Informatics

A Survey of Energy Efficient Multicast Routing Protocols for Wireless Low Power and Constrained Devices

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Stefan Adelmann, BSc
Matrikelnummer 01633044

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Univ.Ass. Dipl.-Ing. Philipp Raich

Wien, 9. Dezember 2021

Stefan Adelmann | Wolfgang Kastner

Informatics

A Survey of Energy Efficient Multicast Routing Protocols for Wireless Low Power and Constrained Devices

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Stefan Adelmann, BSc
Registration Number 01633044

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Univ.Ass. Dipl.-Ing. Philipp Raich

Vienna, 9th December, 2021

Stefan Adelmann   Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Stefan Adelmann, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 9. Dezember 2021

Stefan Adelmann

Kurzfassung

Netzwerke in Wireless Sensor Networkss (WSNs) oder Internet of Things (IoT) werden oft durch eine Mesh-Struktur ausgezeichnet. Dieses generelle Routing-Verfahren wird benötigt, um die räumliche Ausdehnung sowie die Ausfallsicherheit des Systems zu gewährleisten. Diese eingebrachte Redundanz bringt allerdings auch eine gewisse Komplexität in das Routing-Verfahren ein, welches von den typischerweise batteriebetriebenen Niedrigenergiegeräten bewältigt werden muss.
Das Ziel dieser Arbeit ist die Definition, Messung und der Vergleich von Energieeffizienz in WSNs mit Routing Verfahren, welche speziell für die Anwendung in Netzwerken mit Niedrigenergiegeräten und verlustbehafteten Verbindungen ausgelegt sind. Der Fokus liegt dabei speziell auf multicastfähigen Routing Verfahren. Die Resultate und Metriken werden dabei anhand von physikalischer Hardware ausgewertet und aufgenommen, um eine Aussage über die Anwendbarkeit von Energieeffizienz-Metriken basierend auf Simulationen zu treffen. Die Ergebnisse der Arbeit zeigen, dass aufgrund von Hardwareeigenschaften eine Korrelation zwischen Hardware-unabhängigen Metriken und der tatsächlichen Energieeffizienz des Systems nicht immer gegeben ist und von gewissen Größen in Software und Hardware abhängig ist.

Abstract

Networks in WSNs or the IoT rely heavily on mesh-routing in order to achieve their defining spatial distribution and a certain redundancy in the communication. This redundancy, though, introduces a non-trivial routing problem that needs to be handled by typically battery powered low power devices.
The goal of this thesis is the definition, survey, measurement and comparison of energy efficiency in WSNs deploying multicast routing engines in networks defined by low-power devices and lossy connections.
The results and metrics are thereby collected on real hardware, in order to show the applicability of simulation based energy efficiency metrics when comparing them to devices and networks confronted by real-world hardware behavior and influences. The results of this thesis show that with certain settings and configurations of the routing engine and the hardware itself, the correlation between metrics used in hardware-independent studies do not correlate with the actual energy efficiency of the real-world system.

Contents

CHAPTER 1

Introduction

1.1 Motivation

WSNs are becoming an ever-increasing part in the industrial and building automation field. A core part thereof is the wider known IoT that at its heart relies on the communication based on IPv6. For a long time, the introduction of IPv6 on low power devices was hindered by the pure size of the feature set as well as the size of the message header itself. To combat this problem, a workgroup of the IETF introduced IPv6 over Low power Wireless Personal Area Network (6LoWPAN) [MKH$^+$07] in 2007 together with subsequent improvements that enabled low power and low memory devices to encapsulate and compress IPv6 headers. This compressed addressing together with the IEEE 802.15.4 communication protocol can be deployed as the basis for more complex specifications.
When looking at the Open Systems Interconnection (OSI) layer model, IEEE 802.15.4 provides layer 1 and 2, and 6LoWPAN sits in layer 3 [LNJY18], building a base for the upper layers. Both protocols hereby lack the definition of a explicit routing protocol. While 6LoWPAN sits in the routing layer, it itself offers header compression and prerequisites for the addition of routing engines, but no concrete implementation. The range of the targeted low power devices is severely hindered by their energy consumption requirements, which means that alternative ways for increasing the range of a certain network need to be introduced. One solution to this problem is mesh networking. This method provides multiple paths between nodes, which not only leads to seamless range extension but also to increased reliability as more than one path to a certain node can exist. In addition to mesh routing itself, certain use cases of WSN can heavily gain from one-to-many communication, as presented in a work by Philipp Raich and Wolfgang Kastner [RK18]. Use cases for this multicast communication include firmware updates, alert value updates, or general exchange of values that are required for the operation of multiple nodes in the network.

Due to the many use cases of WSN, many routing engines have been developed, creating

an abundance of general and specialized specifications. Since devices deployed in WSN and IoT are typically battery powered, preservation of the charge in these power sources is at the paramount. In order to test these routing protocols regarding their performance in larger installations, simulation can be used. These engines thereby more or less support the emulation of real hardware, or they deploy energy models in order to model the energy consumption of the simulated hardware.
As opposed to the simulation based approach, the goal of this thesis is the definition, survey, measurement and comparison of energy efficiency in WSNs deploying multicast routing engines in networks defined by low-power devices and lossy connections.

1.2 State of the Art

When looking at the term of energy efficiency in WSNs or in routing schemes in general, a number of describing metrics have been defined by other authors.
Sinem Coleri Ergen and Pravin Varaiya [EV05] describe and evaluate two metrics via simulation. The first one is to maximize the minimum lifespan of the nodes in the network, and the second one is to minimize the overall energy consumption of the whole network. In addition, they discuss the influence of two different energy consumption types on these metrics. In particular, they list circuit energy and transmission energy, with the first being the amount of energy spent in the radio electronics while the second one describes the energy spent in the actual transmission by the power amplifier of the radio. In their analysis, they show that for low range transmissions, it is necessary to not only look at the transmission power but also the circuit power as the difference of these two is much smaller in this case than in long range transmissions where a lot of power is needed to boost the signal the required distance. The basis for their conclusion is hereby collected via a simulated system model, highlighting multi-hop communication without regards to actual software running on the nodes.
Going further up from the current draw of the radio circuitry alone, another study by Doku Bandur et al. [BJBJ19] further discusses the influence of different layer/functional parts of the radio transmission on the energy efficiency of a system as a whole. In particular, they look at the current draw of a certain node in different stages, e.g. with the radio in listening or sending mode. By utilizing the Cooja Simulator, they show the influence of different MAC protocols on the radio duty cycle and subsequent average power consumption of the nodes. In regard to energy efficiency, they compare network lifetime, latency and quality based on different unicast routing schemes like LEACH and PEGASIS. The group also discusses the usability of the minimum lifetime metric in their use case of smart agriculture, where they reason that in such systems only certain nodes are needed as key players while the need for power balancing among sheer sensor nodes is not as important.
Both of these studies rely on simulation based results, with the system model of the multi-hop study being a very abstract model and the Cooja simulation which already takes into account the software stack of the running Operating System (OS) but not the real-world behavior of actual hardware nodes.

A comparison between theoretical energy models and real-world hardware is carried out by a group from the Ghent Technology Campus [LCVdPDS20], which focussed on narrowband IoT. In their paper, they present experiments where the power draw of certain nodes is measured over time and the results are then analyzed to show that the real-world hardware and behavior of the system display a less optimistic power profile in comparison to the model. As with the previous paper, the time domain of the overall network configuration is divided into different modes to compare the results of the model with the time and energy spent in the real hardware. In their measurements, the team analyzes the entire lifetime of the narrowband network, from start, to join and network exit.
This thesis, in comparison, aims at combining the approaches of these works in order to evaluate the performance of routing protocols. Taking into account the energy consumption of hardware stages, circuitry, software and the used routing protocol in an experimental manner, in order to improve the evaluation of existing and future routing engines in regard to their energy efficiency.

1.3 Problem Statement

In order to determine if a routing engine running on a particular hardware is energy efficient, a number of facts need to be established. This thesis aims at providing the necessary results and evaluation techniques that are needed to carry out a rigorous evaluation. The overall work thereby covers four main research questions that build towards a definition of the energy efficiency term in mesh multicast routing.

- **RQ1: How is energy consumed by the hardware?**
  In order to evaluate the influence of different parts of both the hardware and the software, it has to be known how much current is consumed at certain points of the node's lifetime and during routing.
- **RQ2: Are hardware-independent metrics enough?**
  Is there a correlation between metrics and evaluations in simulation and the real-world hardware behavior?
- **RQ3: How to introduce hardware behavior into considerations?**
  If metrics do not correlate, developers need a way to introduce the real-world behavior into the energy efficiency considerations in order to determine the best suited routing engine.
- **RW4: What is energy efficiency in mesh-multicast routing?**
  Which metrics need to be applied in order to determine if a routing engine is energy efficient.

1.4 Methodology and Structure of the Thesis

Chapter 2 presents an overview over a hard- and software-independent measurement and evaluation setup, together with network topologies present in IoT use cases. Chapter 3 discusses current multicast routing engines and state-of-the-art operating systems for constrained devices. The hardware used in this thesis as well as the used measurement setup is described in chapter 4. The implementation of the routing engines and the software running and controlling the measurement setup are presented in chapter 5.
Measurement results are collected and evaluated in chapter 6 followed up by a discussion of the results where the recorded data is combined with the theoretical background to form the term energy efficiency in chapter 7.

CHAPTER 2

Design

2.1 IoT-Node

The overall block diagram of a typical IoT or WSN node that is used in this thesis is shown in Figure 2.1. Wireless nodes generally feature a singular Radio Frequency (RF) circuit, consisting of an antenna and a Physical Layer (PHY), which is typically already part of the CPU. The system is powered by an on-board power supply that transforms the external voltage, to the required 3.3V or 1.8V. Depending on the used board, there might be additional periphery connected to the CPU. These parts are omitted from the block diagram, as they are not part of this thesis. In addition to this core functionality, every Unit under Test (UUT) is combined with a measurement node and is instantiated in the test networks as one of two practical nodes, a multicast client that functions as a message source and a multicast server that acts as the message sink. The routing engine is hereby part of the OS running on the CPU. The specific implementations for both the client and the server and their relation to the OS are shown blue in Figure 2.1.

In order to evaluate routing protocols in regard to their energy efficiency, two main data streams need to be produced during the lifetime of the network. Firstly, the routing itself needs to be evaluated in regard to its reception rate, number of packets, redundant packets, etc. Directly sniffing all packets on air is very difficult due to the distances and interferences that occur in real life. Packets are dropped by the UUT but also by the sniffer itself, which is by design required to be part of the same network, therefore experiencing the same limitations. The sniffer would also require an additional communication link in order to be operated by a centralized control entity, i.e., one link that is part of the test network and a second one that is used to start/stop, trigger messages and so on.
Secondly, the current consumption of the UUT must be measured and stored for all devices across the network. This is generating numerous datapoints per time-step, and the size of the data additionally scales with the number of nodes that are deployed.

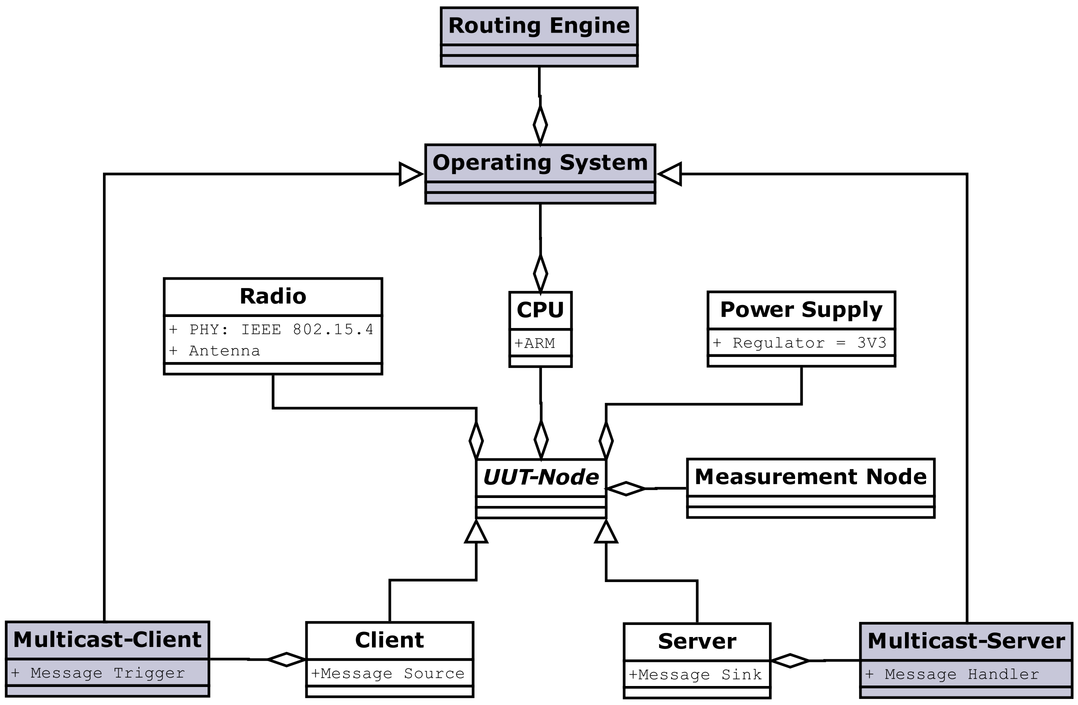

Figure 2.1: Block diagram of a general IoT/WSN node

The following chapter will discuss the overall measurement setup used in this thesis to acquire and store both of these data streams.

2.2 Measurement

2.2.1 Current Measurement

Starting with the current consumption, certain challenges arise when working with low power devices like those described in this thesis. During idle times or sleep periods, the consumption of these CPUs and boards is very low, while during activity periods, the draw increases by factors of 10. This behavior requires a measurement setup that is both capable of measuring very low currents, and fast transient peaks during activity. As this is a well known problem when benchmarking embedded solutions, certain approaches have been discussed in the past.
A paper by Carlos Fernandez et al. [FBV+13] proposes a self energy meter approach, in which the UUT itself is measuring its own current consumption via an additional external circuit attached to it. In this approach, the current is measured via a shunt resistor during the time the CPU itself is active. When the CPU switches to sleep mode, a signal is generated which triggers the Self Energy Meter (SEM) circuit to switch from the direct measurement, to a capacitor which is then charged by the current that is consumed by

the CPU during its sleep period. Once the CPU is active again, the voltage stored in the capacitor, which is equivalent to the consumed current, is measured, i.e. via the SEM circuit, the CPU measures the cumulative value of the current it has consumed while asleep.

For the evaluation in this study, the SEM approach is less suited as it adds additional strain onto the UUT. While the proposed acquisition of the current via a voltage to frequency converter is very efficient and can be handled timely via interrupts, the data which is generated must be stored or transmitted. Storage on the devices itself is not feasible due to the memory constraints of the boards, and additional external storage is typically slower and therefore keeps the CPU active for a longer timespan. Transmitting the data on the other hand would influence the radio of the device, which is the main focus point of the energy efficiency study.
Another approach is discussed by a group around the Linz Center of Mechatronics and the Johannes Kepler University [PBLS14]. Hereby, the current is measured by an external Power Measurement System (PEM) that switches its 16-bit Analog Digital Converter (ADC) between two different shunt resistors. One for small currents during sleep periods and another for higher current draws during active times. The data in this approach is collected by a MCU and sent to a PC via Ethernet.
While this approach provides highly accurate measurements for local systems, again the data accumulation with 250 kHz generates large data streams that either need to be stored or transmitted wireless from the location of the distributed UUT to a centralized entity. Nonetheless, this type of setup provides non-invasive measurement of the current consumption during all stages of the UUT lifetime and is therefore used in a simplified wireless form in this thesis.
The problem of current measurement accuracy, speed and data generation is split into two separate analysis modes, a static level 1 device analysis and a dynamic level 2 network analysis.

Static Analysis

The goal of the static analysis is to evaluate and identify the current draw during different stages of a node's lifetime with high accuracy, high resolution measurement runs. The large amount of data generated in this evaluation requires a localized laboratory setup with a singular static node, or at most one additional node, for capturing the behavior during node interaction. The measurement setup used for this type of analysis can be seen in Figure 2.2.
In order to capture the current consumption of the CPU and the RF-circuitry as accurately as possible, the CPU is the only powered part of the UUT, additional periphery is turned off. The UUT itself is powered by a benchtop power supply which is known to be stable and has a low ripple voltage. The current consumed is measured via a shunt resistor connected to an oscilloscope, which enables a very accurate transient analysis of the consumption pattern. Since the resistance of the shunt resistor is known, the measurements can be converted back into a current timeseries for further analysis. This

level 1 analysis enables a very fine-grained evaluation of the behavior presented by different hardware, running the same software/routing engine. In addition to this evaluation, the data generated can be used to improve existing energy models or to define metrics for energy efficient routing protocols.

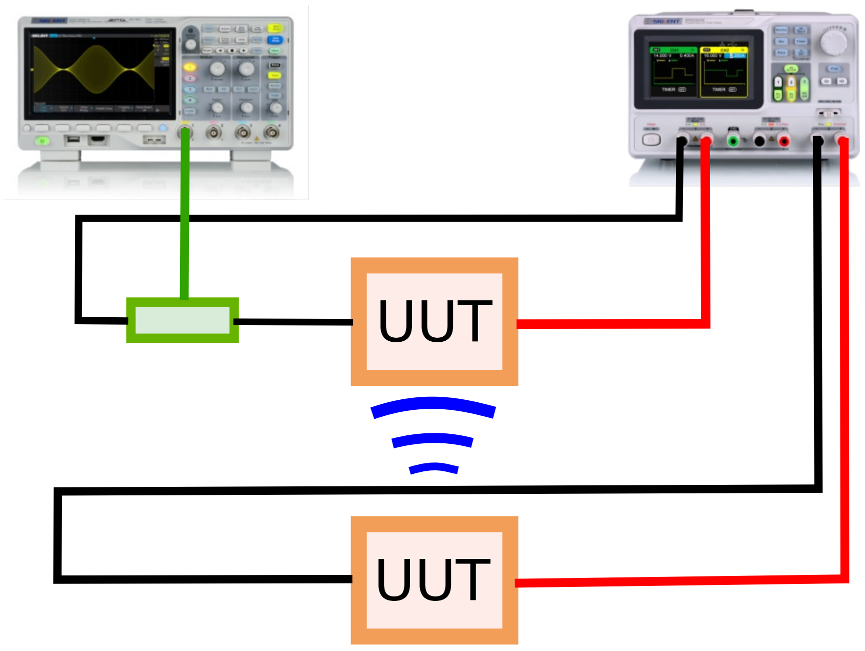

Figure 2.2: Static analysis setup

Dynamic Analysis

The level 2 dynamic analysis is a distributed measurement system utilized during the network lifetime. Its goal is to capture the energy consumed over a stretch of time while typical node interaction, as it can be seen in real-life installations, is carried out. For this analysis, the requirements in regard to speed and accuracy of the measurement are lowered from the static analysis. Extending the overall timeframe from a single event in the level 1 analysis to a benchmark running for upwards of one hour, makes the resolution of the static analysis counterproductive. Huge efforts would need to be undertaken in order to capture and store the information in the distributed setup. Instead, the length of the dataset itself is used to combat the negative effects of a lower resolution in the measurement. Thus, multiple low resolution measurements of the same event will accumulate to an approximation of the high resolution results produced during the static analysis. The block diagram of the utilized simplified PEM can be seen in Figure 2.3.

The measurement node is connected to the base station via its own separate 2.4 GHz network. The channel of this measurement setup is thereby set far away from the one used in the network connecting the UUTs in order to minimize interference. Message

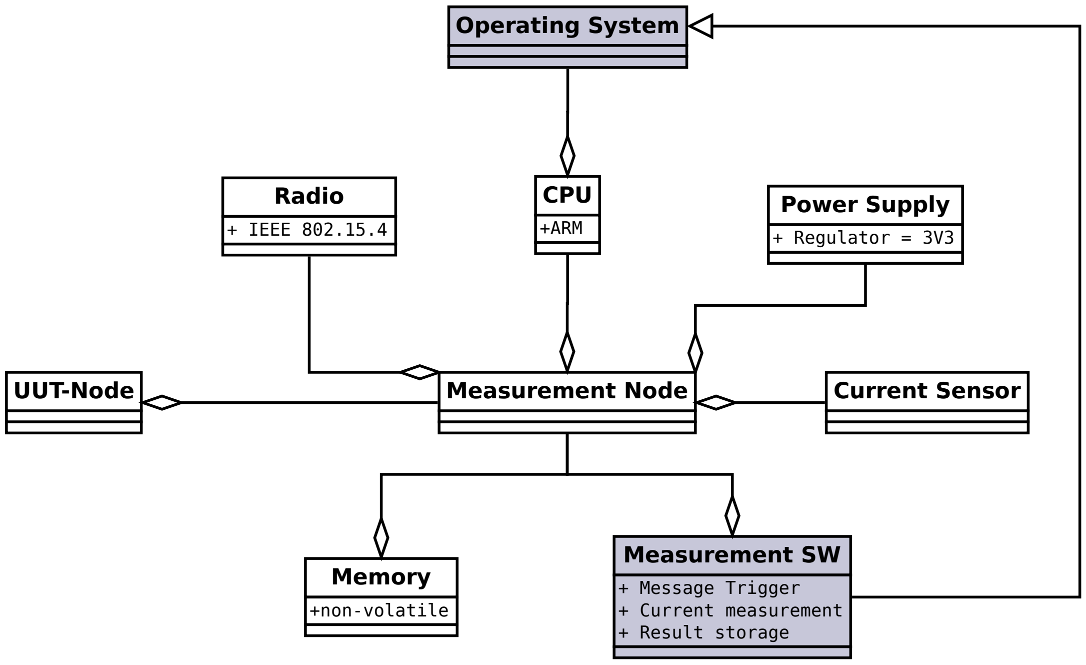

Figure 2.3: Block diagram of a general measurement node

transmissions are triggered remotely on a base station, transmitted to the measurement node via an IEEE 802.15.4 network and in turn to the UUT via a Universal Asynchronous Receiver-Transmitter (UART) connection. The measurement node features its own power supply that converts the external voltage to the internal 3.3V used by the CPU and other peripherals.
In this thesis, the need for measuring the pure CPU sleep current is eliminated by the level 1 analysis, reducing the two shunt approach from [PBLS14], to a singular current sensor. The data generated by this sensor is stored locally in the non-volatile memory and is collected after the benchmark has finished. The capture speed is hereby reduced, so that the overall occurrence and magnitude of the transient computation or transmit event can be measured, but additional information like accurate shape or accurate length of the events is lost. The goal of this analysis is to provide a more global view of the node consumption when arranged in certain topologies. Furthermore, the data can be used to cross-check the results generated in the level 1 analysis in regard to their use in energy efficiency metrics.

2.2.2 Network Metrics

Since the radio of the measurement node is already used by the measurement control network, it is not possible to sniff transmitted packets on air. Instead, the UUT itself internally stores statistics regarding the packets it has sent, received, etc. While this means additional instructions and memory usage for the UUT, the statistics themselves are reduced to simple counters, lowering their influence on the test system to a negligible amount.
The statistics collected by every node include higher level statistics like the number of User Datagram Protocol (UDP) messages sent/received/dropped and lower level routing specific statistics like the number of messages seen by the routing engine, how many of them were needed, i.e. increased the information count for the device, or how many control type messages have been seen if the routing protocol uses these. This enables the evaluation of the routing protocols themselves regarding certain metrics known from the simulation field. Metrics in this regard are the percentage of redundant packets or the general number of packets that were sent.
Once a number of messages has been exchanged in the test network, the measurement node can retrieve this information via the UART bus from the UUT. The data is then stored onto the same non-volatile memory as the current data streams and retrieved from the node by hand.

