data_generator.c/h
    │       ├── slwb_manager.c/h
    │       ├── slwb_network.c/h
    │       ├── slwb_scheduler.c/h
    │       ├── slwb_structures.h
    │       ├── slwb_timer_sync.c/h
    │       └── slwb.c/h
    ├── radio/
    │   ├── semtech/
    │   │   ├── sx126x/
    │   │   │   └── radio.c
    │   │   └── radio.h
    │   └── flora_radio.c/h
    └── time/
        └── hs_timer.h
```

5.2 Flora

This section describes the radio interrupt handling and the timer used for the Gloria and LWB implementation. The interrupt routine was enhanced and the possibility to compensate the drift with the timer was introduced during this thesis.

5.2.1 Radio Interrupts

Figs. 5.1 and 5.2 show the interrupt handling from the radio. The lowest level represents the actual interrupts from the radio (Radio Interrupts). They are handled in the `RadioIrqProcess` function in the *radio.c/h* files. As all interrupts from the radio are mapped to one pin that toggles, it is necessary to check in the interrupt register which interrupt was triggered. Each interrupt is assigned to one bit in the interrupt register. It is possible that multiple bits are set. This is represented by the stacked blocks on the lowest level in the figures. For example when an `RxDone` interrupt occurs, it is necessary to check the `HeaderError` and `CrcError` bits, to know if the reception was correct or not. This information is passed on to the next higher level as a function parameter, indicated by the True/False markers on the arrows.

The second level is the Flora level. These functions are defined in the *flora_radio.c/h* files. The `RxSync` function gets the capture timestamp from the timer and stores it for later use. It does not trigger anything on a higher level. The other functions invoke the callback functions on the next higher level and pass on the parameters. The callback functions must be set before each radio command as they get removed once an interrupt occurred. This can be done with the functions provided in the *flora_radio.c/h* files.

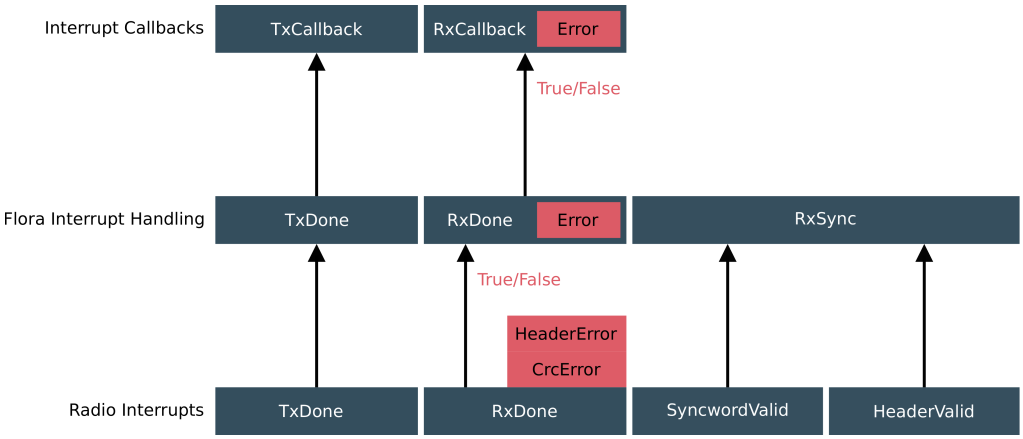


Figure 5.1: Interrupt handling part 1.

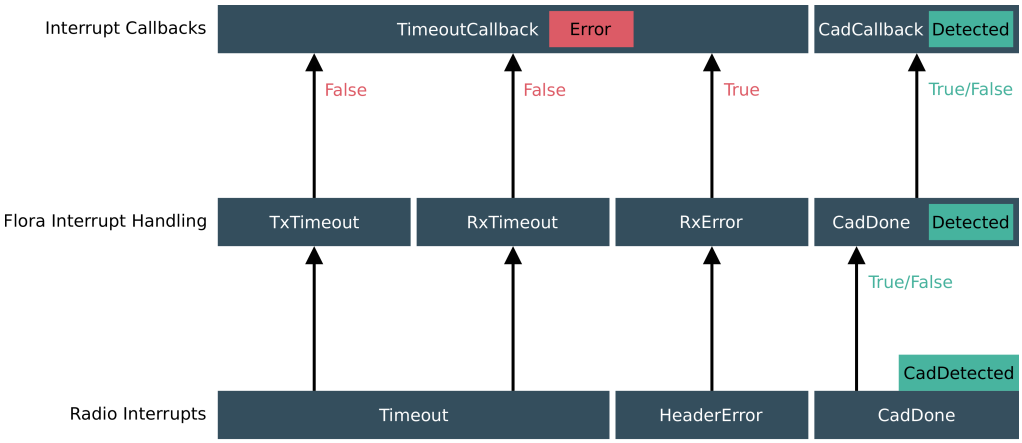


Figure 5.2: Interrupt handling part 2.

The rest of this section explains the handling of the different interrupts in more detail. For more information on the radio interrupts see the radio documentation [3].

TxDone

The `TxDone` interrupt is triggered once a transmission is finished. It is just passed on to the layers above.

RxDone

When an `RxDone` interrupt occurs the `HeaderError` and `CrcError` bits are also checked. The information from those is passed on as a parameter to the `RxDone` function on the Flora level. From there the callback is triggered with the same information.

Sync Word / Header Valid

These interrupts are triggered once the radio receives a valid Sync Word for FSK or a valid header for LoRa. They invoke the `RxSync` function on the Flora level, which saves the timer's capture timestamp. The timestamp can for example be used for synchronization. The `RxSync` function does not have a callback on a higher level.

Timeout

The `Timeout` interrupt triggers the `RxDone` or `TxDone` function on the Flora level. Which one is determined based on the current radio status. The callback on the highest level however is again the same (`TimeoutCallback`).

HeaderError

The `HeaderError` can also occur on its own (without a later `RxDone`). It is then passed on to the `RxError` function and to the `TimeoutCallback` with the `Error` parameter set to True.

CadDone

The `CadDone` interrupt is passed on with the parameter `Detected` from the `CadDetected` interrupt.

PreambleDetected

There is one more interrupt that is not currently handled and thus not shown in the figures.

