Low-power network design: work hard, play hard

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Low-power network design: work hard, play hard

Semester Thesis

Jan Müller
muelljan@student.ethz.ch

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

Supervisors:
Romain Jacob
Reto Da Forno
Prof. Dr. Lothar Thiele

January 18, 2019

Acknowledgements

My thanks go to Romain Jacob, who has shown what it can mean to supervise a project. Your critical thinking and you expertise helped us from the first to the very last stage of the project. Thanks for giving us the opportunity to do this project.

Big thanks also to Anna-Brit, my team-mate. It was nice working with you and discussing the problems that we both struggled with.

Big thanks goes also to the TEC group at ETH for providing a robust testing infrastructure and the opportunity to represent ETH at the EWSN dependability challenge.

And last but not least, thanks to my room-mates which endured me wandering through the flat during the late shifts for this project. I hope you could sleep fine nonetheless.

Abstract

Competing in a well-known competition in a field that you have no experience in can be hard for sure. In this semester thesis, we describe how we managed to design a low-power wireless protocol for a sensor network in less than 3 months of part-time work. Baloo is a novel protocol design framework, which offers an abstraction layer from the underlying hardware and enables fast protocol prototyping. Using Baloo, our goal was to create a protocol to compete in the annual EWSN dependability competition by borrowing different design ideas from various protocols described in literature.

Contents

Chapter 1

Introduction

Years of research has so far not produced a one-fits-all protocol for wireless sensor networks (WSNs). Instead, there are many protocols described in the literature that are usually designed for a very specific use-case. Often, protocols are tuned to perform very well for the scenario they were designed for but will behave unexpectedly if the design-time assumptions don't hold anymore. One of the goals of this year's EWSN dependability competition is to be able to compare different protocols to each other in various different scenarios. For 4 years, this competition has taken place along with the International Conference on Embedded Wireless Systems and Networks (EWSN) and attracts various teams from academia and industry. The competitors have to design a protocol for a wireless sensor network, which competes with the protocols of other teams in terms of reliability, latency and power consumption. This year's edition of the competition features a wide range of different scenarios that should prevent the protocol designers from over-optimise their design. Instead, their protocol has to adapt to an unknown scenario. With this in mind, we think that this year's challenge is highly interesting from an academic point of view and it will be interesting to see if the winning protocol is a step towards a generalised protocol or if it consists of different sub-protocols that are chosen according to the current scenario.

Chapter 2

Background

In this chapter we give a very brief overview over the protocols and ideas that already exist in the literature. After that, we give a brief introduction to Baloo, the framework that was used to design our protocol.

2.1 Design space exploration

Communication protocols in WSNs face various limitations and challenges. The major problem arises from the nature of the communication media. If multiple nodes start sending data at around the same time, the probability of a successful reception of the data is reduced for devices that are in range of more than one transmitting node. This results in the necessity to have a network wide arbitration to prevent or reduce the number of data collisions. There are two general trends to achieve this. There is a more traditional circuit based approach, in which nodes along a path are woken up and data is transmitted along this path. The second approach was made popular by Glossy [4]. These protocols rely on synchronous transmission or the capture effect. Data is forwarded in floods. This offers native support for one-to-many communication schemes. Because of the combined transmission power of the different nodes and the introduced redundancy, this approach usually offers a higher degree of reliability. There are also hybrid approaches, where data is flooded along certain paths. A selection of different protocols is illustrated in fig. 2.1 along with references to the literature describing them.
Glossy and Chaos can be regarded as primitives many other protocols rely on. Glossy uses constructive interference, that allows the radio chip to correctly decode a message, even if it was sent by multiple nodes at the same time. The condition is that all nodes send the same data and that the packets arrive synchronously at the receiver. With Glossy, data is disseminated in floods. Chaos uses the capture effect and allows fast network aggregation of data that needs multiple nodes to cooperate, for example, to compute a maximum value of all current sensor data. Of all protocols mentioned in fig. 2.1, Crystal had the

Figure 2.1: This is a collection protocols for wireless sensor networks, grouped into 3 categories.

most impact on our work. It uses a round-based transmission/acknowledgement scheme of variable length to increase the reliability and uses a distributed termination algorithm to let the nodes know when the communication ends. To tackle interference, Crystal changes the radio channel for each new transmission/acknowledgement pair. Additionally, it uses a noise detection mechanism to be able to change its transmission scheme according to the amount of interference. A2 uses Chaos to have fast network-wide consensus using the capture effect. Less is More shows how the power consumption can be reduced using topology discovery and a learning algorithm. The idea is to turn off or reduce the transmission power of nodes that are redundant for reliable communication.

Common performance metrics for communication protocols in WSNs are latency, power consumption and reliability. Latency describes the delay between the data generation at a sensor and the delivery of the data at a destination. Power consumption matters because WSNs are often running from batteries and deployed in remote locations and should therefore run as long as possible from their energy storage. Reliability describes how much of the data that is collected

actually reaches its destination. The problem that a protocol designer faces is that an improvement of one of these performance criteria often has a negative impact on one of the others. For example, the power consumption of the network can be minimised by sending data in batches once a day. This results in a high latency in the orders of hours. On the other hand, one can have the network always in a ready state and forward data as soon as it is received. While this allows a latency in the order of milliseconds or below, this approach would use a lot of energy. One key task of the designer is therefore to find a sweet spot in the design space that fulfils some weighted representation of the performance metrics and additional criteria that he might have.
We have collected ideas from literature to improve the performance criteria in fig. 2.2. We filtered very specific ideas to have a set of general approaches and used this as a reference during the design process. Not all the ideas are applicable to our use case. For example, pipelining and coding is often used for large data transfer, like the dissemination of a new firmware.

Figure 2.2: This is an overview of ways to improve certain protocol performance metrics.

2.2 Baloo

The implementation of a newly designed protocol often requires a large degree of low-level programming. This means that even testing small protocol improvements can be time-consuming. To facilitate the design of new protocols, researchers presented different abstraction layers with the goal to create a network stack to isolate the low-level mechanics from the protocol logic. One recent framework is Baloo[7].

The protocol described in this thesis uses this novel Baloo framework. This is the first time, it was used outside the initial developing group and by people inexperienced in protocol design. For this reason, we add a short usability evaluation in section 4.3. We focus on the general design principles of Baloo here and invite the interested reader to read on in the aforementioned paper.
The design of Baloo is a balancing act between ease of use and configurability. A network has a single Baloo host node, which can be used to distribute state or scheduling information in the network. This is done using control packets, which are also used to synchronise the network. To join the network, a node has to receive one of these packets to get the scheduling information and to be able to synchronise its local clock to the host. This is important for synchronous transmission to work. After that, the nodes wake up and communicate according to a configurable schedule that is disseminated by the Baloo host. Communication takes places in rounds of data slots. The developer can choose from a set of communication primitives such as Glossy or Chaos for each slot. Baloo adds an abstraction layer to the underlying hardware and allows protocol designers to focus on the protocol instead of low-level programming of the radio. Low-level changes and optimisations can be made through a rich set of parameters without the need for more than a base understanding of the implementation principles.
The benefit of using Baloo is that it allows an easy integration and combination of different communication primitives and enforces isolation of these primitives from the protocol logic. They can be enhanced or changed and new primitives could be tried out without the need to change the more high-level protocol because the interface to the protocol would be fixed. Additionally, it facilitates protocol design because one can immediately start with known-working primitives and a predefined structure for the implementation.