2.3 Network Topology

Using real hardware for gathering results introduces certain challenges into the overall setup. For once, the number of nodes deployed for the tests is significantly lower than in simulation, where networks of arbitrary size can be deployed. Networks spanning kilometers with thousands of nodes are possible in simulation but are non-trivial in reality, especially with the added difficulty of current and statistic measurements. While larger networks are typically used to measure the performance of routing protocols, there are realistic use cases for WSNs with a low number of nodes, e.g. the precision agriculture use case presented in [AEkEB13].
In this study, a team from the Electronics Research Institute in Cairo describes the use of WSN nodes for precision farming in potato fields. The main fields are thereby split into a number of areas that are monitored by a certain number of nodes, and such subnetworks are then connected to one another via border nodes. The distances between the nodes are hereby kept at 6 meters maximum, in order to guarantee a stable connection between them. A routing protocol, is required to work in the small-scale network as well as in larger installations. Thus, the performance of the larger network is dependent on the performance of multiple low node count topologies.
Another small-scale use case for WSN is the private Home and Building Automation (HBA) realm. Typically, the area of these installations is also very limited due to the constraints given by the target environment, but also limited by the distance between nodes, which is influenced by walls, objects, humans and also interference from other

wireless networks like Wi-Fi. A study by Hemant Ghayvat et al. [GMGS15], hereby focuses on the influence of the surroundings on the network in a building environment. They describe the dampening effect of not only certain materials but also the interference of certain household items like microwave ovens, Wi-Fi, Bluetooth and also the number of occupants in a building. They show that (in their case) the package delivery rate in practice is constant up to a certain point, 10 m in their study, after which the delivery ratio falls sharply to 0 within only a few meters. A higher number of occupants in the house hereby worsens the delivery ratio and the error rate.
While both these use cases only feature networks with small distances between the nodes, especially the results of [GMGS15] show that reaching a 100% reception rate between the nodes is not guaranteed. Since the network is to be regarded as lossy, a routing engine needs to cope with these lost messages in order to guarantee a delivery rate of a certain magnitude.
Like in these two studies, the number of nodes in this thesis is kept low due by design, as a smaller network offers the possibility of a more fine-grained control. In the precision farming use case, the nodes were placed in such a way that they have the least amount of signal loss, while maintaining the best coverage of the potato field. Since nodes typically need to span their use case environment, this means that the range of the node and the according drop in reception is part of the network setup process and lossy connections are to be expected, especially in environments where interference from home appliances or other networks are to be expected.
This fact is especially important when the communication in the network is multicast, i.e. where the number of sinks is in general unknown for a specific message. For the benchmarks described in this thesis, the number of nodes is fixed to 3 with 2 distinct network topologies that can either exist on their own or as part of a larger network. Such a network is thereby equivalent to a subnetwork of the precision agriculture use-case in [AEkEB13], where two nodes are placed per one subfield and one node at the border to connect to another carat. They are furthermore designed to highlight certain difficulties as presented in [GMGS15] associated with lossy networks, which in turn typically require higher energy consumption during message transmission. The combination of two separate topologies, one for a typical use case and one showing adverse conditions, thereby presents a good choice when exploring the effects of hardware and the routing engine on the overall energy efficiency of the network.

2.3.1 Line Topology

Bridging the distance between two subnetworks in order to connect them, or the deployment in structures like narrow buildings, will introduce line segments into the network topology. Especially for lossy networks, these structures are a large problem, as main parts of the network can be cut off from the rest. While in unicast traffic it might be known to the nodes that the recipient of the message is on the other side of the bridge, in multicast the bridge has to be taken at all times. An exception to this are thereby routing engines that are based on the generation of trees, i.e. they have knowledge about the other side of the bridge. Due to this fact, the line segment features a test case

that either needs control structures that aids the transmission across the bridge or they require simpler approaches like an added redundancy that makes it more probable that the messages reach the other end of the line.
For the benchmark, the nodes are therefore placed in a line at such a distance that the two end nodes are out of reach and the average message delivery rate to the center node is <100%. Taking into account the results of [GMGS15], there is only a very small distance between 100% delivery ratio and 0%, nodes are therefore placed in such a way that their delivery ratio falls between 80 and 60% leaving a 2 m placement margin in order to accommodate the limitations of the test environment. The theoretical placement of this topology is shown in Figure 2.4.

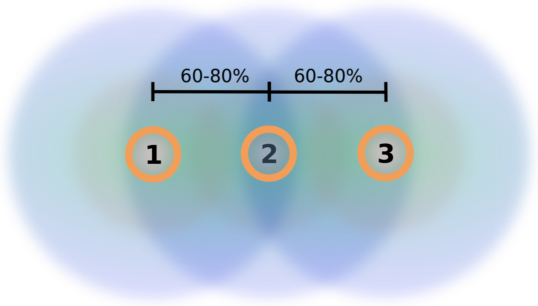

Figure 2.4: Line Topology

Since the package delivery ratio is fixed between the nodes and both distance and interference can be modeled as a drop in package delivery ratio, this type of setup can be used to evaluate routing engines depending on their behavior, independent of the transmit power, the interference or the distance between the nodes. If for example the deployed nodes feature a reduced transmit power, their current consumption will drop, but the work of the routing engine will stay the same, as this difference in transmit power is canceled out by a simple reduction of the distance between the nodes.

2.3.2 Triangle Topology

While the line topology represents a worst-case topology for lossy communication, the triangle topology can be regarded as a typical case of a subnetwork in a larger mesh. As with the line segment, the nodes are placed in such a way, that their package delivery ratio is kept between 80 and 60%. The theoretical placement can be seen in Figure 2.5.

The line topology tests the multi-hop performance of the routing engine in a lossy environment, the triangle evaluates the effect of redundancies in the network, i.e. if the message from node 1 to 2 is lost, node 2 must be reached by either introducing a retransmission of the message, control structures to detect this fault, or the message needs to be transmitted using the redundant route over node 3. Either way, it is paramount for

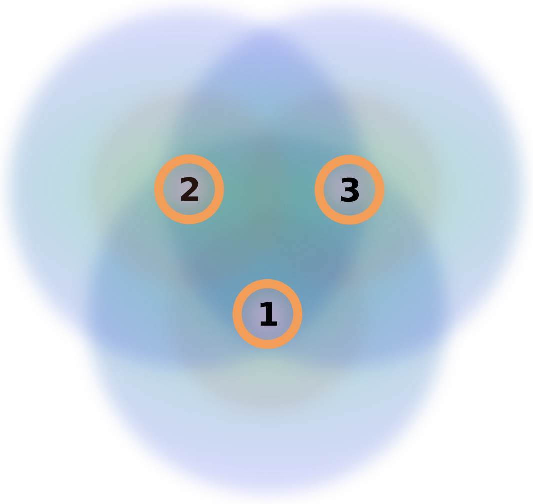

Figure 2.5: Triangle Topology

multicast networks to reach node 2 as the node might be a bridge to another instance of the topology with other nodes that are part of the same multicast group.

CHAPTER 3

Technology Overview

Over the years, a number of routing engines and operating systems for low power devices have been created and discussed. While there is a larger number of unicast routing protocols for a large number of operating systems available, multicast routing still is a less researched field for WSNs and IoT. The following chapter will present some multicast routing engines in used today, as well as operating systems for embedded devices.

3.1 Multicast-Routing Engines

With regard to multicast routing, certain types of engines are used in the field. One popular option is to adapt unicast routing specifications like the common Routing Protocol for Low power and Lossy Networks (RPL) in order for them to support multicast communication. An example of such a routing specification is Stateless Multicast Forwarding with RPL (SMRF) which was first introduced in 2012 by George Oikonomou and Iain Phillips [OP12]. Since then it was developed further, and it today functions as the basis for a number of multicast routing engines, all improving upon certain aspects of the original such as Bidirectional Multicast RPL Forwarding (BMRF) by Guillermo Gastón Lorente et. al. [LLC$^+$17] or Enhanced SMRF (ESMRF) by a team from the Vrije Universiteit Brussel. But at its core all those specifications still capitalize on the existing Destination Oriented Directed Acyclic Graph (DODAG) formation already implemented by RPL. Nodes will thereby add their group membership to their outgoing Destination Advertisement Object (DOA) messages. Parents of these nodes then know which nodes are part of which group and can forward messages to the according nodes, effectively creating multicast by forwarding the messages over the unicast routing tree.
A separate RPL-based approach is Reliable and Secure Multicast Routing Protocol for IoT (REMI) or its successor Reliable Group Communication Protocol (RECOUP), both by the same team from the University of Padua in Italy. While this approach still uses the RPL DODAG tree at heart, they introduce additional clustering, which they describe

as a virtual DODAG, overlayed onto the existing physical tree. These clusters are thereby used to more efficiently transmit messages across the network, as only one neighbor belonging to a cluster will receive the message.
While multicast in WSNs and IoT is still a relatively new topic, approaches on existing ad-hoc network routing protocols have been around a long time. Examples for this are numerous multicast versions of Ad-hoc On-demand Distance Vector (AODV) routing like Multicast Ad-hoc On-demand Distance Vector (MAODV) or the further improved version Improved MAODV (IMAODV), both described in a study by Manmeet Kaur and Amandeep Kaur Virk [KV15]. But while these routing engines provide good results in the highly dynamic networks associated with ad-hoc, they introduce a lot of overhead onto the typically static routing in WSNs and IoT. Routing protocols for ad-hoc networks are based on assumptions regarding their underlying network. Historically, the use of AODV is heavily intertwined with its use in Wi-Fi networks.
Once a new node has become a member of the Wi-Fi network, the connection between the node and its neighbors is regarded as stable and active until the node exits the network. This assumption cannot be made in low power IEEE 802.15.4 networks, where connections have to be regarded as lossy. Due to this fact, the underlying AODV tree would need to be recalculated over and over again, introducing a high number of control messages every time a neighbor is temporarily not reachable. It is for the same reason why in lossy networks, communication is typically handled via UDP and not Transmission Control Protocol (TCP). If messages are lost, devices would need to invest a lot of processing and energy into a reestablishment of the connections which were previously already established via the handshaking process.
All these approaches rely on routing engines originally designed for unicast traffic. I.e. they are based on tree structures that have to be stored and multicast is added via forwarding. Opposed to this are the two approaches implemented and evaluated in this thesis. One is Multicast Protocol for Low-Power and Lossy Networks (MPL), which is similar in its name to RPL but not related, and as a baseline a simple flood routing based approach, which is used in both unicast and multicast routing, as its fundamental function enables multicast routing by default. Together, these two routing engines provide a broad base for energy efficiency considerations. Flooding on the one hand is a common baseline for benchmarks, while the highly configurable MPL provides in itself a high number of routing options with behavior ranging from simple retransmmissions to control messages.

3.1.1 MPL-Routing

At the foundation of MPL is a subform of flooding called gossiping. Hereby, the network is flooded with certain types of messages, but more advanced features lead to retransmission cancellations, i.e. if a node hears a message scheduled for retransmission it cancels this process as the other node seems to already have it. In terms of message types, MPL offers twofold.

- A **MPL Data Message** is used to transmit the actual message payload between nodes of the same MPL domain (nodes that are part of a domain, participate in the message forwarding process)
- A **MPL Control Message** aids in the reliability of the network. Nodes will send out control messages containing information regarding which messages they have in their memory.

The addition of this additional control message type elevates MPL from a simple flood based approach to the real of a dissemination protocol. Through its inherent randomness that is configured via certain settings associated with the MPL specification, it can be counted as a probabilistic (reliable) broadcast protocol [CGR11]. In particular, the following settings are provided by the specification.

- Data Message Timer Expiration
  How often a data messages is retransmitted at most.
- Control Message Timer Expiration
  How often a control messages is retransmitted at most.
- Domain Set Size
  How many domains are able to be stored.
- Seed Set Size
  Number of seeds (other nodes) that are being tracked.
- Buffered Message Set Size
  How many messages of these other seeds are being stored.
- Trickle timer settings for MPL data messages
- Trickle timer settings for MPL control messages

These two Trickle timers build the backbone of the gossiping algorithm. They generate random timer events that can be controlled via three parameters:

- I_{min}
  The minimum interval in ms, i.e. the minimum timespan between the activation of the algorithm and the triggering of a timer event.
- I_{max}
  The maximum interval as number of doublings of I_{min}, i.e. an I_{min} of 100 ms and an I_{max} of 16 will trigger events between 100 ms and 100 ms$*2^{I_{max}} = 6553.6$ s after the activation of the algorithm

- K
  The redundancy constant

Once a Trickle timer is started, it will elapse at a random time between the set minimum and maximum interval, but there are two additional functions that are provided by the specification, consistency and inconsistency. If the used routing engine decides that a received message is consistent, an internal counter of the timer is increased. Once the timer then fully elapses it checks if this counter is higher or lower than the redundancy constant K. If the counter is lower, it will trigger the timer event and tell the routing engine to retransmit the message. If it is higher, the timer event is also triggered but the message retransmission is canceled as a certain number of neighbors seem to already have this message in their buffered message set.

The number of timer elapses and the size of the storage sets are heavily dependent on the network topology itself and are limited by the size of the available memory. As it can be seen in the following list, there are three distinct storage structures in MPL.

- The **Domain Set** stores information like the IPv6 multicast group addresses used for sending data and control messages, an instance of a Trickle timer for sending control messages and a counter that keeps track how often a Trickle event has occurred.

- The **Seed Set** stores information about the seeds that have been seen by the node, i.e. all source nodes from which messages have been received by the node. The structure consists of the seed ID (address) of the source node, a minimum sequence number to avoid duplicates, an instance of a domain structure to keep track of seeds with multiple domains and a list that is used to keep track of the messages originating from that source.

- The **Buffered Message Set** is used to store messages received by the node. Elements of the structure are the sequence number of the message, its size, the payload itself and control elements like a Trickle timer instance used to retransmit the message and a counter that keeps track how often a Trickle timer event has occurred.

Message Generation

Communicating via MPL requires certain headers to be present in order for the messages to be forwarded in a MPL domain. This means that messages of higher level protocols are assembled by an external step and are then encapsulated as a MPL data message. The construction of this MPL options header can be seen in Listing 3.1, the space before the MPL header is thereby occupied by the IPv6 header itself.

Listing 3.1: MPL routing header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |  Option Type  |  Opt Data Len |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| S |M|V|  rsv  |   sequence    |      seed-ID (optional)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The option type is hereby fixed to 0x6D, subsequent data fields provide information about the message in order to short track certain routing decisions, a sequence number that is used to identify the message and avoid duplicates and an optional seed ID which is used to identify the source of the message if needed.
Once a message is passed down from a higher layer, the sequence number, starting from 0, is increased by 1 and included in the header. The message is then stored and run through the internal structures, primarily it is inserted into the buffered message set and depending on the configuration of MPL the Trickle timer for data retransmissions and control message retransmissions is started.
Once this MPL part is complete, the message is passed down to the lower levels of the OS in order to transmit it.

Trickle Timer Expiration

The MPL specification allows for any combination of how often the two Trickle timers can elapse, depending on the settings written to the Data/Control Message Expirations setting. If one of these timers triggers an event, MPL will react the following way:

- Data Message Expiration
  In the corresponding buffered message set entry, the counter keeping track on how often the timer has elapsed is increased by 1, and it is checked against the configuration. If the counter is the same or higher than the given maximum, the Trickle timer is stopped, and the message will not be transmitted again. If the counter is lower, the engine checks if the message should be retransmitted based on the consistency/inconsistency feature of the Trickle timer. If retransmission is allowed, the header is reassembled via the information stored in the buffered message set and given to the lower levels for transmission.

- Control Message Expiration
  Data message retransmissions are coupled with a specific message in the buffered message set. The control message on the other hand is coupled with the domain set of a specific node. It is used to check if neighboring nodes have seen the same messages for the given domain. While data messages encode their header directly after the IPv6 header as an IPv6 Hop-by-Hop options header, control messages are of the ICMPv6 type with a distinct header structure as seen in Listing 3.2.

Listing 3.2: MPL control header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                      MPL Seed Info[0..n]                      .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type and code are hereby constant, the checksum is calculated by the implementation. The MPL Seed info set encodes information for every seed that is known to be part of the given domain. The structure of one such entry can be seen in Listing 3.3.

Listing 3.3: MPL Seed Info

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   min-seqno   |  bm-len   | S |   seed-ID (0/2/8/16 octets)   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.            buffered-mpl-messages (variable length)            .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The min-seqno integer is hereby the lower bound sequence number of the messages encoded in the buffered-mpl-messages, bm-len encodes the number of bytes needed by the buffered messages and S/seed-ID are used to identify the address of the source seed that the information is based upon.
The buffered-mpl-messages are hereby encoded as follows. Every bit of the vector represents a sequence number between the given min-seq and the maximum number currently in the buffered message set of the given seed. If the bit is '1', the sequence number is present in the memory of the node, if it isn't, the bit is set to '0'. I.e. if the buffered message set of the seed contains two messages with sequence numbers 0 and 2, the corresponding vector will be 101.

Message Reception

As with their formation, data and control message receptions have to be handled differently by the MPL routing engine.

- Date Message Reception
  If a message with the correct headers is received by a node, it checks if the domain

is already known, if the seed is already known and if the message already exists in the buffered message set. If one of these checks fails, the message is added to the corresponding set. If the message is only missing in the buffered message set, a new Trickle timer for the data retransmission and the corresponding control Trickle timer for the domain are started. If the message and its source are already part of the sets, the message is counted as consistent and the corresponding data retransmission Trickle timer function is executed.
After handling these routing functions, the message is given to the higher level protocols.

- Control Message Reception
  Since control messages are ICMPv6, they are not checked by the routing engine at the lower level, instead they have to be passed down from the higher level to the engine. MPL will unpack the MPL Seed Info set encoded in the message and compare the neighbor's known messages with its own sets, and check if anyone of the two knows of messages the other has not yet seen.
  If the current node is missing a message present in the neighbor, the control Trickle timer is reset and started. After this Trickle timer elapses, a control message is emitted that informs the neighbor node that the current node is missing messages. If on the other hand the neighbor is missing messages, the data Trickle timer for these messages in the buffered message set is reset and started, retransmitting the message again, in order to transmit them to the neighbor.

3.1.2 Flooding

Flooding is the base case for gossiping based routing engines like MPL. At its core, the functionality is very simple: Every message that is received by a node is subsequently retransmitted, i.e. hence the name flooding. While this is a very inefficient approach for unicast traffic where only one node needs to be addressed, it is very suitable for multicast traffic as the aim of the routing engine is to flood the network anyway. This means that this routing protocol becomes more efficient, in terms of the number of redundant messages, the larger the multicast groups gets.
One negative effect in flood based routing is the possibility of a so-called broadcast storm. In this case, messages are retransmitted again and again until every node in the network is only retransmiting the same message, effectively crippling the whole network. To prevent this, the routing engine needs to keep track of which messages it should retransmit and which to drop. An easy solution for this is again a sequence number for every message, and if a message with the same sequence number is received again it is simply dropped. Another way to prevent this from happening is to utilize the hop counter. Every time a message is retransmitted, its hop counter is decremented until it eventually reaches zero and is dropped. While this can be very effective in regard to storm prevention, the disadvantage of this is that inherent knowledge of the network needs to be present. If for example, the hop limit is set lower than the number of potential nodes in a line, the message will never reach certain parts of the network.

Control messages are not part of this routing engine, as are retransmissions, so while the computation effort and memory usage are reduced from the more advanced MPL, it does not feature any features to repair the state of the network once packets go missing.

3.2 Operating System

OSs for embedded devices in IoT and WSNs need to fulfill certain criteria. They need to fit onto very constrained devices, need to be highly energy efficient, offer radio support for current protocols and they need to support a wide range of boards in order to gain widespread use in the inhomogenous realm of ARM based IoT boards.
Two prominent and long-lived examples for embedded OS in IoT are ContikiOS and TinyOS [JASK18], which were first introduced in 2004 and 2000, respectively. Especially ContikiOS has found widespread use in studies and articles regarding routing engines, mainly due to its Cooja simulator, which enables users to port their implementation onto virtual devices that can then be placed in the virtual environment in order to evaluate function, performance and rough energy consumption. Even though ContikiOS has been around a long time, it has only marginally lost some market share in the last years [AS20], since it has an active developer base and features the most important modules for IoT and WSNs, like IPv6, 6LoWPAN and so on.
While the market share of Contiki has slightly dropped in the last years, a relatively new OS introduced and backed by the Linux Foundation was able to more than double its share between 2018 and 2019 [AS20]. ZephyrOS is an open source OS, that not only provides support for very constrained devices, but also for boards and CPUs with fewer restrictions. Due to this fact, it now supports more than 200 boards. While it originally was just using the Contiki network stack, it has since then parted ways with this approach and redesigned the stack from the ground up, providing IPv6 support.
Due to its recent introduction and active community, ZephyrOS sets itself up as a good choice for this thesis. After recent iterations in the network stack, it is currently shipped with all the necessary interfaces for a routing protocol, but lacks implementations for unicast and multicast routing. A Trickle implementation is already part of the code base, lowering the effort that is needed to improve the OS with MPL and flood routing.

CHAPTER 4

Hardware

When looking at ARM based boards for IoT and WSNs, two of the biggest players in the field are Nordic and Texas Instruments. Both of which are providing a number of boards suitable for deploying not only Bluetooth networks but also IEEE 802.15.4 based ones, like the ones discussed in this thesis. In order to improve the workflow, both the multicast nodes and the measurement nodes are running Zephyr OS and therefore need to be part of the list of supported boards.
The choice in this thesis has fallen upon the Nordic NRF52840 DK and the TI CC1352r1 LaunchXL board. Both these boards provide a good basis for comparing and evaluating the energy efficiency of routing protocols. The NRF52840 features a faster 64MHz CPU in comparison with the 48MHz of the TI board. The flash of the Nordic board is also 3 times the size of the TI board, creating a good spread between both boards, with one being less constrained and fast, while the other one is slower and features less memory. In addition to these performance differences, the RF stages of both devices, together with their included PCB antennas, are of two different power levels and designs as well. In combination, the differences between the two boards build a strong basis for determining if hardware-independent metrics are enough to classify a routing engine as energy efficient.

4.1 Multicast Nodes

Since both boards feature slightly different peripherals and operations, they will differ in their behavior and setup during the benchmark runs. Table 4.1 presents the capabilities of both CPUs side by side.

4.1.1 NRF52840 DK

The NRF52840 DK board is already equipped with a header that can be used to measure the power consumption of the CPU, shown green in Figure 4.1. The development

ARM-CPUs		
	NRF52840	**CC1352r1**
Speed	64 MHz	48 MHz
Flash	1 MB	352 kB
RAM	256 kB	80 kB
Radio	2.4 GHz	2.4 GHz / Sub 1 GHz
max. TX Power	+8dbm	+5dbm

Table 4.1: Comparison between the NRF52840 and the CC1352r1 CPU

board features additional hardware like different power supplies, able to convert different voltage sources to the required 3.3V, an on-board programmer and a set of LEDs/buttons. In order to eliminate the influence of these additional current consumers, the current measurement port of the board is implemented as a high side shunt, meaning that the measurement node needs to be able to measure the current in differential mode. The advantage of this implementation is that the behavior of the programmer and the serial hardware of the board is always independent of the current draw of the CPU and the LEDs/buttons.