5.2.2 Timer

The timer is based on a 32bit counter register with a software extension to 64bit. In the current implementation it runs on the same frequency as the microcontroller, which is 8MHz. This means that one clock cycle or timer tick is equal to 125ns.

The actual counter value is not visible to higher layer software. Instead the timer returns a 64bit virtual time. This virtual time is based on the counter value, the software extension and the drift and offset values. The drift and offset are initialized as 1 and 0 respectively and may be adapted with the functions provided in the *hs_timer.c/h* files. The calculation of the virtual time can be seen in Eq. (5.1).

$$\text{virtual\_time} = (\text{sw\_extension}|\text{counter\_value}) * \text{drift} + \text{offset} \tag{5.1}$$

The | stands for a bitwise or operation. The software extension is shifted by 32 bits before the calculation.

5.2.3 Flocklab Pins

To be able to set the Flocklab pins and use them for debugging, the files *flocklab.c/h* were generated. They offer various functions to set, reset or toggle Flocklab pins. The interrupt routine for GPIO actuations from Flocklab is also defined here. It does not do anything in the current implementation.

The current pin usage is shown in Table 5.1.

Pin Name	Usage
INT1	Set to toggle after each Gloria flood.
LED1	Is high if the radio is in tx mode.
LED3	Is high if the radio is in rx mode.

Table 5.1: Flocklab pin usage.

The general pin assignments for Flocklab can be found in the Flocklab wiki [15].

5.3 Gloria

5.3.1 Gloria Parameters

Before starting a flood certain settings for the flood need to be configured. Every setting can be adjusted in the `gloria_flood_t` struct. To start the flood a pointer to the struct is, together with a pointer to a callback function, handed over to the `gloria_run_flood` function. When the flood is finished the callback function gets executed. The necessary information such as if a message was received, if the message was acked or the received message itself are also stored in the `gloria_flood_t` struct.

An overview of the flood struct can be seen in Table 5.2. Additionally to the parameter name and the data type, the "Usage" column states what kind of parameter it is. "Rx" or "Tx" stands for parameters that need to be configured before listening or initiating a flood. Possible values for those parameters are listed in the "Values" column. "Int" stands for internal and means that the parameter is used during the flood. On the other hand "cb" stands for callback and marks the parameters which might be interesting for higher layer protocols after the flood is finished. Some parameters as for example `msg_received` are used during the flood but are also important after the flood is finished.

Additional to the parameters in the `gloria_flood_t` struct, three parameters can be configured in the `gloria_header_t` struct shown in Table 5.4. These are the message type, the destination and if the flood is a sync flood. The actual payload of the message is copied to the payload array. Both header and payload are part of the `gloria_message_t` struct shown in Table 5.3.

The rest of this section explains the implementation details that are required to successfully use Gloria. If a parameter has already been described in the Chapter 3, a link to the relevant section is given.

Parameter Name	Data Type	Usage	Values
ack_counter	uint8_t	int	
ack_message	gloria_ack_message_t	int / cb	
ack_mode	uint8_t	Rx / Tx	0 - 2
acked	bool	int / cb	
band	uint8_t	Rx / Tx	0 - 51
callback	void*	Rx / Tx	
crc_error	bool	cb	
crc_timeout	bool	cb	
current_tx_marker	uint64_t	int	
data_slots	uint8_t	Rx / Tx	0 - 255
first_rx_index	int8_t	int / cb	
flood_idx	uint16_t	Rx / Tx	0 - 2^{16}-1
guard_time	uint32_t	Rx	0 - 2^{32}-1
initial	bool	Rx / Tx	True / False
last_active_slot	uint8_t	int / cb	
lp_listening	bool	Rx	True / False
marker	uint64_t	Rx / Tx	0 - 2^{64}-1
max_acks	uint8_t	Rx / Tx	0 - 255
max_retransmissions	uint8_t	Rx / Tx	0 - 255
message	gloria_message_t*	Rx / Tx	*
message_size	uint8_t	int	
modulation	uint8_t	Rx / Tx	0 - 10
msg_received	bool	int / cb	
payload_size	uint8_t	Rx / Tx	0 - 251
power	int8_t	Rx / Tx	-9 - 22
received_marker	uint64_t	int / cb	
reconstructed_marker	uint64_t	int / cb	
remaining_retransmission	uint8_t	int / cb	
rssi	int8_t	cb	
rx_timeout	uint32_t	Rx	0 - 2097151750
slot_index	uint8_t	int / cb	
snr	int8_t	cb	
sync_timer	bool	Rx	True / False

Table 5.2: Gloria flood struct.

Parameter Name	Data Type
header	gloria_header_t
payload	uint8_t[255-GLORIA_HEADER_LENGTH]

Table 5.3: Gloria message.

Parameter Name	Data Type	Config	Values
dst	uint8_t	Tx	0 - 255
slot_index	uint8_t		
src	uint8_t		
sync	uint8_t: 1	Rx / Tx	0 / 1
type	uint8_t: 7	Tx	0 - 127

Table 5.4: Gloria header.

ack_counter:
The `ack_counter` is used during the flood to make sure that at most `max_acks` number of acks are sent.

ack_message:
Contains the ack that should be sent. It consists only of the destination described below.

ack_message.dst:
Section 3.4
The destination for the ack message is the node ID of the initiator. It is only valid if an ack has been received during this flood. Check the `acked` parameter to see if this is the case.

ack_mode:
Section 3.3.1.

acked:
Bool that tells if an ack has been received. It is important to check the `ack_message.dst` parameter to be sure for which node the ack was intended.

band:
The band parameter is used to select one of the predefined frequency settings in the variable `radio_bands` in the file *radio_constants.c*. The settings consist of the center frequency, the bandwidth and the associated duty cycles and maximum transmit powers.

callback:
The `callback` parameter saves a pointer to the callback function. The function gets executed as soon as the flood is finished. It has to be specified at flood start and handed over to the `gloria_run_flood` function.

crc_error:
If `crc_error` is true at least one message (ack or data) was received with wrong payload CRC.

crc_timeout:
If `crc_timeout` is true at least one message (ack or data) was received with wrong header CRC.