Chapter 3

Protocol Description

In this chapter, we describe the design of our protocol and highlight a part of the reasoning behind the ideas. This chapter mainly discusses the protocol in the periodic case. The changes needed to support aperiodic data generation are described in section 3.6.

3.1 Scenario and Design Goals

The 2019 edition of the EWSN dependability challenge was about designing a protocol which should be able to adapt to a wide range of parameters. The challenge was held in a testbed at the TU Graz. Up to 70 independent wireless nodes were placed over several floors of the institute of technical informatics of TU Graz in an area of approximately $1440m^2$. While the exact layout for the evaluation was not known in advance, the preparation testbed featured a setup where the network diameter was at least 8 hops on full transmission power on the Telos B replicas used in the testbed.
There were two independent categories. This thesis is about a protocol for the use in the second scenario which was about data dissemination. In this scenario, there are up to 8 nodes generating data packets independently, which have to be delivered to up to 8 destination nodes per source. The rest of the nodes in the network act as relays. The challenge is that there is a wide range of possible input parameters: the payload can be anything between 8 and 64 bytes. There is a periodic case in which data is generated with a period of at least 5s. There is an aperiodic case in which one only knows a lower and an upper bound for the data generation interval. And finally, there can be node failures and different degrees of jamming in the network. Because of the large number of possible combinations of these parameters, the protocol needs to be flexible and adapt itself to the scenario.

As stated by the organisers of the competition, the main performance criteria in the competition were reliability, power consumption and the latency between

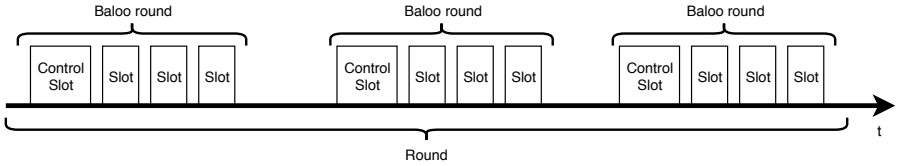


Figure 3.1: Overview of the layers of granularity. A communication round is composed of different Baloo rounds which in turn consist of a number of slots.

data generation at the source and the delivery at the destinations. In the evaluation, the reliability is weighted most strongly. We mainly focused on the periodic case and tried to first achieve high reliability and then tried to reduce the latency and power consumption. Our goal was not to optimise for only one of the metrics, but to achieve a decent, although not the best possible, performance for all of them.

3.2 Protocol description

One important aspect of the scenario is that the topology is uncertain at any given time because of two reasons: First, it is assumed that the protocol has no prior knowledge of the topology at boot. Second, due to interference and the possibility of node failures, from a connectivity point of view, the topology could change from one instant to the next. For example, wide-band jamming could effectively prevent a whole part of the network from communicating and effectively isolate the nodes in this part of the network. While a topology discovery mechanism could be used to resolve the first problem, the uncertainty of the second problem made us design a flooding based protocol instead of a connection based routing protocol. Flooding based protocols are less affected by topology changes as only few assumptions on the underlying topology are made. Additionally, as the scenario is describing a many-to-many communication pattern, a flooding based approach seemed reasonable because of the one-to-many characteristics of network floods.
The task we were given was to use the Baloo framework for developing the protocol. This allowed us to focus on the protocol and to be able to rely on known-working implementations of Glossy and Chaos. Additionally, we expected to achieve a low energy footprint by using Baloo because we would not have to take care on sleep modes and radio on/off states. Being relatively inexperienced with embedded software design, this was a big plus.

To start the description of our protocol, we first want to introduce the terminology that we will use in the rest of the chapter. We distinguish different types of nodes: *Source nodes* produce data periodically and send it to *destination nodes* associated with it. There is a single static *Baloo host* that is in charge of time

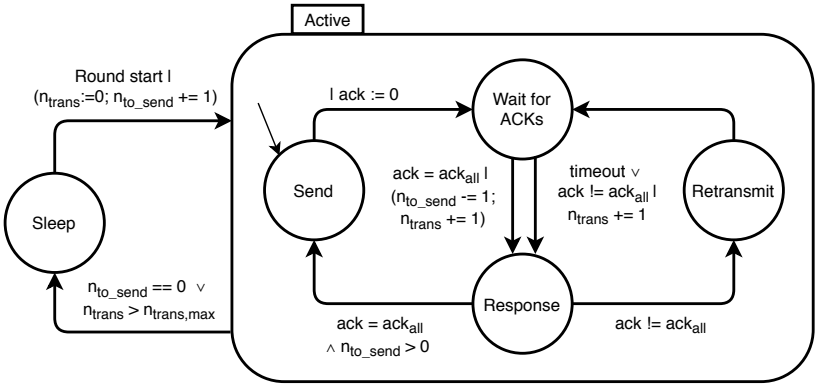
synchronisation and maintains network-wide state. Fig. 3.1 shows an overview of a communication round. The *communication round* or simply *round* is the period of the whole network and matches the period of the data generation in length. We call the duration of a round a *period*. During one round, every source should have the possibility to send its data to its destinations. This is done in different *Baloo rounds*, which each have a maximum duration. We call the nodes communicating in one Baloo round a *cluster* for reasons that will become clear when reading on. We will often use the terms Baloo round and cluster interchangeably. In a Baloo round, a single source sends data to all its destinations. This means, that a round consists of maximally 8 Baloo rounds, because there are at most 8 data sources. If a node is not source or destination in a cluster, we call it a *relay node* for this cluster which simply relays packets during floods. The Baloo rounds themselves consist of different *slots* in which one communication primitive is used. In this project, the primitive is either Glossy or Chaos. In the protocol that we handed in for the competition, only Glossy-slots were used. For the protocol description, most of the time a slot can be seen as best effort atomic network operation after which nodes have received the data if the operation was successful. This will be the lowest level of abstraction used for the protocol description.

The communication itself uses a regular transmission/acknowledgement pattern. In a Baloo round, the associated source starts a network-wide flood with the data it has to send. After that, each of its destinations send a positive or negative acknowledgement indicating if they have received the data or not. These acknowledgements are also flooded in the network. If the data was not received by all destinations, the source retransmits it. There is a maximum number of retransmission slots within the Baloo round to make sure that the time budget is not exceeded and the Baloo rounds do not overlap in time. If a transmission could not be completed, the source retries in a later round. It always tries to deliver the data it generated in the current round first. If this is successful and if there is still time left dedicated to the current Baloo round, the source tries to deliver past data that was not successfully transmitted. The behaviour of the sender/receiver is illustrated in fig. 3.2.