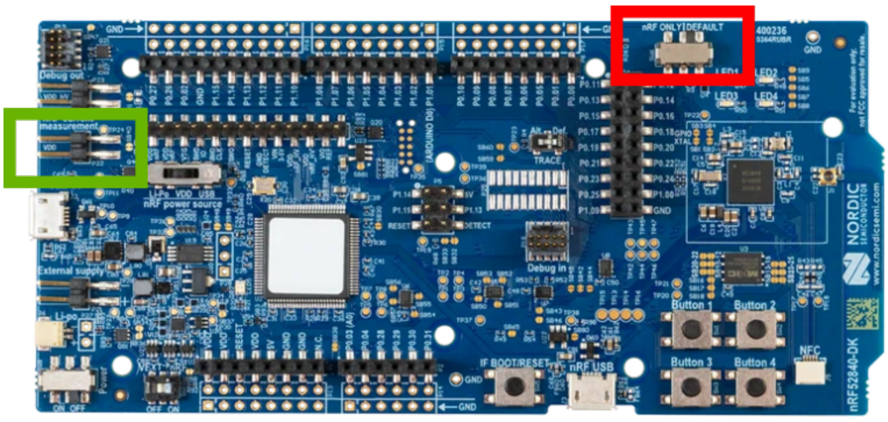

Figure 4.1: NRF52840 DK board, [Nora]

Depending on the type used, LEDs can introduce a current draw similar to the CPU itself. To eliminate this influence, the NRF52840 can be switched to CPU-only mode via the slider shown red in Figure 4.1. In the NRF52840 only setting, the only powered device on the board is the CPU itself, without additional peripherals.
The board offers a number of independent ports for supplying power. The corresponding ports can be seen on the left side of the board in Figure 4.1. Options include the default 5V USB connection, a coin-cell battery on the bottom side of the board, and two connection points for external battery packs.
The amplifier of the 2.4 GHz PHY and antenna assembly can be set in a range of -20 to

+8dBm in 2dBm steps, enabling the developer to tune the range and power required by the PHY stage, depending on the present node environment. The antenna gain of the deployed trace antenna, which is hidden beneath the Nordic logo on the right side of the board shown in Figure 4.1, is 0 dBi.

4.1.2 CC1352r1 LaunchXL

The LaunchXL development board is also equipped with additional debugging/programming and communication hardware. The CC1352r1 CPU and its RF-circuity are located on the left side of the board shown in Figure 4.2. It is connected to the on-board power supply and the programming periphery via the column of jumper connections marked green in Figure 4.2. This enables the developer to specifically connect or disconnect certain functionality from the CPU.

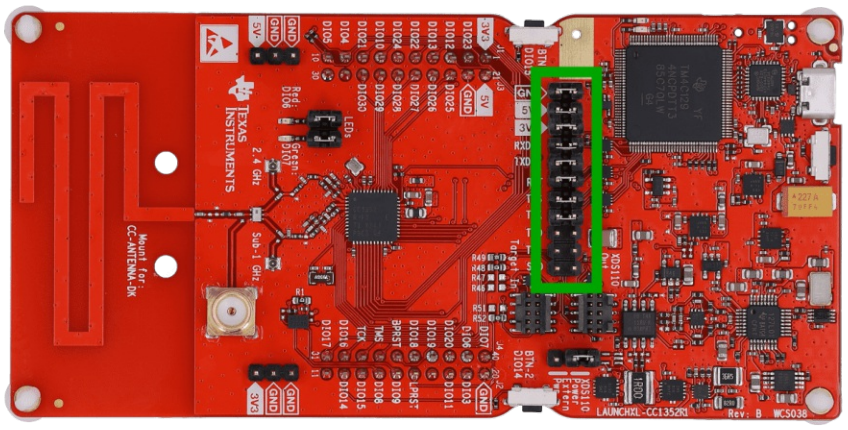

Figure 4.2: CC1352r1 LaunchXL board, [Tex]

High side current measurements can be taken by disconnecting the 3V3 jumper of the pinheader and replacing it with the used current sensor. In the same manner, simpler low side measurements can be achieved by bridging the GND jumper.
The two LEDs can be disconnected via two jumpers, while the two buttons on either side of the board are always connected. The board can be powered via the 5V USB connection on the right of the board or via an external 3.3V power supply. Connections for external battery sources are not present.

4.2 Measurement Node

Since both boards offer the option for high side current measurements, the current sensor needs to be equipped with a differential ADC. The results from the measurement are

communicated to the control CPU via Inter-Integrated Circuit (I2C). Since the current draw even during transmit and active events will be low and the processes involved are fast, the current sensor needs to feature a reasonable high resolution.
The control CPU of the measurement node needs to be able to retrieve the results of the current measurement in a timely manner and store it onto an external memory. Communication to the base station requires a 2.4 GHz PHY and RF assembly.

4.2.1 INA226

As a current sensor, the Texas Instruments INA226 bidirectional current and power controller is chosen. It offers high as well as low side current measurements and is available as a development board, which enables easy integration into other circuits via a pinheader, shown left in Figure 4.3.

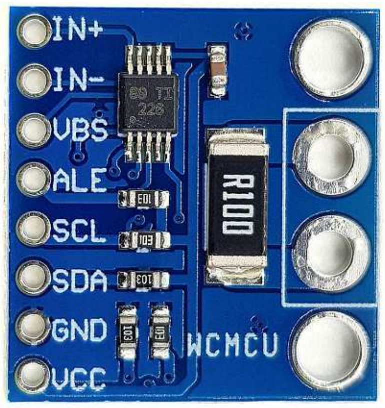

Figure 4.3: INA226 development board, [Wol20]

The built-in ADC has a resolution of 16 bits and a minimum conversion time of 140 us that can be increased to 9s if needed. A control unit is able to switch the ADC from current to voltage measurement and calculate the power from the two samples.
Configuration and measurement retrieval is available via the included I2C bus, and the board can be powered by an external power supply ranging from 2.7 to 5.5V. In addition to the conversion time setting, certain configurations can be made on the device.

- Averaging
  The INA226 is capable of automatically averaging a set number of samples. This process will prolong the set conversion time depending on how many samples are averaged.

- Alarm
  Alarms can be set for certain conditions like current/voltage over a set maximum value or if a conversion has finished. Once triggered, the alarm pin of the device is

toggled and the functionality is reset once the corresponding register has been read via the I2C bus.

- Measurement Mode
  The choice can be made between continuous and triggered operation.

- Type of Measurements
  The developer can choose if the shunt voltage, the UUT voltage, the power or all of the values are captured by the INA226.

- Range and Calibration
  Since the value of the shunt resistor is not fixed, the factor for the used resistor and the current range can be set in their corresponding registers. In the case of the used development board, the shunt has a resistance of 0.1 Ω

4.2.2 NRF52840 Dongle

Since the behavior and the features of the NRF52840 CPU are already known from the multicast nodes themselves, the dongle version of the development board is chosen to be the measurement controller and communication interface to the base station.
In regard to the NRF52840 development board, the dongle version is kept much simpler, only equipped with a programming interface that can be accessed via the regular size USB connector shown right in Figure 4.4. It has to be noted, that this USB connection can be used to program the chip, but not for communication/debugging as no serial interface from the CPU is connected.
As status indicators, two LEDs are available together with a user configurable button and a reset button.

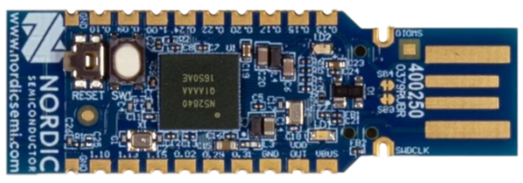

Figure 4.4: NRF52840 Dongle, [Norb]

The board can either be powered with 5V via either the USB connector or a corresponding pin on the bottom pinheader. There is also an option to power the board via an external 3.3V power supply, in which case certain connections on the bottom of the board have to be cut and connected.
Even though the board is much smaller, it is equipped with an antenna and RF assembly

capable of producing the same transmission power as the larger development board, providing 2.4 GHz IEEE 802.15.4 communications.

4.2.3 Measurement Circuit

The INA226 and the NRF52840 dongle are combined with additional periphery to form the circuit shown in Figure 4.5.

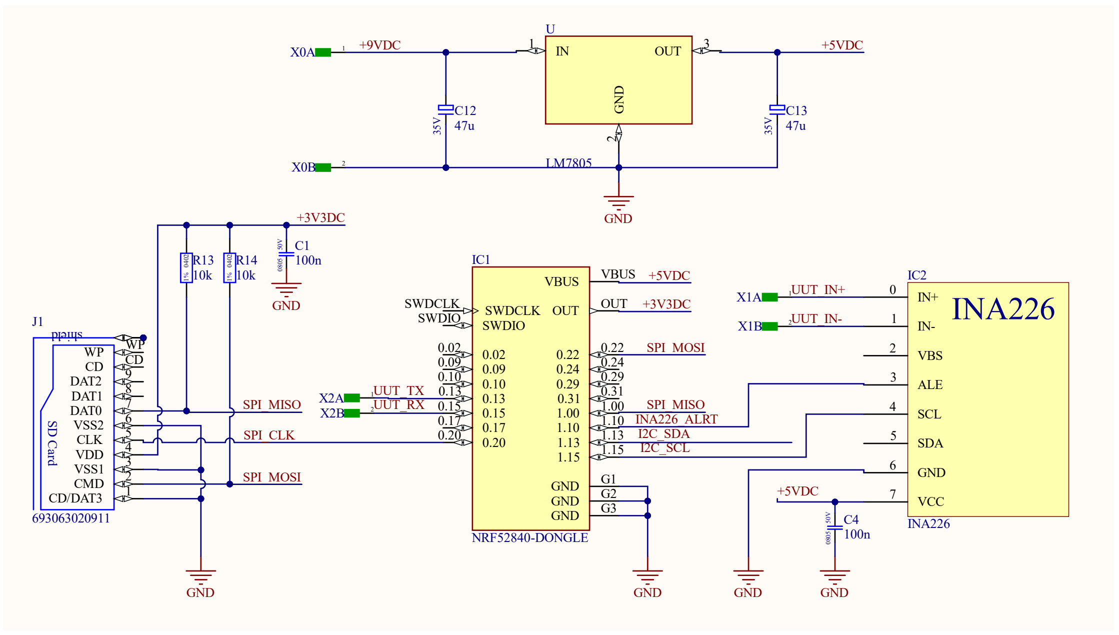

Figure 4.5: Measurement node schematic

In order to store and retrieve the benchmark data streams, an external flash storage in form of a microSD card is attached via Serial Peripheral Interface (SPI) to the NRF52840. This way, the card can easily be attached and removed from the measurement node and the interaction only requires minimal external components: two pullup resistors on the two communication lines and a capacitor used to stabilize the supply pin of the card. The measurement node can either be powered via the USB connector of the dongle or via the 5V regulator circuit that is supposed to be powered by a 9V block battery. Using batteries and the LM7805 linear voltage regulator ensures a stable voltage source for the whole measurement setup and also enables dynamic placement of the setup.

CHAPTER 5

Implementation

5.1 Routing Engines

In Zephyr, network oriented implementations that are independent of the underlying hardware are located in the `[root]/subsys/net` folder of the OS installation. Hereby, the related topics are further divided into:

- `./l2`
  Implementation folder for level 2 specifications like Bluetooth, OpenThread, IEEE 802.15.4, Wi-Fi, etc., that are kept hardware independent.
- `./ip`
  IPv4/IPv6, neighbor discovery, ICMPv4/6, 6LoWPAN and routing related topics such as Trickle and network management in general.
- `./lib`
  Libraries for protocols that typically sit upon these lower levels, like COAP, MQTT, HTTP ...

Keeping to this folder structure, the implementation of the routing engines is included in the `route.c` file located in the `./l2` folder. Since the implementation is kept modular, the implementation can be migrated into a separate library in the future.
In order for the user to configure MPL/Flooding during the build process of the software, Zephyr offers a configuration interface which can be populated by adding a so called Kconfig file. In the case of this thesis, the file is called `Kconfig.mpl`. Inside the file the developer can define a number of configuration settings, e.g. an option to enable/disable the routing engine, the statistics collection or options to configure the overall behavior of the routing engine, like the number of data message expiration's in the case of MPL. A snippet of this configuration file is shown in Listing 5.1.

```
config NET_STATISTICS_MPL
   bool "MPL related statistics"
   default n
   help
     Enable to collect MPL statistics

config NET_MCAST_MPL_DATA_MESSAGE_IMIN
   int "intervall size"
   default 32
   help
     Trickle timer minimum intervall size

config NET_MCAST_MPL_DATA_MESSAGE_IMAX
   int "intervall size"
   default 1
   help
     Trickle timer maximum intervall size
```

Listing 5.1: Snippet of the Kconfig file regarding MPL routing

Following the definition of the configuration interface, the routing engine itself can be implemented.

5.1.1 MPL

In general, the MPL implementation offers methods for the 4 steps described in section 3.1.1: message generation, message reception, control Trickle expiration and data Trickle expiration. In addition to this main functionality, helpers for buffer allocation and interaction with other parts of the OS need to be defined.
To aid readability, the methods are combined into two flows, one that follows a data message though its OS life cycle and the other that follows the control message. Since the underlying network type is non-specific for this implementation, the flow discussed in the following chapters will start and stop at the points where the program flows enters/leaves the methods provided in the `./ip` folder.

Data Message Flow

The sequence diagram, seen in Figure 5.1, presents the flow of the packet during a sending event that is triggered by the software using the higher level UDP sockets implementation. Certain steps inside a method are hereby abstracted and are instead replaced by a description shown in bold. As a whole, the send operation of a data message is handled exclusively by the `net_context.c` and the `route.c` file.

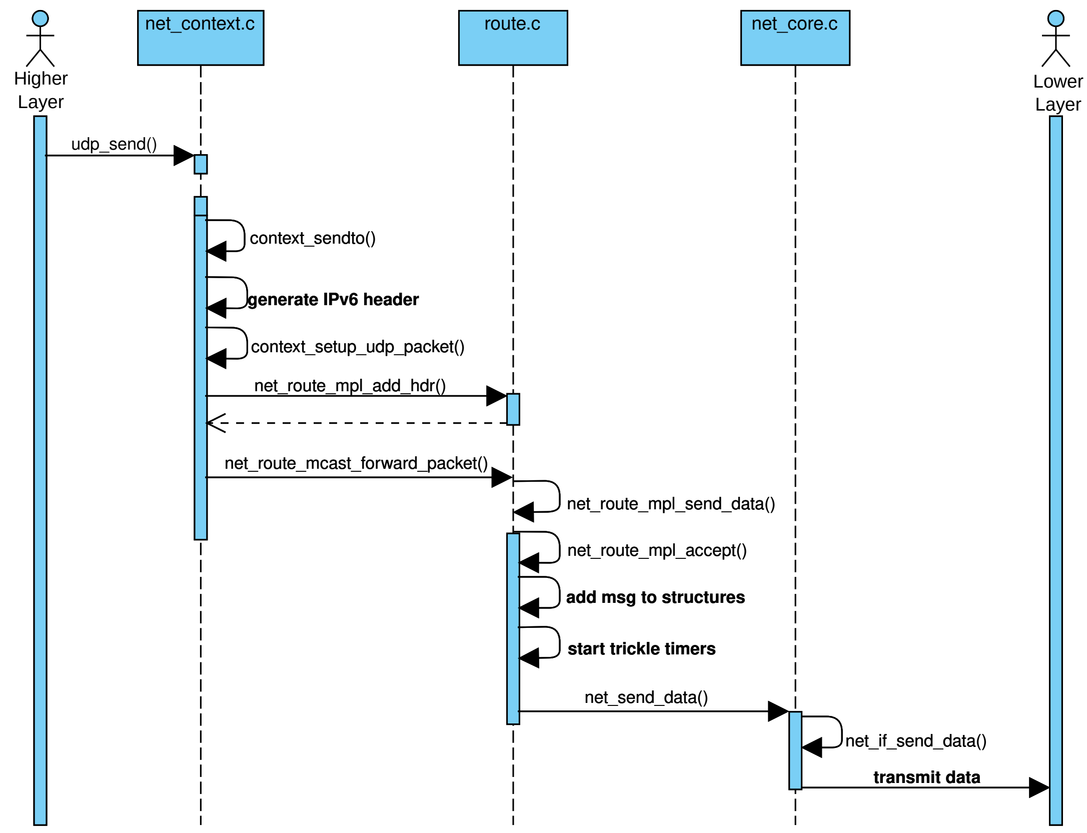

Figure 5.1: MPL message send sequence diagram

If the packet passed down from higher layers is identified as being a UDP message, the context implementation will set up the corresponding option in the IPv6 header. In addition, the MPL routing header, as shown in Listing 3.1, will be added to the packet by calling the `net_route_mpl_add_hdr()` method provided by the `route.c` file. As an example for packet generation and manipulation in Zephyr, Listing 5.2 presents the implementation of this header addition process.

```c
1  void net_route_mpl_add_hdr(struct net_pkt *pkt, size_t *len)
2  {
3    uint8_t mpl_flags = 0;
4
5    net_pkt_write_u8(pkt, IPPROTO_UDP);
6    net_pkt_write_u8(pkt, 0);
7
8    net_pkt_set_ipv6_next_hdr(pkt, NET_IPV6_NEXTHDR_HBHO);
9    net_pkt_set_ipv6_ext_len(pkt, HBHO_TOTAL_LEN);
10
11   net_pkt_write_u8(pkt, HBHO_OPT_TYPE_MPL);
12   net_pkt_write_u8(pkt, 4);
13   mpl_flags |= 0x00; //S=0
14   mpl_flags |= 1<<2; //M=1
15
16   net_pkt_write_u8(pkt, mpl_flags);
17
18   last_seq = SEQ_VAL_ADD(last_seq, 1);
19   net_pkt_write_u8(pkt, last_seq);
20
21   net_pkt_write_u8(pkt, 0x01);
22   net_pkt_write_u8(pkt, 0x00);
23 }
```

Listing 5.2: Implementation of the MPL header manipulation in Zephyr

A packet itself is an instance of the `net_pkt` structure that can be manipulated by utilizing the methods provided by the `net_pkt.c` implementation. Internally, the packet is represented as an array with a cursor indicating the current write position. Calling `net_pkt_write_u8()` will write the given data to the current cursor position of the packet and advance the cursor one element. Other methods for different datatypes or variable length strings are also available. While these packet operations manipulate the packet directly, certain shortcuts for the IPv6 header are available. In the case of Listing 5.2 on line 8 and 9, these methods are used to directly set the IPv6 Next Header option and the total length of this option type used by the MPL routing header.
Once the header has been assembled, the finished packet is given to the `net_route_mcast_forward_packet()` method that is already part of the stock Zephyr stack. Since no routing is implemented by Zephyr, the stock version of the method won't forward any packets. By enabling the MPL option in the configuration, the method will be substituted by the new code that will utilize the MPL engine by calling `net_route_mpl_send_data()`.
The first step in routing the packet is to call `net_route_mpl_accept()`, which will insert the message into the 3 data structures defined by MPL. For this process, the data encapsulated in the packet needs to be extracted from its structure via the snippet shown in Listing 5.3.

```c
1 hdr = (struct net_ipv6_hdr *)net_pkt_get_data(pkt, &ipv6_access);
2 if (!hdr) {
3   NET_DBG("DROP: no buffer");
4   return -1;
5 }
6 net_pkt_acknowledge_data(pkt, &ipv6_access);
```

Listing 5.3: Retrieving data from a net_pkt structure in Zephyr

Line 1 will retrieve the IPv6 header structure from the packet, line 6 is then used to advance the read cursor of the packet to the end of the retrieved structure.
Once the header is checked for errors, e.g. if the packet originates from the node itself or if the options in the header are correct for a MPL data message, the information of the MPL header and the message itself is added to the 3 structures, as described in subsection 3.1.1.
The next step is to start the Trickle timers, according to the configuration chosen by the user. If no errors have occurred during this process and no packet checks indicated a misconfiguration, the packet is given to the `net_send_data()` method that uses the `net_if_send_data()` to access the lower levels of the OS and in turn the hardware that will transmit the packet over air.

Elapses of the data message retransmission Trickle timer feature a much simplified version of the default send operation. Instead of manipulating an existing packet from the upper layers, a new packet is generated and the corresponding headers and options are added according to the information stored in the buffered message set entry. Once the packet is assembled, it does not need to be accepted, as it is by design already part of the right data structure. Instead, the packet is directly given to the `net_send_data()` method for transmission. In accordance with the Trickle specification, this send operation will only occur if allowed by the Trickle timer. In addition, the elapsed timer counter for data messages is increased and checked against the configuration and if the maximum number of expirations is reached, the Trickle timer will be stopped.

If a data message is received, the handling is executed as presented in Figure 5.2.

Events from the lower levels will call the `net_ipv6_input()` method with the received packet as a parameter. The IPv6 header will be extracted from the packet and checked for misconfigurations, in which case the packet is dropped. If multicast routing is enabled and the destination address is a multicast address, `ipv6_forward_mcast_packet()` is called, which in turn will switch to the `route.c` implementation of MPL by calling `net_route_forward_packet()`.
Handling of the received message is carried out exactly as during the message transmit process by executing `net_route_mpl_accept()`, but instead of forwarding the packet onto the lower levels on success, the return value of the method is used to determine if the message should be dropped or handled further. If the message is accepted, its destination address is checked against the addresses that the node is currently subscribed to. If the node is part of the multicast group and there is an additional options header

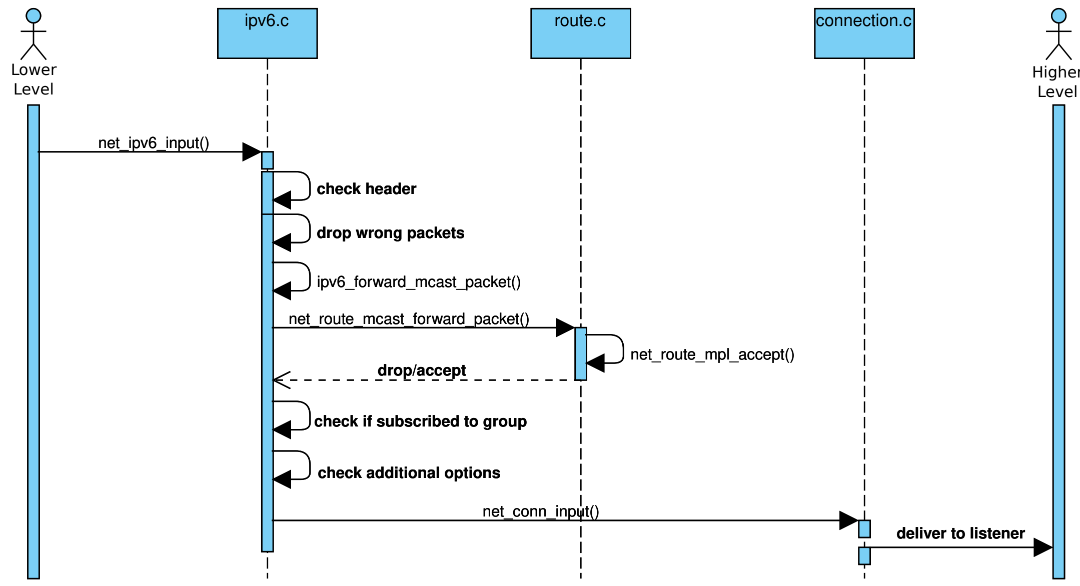

Figure 5.2: MPL message receive sequence diagram

like UDP that's part of the packet, the payload is forwarded to the upper layer protocols via the `connection.c` implementation that checks if there are active listeners for this data and port configuration.