current_tx_marker:
The `current_tx_marker` gets calculated for each slot and marks the beginning of the transmission. The beginning of the reception is scheduled earlier to make sure the radio is ready to receive the message. How much is defined by the `rxOffset` constant (see Section 5.3.3) for each modulation setting. If a guard time is specified the offset is increased by its value.

data_slots:
Section 3.3.1
The `data_slots` parameter is used to set the number of data slots, which is equal to the slot limit. In the actual implementation there are no subslots but the data and ack slots are independent slots as shown in Fig. 5.3.

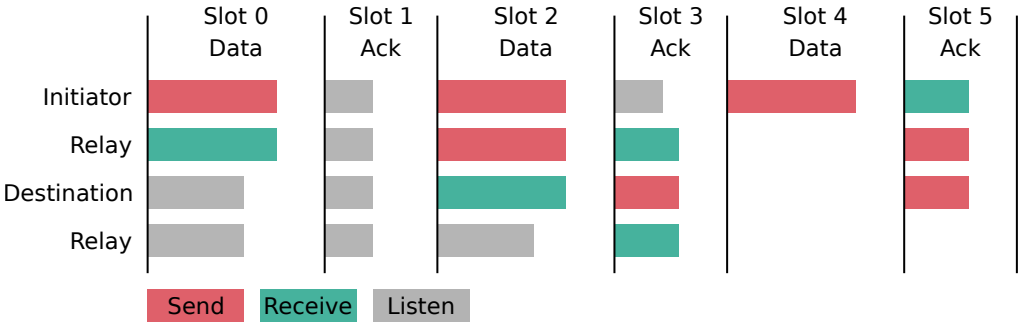


Figure 5.3: Ack flood with slot numbering as implemented.

first_rx_index:
The number of the slot the data message was received. It is set to -1 for normal and -2 for ack floods at their beginning. It is needed to calculate the `last_active_index` and can also be used to for example make an estimation on the distance to the initiator on higher layer protocols. For ack floods it is always even as all data slots have an even slot number.

flood_idx:
This parameter is for debugging and measurement purposes only. It allows to assign each flood an ID to be able to separate the floods for analysis.

guard_time:
Section 3.3.3
The guard time needs to be specified in timer ticks (see Section 5.2.2).

initial:
Set initial to 0 if this node should listen for a flood and to 1 for the initiator.

last_active_slot:
It contains the index of the slot in which the node takes its last action. This can be sending or listening. For the initiator it can be calculated at flood start. For all other nodes it is set to 0 at flood start and is calculated on message reception.

The value that gets calculated depends on the flood type. For ack mode 2 it is set to the slot limit and the flood is only stopped earlier, if all acks were sent. For normal floods and floods with ack mode 1 it marks the slot the last data message should be sent. An example of its calculation for ack mode 1 can be seen in Section 3.2.2.

lp_listening:
Section 3.3.3

marker:
Section 3.3.1

max_acks:
Section 3.3.1

max_retransmissions:
Section 3.3.1

message:
The message contains the `gloria_header_t` and the payload for the flood. It actually gets transmitted. The memory for it also needs to be allocated for listening nodes as the received data gets copied there.

message ->header.dst:
Section 3.3.2

message ->header.slot_index:
Each message header contains the slot index of the current slot, so that the receiving nodes know how many slots are left and when the flood started.

message ->header.src:
Section 3.4

message ->header.sync:
Section 3.3.1

message ->header.type:
Section 3.3.2

message ->payload:
The payload to transmit is stored as an `uint8_t` array and has to be copied before the flood is started. For sync floods the sync timestamp is also stored in the payload and the bytes that can be set are reduced by its length. The length of the timestamp can be set as explained in Section 5.3.3. See also the description on the payload size below.

message_size:
The size of the actually transmitted packet. Consists of the payload size the size of the header and the timestamp length for sync floods. In contrast to the `payload_size` it does not need to be set at flood start but gets calculated during the flood.

modulation:
The possible modulation settings are predefined in the constant `radio_modulations` in the file *radio_constants.c*. There are two versions for FSK with different data rates and seven for LoRa with different spreading factors.

msg_received:
This parameter indicates if the flood message was successfully received. It is always true for the initiator.

payload_size
The size in bytes of the payload to send should be equal to the number of bytes copied to `message->payload`. The maximum payload size is

$$255 - \text{GLORIA\_HEADER\_LENGTH} \tag{5.2}$$

bytes for floods without an appended timestamp and

$$255 - \text{GLORIA\_HEADER\_LENGTH} - \text{GLORIA\_TIMESTAMP\_LENGTH} \tag{5.3}$$

bytes for sync floods. The payload also needs to be known at the receiving nodes for low power listening. If low power listening is disabled, the payload needs only be specified for the initiator. GLORIA_HEADER_LENGTH and GLORIA_TIMESTAMP_LENGTH are constants defined in the *gloria_constants.h* file. The timestamp length can be adapted under certain constraints as explained in Section 5.3.3.

power:
This parameter is used to configure the transmit power in dBm that should be used for sending.

received_marker:
Section 3.4

reconstructed_marker:
Section 3.4

remaining_retransmissions:
The `remaining_retransmissions` is a counter to check if the node should retransmit the message again or not.

rssi:
This parameter contains the RSSI value of the first received message (ack or data).

rx_timeout:
Section 3.3.3
The rx timeout needs to be specified in timer ticks (see Section 5.2.2).

snr:
This parameter contains the SNR value of the first received message (ack or

data) for LoRa modulations. I is not provided for the FSK modulation by the radio.

sync_timer:
Section 3.3.3
For more details on the possibilities to set the timer offset and drift see Section 5.2.2

5.3.2 Gloria Time

The files *gloria_time.c/h* contain all the functions related to time calculations for Gloria. Most of them are used during the flood and not relevant for using Gloria. But `gloria_calculate_slot_time` and `gloria_calculate_flood_time` can be quite useful as they allow to calculate the duration of a slot or the whole flood.

5.3.3 Gloria Constants

Sync Flood Overhead

For sync floods the flood start timestamp is appended to the message. It is 64bit wide and therefore a big overhead for smaller messages. If this overhead should be reduced the `GLORIA_SCHEDULE_GRANULARITY` constant in *gloria_constants.h* can be increased. This results in the timestamp being divided by this constant before it is appended to the message. If the division is high enough the number of bytes that need to be transmitted for the timestamp can be reduced. On the receiver side the received timestamp is reconstructed by multiplying it with `GLORIA_SCHEDULE_GRANULARITY`. The drawback is that all floods need to be scheduled as multiples of `GLORIA_SCHEDULE_GRANULARITY`. Else the rounding errors by dividing and multiplying the timestamp would result in an imprecise synchronization.