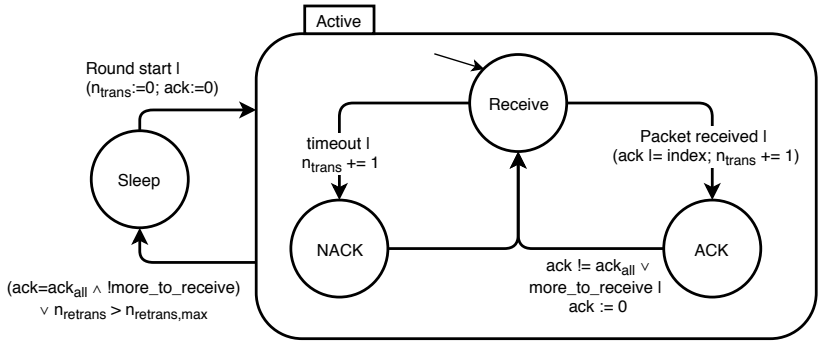
The general protocol was kept simple to limit the overall complexity. In the next sections, we will highlight some more detailed aspects and enhancements of the protocol. The idea was to start with something simple and then add more complex optimisations where it is appropriate.

3.3 Latency Improvements

This section describes how we managed to minimise the latency by adapting the protocol according to the testbed at runtime.

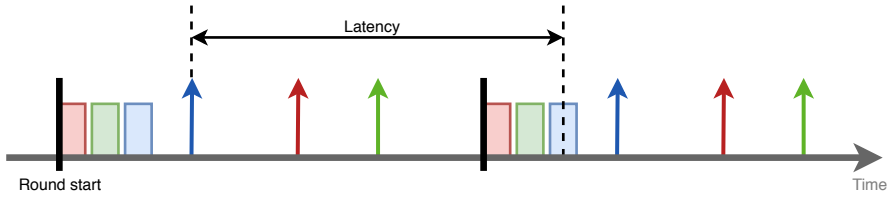


(a) This diagram describes the behaviour of a sender. It sleeps most of the time. If a round starts, it tries to send the current data first and then waits for feedback from the receivers. If the data packet or the acknowledgement is lost, the sender tries to retransmit the data. Otherwise, it tries to send data that it didn't manage to send in previous rounds. There is a maximum number of transmissions in each round.

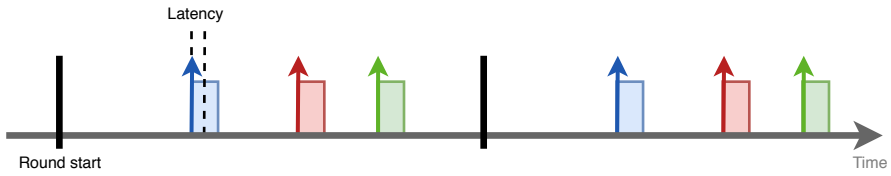


(b) This diagram describes the behaviour of a receiver. It sleeps most of the time. If a round starts, it listens for incoming data packets and returns a positive or negative acknowledgement depending on the transmission being successful of not. If not all nodes have acknowledged the data or if the sender indicated that there are more packets following, the receiver goes back to the receiving state.

Figure 3.2: StateCharts of the logic of the sender and the receiver. The arrows use a *trigger* |*effects* notation.



(a) Initial communication scheme.



(b) Latency-optimised communication scheme.

Figure 3.3: Illustration of the offset compensation. (a) shows the large latency for a possible initial situation. The black bars denote the start of a round. The arrows represent data generation and the blocks are used for the Baloo rounds. Blocks and arrows of the same colour are used for communication and data generation of the same source. (b) shows an optimal schedule with minimal latency.

3.3.1 Offset compensation

To minimise latency, the period of the communication rounds matches the period of the data generation. This makes sure that there is only communication when there is data to send and allows us to send the data immediately after it was generated. For this, the protocol assumes initial knowledge of the period of the data generation. However, the data generation is not synchronised on the source nodes, which means that every source has a different initial offset (while maintaining the fixed data generation period) and that not all sources generate data at the same time. If we assume the optimal case in which every data packet can be delivered at the first try, we still get a worst-case mean latency that is larger than the period if we schedule the clusters randomly. This is the case if the cluster is scheduled just before the data is generated. By creating an offset-aware schedule, this worst-case can be highly improved. Figure 4.1 shows a possible initial situation and how the schedule can be optimised to minimise latency.
As we don't know the offsets initially, we start with a random schedule. When the first data arrives at the source nodes, they save the offset between the moment the data arrived and the moment they read it. During the first few periods, they piggy-back this offset information on the data packets, effectively disseminating the information to the whole network. Note that they measure the offset only in the first round, to make sure that the information is consistent on all nodes after the first few rounds. This way, every node in the network learns the offset of

every source. Because every node has the same information, they can compute a new schedule locally. We use a distributed approach instead of a local one because this way, we don't need to solve the problem of disseminating the new schedule reliably to every node of the network. The schedule is computed in earliest release-time first fashion, because it is fast to compute since it only requires sorting the offsets of the clusters. While this algorithm does not offer an optimal mean latency, it worked reasonably well in our evaluations.

We show under which conditions our greedy algorithm is guaranteed to find a schedule. A schedule is valid if it contains all n configured clusters within a period. We denote the fixed maximum Baloo round duration as t_p. In the worst case, the release times are separated by $2 \cdot t_p - \epsilon$. This would mean, that we cannot schedule a cluster between two release times. For this case, the minimum period P_m is given as

$$(n-1)(2 \cdot t_p - \epsilon) + t_p \leq (2n-1)t_p \leq P_m \tag{3.1}$$

As long as this equation holds, we are guaranteed to find a schedule. As the period and the number of clusters is given at the start of the program, eq. (3.1) defines the maximum Baloo round duration and therefore also the maximum number of retransmissions in a single Baloo round. The maximum number of retransmissions is computed such that the time needed for a Baloo round does not exceed its budget.
Latency-wise, the worst case would be when all sources generate packets synchronously. In that case, the best mean latency L_m that we can achieve is lower-bounded by

$$L_m \geq \left(\frac{n-1}{2}\right) \cdot t_p \tag{3.2}$$

However, due to interference and resulting retransmissions, the expected latency is higher. In the current protocol, we make a trade-off between the probability of the data to be delivered within a single round and the duration of a Baloo round. By increasing the number of possible retransmissions, we make the Baloo rounds longer which leads to longer latencies according to eq. (3.2). On the other hand, if we reduce the number of retransmissions, the probability of unsuccessful Baloo round increases which also leads to higher latencies because the data has to be transmitted in a later round. In the end, our approach was to fix a maximum Baloo round duration and to set the maximum number of retransmissions such that this maximum is not exceeded. Note that we also had to take the payload length into consideration for computing the allowed number of retransmissions, because slots are longer if there is more data to send.