Control Message Flow

By design of the MPL routing engine, the transmission of a control message can only be triggered via its Trickle timer expiration. The initial start of the timer is carried out by either transmitting a data message or by receiving one. The sequence diagram of the control message transmit event is therefore much simpler than the one of the data message, as evident in Figure 5.3.

Once the Trickle timer elapses, it calls the `ctrl_message_expiration()` method, which at first checks for the number of expirations and the Trickle enable signal analog to the data message. If both checks are positive, the actual assembly and transmit is carried out by calling `ctrl_message_out()`.
A control packet is set up by first retrieving the address information from the domain set entry that triggered the Trickle timer. Since control messages are transmitted, the domain control address is set to be the destination address of the packet, with the source address being the IPv6 address that the current node is using to transmit data messages. As a next step, the seed set is iterated. If the current entry is active, i.e. at least 1 message was seen with this source, a new MPL seed info header is generated. Important in this case is that the seed ID in this info header cannot be implicitly given by the source address of the IPv6 packet, as it is the case for a data message. Instead the S=3 option of MPL has to be used which means that the IPv6 address of the current seed

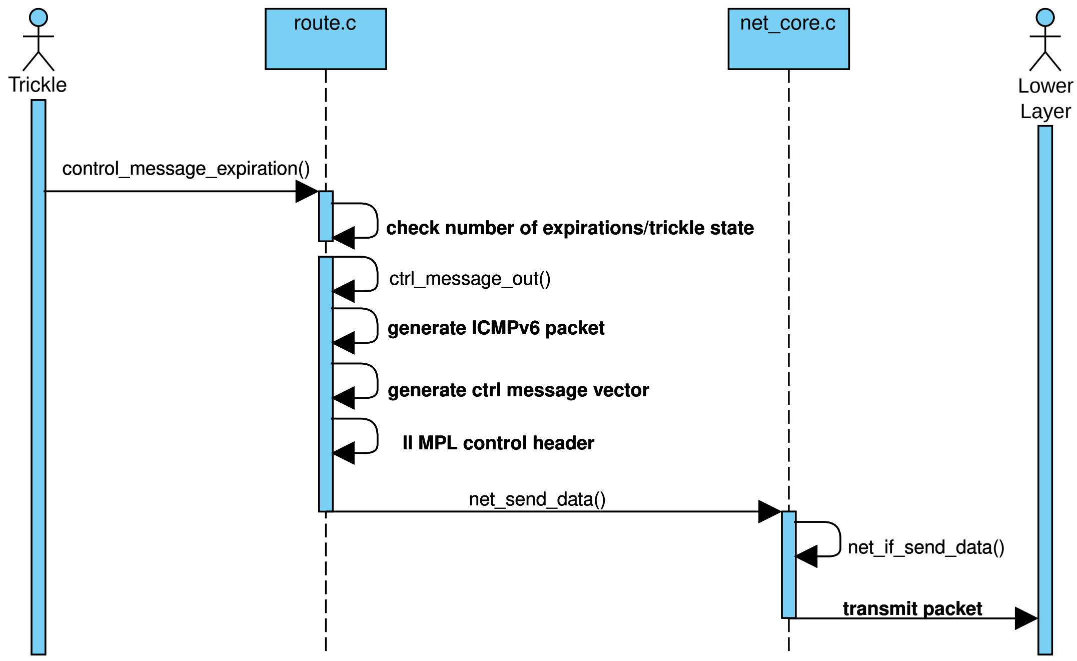

Figure 5.3: MPL control message transmit sequence diagram

entry is inserted directly into this header.
The message vector is generated by iterating through the message set of the current seed entry and setting every bit to 1 if the message with the corresponding sequence number is present in the message set, and 0 otherwise. The packet is then finalized and passed to the lower layers to be transmitted.

Reception of a control message follows the same sequence as the one presented in Figure 5.2. Differences only occur during the execution of `net_route_mpl_accept()`, since the packet is of type ICMPv6, it will not be entered into any of the MPL data structures, instead the method is stopped early and accept is returned in order to further process the message in the higher levels.
ICMPv6 packets will be routed into the `icmpv6.c` implementation by the `net_conn_input()` method. If MPL is enabled in the configuration, the ICMP handler will find a definition of the MPL option in its data set and via a callback will trigger the execution of the `route.c` implementation via the `icmpv6_handle_mpl_ctrl()` method. By analyzing the information in the control message, three outcomes can arise.

- Remote Inconsistent
  The message vector of the other node is missing a message that is present in the seed set of the current node.

- Local Inconsistent
  The seed/message set of the current node is missing a message that is present in the source node of the control message.

- Consistent
  The message vector and the seed set of the current node match up.

The simplest case is that the domain is consistent, in which case the consistency method of the domain Trickle timer is called in order to reduce the number of control messages that are sent.
Remote inconsistency is detected by multiple checks run on the package. At first the header is iterated and the seed set of the current node is checked against the seeds that are included in the control message. If there is a seed ID that is present in the seed set but not in the control message, the domain is to be regarded as inconsistent and the data retransmission Trickle timers for the given missing seed are restarted. The same check is also carried out for the other direction, in which case a missing seed ID in the seed set leads to a local inconsistency. The next check is run against the message vectors of the included seed info structures. Messages missing in the remote node will always trigger a restart of the data retransmission Trickle timer. Local inconsistencies on the other hand are solved by starting the control Trickle timer of the domain. The remote node will then in turn receive a control message that will trigger a retransmission of the missing messages.

5.1.2 Flooding

As described in subsection 3.1.2, there are many ways to prevent a broadcast storm in flood based routing. Since the MPL implementation already features an option header that includes sequence numbers for messages and corresponding algorithms to find and deal with duplicates, a main part of the MPL routing engine is used to implement flooding in this thesis. The transmit event therefore unfolds as shown in Figure 5.1 with only a minor change in the code execution: no Trickle timers will be started.
For message reception, the `net_route_mpl_accept()` method is changed by the flooding configuration to exclude all Trickle operations. Instead, the data message is reassembled into a packet, analog to data message retransmissions, and transmitted by calling `net_send_data()`. The rest of the handling is then unchanged, delivering the packet to the upper layers for further computation.

5.2 Multicast Nodes

As described in chapter 2, there are two types of nodes used in this thesis. The multicast client features an UART interface and the UDP client configuration so that messages can be triggered and sent to the multicast group. The multicast server on the other hand features a simplified UART interface and the UDP server implementation to receive the transmitted messages. The choice for a client/server architecture is thereby made to analyze two behavior types of the software, a node that is mostly asleep in the form of the client, i.e. a very low power node, and a node that constantly waits for messages from other nodes in the form of the server node.

5.2.1 Multicast Client

The multicast client features two main flows, one specifically for the benchmarking process by the means of an UART interface and a test process including the use of a button in order to transmit a message over the network. At startup, the node will initialize GPIOs for a singular button and LED. The LED is subsequently switched on in order to visually differentiate between client and server nodes. In the next step, the UART and the UDP port is set up and activated. The UART is thereby used for receiving benchmark related commands, while UDP is used to transmit messages in the multicast group.
In order to aid with static level 1 measurements, the transmission of the UDP message can be triggered by either receiving a message send command via UART for dynamic installments or by pressing the setup button on the board itself, limiting the number of devices that need to be active and running in the static analysis.

UART

In order to keep the influence of the benchmarking interface as low as possible, the UART is set up to use interrupts for both sending and receiving. This means that the node is asleep for most of the time and no buffers have to be polled, which would increase the current consumption of the device. The following commands can be processed:

- ’s’
  Reception of this command, triggers the generation of a statistics Java Script Object Notation (JSON) response, which is then returned to the measurement node via the same UART connection.

- ’m’
  Triggers the transmission of a UDP message to the multicast group.

- ’l<n>’
  n is an integer and determines the payload in bytes (default = 20) for subsequent message transmissions

- ’p<n>’
  n is an integer and determines the transmission power (default = +4) for subsequent message transmissions

In order to not disturb the measurement statistics, length, power and statistic commands should not be issued while a current measurement is running as every message will wake up the CPU. As in real-world applications, the CPU will be asleep for most of the time, and it will wake up in order to react to some external stimulus. In the case of the client node, this stimulus is either the reception of a command via its UART interface, or the press of the button. Since the computation times of these events is kept as low as possible, current measurements primarily show the consumption of the lower OS layers and the routing engine that build the basis for the client/server application.

UDP

While it is possible in Zephyr to add other even higher level protocols onto the socket, e.g. COAP or HTTP/REST, the implementation in this thesis utilizes the sockets directly, as this requires the least amount of changes in the protocol stack to introduce multicast capabilities. The multicast client is thereby setup as a UDP client by creating a socket and connecting to the multicast remote. Reception of server replies is implemented in the `udp.c` file, but is not used in this thesis, as bidirectional communication in a multicast group would flood the client with as many replies as there are members in the group. In order to receive messages during execution of the program, the multicast group has to be joined via the `net_if_ipv6_maddr_join()` method provided by Zephyr. The group address is set to be `ff03::fc` for all nodes. For easier traceability during debugging of both devices, the default selection of an IP address is disabled by manually adding the designated IPv6 address to the default communication interface of the device (IEEE 802.15.4 interface due to the configuration). A code snippet of this process can be seen in Listing 5.4.

```c
iface = net_if_get_default();
if (!iface) {
  LOG_ERR("Could not get te default interface\n");
  return false;
}

if (net_addr_pton(AF_INET6, CONFIG_NET_CONFIG_MY_IPV6_ADDR, &my_addr) < 0) {
  LOG_ERR("Invalid IPv6 address %s", CONFIG_NET_CONFIG_MY_IPV6_ADDR);
}

ifaddr = net_if_ipv6_addr_add(iface, &my_addr, NET_ADDR_MANUAL, 0);
if (!ifaddr) {
  LOG_ERR("Could not add unicast address to interface");
  return false;
}
```

Listing 5.4: Defining IP-addresses in Zephyr

5.2.2 Multicast Server

While the client is asleep until an external event triggers a transmission, the server continuously waits for transmission on its UDP interface. In order to retrieve the reception and routing statistics from the device, it also features a simplified UART interface for benchmarking purposes.

UART

The following commands are defined for the multicast server serial interface:

- 's'
  Reception of this command, triggers the generation of a statistics JSON response, which is then returned to the measurement node via the same UART connection.
- 'p<int>'
  Even though no direct messages originate from this node, there are routing messages and retransmissions of received data messages. Due to this, the transmission power of the RF circuit can be changed for the server node in the same manner as the multicast client.

UDP

Since messages are to be received from the client and not transmitted on their own, the UDP socket is created and bound to the multicast group address. This way, there is a registered listener for UDP data messages destined for the group.
Even though a listener is registered by this process, multicast messages will not be given to higher levels by the `ipv6.c` implementation. In order for this to happen, the multicast group has to be joined, analog to the multicast client. The code used to set up both the client and the server can be seen in Listing 5.4.
Waiting for incoming messages is implemented using the poll interface of Zephyr, which can be regarded as a type of semaphore between threads. In the case of this implementation, the method is told to wait for the UDP socket signal for message reception. If the signal is raised by the lower levels, the execution of the thread continuous and the payload is read from the input buffer via the socket `recv()` method, which will return the payload and the length of the received message.
If the message starts with the expected character, the main loop of the multicast server will toggle two LEDs in order to visualize the reception of a message.

```
join_multicast_group();

if(is_server) {
  udp_sock = socket(addr6.sin6_family, SOCK_DGRAM, IPPROTO_UDP);

  if (udp_sock < 0) {
    LOG_ERR("Failed to create \Gls{udp} socket: %d", errno);
    return -errno;
  }

  ret = bind(udp_sock, (struct sockaddr *)&addr6, addrlen);
  if (ret < 0) {
    LOG_ERR("Cannot bind to \Gls{udp} socket: %d", errno);
    ret = -errno;
  }
} else {
  udp_sock = socket(mcast_addr.sin6_family, SOCK_DGRAM, IPPROTO_UDP);

  if (udp_sock < 0) {
    LOG_ERR("Failed to create \Gls{udp} socket: %d", errno);
    return -errno;
  }

  ret = connect(udp_sock, (struct sockaddr *)&mcast_addr, sizeof(mcast_addr))
    ;
  if (ret < 0) {
    LOG_ERR("Cannot connect to \Gls{udp} remote: %d", errno);
    ret = -errno;
  }
}
```

Listing 5.5: UDP socket setup process

5.3 Measurement

While the test network consists of only two types of nodes, the measurement setup features a control software running on a PC that communicates the contents of a command file to the measurement nodes via a base-station. The evaluation of the measurement data streams is again handled by a software run on the control PC.

5.3.1 Measurement Client

The measurement client is the main application for capturing the benchmark data streams. It is running on the NRF52840 dongle, combining the functionality of the INA226 current sensor with a SD card storage to form the measurement node described in chapter 2. Its job is to set up the interaction with the INA226 and the storage, and to collect and store the current datapoints in a timely manner during operation.

INA226

Control and usage of the current sensor are implemented in a separate `ina226.h/c` library. The library provides the structure shown Listing 5.6, which can be used to configure the sensor according to the measurement needs. The resistor is thereby fixed by the used development board to 0.1Ω, the range can be set to 400/800 mA.

```c
struct ina226_conf{
    enum ina226_mode mode;
    enum ina226_average avg;
    enum ina226_conv_time conv_time;
    enum ina226_range range;
    enum ina226_alert alrt;
    float resistor;
    float range_ma;
};
```

Listing 5.6: INA226 configuration structure

In order to eliminate the use of polling to check if a conversion is done, the configuration `alert` is set to conversion ready, meaning that the `alr` pin of the development board is toggled every time the set type of conversion is finished. This toggling process in turn triggers a pin change interrupt executing `ina226_work_handler()`.
The job of the handler is to read the result of the current measurement via I2C, convert it to a fixed point number with 0.1 mA resolution and insert it into the buffer of the SD card. With `average` set to 1, `mode` to current measurement only and a conversion time of 140 μs, this process leads to a new datapoint in the buffer every 1.1 ms, as additional overhead of the I2C communication and the sample rate of the ADC itself needs to be run through. Nonetheless, this value very close to 1 ms means that the result of the current sensor can be used directly as the integral over the current, which gives mAms.

Data Storage

Internally, the SD card implementation features two separate buffers. While one is written to by the INA226 handler, the contents of the other one are written to the SD card via the Zephyr `disk-access` library. This library uses the SPI of the NRF52840 to interact with the micro-SD card. In order to avoid overwrites or changes during this store operation, the switch operation from one buffer to the other is protected by semaphores. If this semaphore access cannot be granted in the event that the communication with the SD card takes longer than expected, a certain number of samples will be lost. One option to combat this would be to introduce another buffer that is written to the SD card once the semaphore has been granted. The problem with this solution although is, that this again delays the write process of the pending larger main buffer, which results in more locking cases and therefore worse performance. The implementation of this process can be seen in Listing 5.7.

```c
datum_len = sprintf(buffer, "%d,%d\n", num++, value);

if(len[selector] + datum_len >= MAX_BYTE_TO_WRITE) {
  if(k_sem_take(&sem_sd_done, K_NO_WAIT) == 0) {
    if(k_sem_take(&sem_selector, K_NO_WAIT) == 0) {
      selector = (selector + 1) % 2;
      k_sem_give(&sem_selector);
      len[selector] = 0;
      ret = 2;
    } else {
      num--;
      return 0;
    }
  } else {
    num--;
    return 0;
  }
}
```

Listing 5.7: SD card buffer interaction

As it can be seen in the first line, the current datapoint is stored into the buffer in a Comma-separated values (CSV) format together with an index. Should the storage process of the SD card fail quietly, there would be an index jump larger than 1 for two subsequent samples. Data sets that present such an error can then be eliminated from further analysis.
The actual write process of the data to the flash memory is started once one of the buffers is completely filled. This event is indicated by setting another semaphore, that in turn releases the write process that is polled in the main loop of the software. The detailed write operation can be observed in Listing 5.8.
At first, the semaphore indicating pending data is checked. If the semaphore is available, the `k_sem_take()` method will return with '0' and the `if` statement is executed, otherwise it will return immediately without a timeout, as indicated by the `K_NO_WAIT` keyword. The next semaphore check is responsible for keeping the buffer consistent during the write operation, by locking the buffer switch operation, shown in Listing 5.7. Since `selector` always points to the buffer currently interacting with the INA226, the local selector is switched over for the write process. It is important to note that executing `fs_write()` will not store any data in the flash memory, instead it fills an internal buffer that is only written to the card if the file descriptor is closed or `fs_sync()` is executed. Overfilling the buffer will lead to a fault condition of Zephyr, crashing the application without recovery. In order to visualize a working SD card interaction, the red LED of the NRF52840 dongle will be toggled every time data is written to the flash memory.

```c
if(k_sem_take(&sem_write_ready, K_NO_WAIT) == 0) {
  if(k_sem_take(&sem_selector, K_NO_WAIT) == 0) {

    local_selector = (selector + 1) % 2;

    k_sem_give(&sem_selector);

    ret = fs_write((struct fs_file_t*)&current_file,
          (char *)current_buffer[local_selector],
          len[local_selector]);

    if (ret < 0) {
      LOG_ERR("Failed writing to file [%zd]", ret);
      return ret;
    } else {
      fs_sync((struct fs_file_t*)&current_file);
      len[local_selector] = 0;
      k_sem_give(&sem_sd_done);
    }

    return 1;
  }

  k_sem_give(&sem_write_ready);
}
```

Listing 5.8: SD card write operation

UDP

In order to control the benchmark remotely, the measurement node features an interface very similar to the UART interface of the multicast nodes. Since the measurement nodes cannot have a direct UART connection to the base-station, the interface is instead provided via UDP. The following commands are available:

- 'l', 'm', 'p'
  These commands do not concern the measurement node directly, instead, they are directly forwarded from the UDP interface to the UART connection of the UUT.
- 's'
  The statistics command will be forwarded straight to the multicast node via the UART connection. The UDP handler will then wait for a reply from the node. If the statistics string is received and successfully stored onto the SD card, `OK` will be returned to the base-station, otherwise the process will time out if the UUT is not reachable and a `NOK` is returned.
- 't'
  Receiving 't' for the first time starts the measurement process, i.e. it will create

a CURRENT<nodeid>.CSV file on the SD card and trigger the INA226 to start producing samples. Receiving 't' a second time will write the remaining data to the SD card and close the file access. In addition, the INA226 is stopped. If a measurement is running, the green LED of the NRF52840 dongle will be active.

5.3.2 Measurement Base-Station

The base-station node is used to form the link between the control PC and the IEEE 802.15.4 network used to control the measurement nodes. It is therefore a simple medium switcher, receiving commands via a USB cable and transmitting them over its 2.4 GHz radio. In order to guarantee correct behavior of the benchmark, it will check if commands are correctly received by the measurement nodes. For this process, all commands are implemented as unicast, meaning that even a command destined for all remote nodes, e.g. the test start command, will be sent to each node individually which in turn must answer in a set amount of time. If a node fails to do so, the benchmark will be canceled by transmitting this error back to the PC. Since, as already described, the IEEE 802.15.4 network is always to be regarded as lossy, the messages are retransmitted a set number of times in order to differentiate between an inactive node and one that is only temporarily not reachable. If the packet is lost on the way from the base-station to the remote node, this retransmission will fix a certain number of faults. If the command is received by the measurement node, its computation is started and the corresponding response is returned. If this response is lost and the base-station reissues the same command, this could lead to wrong states, i.e. a reissue of the test start command will end the test which in fact is already running. To combat this, every command is transmitted together with a sequence number. If the measurement node receives the same sequence number twice, the command will not be executed again, instead `OK` is returned directly.

5.3.3 Measurement Control

The job of the control software is to read a flow of commands from a command file and transmit them to the base station via a serial console. It thereby accepts the same type of commands as the measurement node with one addition, the sleep command. Listing 5.9 depicts a command file example.

```
w1;
t;
w1;
m1;
w60;
m1;
w60;
t;
s;
```

Listing 5.9: Example benchmark code

Executing the control center on this benchmark will cause the control software to initially sleep for 1 second. Afterwards, the test is started via the ’t’ command followed by another 1 second sleep in order to create a break between the startup of the INA226 and the first message event that is subsequently triggered by the ’m’ command. The control center will then wait for 60 seconds and repeat the same process once more, after which the test is stopped and the statistic data sets are written to the SD-cards via the statistics command ’s’.

CHAPTER 6

Evaluation

6.1 Static Analysis

During the lifetime of the node, it cycles through certain stages of current consumption. The amount of current consumed depends on the hardware itself, the state of the CPU, and also the configuration of the node in software.

6.1.1 Idle Behavior

For the most time, a node will spend its network lifetime in its idle stage, consuming a set amount of power. In the case of this thesis, the idle behavior is defined as an asleep CPU, with only the RX circuitry being active in order to receive and subsequently route packets. Since this describes the majority of the node's lifetime, it will also have a great impact on the consumption of the overall network.
Table 6.1 presents the idle current consumption during this phase, together with the current consumption during an active load and during a sleep without an active RX circuitry.

It can be seen from this data, that the presented current consumption of the NRF52840 CPU is lower than the one of the CC1352r1 across the board. The two left most columns thereby depict the idle state during network lifetime, the two right columns the

Current Consumption												
	RX-idle Client			**RX-idle Server**			**Active**			**Sleep**		
CPU	max	min	avg	max	min	avg	max	min	avg	max	min	avg
NRF52840	6.3	6.28	6.29	6.36	6.28	6.30	3.3	3.29	3.29	0.56	0.55	0.56
CC1352r1	6.76	6.72	6.74	6.86	6.79	6.81	3.4	3.37	3.38	1.67	1.66	1.67

Table 6.1: Current consumption during idle phases given in mA

consumption of the chip if the radio is switched off. Even in this lowest power case, the NRF52840 consumes 3% less power than the CC1352r1.
While this 450 μA decrease in current does not seem much, it can add up over time. Assuming a 2600 mAh battery powering both CPUs, the Nordic board will last 27 hours longer than the CC1352r1 with the multicast client running in RX-idle mode.
Another difference between the boards can be observed in the behavior between client and server. Implementation-wise, the client spends all its lifetime asleep, only waking up periodically in order to go back to sleep (10s sleep interval in the main loop) or due to an external event. The server on the other hand uses the Zephyr polling method to check for incoming messages. Due to this setup, one could expect the server to consume more power than the client, when run on the same hardware. While this is the case for the CC1352r1, the internal software stack of the NRF52840 handles this case differently, consuming practically the same amount of current for both cases. In terms of energy efficiency, this raised consumption on the CC1352r1 server leads to a 4 hour shorter lifetime of the node when both are powered by the same 2600 mAh battery.
The periodic wakeup from the sleep phase in the client can also be observed in the current consumption, as presented in Figure 6.1.