If for example `GLORIA_SCHEDULE_GRANULARITY` is set to 2^{16}, the floods need to be scheduled as multiples of 2^{16}*125ns = 8.192ms. But the timestamp appended to sync floods can be reduced by 2 bytes.

Gloria does not check if the flood start marker is set to a multiple of `GLORIA_SCHEDULE_GRANULARITY`. This has to be considered before setting the marker on higher levels!

Timing Constants

In the file *gloria_constants.c* some timing constants are defined, which are different for each modulation setting. The file was originally generated by the python

script *gloria.py* located in the flora tools git [16] under */flora_tools/*. The different constants get calculated according to measurements done in a previous thesis [2]. Some of them were adapted during this thesis and are marked with "(adapted by kelmicha)". A brief overview of the different constants is given below.

floodInitOverhead:
The initial overhead is based on the wakeup time for the radio and the time needed for the rx or tx setup. It makes sure that a node has enough time to initialize and start the flood.

rxOffset:
The rx offset is the time a node starts receiving before the message is actually sent. This is only relevant for low-power listening or if a node has received the data message and listens for acks. It was set to a few symbols time based on experience in the previous thesis [2].

slotOverhead / slotAckOverhead:
The slot overheads are used for processing of received messages and the configuration of the radio. They were calculated based on the transition times between the radio modes. For the FSK modulation settings they were increased during this thesis as they were too tight. The increase is based on rough estimates and observations of timing issues.

txSync:
This constant is the delay between the start of the transmission and the header / Sync Word interrupt. It is used to calculate the flood start as shown in Section 3.5.1. It has been adapted based on Flocklab measurements during this thesis.

5.4 LWB

5.4.1 Linked List

To store for example the streams or the data messages a linked list was implemented. It can be used for any data type. The data is stored in a `uint8_t` array and can be cast to the desired data type or copied to another variable. For more details on the different operations on the list see the *linked_list.c/h* files.

5.4.2 Linear Regression

The linear regression for LWB is done in the *slwb_timer_sync.c/h* files. The number of markers to use can be set with the constant `NUMBER_SAVED_MARKERS`. The constant `SLWB_MIN_MARKERS_FOR_DRIFT_COMP` specifies how many markers are minimally needed for drift compensation. A regression on for example only two points can be very inaccurate. Especially if one of the points itself is inaccurate. Because of this, only the offset is corrected until enough markers are available. It is calculated based on only the latest marker. Both constants are defined in *slwb_constants.h*. The actual computation of drift and offset is shown below.

X is the array of the local timestamps and Y the array of the corresponding timestamps at the initiator. The drift and offset can then be calculated as follows.

$$\text{drift} = \frac{\bar{X} * \bar{Y} - \overline{X * Y}}{\bar{X} * \bar{X} - \overline{X^2}} \tag{5.4}$$

$$\text{offset} = \bar{Y} - \text{drift} * \bar{X} \tag{5.5}$$

$\bar{X}$ and $\bar{Y}$ stand for the average of all the values in X respectively Y. $X * Y$ for the element wise multiplication of the two.

To avoid overflows for the calculations, all markers are reduced by the value of the smallest one. At the end the variable transformation is reverted to get the correct offset value.

5.4.3 LWB Constants

All the constants for the LWB protocol are defined in *slwb_constants.h*.

The slot overhead is set to 50ms. The actual time needed for the processing in the different slots was not measured. The current value makes sure that it is more than enough and there is also time for more processing if the protocol is extended. For a final version of the protocol the overhead could be measured

and reduced to save some time. Also an overhead per slot type would reduce the total time needed for each round.

5.4.4 LWB Slot Times

The slot times for the different slots are predefined in the *slwb_constants.c* file for each modulation. They are composed of the flood duration for the respective floods and the `SLWB_SLOT_OVERHEAD` constant defined in *slwb_constants.h*. The number of slots for the floods is also defined in *slwb_constants.c*. The slot overhead is used for processing of the last flood and preparing the next flood. It is the same for all slots.

$$\text{slot\_time} = \text{flood\_duration} + \text{SLWB\_SLOT\_OVERHEAD} \tag{5.6}$$

The file *flora_tools/codegen/gen_slwb_slot_times.py* in the *flora_tools* development fork for this thesis [17] contains the code used to generate the slot times. The paths probably need to be adapted.

5.4.5 LWB Network

The partitioning of the network for Flocklab is defined in *slwb_network.c*. The current partitioning defines two clusters. The two node lists `cl_0` and `cl_1` define the bridge nodes for them. The remote nodes get added from the host node as soon as a stream request from them is received.

5.4.6 LWB Helpers

In the files *helpers.c/h* multiple smaller functions are define. On one hand these are the functions for getting and setting the node configurations as for example the node ID.

On the other hand there are multiple functions used to print information about the protocol evolution over the serial interface. For those information printed a priority may be defined in the header file. It is called `PRINT_PRIO` and can be used to configure how much information should be printed for a certain measurement. The data messages that are sent, received at the host or generated are printed with priority 10. Also the information about the data queue size and the stream requests are printed with this priority. Other things like the schedule of the current round are printed with lower priorities.

Chapter 6

Conclusion

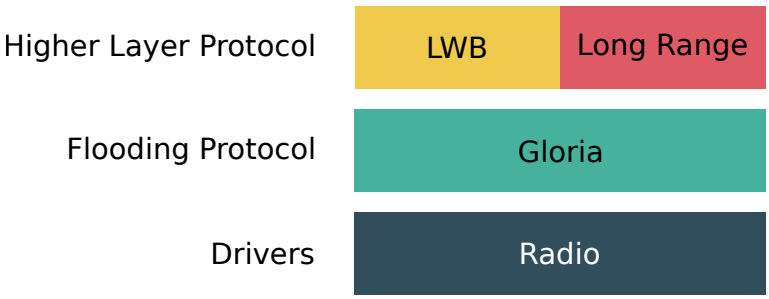


Figure 6.1: Software stack.