3.3.2 Period correction

The period of the data generation is known in advance and the protocol period is set accordingly. However, during testing, we noticed that the actual period

differs from the one in the configuration. It seemed that the testbed introduced a payload dependant delay of more than 20ms during data generation that was added to the data generation period. For example, if the specified period was 5000ms the period in which the data was generated could be 5021.6ms. Without compensation, this would lead to a considerable drift of the data generation events with respect to the schedule. To overcome this issue, we used a two-step approach: first, we measured the introduced delay for different payloads. With the measured data we computed an offline approximation of the shift depending on the payload size. We add this approximation to the period that is advertised by the testbed at boot. This allowed the reduction of the drift to a sub-microsecond area. After the offset compensation described in section 3.3.1, the Baloo host periodically measures the change between the initial (after the rescheduling) and the current offset and piggy-backs this value on the synchronisation packet. This way, the nodes learn the value of the current drift and can compensate it.

3.3.3 Latency upper bound

To be able to retransmit data if the transmission was not successful during a single Baloo round, the source nodes store the data packets. They use a ring-buffer and store the 8 most recent received/sent packets. We limit the number of stored packets to 8 to have an upper bound on the latency and guarantee some degree of freshness of the delivered data. We guarantee an upper limit L_u on the latency depending on the period P as

$$L_u \leq 9 \cdot P \tag{3.3}$$

This is a trade-off between reliability on one side and latency and power consumption on the other side. More retransmissions would allow to have always perfect reliability but the energy cost and the latency would eventually become large. By defining a maximum amount of latency in terms of rounds, we have a good amount of reliability while not having a too large performance penalty in terms of power consumption and latency.

3.4 Energy Improvements

In this section we describe different mechanisms that helped to reduce the power consumption of our protocol.

3.4.1 Clustering

At one point during the preparation for the competition, we learned that during the first minute of the experiments, there would be no data generated and the

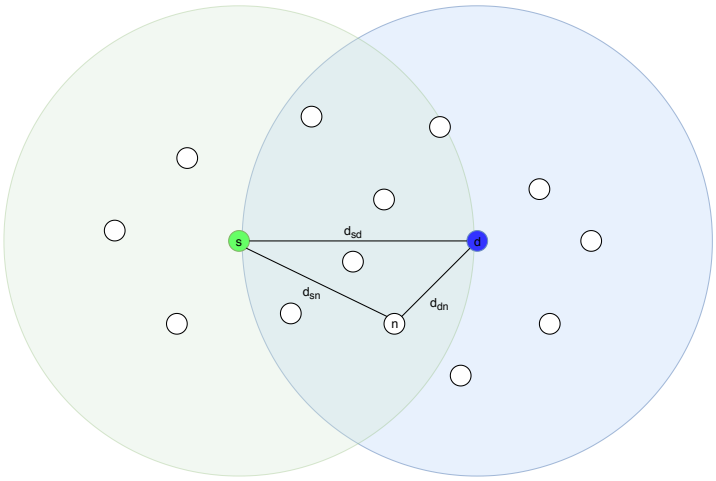


Figure 3.4: Illustration of the clustering algorithm used. A node is part of a cluster, if it is on a quasi-direct path between a source s and a destination d. In the extreme case where the threshold is 0, this would include all nodes that are at most d_{sd} away from the source s and the destination d. This can be interpreted graphically as the nodes within the intersections of two circles of radius d_{sd} from s and d.

power consumed during the first minute would not be taken into account when evaluating the energy metric. This means that the first minute can be used for topology discovery. To save energy, one straightforward approach is to turn off nodes that are most probably not needed for a successful transmission. Examples for such nodes would be the ones at the edge of the network. Our approach was to turn off nodes on a per cluster basis. The goal is to create a group of active nodes per cluster, containing the source and the respective destinations, as well as the relay nodes on a quasi-direct path between the source and any destination of this source. By quasi-direct, we mean that we don't only want the shortest path but want also to include almost shortest paths up to a threshold of additional length. By now, it should also be evident why we used the term *cluster* to describe the active nodes during a Baloo round.
To create the clusters, we made the nodes detect their relative position to the source and the destinations. For every source/destination-set, the nodes note their distance from the source (d_{sn}) and the respective destinations(d_{dn}). We use the hop count as a measure of distance. The nodes get this information directly out of the header of the Glossy packets that are exchanged. The destinations also broadcast their distance from the source (d_{sd}). Every node now knows 3 things: it's distance to the source and the destination and the distance between source and destination. With these 3 values and a configurable threshold, the nodes can decide if they are part of a cluster using the following criteria:

$$d_{sn} \leq th + d_{sd} \tag{3.4}$$

$$d_{dn} \leq th + d_{sd} \tag{3.5}$$

If eq. (3.4) and eq. (3.5) hold for any of the destinations in the communication pattern, the node should participate in this pattern. To get some interference resilience, the nodes default to participate in all communication patterns, and they only turn off, if they have multiple measurements that come to this conclusion. This should prevent the case in which a lot of packets are lost during the clustering and too many nodes turn off. Note that we assume fixed node positions for this approach. A graphical explanation of the criteria is depicted in fig. 3.4. One thing that we observed empirically is that it can be beneficial to reduce the transmission power during such clustering tasks, because this increases the number of hops in the network which leads to a more fine-grained clustering.
Besides clustering, in this first phase, we make sure that we have a high probability that all nodes join the network. In order to join a Baloo network, a node needs to receive a single control packet. We therefore make sure that we send control packets at a higher rate than during later stages of the protocol.
Unfortunately, this optimisation was not added to the protocol that we handed in for the competition. The reason is that we were not sure about its impact on the reliability of the Glossy floods. With the limited time that we had available on the evaluation testbed, we could not collect enough data for all possible scenarios to add this addition confidently. Such an optimisation, when added prematurely, can lead to an increase in power consumption. If the reliability of Glossy floods is reduced, there are more retransmissions which lead to more radio-on time. This means that an optimisation that is designed to reduce the power consumption may need more energy in the end. However, first evaluations that are described in section 4.2 would look promising.