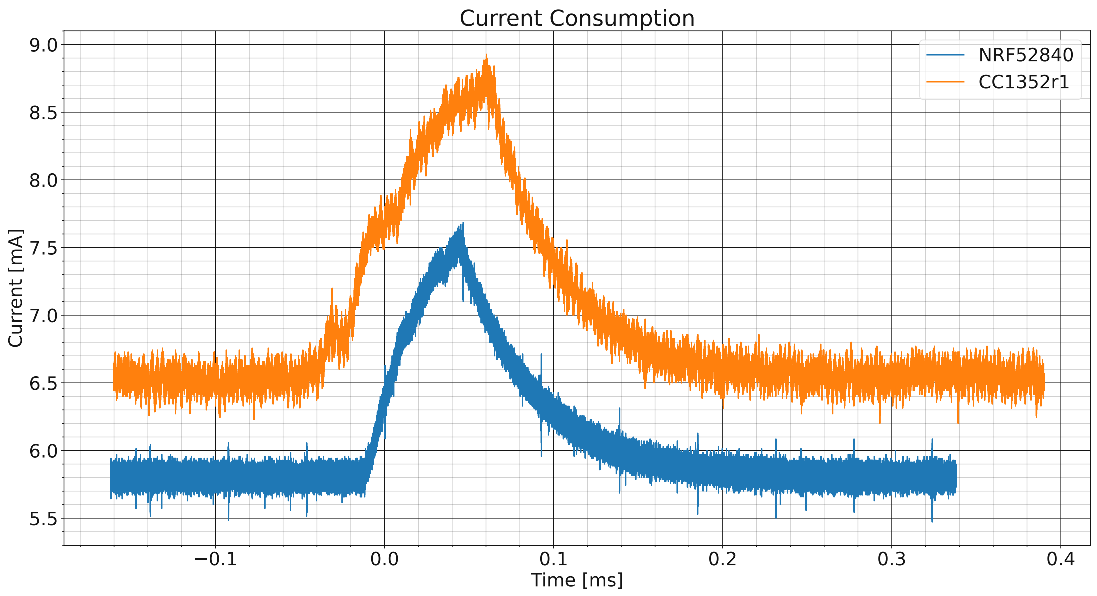

Figure 6.1: Wakeup phase NRF52840/CC1352r1

For the NRF52840 the current draw peaks at RX-idle plus 1.71 mA, i.e. 7.99 mA with the whole event lasting 250 μs. For the CC1352r1, the current increases 2.21 mA with an absolute peak height of 8.83 mA. In theoretical energy models, one might expect that the current consumed during this stage is the active current from Table 6.1 added onto the RX-idle current of the board, over the duration it takes the CPU to wake up and sleep again. In reality, the behavior of the current conforms to that of an RC charging curve, never reaching the constant active current, due to a event lasting less than the 4 times the time constant which would be required for reaching 98% of the peak current consumption.

6.1.2 TX Behavior

In order to capture the TX-behavior of the boards, message events are triggered manually by pressing the button on the multicast client. The following behavior is observed for both boards. In order to discuss the states of the message send event, the transmission power is fixed at 4dBm for both boards Afterwards the evaluation focuses on the effects the (adjustable) transmit power has on the current consumption.

Nordic NRF52840

Figure 6.2 presents the current draw observed when a 20 byte payload is transmitted at +4dBm with the NRF52840 DK development board.

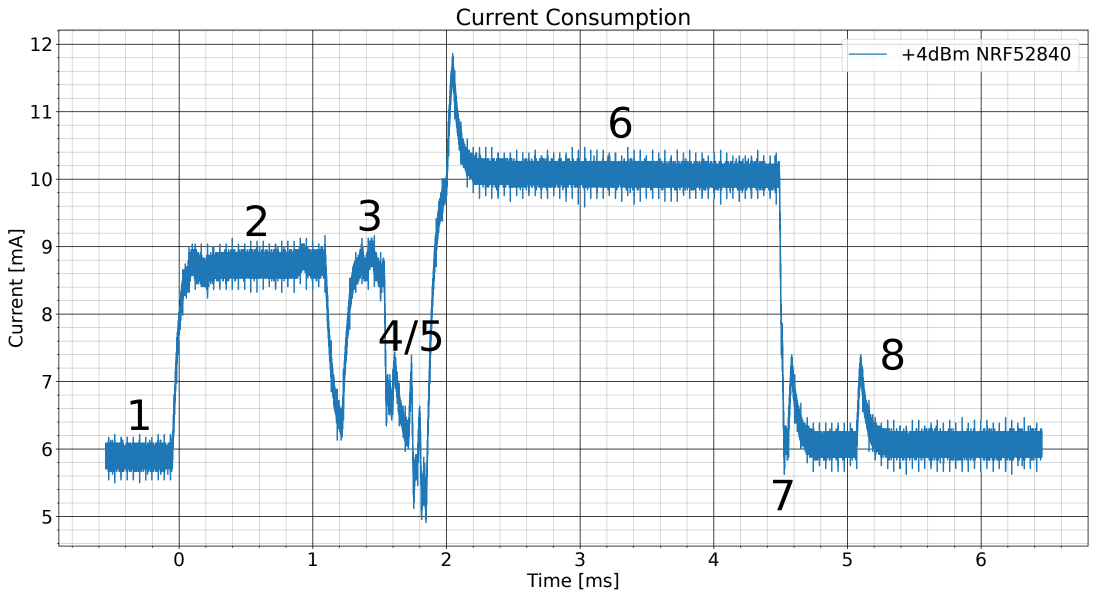

Figure 6.2: NRF52840 transmit event

As evident in the transient of the current consumption, the device cycles through a defined set of stages that can be analyzed and correlated to specific actions of the OS and hardware combination that forms the multicast client. Going through them step by step, the following key actions are identified by features in Figure 6.2. The phases and their function assignment is thereby evaluated twofold. Radio interactions are featured as a flow chart in the datasheet of the chip and can be assigned to the corresponding peaks in the timeseries. Software and OS phases on the other hand need to be identified iterative by breaking off certain processes at certain times. Returning before entering the routing engine, for example, will remove phase 3 from the graph.

- Phase 1:
  The CPU starts in the RX idle state with the corresponding current draw of 5.9 mA.

- Phase 2:
  The external event triggers the CPU, and the software in turn issues a UDP packet destined for the multicast group. The current draw ramps up with the corresponding time constant until the current reaches RX-idle plus the current draw of an active CPU, increasing by 2.8 mA to 8.75 mA.

- Phase 3:
  The UDP packet is assembled by the OS and the routing engine.

- Phase 4:
  The finished packet is given by the OS to the hardware radio of the NRF52840. The CPU returns to sleep, and the radio returns to RX idle in order to check if the channel is clear.

- Phase 5:
  If the channel is determined clear, the radio triggers a corresponding callback event that is indicated by a peak from the idle line. The RX circuit is subsequently powered off and the TX circuit is starting to ramp up.

- Phase 6:
  Once the circuit is fully powered up, the hardware triggers a callback at the top of the charging curve, indicating the entry into the transmit stage. The current draw during the actual transit increases by 4.17 mA to 10.08 mA. The peak at the start of the transmit process peaks at 11.67 mA.

- Phase 7:
  After the 2.5 ms transmission phase, the radio ramps down.

- Phase 8:
  The power down of the TX circuit and the return to RX-idle triggers callbacks in the software stack that can be seen as two peaks from the RX-idle current draw. The transmission of the packet is finished. In Figure 6.2, it can be seen that the current does not return to RX-idle in this phase, this behavior is only present if the message transmission is triggered by the button press. In this case, it will last between 160 and 200 ms. If the transmission is triggered via the UART command as it is the case in the benchmark, the current does in fact return to RX-idle immediately.

If the flooding routing engine is deployed, the transmission of the message is finished after these cycles have been run through. For MPL, two additional transmit events occur, the retransmission of the data/control messages. While the radio will always cycle through the presented steps, the computation of these special message types is distinct from the one that is triggered by the higher level implementation. The current draw during a data message retransmission can be seen in Figure 6.3.

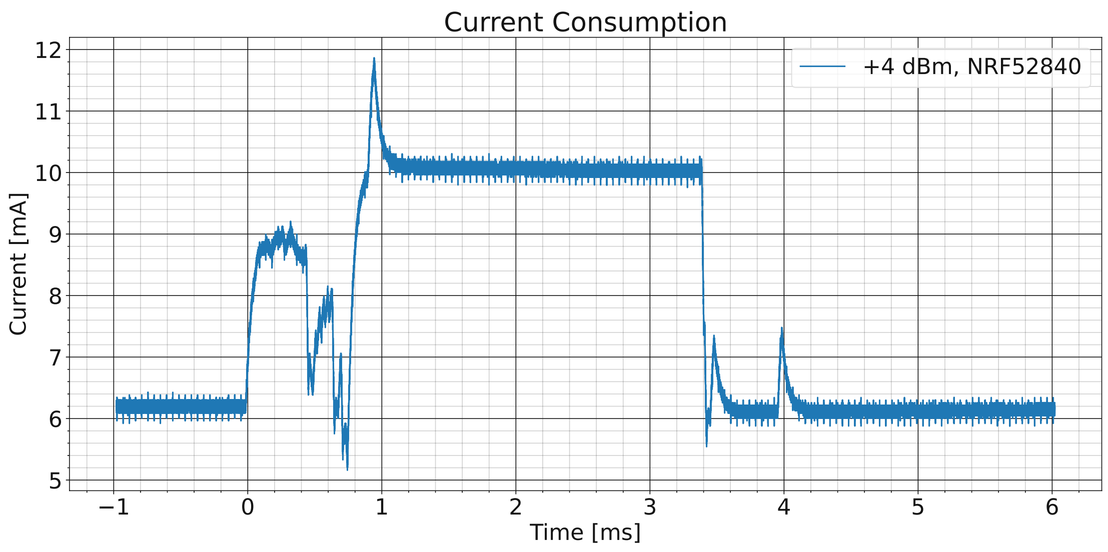

Figure 6.3: NRF52840 data message retransmission event

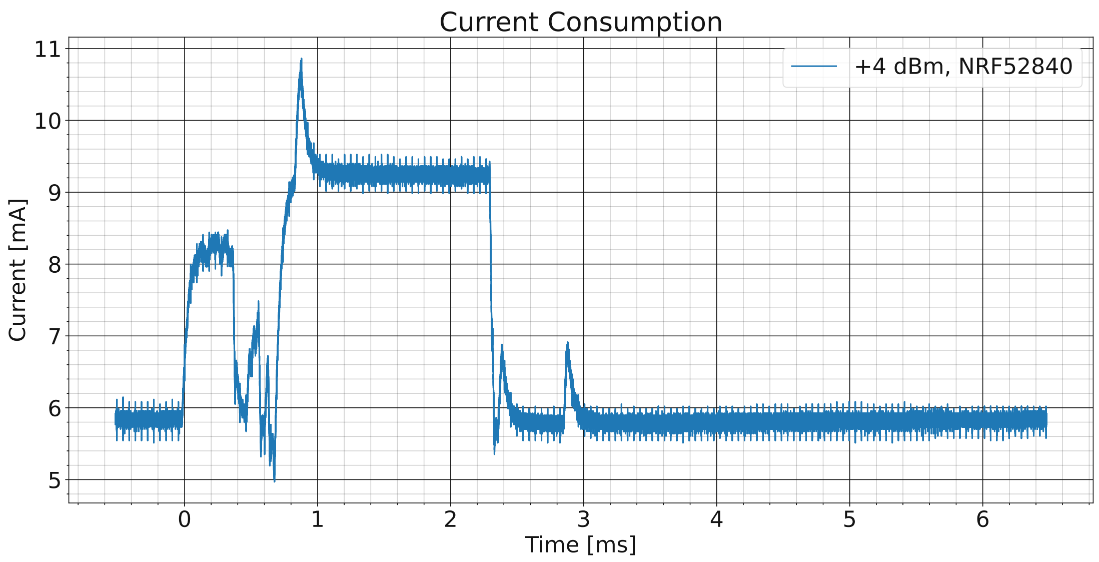

Figure 6.4: NRF52840 control message event

During the data retransmission, the overall structure of the current consumption stays the same with only one major difference, the software phase is shortened from 1.6 ms to 450 μs. This change highlights, that the packet trigger of the MPL data retransmission is the routing engine itself and not the client application running on top of the OS, which requires more computational effort.
The same can be observed in the control message current consumption presented in Figure 6.4.

In comparison to the direct message transmission, the time spent calculating and assembling the message is cut from 500 us to 400 μs, and the time to transmit the message is decreased from 2.5 ms to 1.48 ms, since in this case the payload and information within the control message is smaller than the data message including the 20 byte payload. The seed info set of the transmitted packet consists of only 1 seed with a singular message in its message set. After transmitting 12 additional messages, the software phase stays the same length, but the TX time of the radio increases from the 1.48 ms in Figure 6.4 to 1.51 ms. Since the transmission rate of IEEE 802.15.4 is fixed to 250 kbps the timing difference of 0.03 ms in these two control messages equals the length of one additional byte being transmitted. This observation therefore matches the implementation of the MPL control algorithm, which will require an additional byte for every 8 messages in the message set of the seed.
Overall, the following observations can be made for the transmission events: Direct messages and data messages feature a constant radio TX time but different computation times when the payload size stays the same. The current consumption of the control message does not depend on the payload, but on the number of seeds and the number of messages issued by those seeds. By increasing the payload of the data packets, the length of the radio TX stage is increased by the same amount measured in the control message transmission, 0.03 ms per byte added, but no measurable length changes were observed in the software phase. While the current consumption over time lends itself well to analyze the specific states taken by the device during transmission, energy efficiency analysis focuses more on the amount of current consumed. In order to do that easily, the time plot is integrated and combined with the state information obtained by the previous analysis. Applying this process to the plots shown in Figure 6.2, Figure 6.3 and Figure 6.4 yields the result shown in Figure 6.5.

Before iterating the datapoints over the time steps, the RX-idle consumption is subtracted from all datapoints, in order to just obtain the additional current consumption during the transmission event. The phases from Figure 6.2 are simplified to a software phase, i.e. the client application is running, an OS phase in which the routing protocol etc. is executed, a radio setup phase for Carrier Sense Multiple Access (CSMA) etc., the transmit phase (TX), in which the radio is actively ramping up or transmitting and the disable phase, i.e. operations and behavior after the TX event has finished. It has to be noted that the os and software phase are only possible to differentiate in certain situations, software and os consumption should therefore be seen as one block and are just shown in different colors in order to highlight the times when they can be separated.

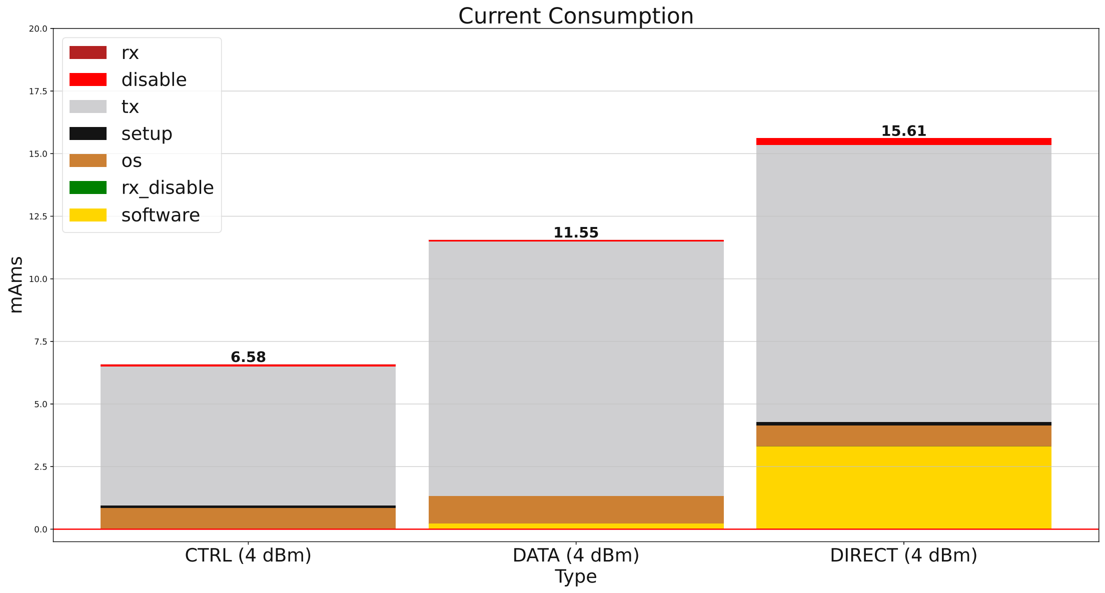

Figure 6.5: NRF52840 MPL transmission events current consumption

Using Figure 6.5, the shortened computation times for control and data retransmissions is more evident in comparison to the combined software time of the direct message. As expected, the TX phase consumption of the data retransmission and the direct message stays the same. The direct message transmission therefore presents the most expensive type, in regard to current consumption, for the MPL routing engine.

The power consumption during the TX-phase of the radio is heavily dependent on the amplifier power level of the circuit. For the NRF52840, the amplifier can be set in a range from +8 dBm to -20 dBm with 2 dBm steps. A comparison of the current consumption during these power levels can be seen in Figure 6.6.

Translating the information of Figure 6.6 in the same manner as in Figure 6.5 yields the comparison over all power levels shown in Figure 6.7.

Over all power levels it can be seen that the software phase consumption can be regarded as constant with a spread of only 0.0778 mAms between the maximum and minimum consumption. The consumption of the OS phase on the other hand displays a spread of 0.3 mAms. When looking at the positive transmission powers, the TX consumption increases on average 1.25 times every 2 dBm step. In the negative direction, the behavior of the system differs from the information given in the datasheet. While in theory the resolution step of the transmission power is 2 dBm, only 4 dBm steps changed the current consumption of the chip.
Summing up all the consumption phases for one transmission power event gives the values presented on top of the shown bars. It is interesting to note that at -20 dBm, the overall current consumption is lower than 0. This behavior indicates that sending the direct message with a 20 byte payload is in fact cheaper in terms of current consumption

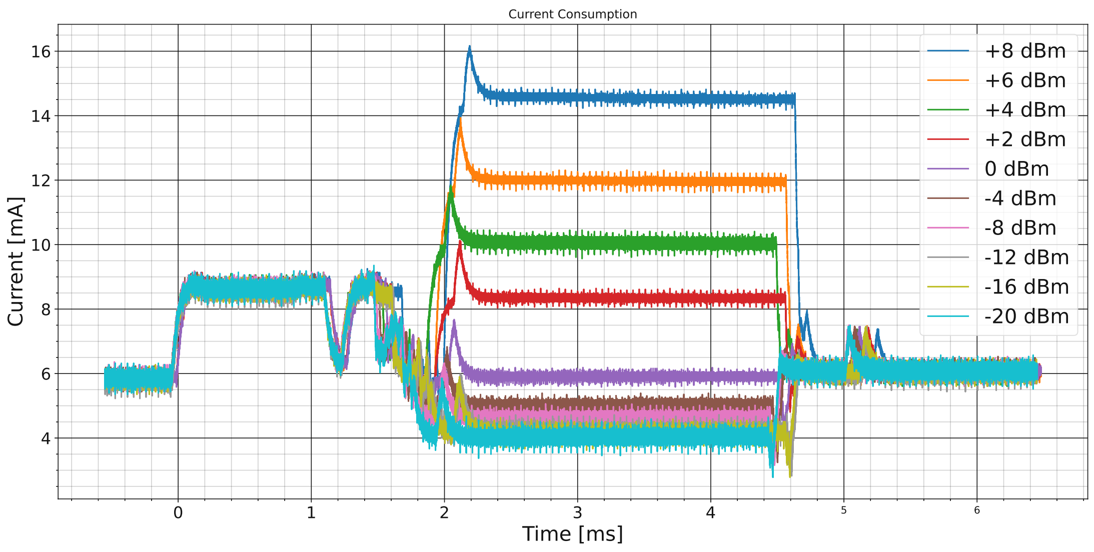

Figure 6.6: NRF52840 message current consumption with varying power levels over time

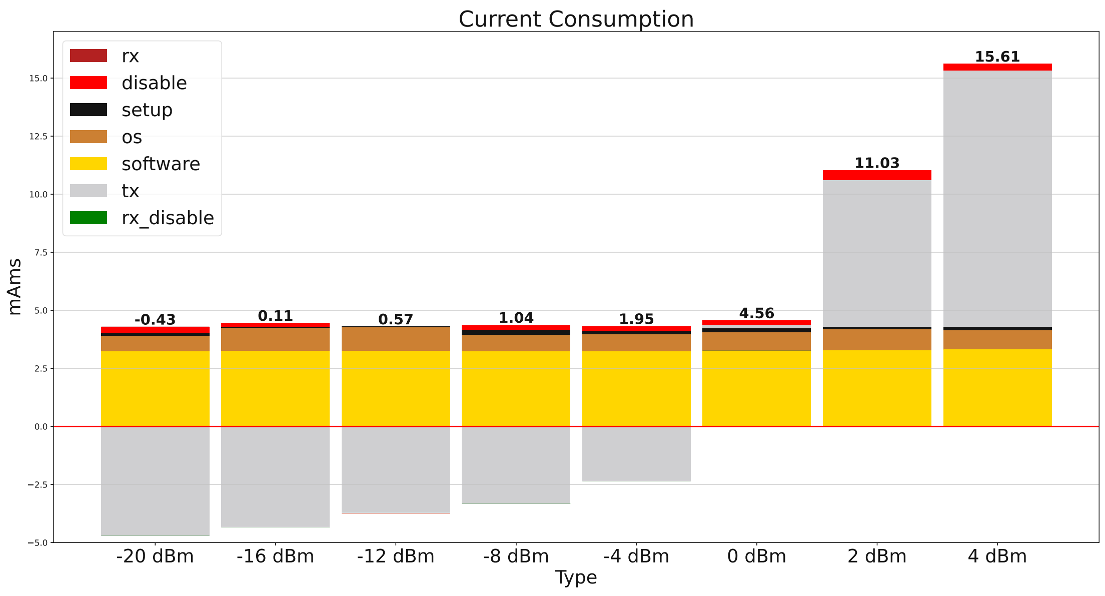

Figure 6.7: NRF52840 direct message current consumption with varying power levels

over time, as staying in RX-idle. At 0 dBm, the current consumed by the TX phase is practically zero, which means that at 0 dBm, the current consumption of message transmissions is practically just the amount of current consumed by the software running on the CPU.

TI CC1352r1

After programming the TI CC1352r1 CPU with the same application and configurations as the NRF52840, the current consumption shown in Figure 6.8 is observed in the measurement. It can be seen in the plot, that the behavior of this chip is different from the one observed in section 6.1.2.

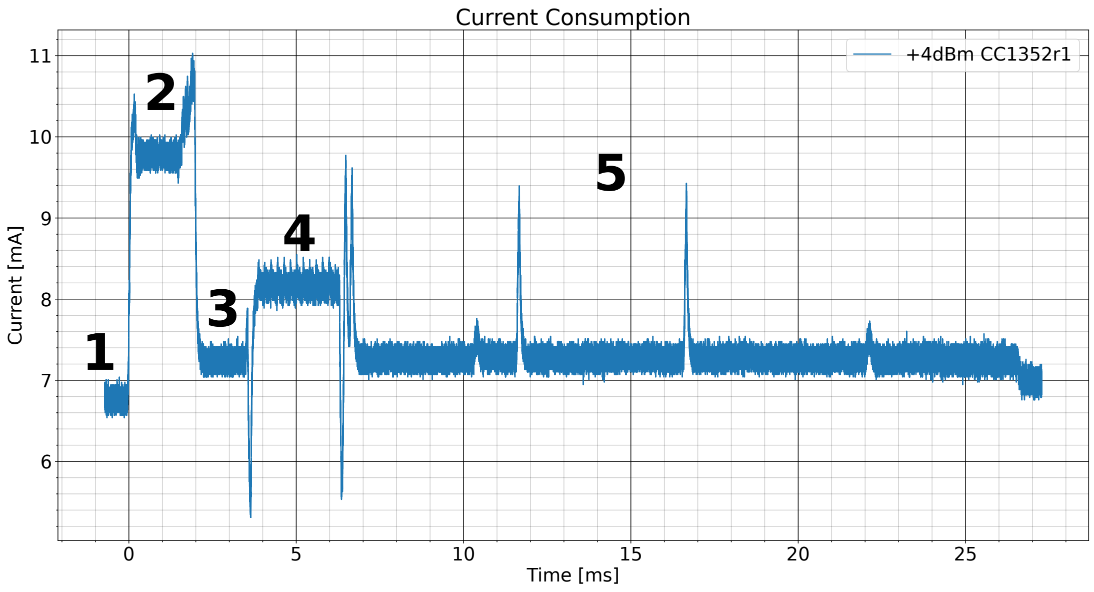

Figure 6.8: CC1352r1 message transmission

Analog to the Nordic analysis, the following stages can be defined by analyzing the plot.