Fig. 6.1 shows the resulting software stack of this thesis. Apart from a large contribution to Gloria and the implementation of LWB also the radio drivers were enhanced.

Gloria is a network flooding protocol which offers time synchronization in the order of a few microseconds. It is highly configurable and for example offers the possibility to chose between the two modulations FSK and LoRa. This makes it useful for a wide range of scenarios which may also need communication over long distances. It has been extensively tested and enhanced to work with high reliability.

On the other hand the implementation of LWB is very simple. It was also tested, but not as thoroughly as Gloria. It was used as basis for the long range extension. The long range extension is the implementation of one possibility to include nodes that are further away into an LWB network, without introducing more state at the normal nodes. Tests on Flocklab showed that it works without discriminating the remote nodes. The main goals of a first and imple implemantation were met. But as the testing of Gloria took more time than expected, the LWB implementation is not yet in a state to perform extensive performance tests.

Chapter 7

Future Work

Although the radio interrupt routine has been updated, the rest of the radio drivers are not well commented and sometimes confusing. For example, there are multiple functions for sending and receiving, which can be used to do the same thing. It would make them a lot easier to use if they were cleaned up and simplified.

Gloria is tested and working as it is. Nevertheless there are some improvements which could be made. For example, the slot times for FSK could be reduced, by measuring the timing more accurate and trying to set / read the radio buffer while sending / receiving. This would reduce the total time needed for a flood to propagate through the network. For LoRa the reduction would be less significant, as the time on air is much larger.
The worse behavior in terms of reliability compared to the Glossy FSK implementation needs to be examined. If it is due to the protocol implementation the reliability can hopefully be increased. It could also be because of different test setups or hardware used.

For LWB the future work can go in many directions. One very important part that is missing now is the bootstrapping and the adaption to network changes. Furthermore the scheduler serves the stream requests in FIFO order and thus is not fair. Also the support of data periods below the round period and the adaption of the round period to the traffic demand are not yet implemented.
The extension for the remote nodes could be extended to not only make use of node to node communication between the remote nodes and the bridge nodes, but also include communication over multiple hops, with remote nodes as relays.

Appendix A

Miscellaneous

A.1 Flocklab Synchronization Issues

During the Gloria measurements synchronization problems were observed. At first it was not clear if they are due to problems in the implementation or on Flocklab. However, often the errors were arranged in a certain pattern. Fig. A.1 shows the synchronization error relative to the initiator over multiple floods. Per flood and node one point marks the error. As can be seen multiple nodes drift away and come back to the normal synchronization in a similar pattern. Around flood 1800 all the nodes have an error which is negative. This is most probably due to synchronization problems on the initiator node.

Due to this observations the error was expected to be on Flocklab side and not in the implementation. To confirm this thesis another measurement was performed. In this measurement half of the nodes set a pin every 500ms and do nothing else. The others sleep for the first half of the measurement and then begin to flood the network with Gloria floods. The modulation was set to LoRa SF5 and the power to 22dBm.

The drift of the nodes that only set the pins is shown in Fig. A.2. In the first 600s the drift is more or less linear without massive variations. But after the flooding starts at second 600, there are major spikes in the drift.

It is suspected that the extensive flooding generates enough interference to disturb the Flocklab synchronization performed with Glossy. This also means that the synchronization measurements for LoRa modulations can be inaccurate because of imprecise synchronization on Flocklab. For FSK the problem is less significant as the time on air and thus the probability to interfere with the Glossy synchronization is much lower.

More measurements are needed to be able to precisely determine the impact of interference on the Flocklab synchronization.

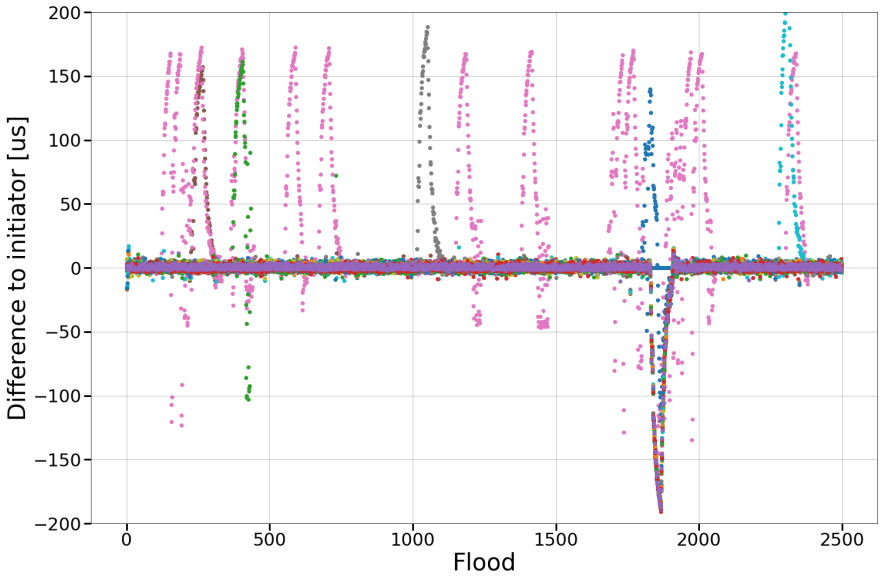


Figure A.1: Synchronization relative to the initiator.

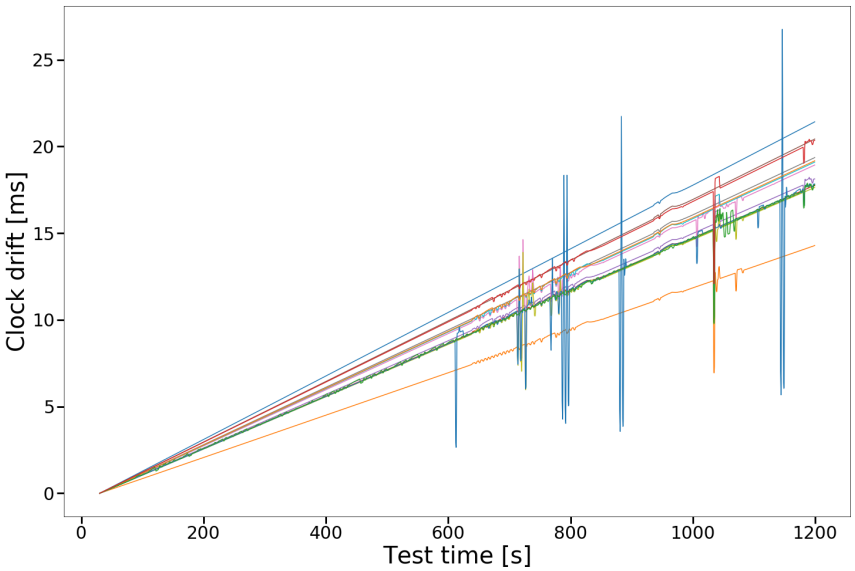


Figure A.2: Clock drift on the different nodes.