3.4.2 ACK optimisation

When implementing the acknowledgement scheme, we had 3 possible candidates. First, we had a Chaos based system. With this approach, the nodes send packets which contain 2 flag bytes. One of them is used for the actual acknowledgement and the other is used as a consensus progress indicator. In the first byte, the nodes can indicate if they want to acknowledge a packet. In the second byte, every destination that added its acknowledgement information to the packet sets its flag to 1. One node now starts the slot by advertising its acknowledgment information. If one of the other nodes receive the packet, it adds its own acknowledgement information, sets its flag in the progress byte and forwards the packet. Gradually, the information in the packet increases and when all bits in the progress flag are set, a final flood ensures that the whole network has the same information. According to [10], this scheme can complete faster than multiple floods on large networks.
The second approach is entirely Glossy-based. Each destination is assigned a

slot in which it broadcasts its acknowledgement information to the network.
The third approach works exactly as the second one, but there is an additional slot in which the source broadcasts the information of what acknowledgements it has received. The intention behind this is to prevent single nodes from going to sleep too early. If an acknowledgement is lost but a part of the network managed to receive it, it can happen that a part of the network decides that the round is over and goes to sleep. The other half of the network tries a retransmission which is not needed and which will also not be successful because the node, that did not receive the data according to the information that the source has, has already turned off its radio.
We first tested the Chaos based approach, because the use case it was designed for, seemed to match exactly our acknowledgement collection problem. First tests on flocklab looked fine but we had issues getting it to work on the larger competition testbed. Chaos offers no completion guarantee. This means that we are not sure if a Chaos aggregation finishes or will time out. To have some reliability, we needed to increase the duration of the acknowledgement slot up to a point where the slot was longer than multiple successive Glossy slots. For this reason, we decided to use one of the Glossy based approaches because we wanted to keep the Baloo rounds short.
To decide if we should add the extra acknowledgement slot described in the third approach, we've run experiments on flocklab to check if the extra slot has a positive impact on the power consumption. We measured the radio-on time for both approaches on flocklab and came to the conclusion that the extra slot has a negative impact on the radio-on time. Therefore we settled for the second approach and used a single Glossy flood per destination.

One way of reducing the power usage is to reduce the number of unnecessary retransmissions. One source of unnecessary retransmissions are acknowledgements that don't reach the source. The source will interpret the missing acknowledgement as packet loss to maintain reliability and eventually trigger a retransmission. To reduce this risk, all destinations keep track of who acknowledged a packet. Instead of simply sending a 0 or 1 to signal that they received a packet, every node has an index and sets the bit at this index in a bit vector to 0 or 1. With this, if an acknowledgement gets lost between a destination and a source but is received by other destinations that are scheduled later, an unnecessary transmission can be avoided.

3.4.3 Gap time optimisation

Between two slots, Baloo reserves time for computational tasks and for preparing the payload for the next slot. This time is configurable at runtime but is fixed for the duration of one Baloo round. However, we found this to be inadequate because we have more time-consuming tasks after certain slots. For example: after the first slot in which the source sends data, additional time is needed

because the destinations have to write the data to a bus to deliver it. This task is time-consuming compared to simply toggling a bit between the acknowledgement rounds. We made a small modification to Baloo that allows us to set the gap time for the next slot during runtime. This gives us more fine-grained control and allows us to reduce the overall Baloo round time.

3.5 Reliability improvements

In this section, we present our approach for having a network-wide consistent state and how we deal with interference.

3.5.1 Reliable state switching

With the optimisations described above we have different protocol states in which we have different schedules (see section 3.3.1 and section 3.4.1). We need a way to consistently switch between these states in the whole network, otherwise there would be chaos in the network. The fact that the underlying network is lossy makes this task more challenging. We cannot use a single Glossy flood to signal phase transitions because there is no guarantee that the messages are received at every node. We did not want to use a separate round to get a reliable consensus as described for example by A2[1], because this would generate larger power consumption overhead. Instead, we added additional information to regular control packets and boost reliability with additional state-keeping at the nodes. The tradeoff for the simpler design is a longer transition phase from one state to the next.
We use the Baloo host as a network wide controller. Besides the tasks of an ordinary source node, the Baloo host is responsible for network synchronisation. The Baloo host sends a control packet at the start of each Baloo round. This control packet consists of a single byte and contains the current phase, as well as phase-dependant additional information. In the bootstrap (see section 3.4.1) and learning (see section 3.3.1) phases, the Baloo host sends a decreasing counter indicating the number of rounds until the next phase transition. Every node knows the predefined initial value of this counter and decreases it every round. If a control packet is received, the nodes compare their value with the value in the control packet. This mechanism allows to resyncronise nodes that got out of sync and is especially useful in the bootstrap phase, as not all nodes join the network at the same time. With this, the Baloo host could also react to different network conditions and for example signal a shorter or longer synchronisation phase than the predefined phase duration. With this, the nodes don't have a strong dependency on the control packets and only need to receive very few of them. In a scenario where the phase durations stay as preconfigured, they only need to receive a single control packet to guarantee state synchronisation.

However, due to clock drift, some control packets would still need to be received in order to maintain the time synchronicity needed for synchronous transmission.

3.5.2 Fighting Interference

To route traffic around interference, we use a system similar to the one used in the Crystal protocol in [6]. We rely on channel hopping and hop between 4 802.15.4 radio channels and make sure that the respective next frequency is sufficiently far away from the current one in the sequence. To guarantee synchronisation between the nodes, we have a deterministic algorithm that runs on every node to determine the next frequency in the hopping scheme. We switch frequency at every start of a transmission round and on retransmissions. We stay on the same frequency for a single transmission and the following acknowledgement slots. The intention is that if a transmission is successful, the probability is high that an immediately following ACK slot is also successful on the same frequency.
Due to time constraints, no channel hopping was added for the control packets. Because of the limited testing time available towards the end of the preparation, we did not dare to change such an essential part of the protocol towards the end of the preparation phase. This should not be a problem once synchronised, because the protocol is designed in a way such that most of the control packets can be missed without consequences. However, active and heavy interference on channel 26 will hurt the bootstrapping process of the network. We hope that by using many retransmissions during the Glossy flood and by having many rounds with equally many control packets during the first stage of the protocol, at least one of the floods will be successful. The approach we would have implemented was the following: The network stays on the same frequency for its control packet for the duration of a period and switches frequency deterministically. The Baloo source sends out many control packets during this period during the bootstrapping phase. Nodes that already joined the network, know the current frequency and therefore the next frequency in the hopping sequence. Nodes that have not yet joined the network split the duration of the period into equally many segments as there are channels in the hopping sequence. They listen to incoming control packets on one frequency for the duration of this segment and switch frequency for the next segment. This ensures that during the length of a segment of one period, the network and the node which wants to join it are using the same frequency. It would have to be evaluated if the nodes join fast enough when using 4 channels or if the control packets should use fewer radio channels.

3.5.3 Large Period Reduction

We don't know the values used for the period during the evaluation of the protocol. Due to clock drift, we don't assume the nodes to be able to maintain

synchronicity that is exact enough to allow synchronous transmission if the period is above a configurable threshold. If the data generation period is too large, we artificially split it up and reduce the protocol period accordingly. While there are periods during which there is no data to send, the control packets are sent nonetheless. They are used to resynchronise the network. A beneficial side effect is that this reduces the latency of packets that the sources were not able to deliver on the first attempt.