- Phase 1:
  The chip is asleep, drawing the RX-idle current.

- Phase 2:
  The external event is registered, and the CPU switches to active mode, while the Nordic board allows the measurement of software and OS times, the CC1352r1 does not switch between the two and instead just features an additional current draw when the radio interaction is started.

- Phase 3:
  The CPU goes back to sleep, but internal processes still interact with the radio, i.e. CSMA is executed.

- Phase 4:
  The RX circuit is switched off and the TX circuit is used to transmit the packet.

- Phase 5:
  The radio is switched off again and the RX circuit is activated. During this disable phase, a number of events indicate state changes of the radio hardware.

While the settings enforce the same transmission power as with the Nordic board, the current consumption of the TX phase is significantly smaller than with the NRF52840. Since the transmission power of the TI board peaks at +5 dBm, the major energy is consumed by the software phase. The length of this phase is also longer due to the slower clock speed of the board. The combined software phase in the NRF52840 is 1.5 ms in comparison to the 2 ms of the CC1352r1, which corresponds to the 75% slower clock speed.
The consumption during data and control retransmissions correspond to the observations made in the previous section. The software times are reduced as only the OS is utilized in their computation. A comparison of the three transmit events can be seen in Figure 6.9.

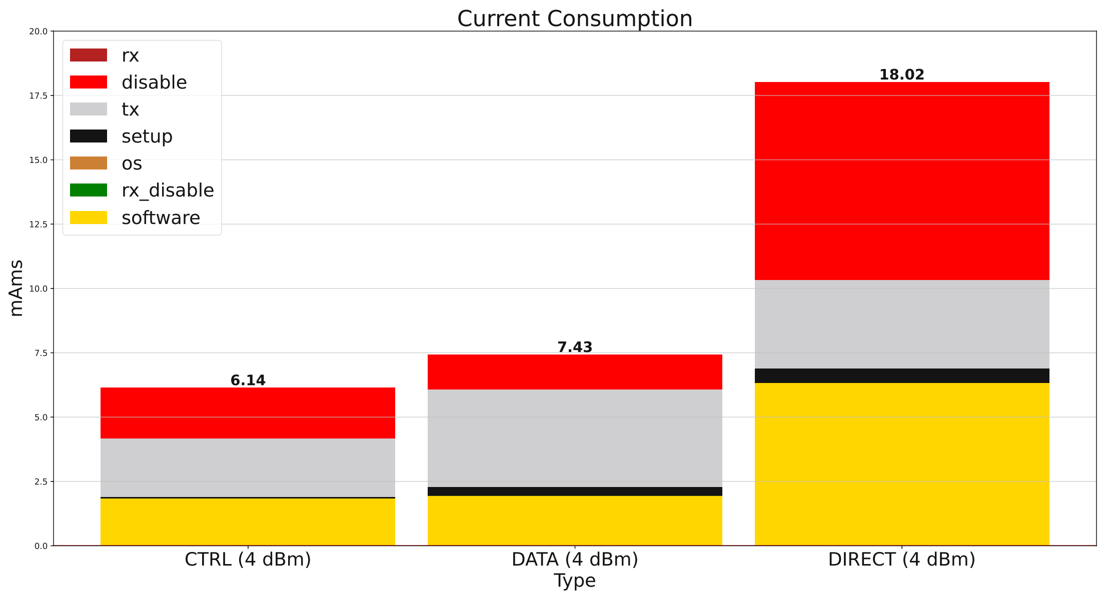

Figure 6.9: CC1352r1 TX event current consumption comparison

While the NRF52840 features a very notable disable phase when using a button as the external wakup event, the CC1352r1 features one for all trigger events. But the length of

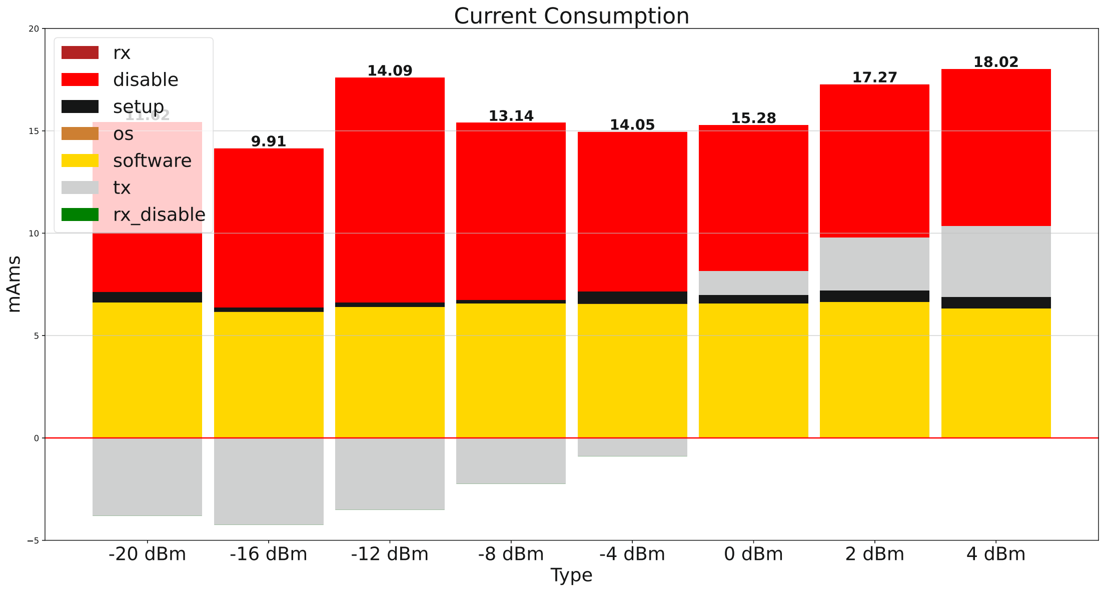

Figure 6.10: CC1352r1 direct message current consumption comparison

this phase is reduced from the 160-200 ms in the case of the Nordic chipset, to a more manageable 20 ms. Nonetheless, for the direct message transmission this phase consumes the most, followed by the software phase which consumes 1.36 mAms less. Combined, the TI solution consumes 15% more current when transmitting a direct message in comparison with the Nordic chipset. This fact is mainly caused by the high disable phase current consumption. As it can be seen from the bar chart, the TX phase itself consumes only a third of the consumption observed on the Nordic board. The positive effect of this can be seen in the data and control retransmission events, where the TI chipset consumes 35.6% less in the case of the data retransmission and 6.6% less in the case of the control retransmission. Except for the data retransmission, the chip consumes more power in the non-TX phases. In case of the data retransmission, the consumption of non-TX and TX phase are equal. The dominance of these software and setup phases can be seen across all power levels, in this case from -20 dBm to +4 dBm, as shown in Figure 6.10.

While the results of the Nordic static analysis shows a constant decline in current consumption when the transmission power is lowered, the results shown in Figure 6.10 present a more varying result. Since the disable phase is not fixed and the majority of the current is consumed in the non-TX phases, the reduction in transmission power is not enough to cancel out the more drastic length variations of the disable phase in particular. This reduced TX phase influence also leads to the fact that the current consumption stays positive across all power levels. The spread in the software current consumption is equal to the one of the software and OS current consumption of the NRF52840. Another interesting detail is the difference in behavior when using -20 and -16 dBm. The datasheet of the CC1352r1 defines a maximum power of +5 dBm but no minimum power level.

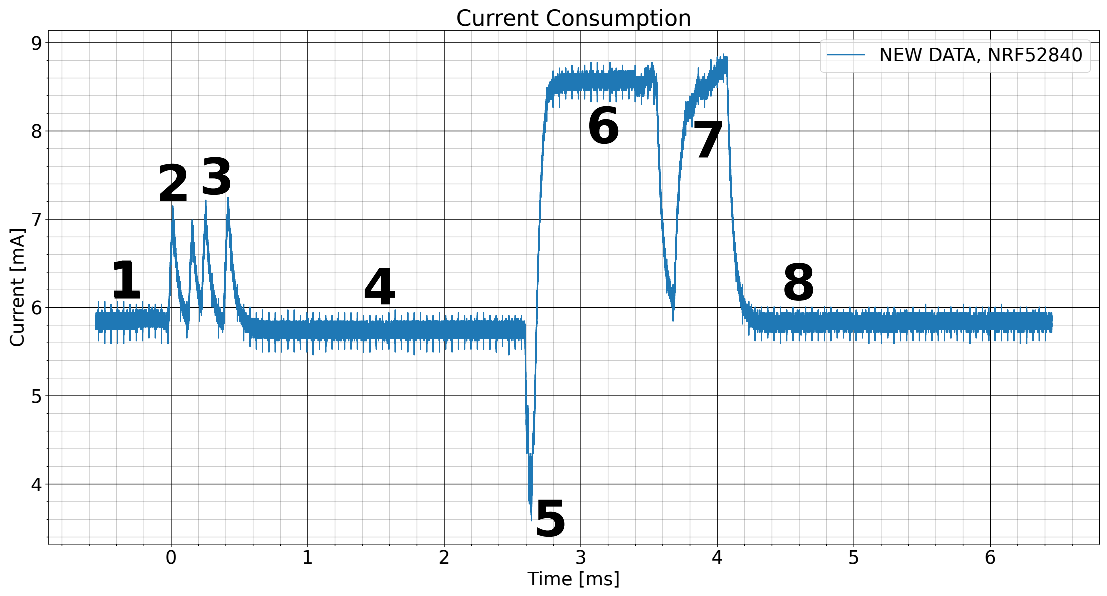

Figure 6.11: NRF52840 new data message reception

While the Bluetooth radio is given as min. -10 dBm, it can be seen in the chart that lower levels are possible with the IEEE 802.15.4 radio. The practical limit thereby seems to be -16 dBm, as an even lower transmission power consumes more energy. This might be caused by the fact that -20 dBm is smaller than the actual minimum transmission power, and the hardware reverts to a specific level. As with the Nordic board, the positive transmission power changes in 2 dBm steps and the negative with 4 dBm.

6.1.3 RX Behavior

The second type of routing event is the reception event. The two CPUs are again powered by a benchtop power supply, and their current consumption is measured when they receive the packet types transmitted in the previous transmission event analysis. The results of this measurement are shown in the following sections.

Nordic NRF52840

The current consumption during reception of a new data message is shown in Figure 6.11.

During message reception, certain phases can be identified.

- Phase 1:
  The board starts in its RX-idle stage, drawing the RX-idle current.
- Phase 2:
  The synchronization header of the IEEE 802.15.4 package is received, indicated by the first peak triggering a callback in the software stack.

- Phase 3:
  Additional header information reception is indicated by the further 3 peaks.

- Phase 4:
  The actual payload data follows after the last peak and reception lasts for the rest of the 2.5 ms on air time of the packet.

- Phase 5:
  According to the datasheet, the radio is switched off after a reception event, indicated by the negative spike in the plot. Due to the configuration of the OS, the radio is immediately switched on again.

- Phase 6:
  The reception of the data triggers the software stack and in turn the OS, which wakes up from its sleep mode and starts deconstructing the packet until it reaches the routing engine. The current draw in this phase is increased by 2.8 mA to 8.58 mA.

- Phase 7:
  The packet is determined to be destined for the reception node and forwarded to the higher layers of the OS and the main application.

- Phase 8:
  Due to the implementation presented in subsection 5.2.1, the node returns to sleep with a RX-idle current draw. The message is received and handled.

The spikes at the beginning indicate the reception of certain parts of the message header to the lower parts of the OS, the rest of the message is then received between the last spike and the switch off phase of the radio. The message is then first run through the lower parts of the OS and the routing engine and is then handled by the higher parts in a separate current consumption phase. It can be seen in this plot that no disable phase is present, as the current drops back to RX-idle after the message handling.
There are three reception types for MPL and two for flooding. For MPL a data message can be received twofold: One is the option shown in Figure 6.11, where a message is received that was not previously seen by the node. Two is the reception of a message that was already seen before, in which case the event follows the same initial pattern, but the computation time is cut short, as the packet is immediately dropped by the routing engine. The third type is the reception of a control message. A comparison of these MPL reception events can be seen in Figure 6.12.

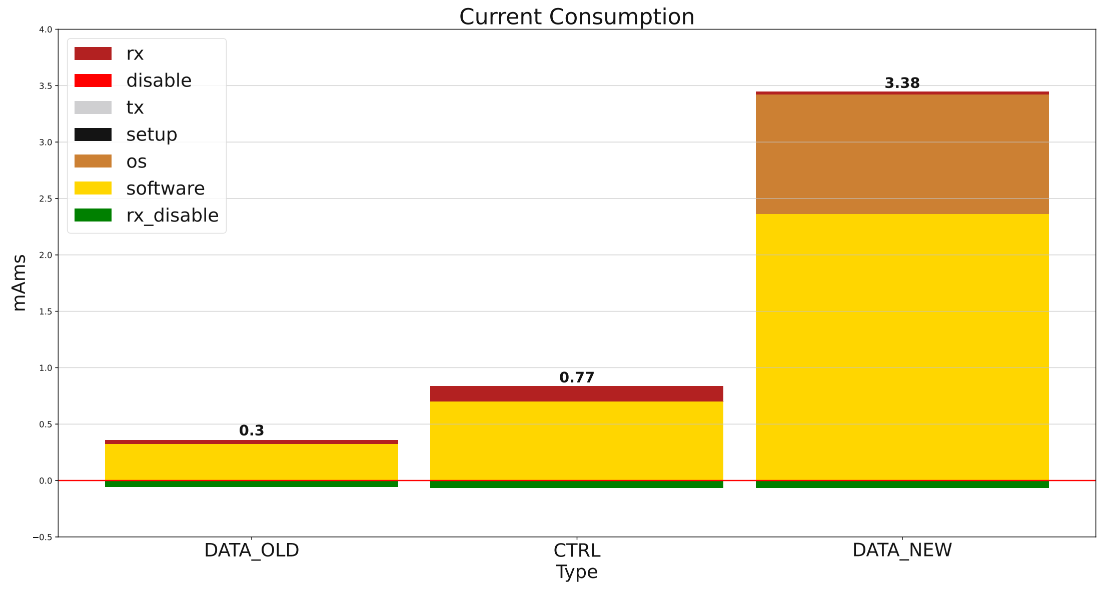

Figure 6.12: NRF52840 MPL reception event current consumption

Looking at these messaging events left to right, orders the bars not only in their current consumption, which correlates to their computational effort. Since an already received data message is dropped very early in the message handling, this option requires the least amount of computation. The control message on the other hand needs to be run further in the implementation, as it needs to check the received seed information against its own internal memory structures. The current draw of the software phase is thereby more than double in the case of the control message reception. The most power is obviously consumed by the reception of a new data message, as this message not only needs to be run through the OS layers, but also through the server application on top.
When receiving a message in flood based routing, the message is either discarded as old in the same manner as in MPL (left most bar in Figure 6.12), or it is received, computed and retransmitted, in which case the current consumption unfolds as shown in Figure 6.13.

In practice, this consumption pattern presents a combination of an ordinary MPL data reception event with an added TX phase. The length of the software/OS phase is thereby comparable to the external event handling in Figure 6.2.

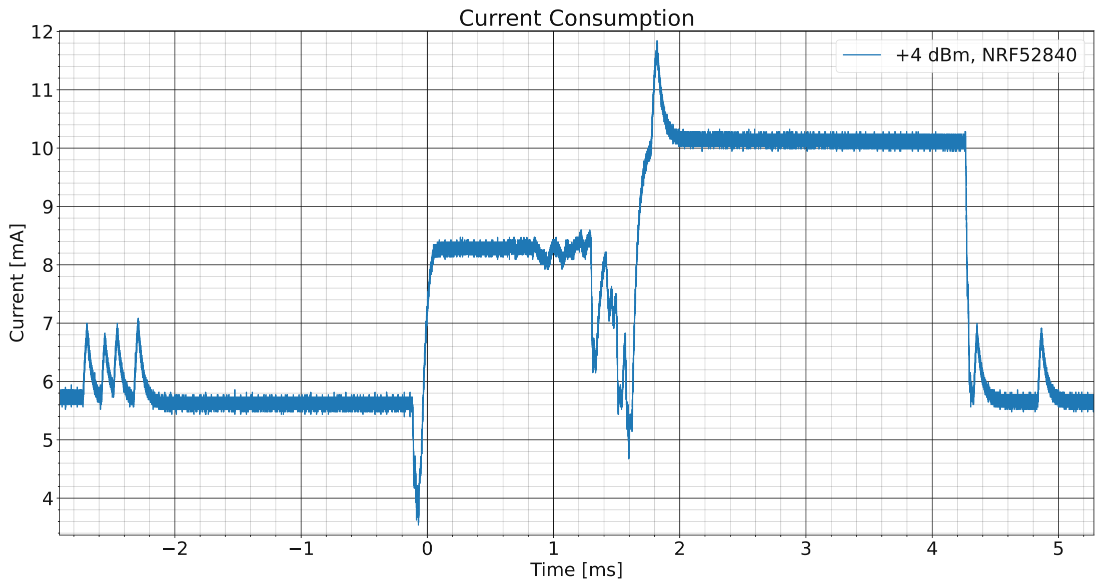

Figure 6.13: NRF52840 flood reception event current consumption

TI CC1352r1

The consumption pattern during the reception of a new data message on the CC1352r1 is shown in Figure 6.14.

While the reception of the actual packet in the early phase 1 of the reception event, consumes marginally less than RX-idle in the Nordic chipset, except for the dominant initial spikes, it features a positive current draw in the case of the CC1352r1 with additional event spikes added upon this consumption. The RX phase is then followed by the typical computation current draw and analog to the Nordic board with a disable phase, which ends 24 ms after the completion of the computation. For the CC1352r1, it can be said that a disable phase is present, meaning a positive deviation from RX-idle, as soon as higher levels of the OS or the application itself are active. Notable are also 3 additional event spikes during the disable phase, which indicate state changes in the radio hardware.
Reception of the other two message event types mirrors the behavior of the Nordic chipset, as evident in Figure 6.15. The computation times are cut short, with the reception of an old data message having the lowest current consumption. While the control message consumed 2 times the current in the NRF52840, for the TI chipset the difference is less drastic, with control message handling consuming only 35.1% more power. Due to the addition of the dominant disable phase, the reception of a new data message is 8.3 times more expensive than the reception of the old message. In comparison, due to the faster execution of the message drop, it is 11.26 times more expensive in the case of the NRF52840.

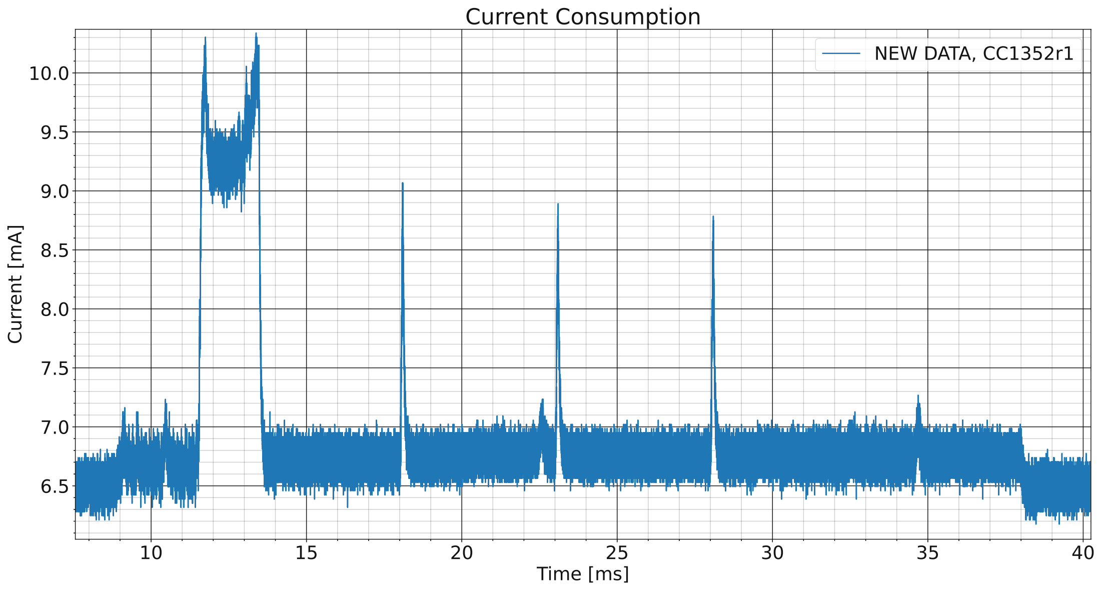

Figure 6.14: CC1352r1 data reception event current consumption

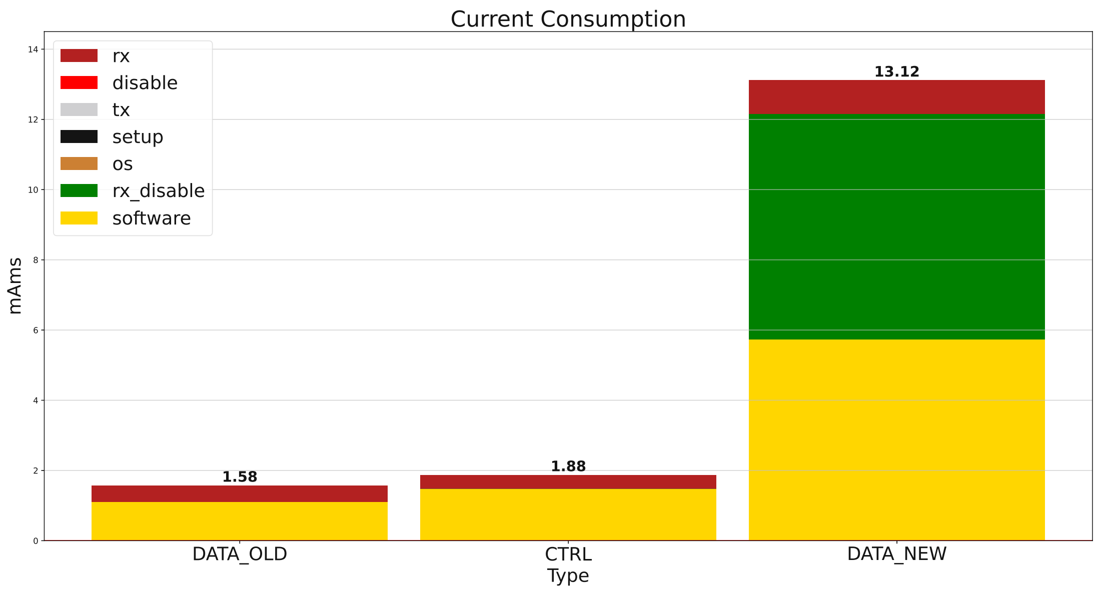

Figure 6.15: CC1352r1 reception event current consumption

Reception events during flood routing feature the same behavior as shown in Figure 6.13 with them being a combination of a data message reception event with an added TX phase, as shown in Figure 6.16.