A.2 RxDone Interrupt

During this thesis it was observed that sometimes the rx done interrupt of the radio seems to occur too late. This led to timing problems for the Gloria floods, as it could happened that a node did not have enough time to prepare for the next slot if the interrupt was delayed. A too late interrupt always meant that the CRC failed. Mostly it was observed for FSK modulations. It is not clear what causes the problem as it is also very rare.

Fig. A.3 shows an example of an interrupt arriving too late. The blue bars indicate that a node is listening, the red ones that a node is transmitting and the green ones stand for radio interrupts. For the transmissions there is only the tx done interrupt at the end. For the receptions there is a Sync Word valid interrupt earlier and an rx done interrupt at the end. Node 10 listens three times in the example. The first time a message was received but with wrong CRC. This is why it listens again. The rx done interrupt occurs almost at the same time as the rx done interrupt of the sending nodes. For the second reception the rx done interrupt arrives 1.68ms after the tx done interrupt. For the third reception the interrupts are again simultaneous.

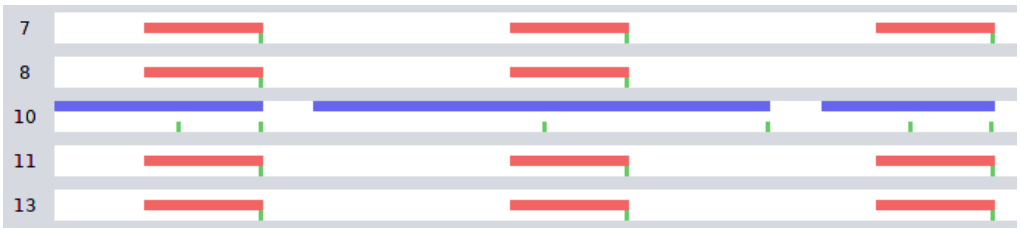


Figure A.3: Example of the RxDone interrupt arriving too late.

Appendix B

Flocklab Measurements

This chapter explains how the measurements on Flocklab where done and gives an overview of the python scripts used for their analysis.

B.1 File Overview

Below is an overview of the files used for the measurements on Flocklab and the analysis of the data.

```
flora_tools/
├── analysis/
│   ├── analysis_helpers.py
│   ├── analyze_gloria_kelmicha.ipynb
│   ├── analyze_gloria_kelmicha.py
│   ├── analyze_links_kelmicha.ipynb
│   ├── analyze_links_kelmicha.py
│   ├── analyze_slwb.ipynb
│   ├── analyze_slwb.py
│   ├── analyze_sync_kelmicha.ipynb
│   ├── analyze_sync_kelmicha.py
│   ├── gloria_ack_analysis_calc.ipynb
│   ├── gloria_ack_analysis_calc.py
│   ├── gloria_ack_analysis_gpio.ipynb
│   ├── gloria_ack_analysis_gpio.py
│   ├── radio_const.py
│   ├── test.ipynb
│   ├── time_calc.ipynb
│   ├── time_calc.py
│   └── time_const.py
├── codegen.py
│   └── gen_slwb_slot_times.py
├── flocklab/
│   ├── flocklab.py
│   ├── measure_gloria_kelmicha.py
│   ├── measure_links_kelmicha.py
│   ├── measure_slwb.py
│   └── run_nodes.py
├── __main__.py
└── node.py
```

B.2 Flora CLI

Flora offers a command line interface to start and control measurements. The commands that can be used are defined in the command files for the respective protocol. These are *gloria_commands.c/h* for Gloria and *slwb_commands.c/h* for LWB. On overview of the commands and the corresponding parameters is given in Tables B.1 and B.2. These are also the commands that were used for the

measurements described in Appendices B.3 and B.4.

Command	Parameter	Flag
gloria rx	ack mode	a
	rx timeout	c
	guard time	g
	flood index	i
	maximum number of acks	k
	slot limit	l
	modulation	m
	power	o
	retransmissions	r
	sync the timer	s
	delta marker	t
	continuous measurement	u
	tx period for continuous measurements	x
gloria tx	ack mode	a
	destination	d
	flood index	i
	maximum number of acks	k
	slot limit	l
	modulation	m
	number of floods for continuous measurements	n
	power	o
	payload	p
	retransmissions	r
	sync flood	s
	delta marker	t
	continuous measurement	u
	tx period for continuous measurements	x

Table B.1: CLI commands for Gloria with the according parameters.

Command	Parameter	Flag
slwb start	long range modulation	l
	modulation	m
	long range power	o
	power	p
	round period	r

Table B.2: CLI commands for LWB with the according parameters.

B.3 Gloria

B.3.1 Measurement Script

flora_tools/flocklab/measure_gloria_kelmicha.py

B.3.2 Parameters

The following parameters can be set in the *measure_gloria_kelmicha.py* script.

ITERATIONS:

The number of iterations for this measurement. For each iteration all initiators send with all specified modulations.

SLEEP_TIME:

The sleep time defines the time the script waits before starting the next flood. It is defined as an array to be able to set it independently for each modulation.

TX_DELAY:

After the commands for the receiving nodes are sent, the script waits TX_DELAY seconds before issuing the command to the initiator of the flood. This is done because the server relaying the commands to the Flocklab nodes sometimes has a delay. With the timeout the commands for the receiving nodes should have arrived at the nodes before the flood is started.

RX_TIMEOUT:

The time the nodes should listen before going back to sleep.