3.6 Aperiodic case

We mainly focused on the periodic case. In the aperiodic case, data is generated aperiodically. The nodes know the lower and the upper bound of the time between any two data release times on the same source. The exact time is not known and may change for every packet. To make sure that the protocol also works reasonably well in this case we did some minor changes to the protocol described above. First, we removed the offset compensation described in section 3.3.1. The data generation is unpredictable and therefore we can't compensate the offset. Second, we adapt the structure of the Baloo round. We add an additional slot in which the source sends one byte indicating if it has data to send. If a node receives this byte, it can decide if it needs to listen to further packets from the source or if it can go back to sleep early and can skip the rest of the Baloo round. Because this is an optimisation, the nodes default to "on", meaning if they don't receive the flag-message, they assume that the source sent a flag indicating that it has data to send and listens to further packets from the source.
The obvious question is how often the nodes should wake up to check for transmissions. This is a trade-off between power consumption and achievable latency. To get the feeling for the expected mean latency we can achieve with a given period P, we do the following simplified calculation. If we assume that the data generation takes place at a given moment in time, we can assume the probability of the start of the next period to be distributed equally in $(0, P]$ leading to an expected start of the next period of $P/2$. This expected start of the next period can be used as a rough estimate of the mean latency that neglects the time needed to send the data and the latency introduced due to packet loss.

3.7 Further enhancements

Some short-comings and possible enhancements were already mentioned while describing the protocol, and we won't repeat them here. However, there are some other enhancements that we had in mind but that we have not implemented due to prioritising other changes first.

3.7.1 Better Topology Discovery

We don't know the final evaluation topology. This means that we don't know the total size or the diameter of the network and don't know about the local node density. In an enhanced protocol, these values could be learned at runtime and the configuration could be adapted accordingly. At the moment, we assume a large network with a diameter of 12 hops in the configuration. The diameter is an important factor for determining the maximum time needed for Glossy floods to complete. If we measure the diameter, we can reduce the slot lengths which would allow us to save energy in case a transmission is not successful.

By learning about the local density of the network, we could use a more sophisticated forwarder selection mechanism. We could save energy by allowing a number of nodes to turn off depending on their location. If they are in a dense part of the network, there might be no decrease in reliability if some of them are turned off. For example, Zhang et al. show a more sophistic way of forwarder selection using machine learning in [18] that one could use.

3.7.2 Avoid interference

Our approach to interference is to try retransmissions on different channels to find a way around the unusable frequencies. This could be taken to the next level. Instead of per slot frequency hopping, one could think of a scheme in which the frequency is changed within a single Glossy flood as described in [11]. We expect such an approach to offer more reliable floods as they exploit a larger frequency diversity within a single flood.

Another approach to avoid interference would be to learn about "good" or "bad" channels by measuring the amount of interference on different channels. Nodes could decide which channels should be used and which ones should be skipped in the hopping sequence as it is done in Crystal[6].

3.7.3 Allow late joining of the network

If a node is not able to synchronise to the network within some time, we turn this node off completely. While this allows us to save some energy that would otherwise be wasted, this is a dangerous approach. If there is strong interference during the first 2-3 minutes, and too many nodes turn off, no communication is possible any more. To improve the protocol, one could find an algorithm which allows a node to join the network at any point in time and learn the currently used configuration.

3.7.4 Asynchronous start, Synchronous schedule

In the periodic case of the protocol, the nodes learn the offset of the data first and reschedule accordingly. To ensure that every node received the information, the phase in which the offset is not compensated for can harm the performance in the competition. For a real world example, it would not be a problem if the network needs 1 minute at boot until it is able to deliver data at low latency. However, in the competition, where a protocol only runs for some minutes, this can make a difference in the final performance evaluation. The latency could have been improved in this first offset-discovery phase by assuming asynchronous data generation and using the protocol that we designed for the asynchronous case. Then, after the offset is learned, we would switch back to our round-based scheduled algorithm.

Chapter 4

Evaluation

The protocol described in the previous chapter was implemented for the Tmote B platform using the Contiki OS. The source code will be made publicly available together with the scripts used for the evaluation. During the implementation, most evaluation was performed on the testbed at the TU Graz where the competition will be held, as well. The final performance assessment of the protocol will be the competition. The results of the competition is not added to this report because they were not published at the time of writing.
In this chapter, we first analyse the general performance of the protocol with respect to the competition and then show individual evaluations of ideas used in the protocol that were performed during the design phase. We dedicate section 4.3 to a short evaluation on the usability of Baloo.

4.1 Metric-wise performance

In this section, we evaluate the performance of our protocol according to the competition metrics. After that, we will discuss the performance in general, taking all criteria into consideration.

4.1.1 Latency

We tested the offset compensation on the testbed in Graz. We used 5s as period, payloads of 5B and configured the nodes to perform the compensation after 10 rounds. 3 sources and a total of 13 destinations were configured. The experiment run for 8 minutes, including the 1 minute available for the setup. To calculate the latency, the testbed measured the time between the moment data is available at the source and the moment when a destination delivers the data. The offset of the experiment is plotted in fig. 4.1. The compensation is clearly visible at around 50s where the latency drops from around 760ms to around 50ms. There are multiple reasons why the latency does not drop further: first, the source reads the data before the start of the round which means that the time needed

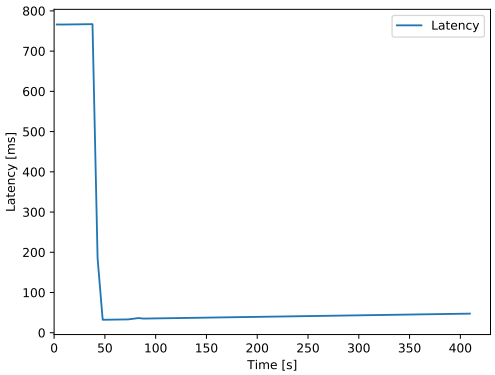


Figure 4.1: This figure shows the mean latency over all 3 clusters and the clear drop of latency after the offset compensation.

for the control packet and reserved to process it adds to the latency. Second, we added a safety margin to make sure that we don't check for new data too early which could result in latency that is roughly equal to the period. Additionally, late retransmissions would also increase the observed latency.
Note that there is a small increase in latency over time. This is due to the issue explained in section 3.3.2 and this experiment was run without compensation for the drift.

We performed the same experiment for the aperiodic case. We used the same setup as above but this time, the data generation period is aperiodic with higher and lower bounds of 5s and 30s respectively. We define a period for the protocol of 3.5s. We achieve a mean latency of 1972ms over the whole experiment. The smallest latency was 236ms and the maximum observed latency was 3433ms. These values are not very satisfactory but are in an order of magnitude that can be expected using the above mentioned period according to the reasoning in section 3.6. By reducing the period, the latency could greatly be improved at the cost of a bit of extra energy. However, this claim cannot be tested as the testbed in Graz is not available any more and it is not in the scope of this thesis to reproduce the capabilities of the Graz testbed in another testbed.