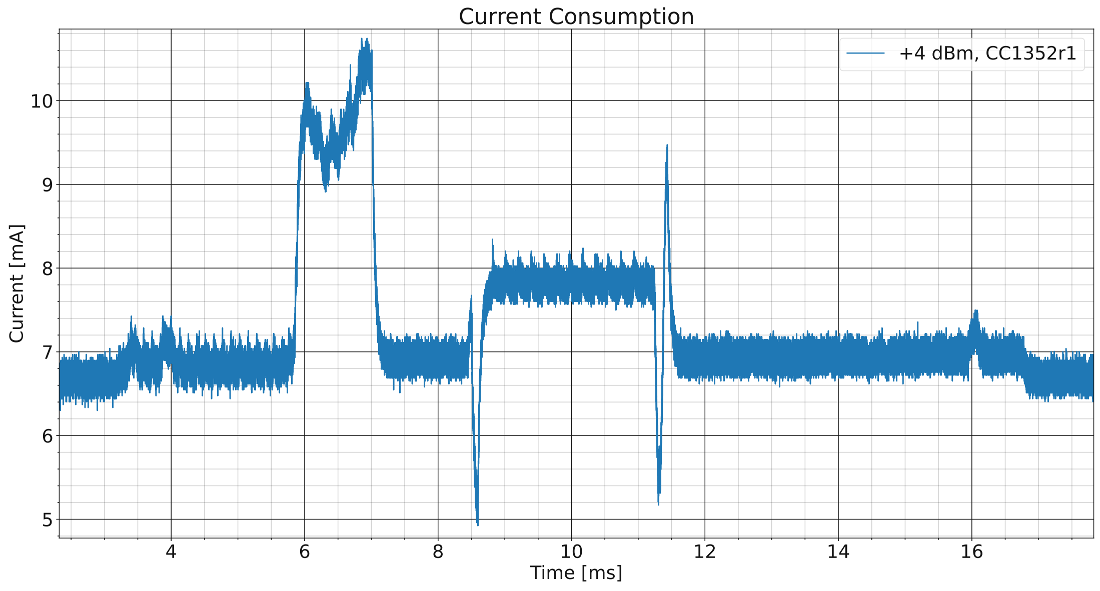

Figure 6.16: CC1352r1 flooding reception event

Since a new data message is received in the shown event, a disable phase occurs, but while during MPL message reception the whole duration of the disable phase is practically wasted current, flood routing uses the phase to retransmit the received message. This additional hardware interaction is observed to not prolong the disable phase. Overall, this therefore presents a more energy optimized message event than the two separate events that at least have to occur with at least 1 data message retransmission.

6.1.4 Comparison

Combining the transmission and reception events of both routing engines yields the chart shown in Figure 6.17 for the NRF52840.

The three right most bars represent all types of events associated with the flood based routing, the remaining bars on the left represent MPL routing events. The transmission of a direct message and the reception of an already handled data message is thereby the same for both routing engines. When looking at the flood retransmission and the direct message side by side, it can be seen that the combination of software and OS phase consumes less current in the case of the retransmission and the same amount as the MPL new message event. The radio trigger action and subsequent ground up execution, going first into the routing engine and only activating the application afterwards, distributes the current differently between OS and software phase, but ultimately the same amount of current is consumed as in the MPL new message reception event. This means that the retransmission of a message in flooding is cheaper than the data message retransmission in MPL. The flood retransmission is a new data event with an added TX phase, the overall MPL retransmission on the other hand features two software/os phases as the

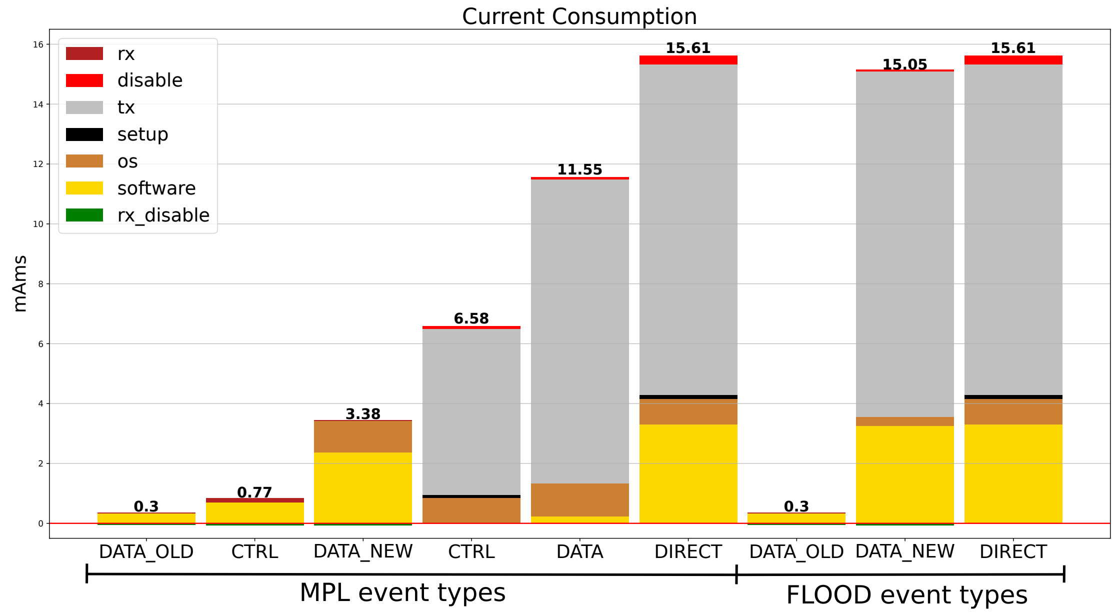

Figure 6.17: NRF52840 routing event type comparison

actual TX phase needs to be triggered, and the message assembled after the elapse of a Trickle timer.
In comparison, Figure 6.18 presents the same result for the CC1352r1.

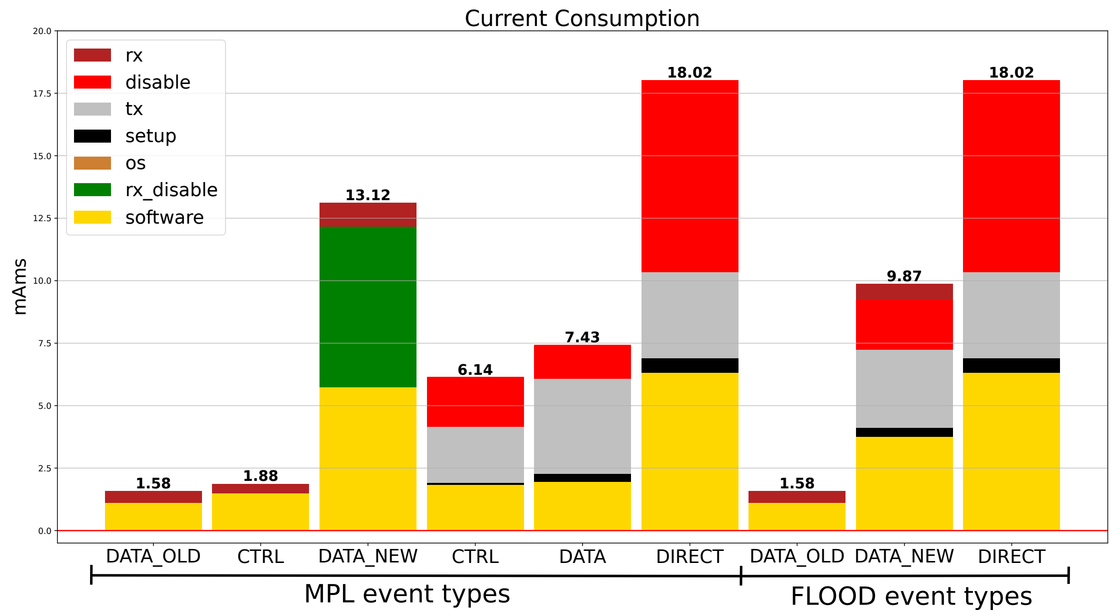

Figure 6.18: CC1352r1 routing event type comparison

For the TI chipset, it can be seen that the reception of a new data message is in effect, more expensive than the data/control retransmission of MPL and the flooding retransmission. This is mainly caused by the addition of the disable phase. What this also means, is that flooding running on this board is particularly cheap in comparison. Even without the disable phase, the slower clock speed and therefore higher current consumption in the software/OS phase benefits a lot from the reduction in computational effort that is offered by the flooding retransmission in comparison to the two separate events in MPL. To underline this and to determine the most energy efficient routing protocol/setting for the used protocol, these results can be combined, to form the worst-case analysis for a very simple network that features only two nodes. In this theoretical network, the nodes are placed in such a way that all messages are delivered, i.e. a reception rate of 100%. In the case of this thesis, 2 routing engines, totaling 5 overall configurations, are deployed. Four of them are MPL with varying settings and one is the flood based routing as a baseline. The settings of the MPL routing engine are listed below.

- 1D0C
  Every message transmission triggers 1 data message retransmission and no control message.
- 2D0C
  Every message transmission triggers 2 data message retransmissions and no control message.
- 1D1C
  Every message transmission triggers 1 data message retransmissions and 1 control message.
- 2D2C
  Every message transmission triggers 2 data message retransmissions and 2 control messages.

In order to form this analysis, one message is transmitted from one node to another. The redundancy constant of the MPL routing engines is set to 1, meaning that only half of the retransmissions are carried out. E.g. four Trickle timer retransmission events are scheduled in the case of 2D0C, every elapsed Trickle timer will cancel exactly one Trickle timer of the other node. Figure 6.19 presents a comparison between the NRF52840 and the CC1352r1 running these routing engines with 4 dBm transmission power.

Side by side, it can be seen that except for the flooding based routing engine, the NRF52840 consumes less energy than the CC1352r1 in the small-scale 2 node scenario. The reason for the paradigm shift, when looking at flood routing, is again that the involved message types in the message exchange are particularly cheap for the CC1352r1. The faster NRF52840 does not profit as much from the saved execution time, as its current consumption is dominated by the TX phase. It is for this reason that flooding

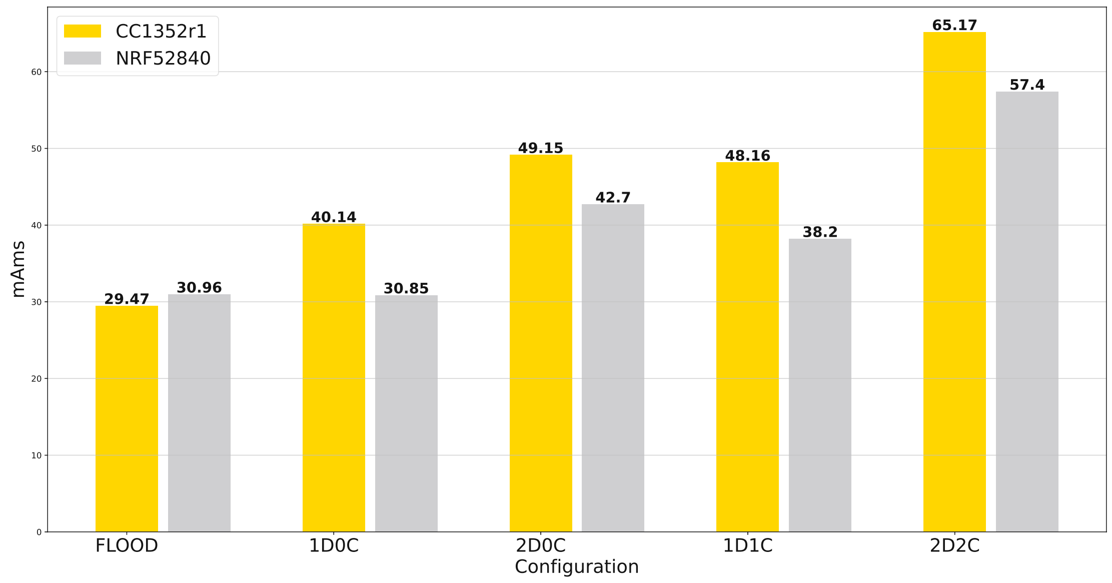

Figure 6.19: Static routing current consumption comparison at 4 dBm

and 1D0C MPL lie only 0.11 mAms or 0.35% apart.
The next two columns are 2D0C and 1D1C routing. Here both chipsets fall in line with the option of using a control message instead of an additional data retransmission consuming less. The difference is hereby larger for the NRF52840 since the dominant consumption is the TX phase, the shorter control message imprints stronger on these results. While the CC1352r1 also gains from this TX phase length decrease, the effect is mostly negated again by the software phase, which is longer for control messages.
Due to the much higher number of messages, 2D2C routing sticks out as having the highest current consumption over all.
What the results in Figure 6.19 as well as the presented transmission/reception bar charts omit is the additional difference in RX-idle current consumption. As shown in the previous chapters, the CC1352r1 consumes more power in this regard than the Nordic chipset. As with the long but small magnitude disable phase in the CC1352r1, these small changes can have a large impact over time. If this higher RX-idle consumption is added onto the results for the CC1352r1 inFigure 6.19, the results are shifted, as shown in Figure 6.20.

If the developer is required to choose between the two boards, this 0.19 mA increase in RX-idle consumption has a great influence on the choice of the more energy efficient system. While routing stays as expensive as in the previous case, the board itself is now a lot more expensive for continuous operation. This is especially true if the transmission power of the system is lowered from the previously used +4 dBm.
The result of the same small-scale simulation with a transmission power of 0 dBm can be seen in Figure 6.21, the RX-idle consumption is thereby omitted again.

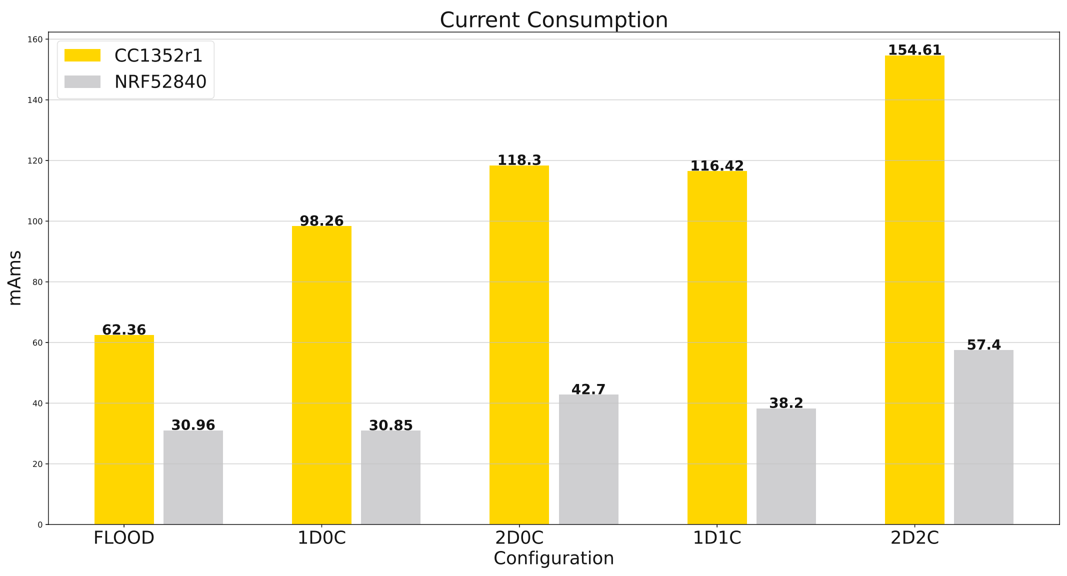

Figure 6.20: Static routing current consumption including the RX-idle difference

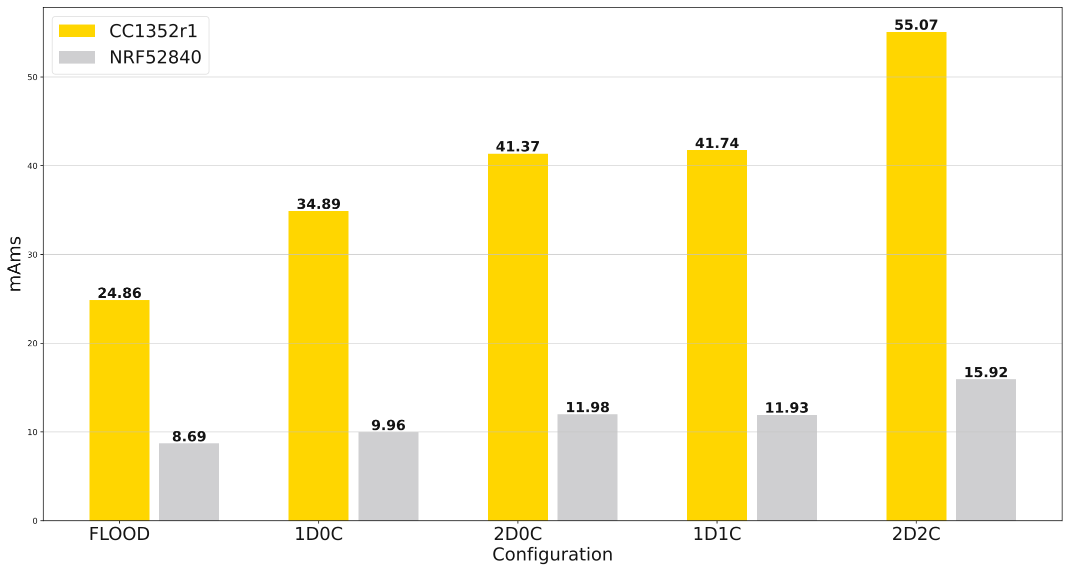

Figure 6.21: Static routing current consumption comparison at 0 dBm

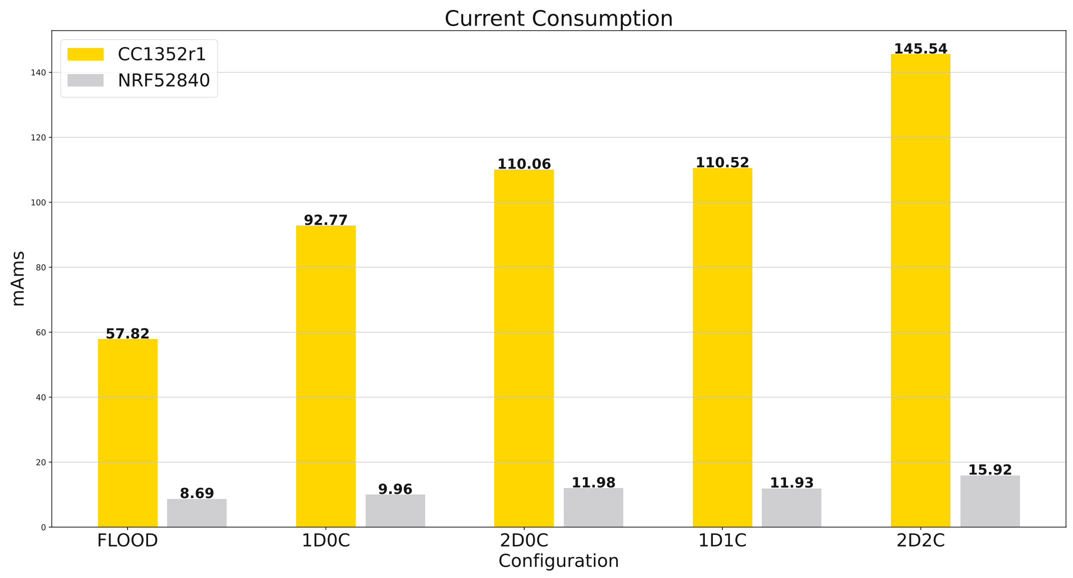

Figure 6.22: Static routing current consumption (0dBm) including the RX-idle difference

Reducing the transmission power from 4 to 0 dBm, lowers the routing consumption of the NRF52840 on average 42.7%. The CC1352r1 on the other hand only consumes 14.67% less power. Factoring in the RX-idle consumption, this becomes even more drastic for the TI chipset, as shown in Figure 6.22. Since the current is now shifted by this factor, the reduction in current consumption is decreased to only 6.14% less current consumed.

The main reason why the NRF52840 drops more than the CC1352r1 when reducing the transmission power is that the TI current consumption is software instead of transmission dominated. Even when the transmission power of the TI chipset is set to its maximum value, the software/routing phase is still consuming more current, as seen in Figure 6.17. Interesting to note is that the consumption of the routing engines running on the NRF52840 have started to equal out, while 2D2C consumed 33% more at 4 dBm in comparison to 1D1C, 0 dBm decreases the difference to 25%, making 2D2C more attractive to use if it offers a better network performance.
In this very small case of only 2 nodes with a 100% message reception rate, it is easy to determine the number and the type of the messages that are transmitted, once difficulties like lossy connections/interference's or multiple nodes are introduced, this process starts to get very difficult. Here the use of a simulation or real-world hardware comes into view. The network can be set up and a benchmark run produces the number and the type of the messages that are exchanged. These results can then be combined with the results of this static analysis in order to estimate the current consumed by the routing engine on different hardware and using different transmission powers. In this thesis, the benchmarking and the generation of the network performance data is part of the dynamic analysis.

6.2 Dynamic Analysis

The routing engines are benchmarked based on the two low scale network topologies presented in chapter 2. Every benchmark run hereby has 60 messages, with one being triggered every minute.
As described in chapter 5, every run of the benchmark will produce the measured current consumption over time as well as the statistics file.
The nodes are deployed standalone during the benchmark, meaning that they are powered via suitable battery packs instead of the laboratory power supply in the previous chapter. These variables introduce additional effects when looking at the current consumption measurements.
Executing the triangle benchmark running MPL with 1 data and 1 control message retransmission yields the current data set shown in Figure 6.23 for the NRF52840 DK board. The constant K is thereby set to 1 for both data and ctrl messages.

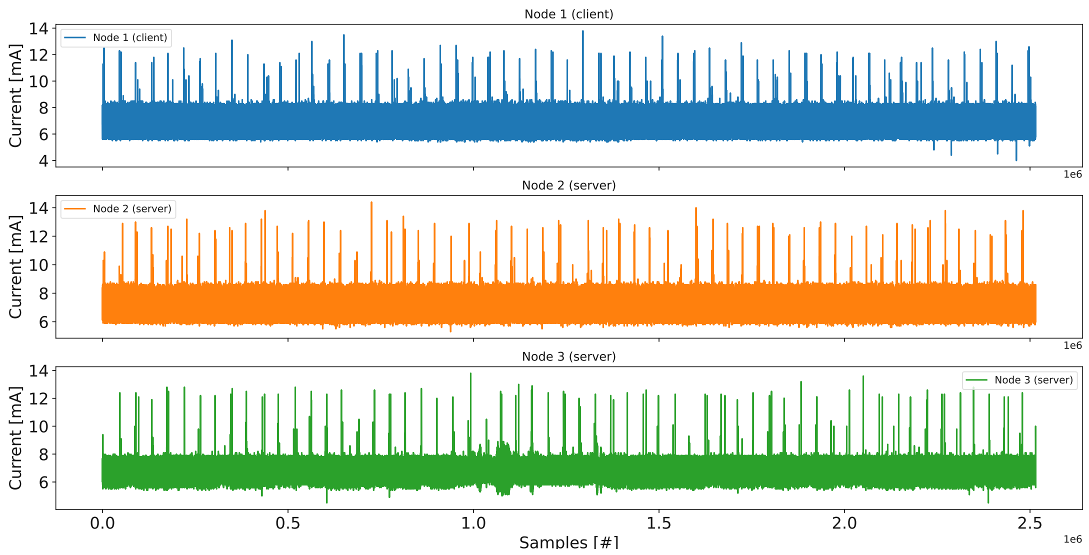

Figure 6.23: NRF52840 DK network current consumption with 1 data and 1 ctrl retransmission

Comparing the current consumption of the Nordic board in Figure 6.23 with the behavior of the TI LaunchXL board shown in Figure 6.24 yields certain differences. Most notable hereby is the amount of noise around the RX-idle current of both boards, for the NRF52840 DK, this baseband ranges from 5.5 to 8mA (2,5mA peak to peak) while the LaunchXL board has an average noise band only one-sixth as wide with 7 to 7.4mA (0.4mA peak to peak).