RADIO_MODULATIONS:

One of the radio modulation settings defined in *flora/lib/radio/radio_constants.c*. For each iteration a flood with each modulation is started.

POWERS:

The transmit power in dBm.

SLOTS:

The slot limit for the floods. Can be specified for each modulation independently.

RETRANSMISSIONS:

The number of retransmissions.

SYNCFLOODS:

Specifies if the floods should be sync floods.

SYNC_TIMER:

Specifies if Gloria should adapt the timers offset according to the received timestamp in sync floods.

ACK_MODE:

The ack mode for the floods.

MAX_ACKS:

The maximum number of acks.

MESSAGE:

The message defines the payload that should be sent with the flood.

CONTINUE:

This parameter allows to do continuous measurements. This means that the initiator starts the next flood without any new cli command. The other nodes also continue listening for further floods if CONTINUE is set. This only works if ITERATIONS is set to 1 and only one initiator and one modulation is specified. Also the other parameters stay the same during all the floods.

TX_PERIOD:

The period between flood starts for continuous measurements.

GUARD_TIME:

The time the receiving nodes start to listen before the initiator starts the next flood for continuous measurements. Should be set to zero for other measurements.

NODES:

The node IDs of the nodes that should participate in the measurement. These also have to be specified in the Flocklab xml configuration file.

TX_NODES:

The node IDs of the nodes that should initiate a flood. For each iteration all tx nodes send a flood after each other.

BASE:

The destination of the floods. Can be set to 0 for broadcasts.

B.3.3 Measurement Timings

The script *flora_tools/analysis/time_calc.ipynb* can be used to calculate the duration of the different floods and also the duration of the whole measurement.

B.3.4 Analysis

Each node prints a json like string over the serial interface after a flood is finished. The json contains information about the flood. The file *analyze_gloria_kelmicha.py* offers functions to parse those strings convert them to data frames and save the information as .csv files. The *analyze_gloria_kelmicha.ipynb* uses the generated data frames to generate plots of the data or analyze it in other ways.

Each node toggles the "INT1" Flocklab pin 50ms after the flood is finished. This pin toggles can be used to examine the synchronization of the nodes. The script *analyze_sync_kelmicha.ipynb* was used to do this.

B.4 LWB

B.4.1 Measurement Script

flora_tools/flocklab/measure_slwb.py

B.4.2 Parameters

The following parameters can be set in the *measure_slwb.py* script.

NODES:

The node IDs of the nodes that should participate in the measurement.

HOST:

The node ID of the host node.

LR_NODES_FL:

The IDs of the remote nodes for measurements on Flocklab.

LR_NODES_LC:

The IDs of the remote nodes for local measurements.

ITERATIONS:

Should be left at 1 as the protocol runs by itself.

RUNTIME:

After the runtime the nodes are reset and disconnected and thus the protocol is stopped.

LR_MODULATION:

Modulation for the communication between remote and bridge nodes.

LR_POWER:

Power for the communication between remote and bridge nodes.

MODULATION:

Modulation for the floods in the basic network.

POWER:

Power for the floods in the basic network.

ROUND_PERIOD:

Round period in seconds for the protocol.

B.4.3 Analysis

The serial outputs from the LWB protocol can be parsed with the *analyze_slwb.py* script. The *analyze_slwb.ipynb* script uses the parsed information to generate some performance plots.

B.5 Start the Measurements

To be able to run the measurements with the bash commands, the flora tools need to be installed as described in the README in the git repository [16].

B.5.1 Flocklab

To start a measurement on Flocklab do the following:

1. Compile the C code.
2. Prepare the Flocklab xml file with the binary version of the code.
3. Save it under "/flora_tools/flocklab/flocklab-dpp2lora-flora_cli_kelmicha.xml".
4. Start the measurement script with the bash command: "flora_tools flocklab_measure_gloria -r" or "flora_tools flocklab_measure_slwb -r"

The name for the xml file can also be changed. But the measurement scripts have to be adapted accordingly.

B.5.2 Local

To do the measurement only on local nodes the following steps are needed:

1. Compile the C code.
2. Flash the code onto the nodes.

3. Start the measurement script with the bash command: "flora_tools flocklab_measure_gloria -l" or "flora_tools flocklab_measure_slwb -l"

The serial output for the local measurements is currently saved in the folder */ma_kelmich/workspace/flocklab/test_results/local/*. The path should be adapted.

B.5.3 Aliases

The *.bash_aliases* file in the git repository of this thesis [18] contains some bash aliases to start the measurements. They are described below.

fl_xml $1

Convert the */flora/platform/gloriaLwb_flocklab/Debug/gloriaLwb_flocklab.elf* file to binary, copy it to the Flocklab xml file and set the test duration to $1.
The paths need to be adapted correctly.

flasha

Flashes the code of the */flora/platform/gloriaLwb_comboard/* project onto the local nodes. It is an alias for the bash command "flora_tools program_all -v comboard -d /ma_kelmicha/workspace/flora".

flashaf

Flashes the code of the */flora/platform/gloriaLwb_flocklab/* project onto the local nodes. It is an alias for the bash command "flora_tools program_all -v flocklab -d /ma_kelmicha/workspace/flora".

measure_gloria r/l

Start the measurement script for Gloria. The "r" is used for measurements on Flocklab and "l" for local ones.

measure_slwb r/l

Start the measurement script for LWB. The "r" is used for measurements on Flocklab and "l" for local ones.