4.1.2 Reliability

To evaluate the reliability during the preparation for the competition, we did measurements both on the testbed in Graz and on flocklab. On flocklab, we took a random topology and measured the packet delivery ratio by comparing the packets sent by the sources and the packets arriving at the destinations. Normally, we get a delivery ratio of 100% meaning that all packets sent by the sources arrived at the destinations. However, there are rare cases where protocol behaves badly. We see this because single nodes miss packets. We were not able

to pinpoint the error as it only appears sporadically. However, there is normally only one node affected, so we assume that the affected node temporarily looses synchronicity to the other nodes for some reason.
On the testbed in Graz, the packet delivery ratio was slightly worse than on flocklab. However, in the end we managed to get a ratio of 95-100% consistently over the last few experiments. This was true even in case of interference.

4.1.3 power consumption

It is hard to have a meaningful evaluation of the power consumption. The power consumption depends on many factors: the payload length, the data generation period, the topology, the number of sources and destinations, the degree of interference, to name a few. This means that a complete analysis of the power consumption of the protocol needs a lot of time and experiments and is clearly out of scope of this project.
During the design phase of the protocol, we continuously evaluated the power consumption on the testbed in Graz. There, an external device measured the power consumption of all nodes. The good thing about this was that we could directly compare our performance to the performance of other teams. In the end, our performance was comparable to the one of other teams. We neither had a much better or a much worse performance energy-wise. One interesting thing that we observed was that reliability is strongly linked to the power consumption. If there are many retransmissions, for example when there is heavy interference or a bug in the code, it can happen that the power consumption increases by a factor of 2, due to the extra time that the radio needs to be turned on.

4.1.4 Conclusion

During the preparation phase for the competition, there was a leader board which contained information about the performances of the different teams. While we cannot say anything about the final ranking and are not sure of the state of the other protocols during their experiments, it seemed that in general our protocol was a little above average for most of the metrics. Our goal was to find a reasonable balance between reliability, power consumption and latency. This means that we did not push to the limits of what is possible for any of the criteria and our protocol is therefore not the best performing in any of the metrics. Other teams might have weighted the criteria differently and may for example offer a better latency at the cost of a higher power usage. It has to be seen what weights the organisers give the individual criteria in their evaluation. In general, we are satisfied with the trade-off that we achieved. We are able to deliver data reliably with only few tens of milliseconds latency with a low energy food-print. The only things that we would like to further optimise are

Experiment	Radio Duty Cycle	Retransmissions
Random without clustering	2.4%	14
Random with clustering	1.6%	14
Designed without clustering	3.6%	17
Designed with clustering	3.2%	17

Table 4.1: Results of the clustering evaluation

the parameters used for the aperiodic case to be able to offer a lower latency there as well.

4.2 Clustering

We test the clustering algorithm described in section 3.4.1 on flocklab. We want to check if the clustering reduces the mean power consumption and if it has an impact on the reliability. For this, we use two different topologies. The first one is just a random topology with 3 sources and 8 destinations. For the second topology, we choose sources and destinations such that they are close to each other, with the intention that this should enable us to shut down most of the network and save a lot of energy. We have 3 sources and 10 destinations in the second topology and we run the protocol once with and once without clustering with both topologies. For the power consumption, we assume that most of the power is used by the radio. Therefore, we compare the radio duty cycles of the different experiments. We measure the duty cycle using the Energest profiler module of Contiki which is a software module that measures the time the radio is assumed to run. We mainly need the result to compare the experiment runs, and therefore this approximation should be good enough. We use the mean of the duty cycles of all nodes to compare the performance. To measure the reliability, we count the number of retransmissions. We run each experiment for 15 minutes.
The results are summarised in table 4.1. The packet delivery rate was 100% for all experiments. From the number of retransmissions, we see that there is no impact on the reliability of the protocol. For the topology that we designed to have a small clusters, we see that we can reduce the radio duty cycle by 33%. As expected, for the random topology we get a smaller reduction in the duty cycle. We observe a reduction of 11%. In general, we conclude with this short evaluation that our clustering algorithm can be used to reduce the mean energy usage of the network.
To give a small glimpse at the clusters that our algorithm creates, we illustrated one of the clusters that was created in the random cluster in fig. 4.2.

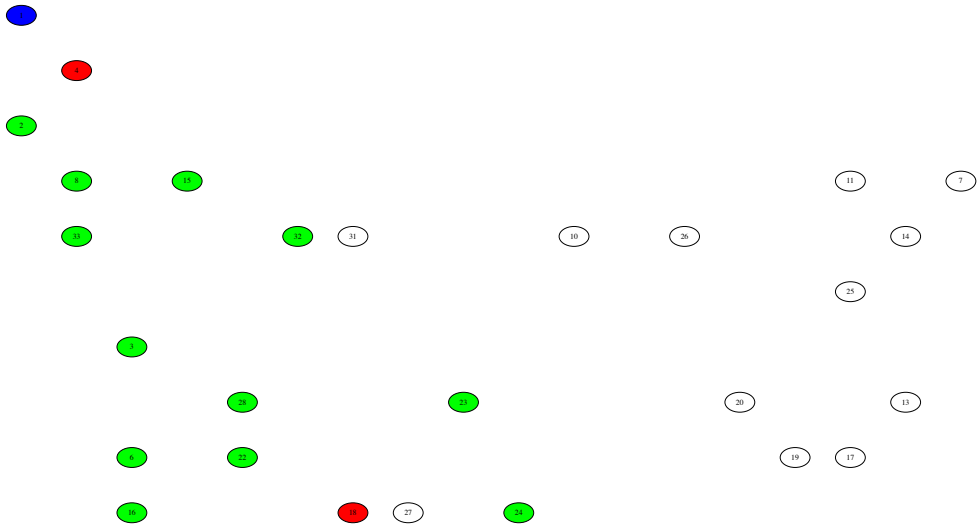


Figure 4.2: Example of a cluster computed by our algorithm in flocklab. The circles correspond to nodes in the flocklab testbed. The blue node is the source and the red nodes are the destinations. Relay nodes that are part of the cluster are marked green. The white nodes are turned off during transmissions of this cluster.

4.3 Evaluation of Baloo

We want to dedicate the last part of the evaluation to the Baloo framework. We were the first people that used Baloo outside the initial developer team. We had not much prior experience in WSN protocol design. While the result is not perfect, we managed implement our protocol within 8 weeks of part-time work. We would not have been able to do that without Baloo.
To get started, the authors provide a collection of example projects that are quite useful for understanding the framework. The examples range from minimal ones to reimplementations of known protocols. This allows you to implement you first toy example without knowledge about Glossy or synchronous transmission. The callback-based design predefines a certain structure of the communication part of the protocol. This is good for beginners, as this acts as a structured skeleton one can build its logic upon. While the first steps are easy, there is a steep learning curve when working on a more involved project. The authors of Baloo made sure that it is as versatile as possible, which means that there are a lot of possible parameters. Currently, the documentation lives mostly within the code, which makes searching for the right parameter to set harder. Additionally, most of the parameters have a default value that might make sense for most protocols but can make your protocol stop working at one point during development if you start to exceed certain bounds. Currently, we see this as the most limiting factor