Appendix C

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

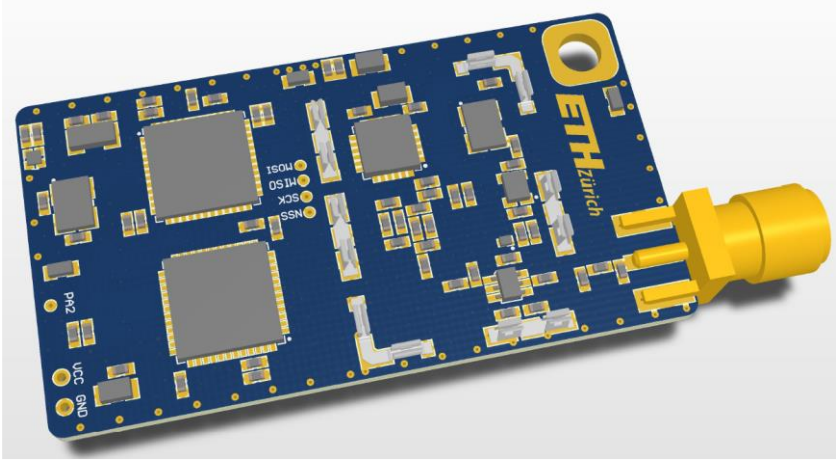
Master Thesis Project
For
Michael Keller

Supervisor: Jan Beutel, Roman Trüb

Start Date: October 1, 2018
Initial Presentation Date: xxx xx, 2018
Final Presentation Date (tentative): xxx xx, 2019
End Date: April 1, 2019

LWB with Long-Range Modulation

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.



This thesis project aims at evaluating and extending the previous investigation on the next-generation data collection protocol running for the new communication board of the Dual Processor Platform (DPP). The features of long-range, low-power sub-GHz RF transceiver by Semtech (SX1262) shall be exploited and incorporated into the communication scheme. The work bases on the existing Glossy/LWB approach and aims at obtaining a more parametrizable implementation and evaluating this implementation in the context of different extensions to Glossy which makes it more adaptive to a large range of application scenarios. Extensive evaluations shall be performed with the real hardware using the FlockLab testbed environment.

Tasks

- Formulate a time schedule and milestones for the project. Discuss and approve this time schedule with your supervisors.
- Implement a layer, inspired by LWB, on top of Gloria (port of Glossy by Markus Wegmann) on the SX126x LoRa communication platform. The goal is a simple implementation with a limited number of features (rendezvous, slot allocation, selectable modulation)
- Evaluate the implementation in terms of scalability (10-20 nodes) with FlockLab experiments.
- Work out a scheme to dynamically assign a modulation type (LoRa SFs, FSK) to each node. The scheduling/arbitration should run centralized on a host node. The scheme should cover: bootstrap, fallback, and rendezvous strategies.
- Implement the proposed scheme on the SX126x LoRa communication platform.
- Evaluate the implementation and compare the results to LWB/Glossy.
- 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.

Deliverables

- Time schedule (at the end of first 2 weeks)
- Initial Presentation (5 min)
- Final Presentation (20 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 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 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] 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

[3] 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.

[4] 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.

[5] Sutton, F., Da Forno, R., Gschwend, D., Gsell, T., Lim, R., Beutel, J., & Thiele, L.: The design of a responsive and energy-efficient event-triggered wireless sensing system. Proc. of ACM EWSN, 144-155, 2017

[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.

[8] Markus Wegmann: Reliable 3rd Generation Data Collection. Master Thesis, August 2018

Bibliography

[1] "Permasense," accessed April, 2019. [Online]. Available: www.permasense.ch

[2] "Master Thesis Markus Wegmann," accessed April, 2019. [Online]. Available: https://gitlab.ethz.ch/tec/students/projects/2018/ma_mwegmann

[3] "Semtech SX1262 Transceiver," accessed April, 2019. [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1262

[4] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with Glossy," in *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, April 2011, pp. 73–84.

[5] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power Wireless Bus," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '12. New York, NY, USA: ACM, 2012, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/2426656.2426658

[6] "STM32L443CC Microcontroller," accessed April, 2019. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32l433cc.html

[7] Semtech. (2015, May) LoRa^{TM} Modulation Basics. Document Version 2. Accessed April, 2019. [Online]. Available: https://www.semtech.com/uploads/documents/an1200.22.pdf

[8] "Flocklab," accessed April, 2019. [Online]. Available: https://gitlab.ethz.ch/tec/public/flocklab/wikis/home

[9] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and Efficient All-to-all Data Sharing and In-network Processing at Scale," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '13. New York, NY, USA: ACM, 2013, pp. 1:1–1:14. [Online]. Available: http://doi.acm.org/10.1145/2517351.2517358

[10] M. Mohammad, M. Doddavenkatappa, and M. C. Chan, "Improving Performance of Synchronous Transmission-Based Protocols Using Capture Effect over Multichannels," *ACM Trans. Sen. Netw.*, vol. 13, no. 2, pp. 10:1–10:26, Apr. 2017. [Online]. Available: http://doi.acm.org/10.1145/3043790

[11] C. Liao, G. Zhu, D. Kuwabara, M. Suzuki, and H. Morikawa, "Multi-Hop LoRa Networks Enabled by Concurrent Transmission," *IEEE Access*, vol. 5, pp. 21 430–21 446, 2017.

[12] T. Istomin, M. Trobinger, A. L. Murphy, and G. P. Picco, "Interference-resilient Ultra-low Power Aperiodic Data Collection," in *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*, ser. IPSN '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 84–95. [Online]. Available: https://doi.org/10.1109/IPSN.2018.00015

[13] R. Jacob, J. Bächli, R. Da Forno, and L. Thiele, "Synchronous Transmissions Made Easy: Design Your Network Stack with Baloo," 2019, 16th International Conference on Embedded Wireless Systems and Networks (EWSN 2019); Conference Location: Beijing, China; Conference Date: February 25-27, 2019.

[14] "Flora Git Repository," accessed April, 2019. [Online]. Available: https://gitlab.ethz.ch/tec/research/dpp/software/communication_platforms/sx126x_lora/flora

[15] "Flocklab Pin Assignment," accessed April, 2019. [Online]. Available: https://gitlab.ethz.ch/tec/public/flocklab/wikis/Man/GpioAssignmentTargetAdapter

[16] "Flora Tools Git Repository," accessed April, 2019. [Online]. Available: https://gitlab.ethz.ch/tec/research/dpp/software/communication_platforms/sx126x_lora/flora_tools

[17] "Flora Tools Development Fork Git Repository," accessed April, 2019. [Online]. Available: https://gitlab.ethz.ch/tec/research/dpp/software/dev_forks/ma_kelmicha/flora_tools

[18] "Master Thesis Git Repository," accessed April, 2019. [Online]. Available: https://gitlab.ethz.ch/tec/students/projects/2018/ma_kelmicha.git