from a usability point of view. To work around this, we suggest adding a default configuration, that contains all configuration switches and values including a description for each of them. Additionally, it would be beneficial to have more error checking during run- and compile-time. For the inexperienced user, if a tool just stops working without giving an explanation, a lot of time is lost for debugging.
The good thing about Baloo is that once you got some experience, you can be quite fast at changing the communication pattern used. For example, at one point we experimented with a different version of Glossy that does aggressive channel hopping. As it used the interface defined by Baloo, to test it we just had to add the sources and tell Baloo to start using it. This also allows orthogonal development on the communication primitives such as Glossy and the higher level protocol.
Energy-wise, the performance you get from Baloo is quite good, even without long parameter optimisation. Furthermore, the middleware normally provides you with all the information you need for the higher level protocol. The only thing that we needed to change from the upstream Baloo is that we wanted to be able to configure some values during runtime. Thanks to the well-structured codebase, changing the Baloo source was easy.
To conclude, we think that Baloo is a very useful tool for prototyping protocols and communication primitives and makes this task much easier. We are looking forward to further development in this direction.

Chapter 5

Conclusion and future work

5.1 Conclusion

In this thesis we designed a protocol to compete in this years EWSN dependability challenge. We used the Baloo framework and created a protocol that was inspired by different well-known protocols. We relied on a simple round based approach and optimised it in terms of reliability, latency and power usage while maintaining a balance between these performance metrics. With this design principle, it is clear that the protocol is not the best-performing one for any of the mentioned criteria, but it performs reasonably well for all of them.
Being the first group to design a protocol with Baloo outside the initial developing team, we added a short usability evaluation of the framework. In our opinion, it is well suited for the protocol design task. It makes the design and implementation process faster and manages to find good a middle-ground between ease of use and configurability.

5.2 Future work

The protocol that we described in this project was designed for the 2019 edition of the EWSN dependability challenge and is tailored according to the scenarios that we expect there. Additionally, it uses ideas from other protocols and does not introduce something fundamentally new. We therefore don't think that this protocol should be further developed.

Despite participating in the competition, the goal of this project was to see how the Baloo framework is suited to create new protocols. We think it is well suited and should be developed further by adding new communication primitives such as a version of Glossy that does frequency hopping within a Glossy flood. Additionally, the error reporting and the documentation should be enhanced.
We think, that Baloo might be an entry-point to interesting research topics. On one side, such a framework could become a de facto standard. If researchers

would use a common framework to implement their protocols, they could be compared more easily and knowledge transfer could be enhanced. Protocols would be reproducible by others if they know the parameters that were used. Together with a testbed that uses configuration files for creating different scenarios, one could think of a general evaluation framework to assess the overall performance of a protocol.

One of the great things about Baloo is its modular design. It is easy to experiment with different communication primitives. If we take this approach to the next level, protocol design could be made easier by having different common building blocks. For example, if a designer needs a network-wide consensus, he would be able to pick from different modules which each has a slightly different focus on delay, power consumption or reliability. At this point, one could then try to automate the protocol design. An optimisation algorithm could be designed that takes a topology and weighted performance metrics as input and then proposes a new protocol tailored to the specific scenario.

Bibliography

[1] Beshr Al Nahas, Simon Duquennoy, and Olaf Landsiedel. Network-wide consensus utilizing the capture effect in low-power wireless networks. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, page 1. ACM.

[2] Manjunath Doddavenkatappa, Mun Choon Chan, and Ben Leong. Splash: Fast data dissemination with constructive interference in wireless sensor networks. In *NSDI*, pages 269–282.

[3] Simon Duquennoy, Olaf Landsiedel, and Thiemo Voigt. Let the tree bloom: Scalable opportunistic routing with orpl. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, page 2. ACM.

[4] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. Efficient network flooding and time synchronization with glossy. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 73–84. IEEE.

[5] Timofei Istomin, Amy L Murphy, Gian Pietro Picco, and Usman Raza. Data prediction+ synchronous transmissions= ultra-low power wireless sensor networks. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, pages 83–95. ACM.

[6] Timofei Istomin, Matteo Trobinger, Amy L Murphy, and Gian Pietro Picco. Interference-resilient ultra-low power aperiodic data collection. In *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 84–95. IEEE Press.

[7] Romain Jacob, Jonas Bächli, Reto Da Forno, and Lothar Thiele. *Synchronous Transmissions made easy: Design your network stack with Baloo*. 2019.

[8] Jirka Klaue, Angel Corona, Martin Kubisch, Javier Garcia-Jimenez, and Antonio Escobar. Competition: Redfixhop. In *EWSN*, pages 289–290, 2016.

[9] Olaf Landsiedel, Federico Ferrari, and Marco 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*, page 1. ACM.

[10] Olaf Landsiedel, Federico Ferrari, and Marco 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*, page 1. ACM.

[11] Roman Lim, Reto Da Forno, Felix Sutton, and Lothar Thiele. Competition: Robust flooding using back-to-back synchronous transmissions with channel-hopping. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*.

[12] Xiaoyuan Ma, Peilin Zhang, Xin Li, Weisheng Tang, Jianming Wei, and Oliver Theel. Decot: A dependable concurrent transmission-based protocol for wireless sensor networks. *IEEE Access*, 6:73130–73146, 2018.

[13] Mobashir Mohammad and Mun Choon Chan. Codecast: supporting data driven in-network processing for low-power wireless sensor networks. In *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 72–83. IEEE Press, 2018.

[14] Maria Rita Palattella, Nicola Accettura, Mischa Dohler, Luigi Alfredo Grieco, and Gennaro Boggia. Traffic aware scheduling algorithm for reliable low-power multi-hop ieee 802.15. 4e networks. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on*, pages 327–332. IEEE.

[15] Chayan Sarkar, R Venkatesha Prasad, Raj Thilak Rajan, and Koen Langendoen. Sleeping beauty: Efficient communication for node scheduling. In *Mobile Ad Hoc and Sensor Systems (MASS), 2016 IEEE 13th International Conference on*, pages 56–64. IEEE, 2016.

[16] Felix Sutton, Reto Da Forno, David Gschwend, Tonio Gsell, Roman Lim, Jan Beutel, and Lothar Thiele. The design of a responsive and energy-efficient event-triggered wireless sensing system. *Proc. of ACM EWSN*, pages 144–155, 2017.

[17] Yin Wang, Yuan He, Xufei Mao, Yunhao Liu, and Xiang-Yang Li. Exploiting constructive interference for scalable flooding in wireless networks. *IEEE/ACM Transactions on Networking (TON)*, 21(6):1880–1889, 2013.

[18] Peilin Zhang, Alex Yuan Gao, and Oliver Theel. Less is more: Learning more with concurrent transmissions for energy-efficient flooding. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 323–332. ACM.

[19] Peilin Zhang, Olaf Landsiedel, and Oliver Theel. Mor: Multichannel opportunistic routing for wireless sensor networks. In *Proc. of EWSN*, 2017.