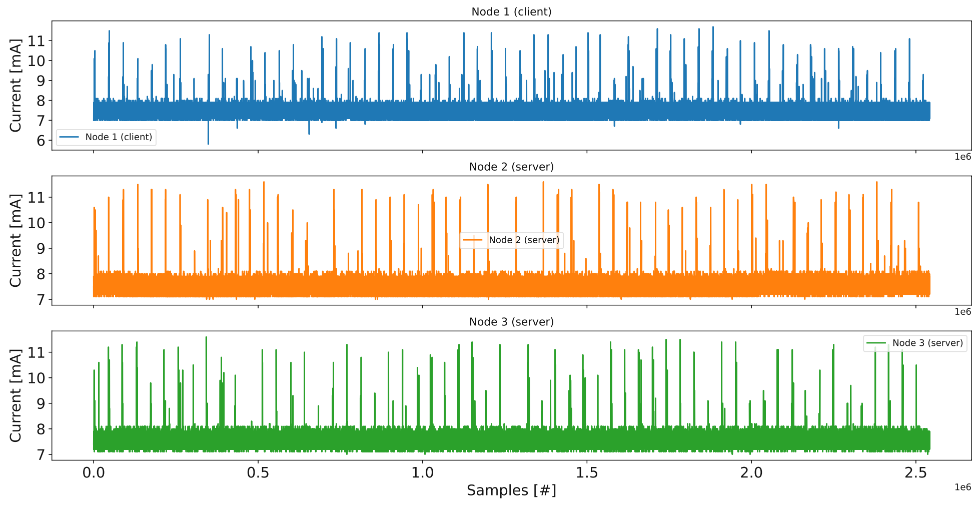

Figure 6.24: TI CC1352r1 LaunchXL network current consumption with 1 data and 1 ctrl retransmission

As with the current plot in the static analysis, the main feature that needs to be extracted from these datasets is the overall consumed current. In order to accomplish this for this large dataset, the datapoints are translated into a histogram. For this, the current range between 4 and 15 mA is divided into 110 bins, the measured datapoints are then assorted into these bins by summing up how often this current is measured during the benchmark. Executing this algorithm on both datasets, shown in Figure 6.23 and Figure 6.24, yields the histogram shown in Figure 6.25.

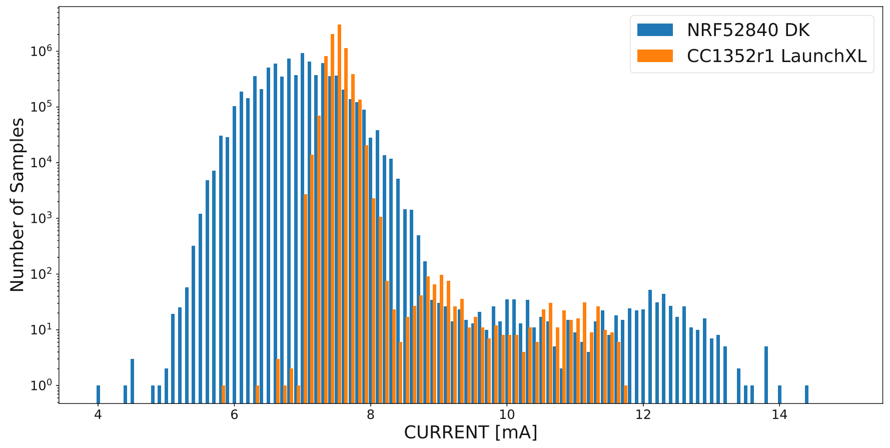

Figure 6.25: Current consumption histogram comparison

Analyzing the bins of the histogram presents certain information about the behavior of both boards. Overall, three main regions can be identified in the plot, separated by the

RX-idle current. Since power supplies are inherently noisy, the RX-idle current is not a singular bin but a range of values, which is called the baseband region in the further analyses of this thesis.

- Region 1:
  A few samples can be seen below the bulk of the baseband region. In general, they depict samples of currents that lie below the RX-idle current consumption, for the Nordic board these lie relatively close to the baseband as radio mode switching of the CPU is a very fast transient event in which the current does not drop a lot under the RX-idle current. The TI board on the other hand has a higher number of samples that deviate more from the baseband, as every TX and RX event introduces negative spikes due to radio mode switching (as seen in section 6.1). The higher the product of current and number of samples on this region, the more energy is saved, compared to the node always running in RX-idle.
  In addition to this, TX events with a transmission power lower than 0 dBm will also show up in this region, i.e. saving energy by using the low power mode of the TX phase.
- Region 2:
  Region 2 is the baseband, i.e. the RX-idle current consumption of the board convoluted with a noise of varying amplitude creating a high number of samples as the board spends most of its time idling in this state. Analysis of this region lends itself more into the performance of the power supply itself than to compare routing protocols, since RX-idle is a mandatory current draw of the system. For advanced network types and routing protocols, this region could also be of interest if radio duty cycling is enabled. In this mode of operation, the RX circuit of the CPU is not always on, but instead is switched off at certain times, reducing the power consumption of the CPU to CPU-sleep. Since in comparison to the other two regions, most of the samples fall into this region, this by default offers the best vantage point to reduce the overall current consumption of the board during use. As shown in section 6.1, the inclusion of a higher or lower RX-idle consumption can greatly shift the results of two boards against each other.
- Region 3:
  For the routing engines described in this thesis with power levels >0 dBm, this region is the most important in regard to energy efficiency. Samples in this region depict current draws over RX-idle, e.g. during CPU-active times or during TX events. The higher the product of current and number of samples is in this region, the more power is used up by the node communication. For transmission powers lower than 0 dBm this region will only show the computation consumption of the routing protocol.

With these three regions in mind, certain measurements of the static analysis are confirmed, while new information is also generated and added by the introduction of the additional

board hardware, primarily the on-board power supply.
The higher RX-idle current of the CC1352r1 is confirmed by the histogram, the main peak of the TI board is located more to the right than the main peak of the NRF52840. The current consumed during TX events is also higher in the case of the Nordic board, since the distribution of the bins reaches higher, current wise, than in the case of the TI board.
The most striking difference between the two boards is the width of the baseband region. The power supply of the Nordic development board is much noisier than the one powering the CC1352r1. This also impacts the TX event current consumption, as this noisiness is added upon their base current consumption as well. In order to analyze the impact of the real-world hardware onto the 4 dBm 1D1C case, the bins responsible for the TX operation are extracted from Figure 6.25. While the timing information is lost by the histogram generation, the TX event current consumption can be approximated by integrating over region 3 and subtracting the integrated value of region 1. In order to do this, the number of samples in the bins is multiplied by the current associated with the specific bin and the results of this are then summed up. Region 3 is thereby set to start at the bin after the baseband region with a number of samples smaller than 0.1% of the maximum height bin.
While the static analysis offers an optimistic view in regard to the power supply and the behavior of real-world hardware in a laboratory setup, the dynamic analysis adds upon these effects caused by the software, which are not part of the routing engine and also the influence of additional board hardware as already discussed in the analysis of Figure 6.25. Static and dynamic analysis therefore offer two levels of analysis of the same system. Static lends itself more to the evaluation of routing engines running on a specific CPU, while the dynamic analysis can be used to evaluate the routing engines in combination with a specific IoT device assembly. Since the statistics file of the benchmark run keeps track of the number and type of packets that are sent, both types of analyses can be compared against each other. In case of the Nordic board, the result is the bar chart shown in Figure 6.26. Since the client (node 1) and the servers (node 2, node 3) by design handle different types of messages and functions, the consumption is split up into the three nodes that are part of the network. The three bars on the left are thereby the result of the histogram analysis, while the three bars on the right are a combination of the static analysis with the messages handled by the corresponding node. Since all following analyses feature a higher number of events, the cumulative current consumption of whole benchmark runs is divided by 1000, transforming mAms into mAs.

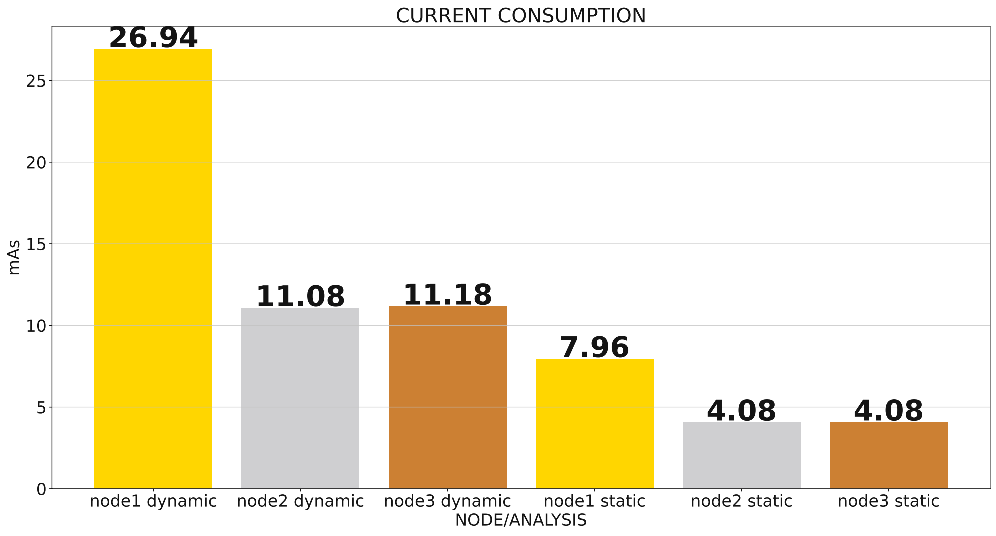

Figure 6.26: Nordic 1D1C dynamic vs. static analysis comparison

While the relation of the bars stays the same in regard to their height in both modes of analysis, the results of the dynamic analysis are on average 2.95 times as high as the results of the static analysis. While the lower resolution measurement setup of the dynamic analysis introduces a certain error into the overall current consumption, the expectation is that this type of error manifests as a lower measured value. This discrepancy in the results is actually caused by the influence of the noisier power supply. Overlaying a ripple not only broadens the RX-idle consumption, but the current consumption of the chip overall.
Applying the same process to the TI benchmark results in Figure 6.27.

Again the client node consumes more power than the two server nodes, but due to a power supply that is less noisy, the results of both modes of analysis lie much closer together, while the dynamic analysis is 2.95 times the static analysis on the NRF52840 DK, it is only 13% more when analyzing the CC1352r1 LaunchXL. Nodes with a lower number of messages thereby show the expected lower measured current consumption, while a higher number of messages is still influenced by the noise level which is much smaller than the one observed on the Nordic board, but larger than the one in the static analysis.
The statistics files produced by the benchmark also provide the possibility to compare the routing engines in regard to their routing performance in a hardware independent manner, which is typically used in simulation. Hereby, the collected number of messages and their types are translated into a number of metrics. In particular, the following data is generated:

- Number of messages received
  In order to keep the metric independent of the used routing engine, there is no

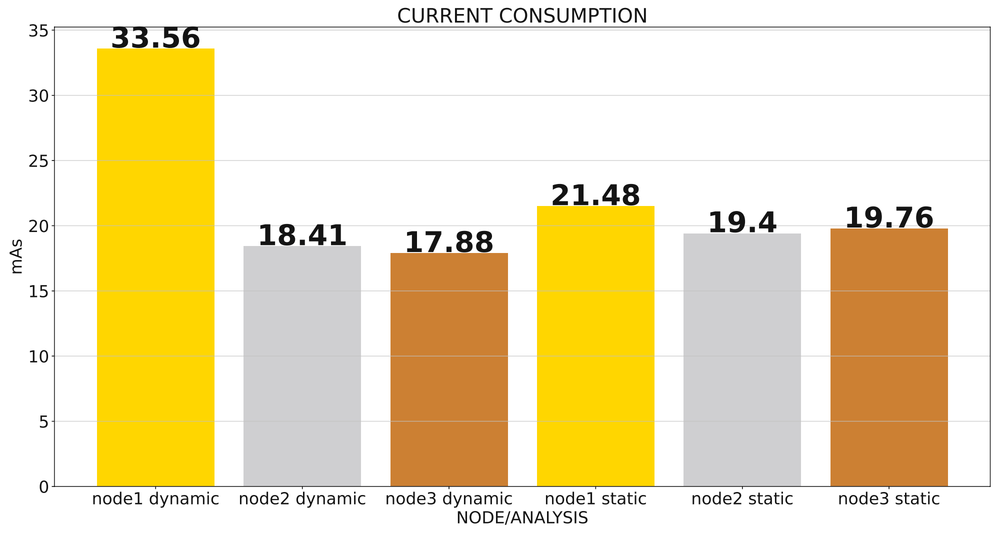

Figure 6.27: CC1352r1 1D1C dynamic vs. static analysis comparison

differentiation made between data and control messages when applying it to MPL. This choice is made in order to be able to directly compare the two used main routing engines of this thesis.

- Number of messages sent
  As with the number of received messages, this metric does not differentiate between the message type.

- Number of redundant messages received
  A received message is counted as redundant if its contents do not improve the information set of the particular node. This could be the reception of a data message that was already seen or in case of MPL, or the reception of a control message that indicates a consistent domain.

- Number of needed messages
  A needed message is counted as the polar opposite to the redundant metric, i.e. the received message improves the information set of the node. Here the influence of more advanced types of messages will show up. For routing engines that only use data type messages like flooding or MPL with only data message retransmission, this metric will directly present the completion rate of the network. When 60 messages are sent by the client, two 60 message bars of the servers will indicate a 100% completion rate, as only these 60 messages really benefit the application running on the node.
  Once more advanced features like control message retransmissions are introduced, the information contained in these bars is increased. If an inconsistent domain is detected, the information set of the overall network and the particular node is improved, as a fault in the previous routing steps is detected. Any number of

messages above zero for the client node, or above the number of messages sent by the client, therefore shows the positive effect of an additional message type.

Applying these metrics to the NRF benchmark statistics file yields the result shown in Figure 6.28.

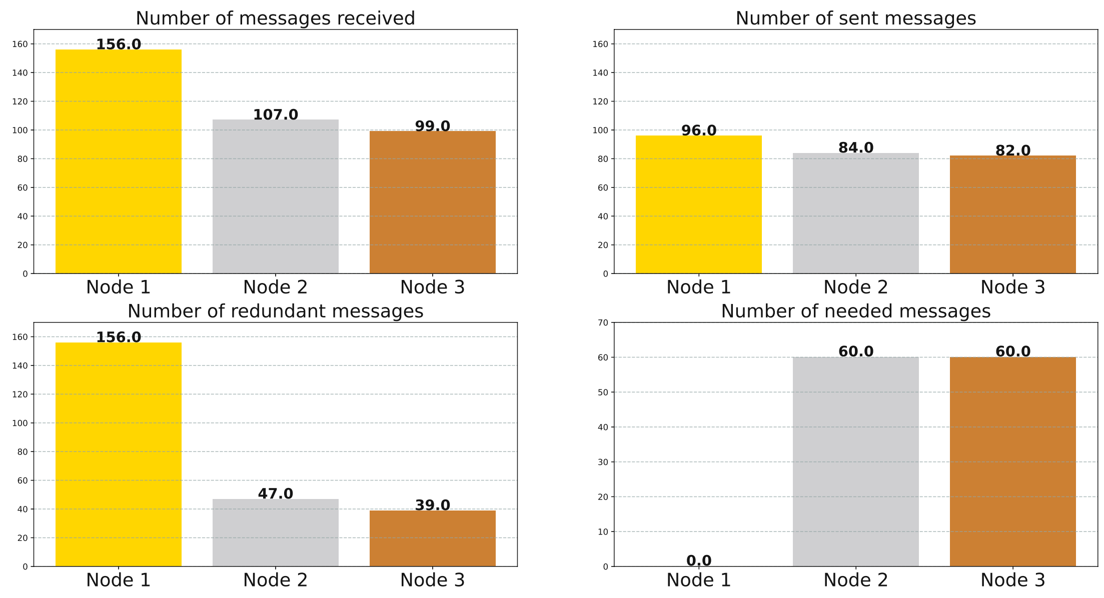

Figure 6.28: NRF52840 network performance with 1 data and 1 ctrl retransmission

In addition to these metrics in bar form, the evaluation also produces the percentage of completion, i.e. how many percent of the messages sent by the client were received by the servers and the average percentage of redundant messages from all messages that were received. The two numbers are thereby an average over all nodes, generating one defining number for a singular benchmark run.

- Completion: 100%
- Percentage redundant messages: 66.85%

Looking at these results yields the following conclusions: The routing engine is able to transmit all sent packages to all listening servers in the network. In accordance to chapter 2, the average message reception rate of the nodes is 75.16%. This number is thereby calculated by looking at the number of messages sent and received by the three nodes. By accepting 66.85% redundant messages (data and control), the routing engine is able to achieve a 100% completion rate. When looking at the number of needed messages, it can be seen that during the benchmark every control message was

determined as consistent, the overall performance is therefore achieved by the use of data retransmissions alone.
It can also be seen in the two top metrics that the ordering of the bars corresponds with the current consumption presented by the dynamic and static analysis in Figure 6.26. The most active node in both cases is the client, as it must at least send the 60 direct messages in addition to the routing messages, while the server nodes feature a reduced load due to the MPL routing engine and its used Trickle timers.
The four presented metrics lend themselves well to analyze the load and the consumption of the three individual nodes during one or multiple runs of the benchmark. In order to compare the routing engines against each other, three individual runs are combined into a single statistic and the individual node measurements are combined into a singular per network/routing engine metric evaluation. In order to counter changing network interferences in the real-world setup, a range test is run before every benchmark and the distance between the nodes is adjusted in order to guarantee a reception rate in the 60-80% range. The results of this analysis are presented in the following chapters.

6.2.1 Triangle Topology

For the Nordic NRF5840 DK board, the network wide analysis yields the results shown in Figure 6.29. Some metrics of the single run benchmark analysis shown before are directly used again in the more global view, while some new ones are generated and added.

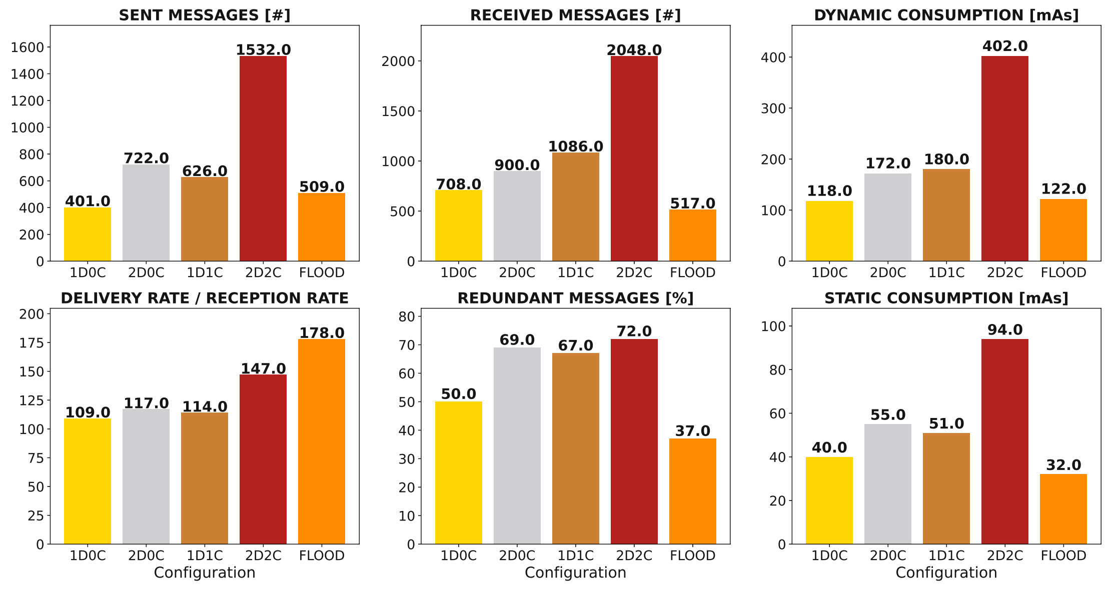

Figure 6.29: Nordic triangle benchmark routing engine comparison (+4dBm)

The number of sent/received messages as well as the dynamic/static current consumption are already used in the more fine-grained per-node analysis. The current consumption is hereby the sum of the three individual node consumption data. Newly added are the percentage of redundant messages and the Delivery/Reception rate factor (D/R-Factor).

- D/R-Factor
  As presented by Hemant Ghayvat et al. [GMGS15], the reception rate of a node is influenced by a number of external effects. While the distance and the location of the nodes is kept constant during the benchmark runs in this study, additional effects like humidity, the number of people in the test area and the influence of other radio devices is more volatile. Due to this fact, some runs of the benchmark feature a poorer overall message reception rate than others. If a routing engine is confronted with a higher reception rate, it is easier to achieve a high delivery rate as well. In order to combat this influence, the factor of the two is used as a metric. If a routing engine achieves a high delivery rate, although the message reception rate is poor, it will score higher. Certain other information can also be extracted from this factor: If the number is higher than 100, the delivery rate beats the reception rate of the network, a greater percentage of actual messages are successfully exchanged than overall packets are lost due to interference.

- Percentage of redundant messages
  If a routing engine uses more messages during its lifetime, it will per design introduce an overhead onto the otherwise low number of actual data messages that need to be transmitted. A routing engine like 2D2C MPL is practically built around redundancy to function, as evident in the high number of redundant messages in Figure 6.29. But if these increased number of messages is used to achieve a higher delivery ratio, the so-called needed messages in the per-node analysis will also increase. Since these messages actually increase the information in the system, they need to be introduced as a positive effect in the analysis. In order to do this, the percentage of redundant messages is used. While the number of sent messages more than doubles from 1D1C MPL to 2D2C, the percentage stays practically the same, as a higher number of needed messages is transmitted, further improving the delivery rate to reception rate factor.

Using these metrics, the following conclusions can be drawn for the triangle network topology: In terms of performance over current consumption, flood based routing is the clear winner in the triangle topology. It features the highest delivery factor with the lowest energy consumption in the dynamic and the second lowest in the static current consumption. 1D0C, 2D0C and 1D1C perform very similarly in the delivery factor, indicating that the additional number of messages or the introduction of a new message type is not really improving the network performance. Combining this metric with the percentage of redundant messages or the current consumption, the best choice for MPL is 1D0C routing in the case of this study. Only the introduction of, theoretically double the messages, with 2D2C routing manages to improve the network performance above the rest of the MPL level. This improvement though is not cheap in terms of energy

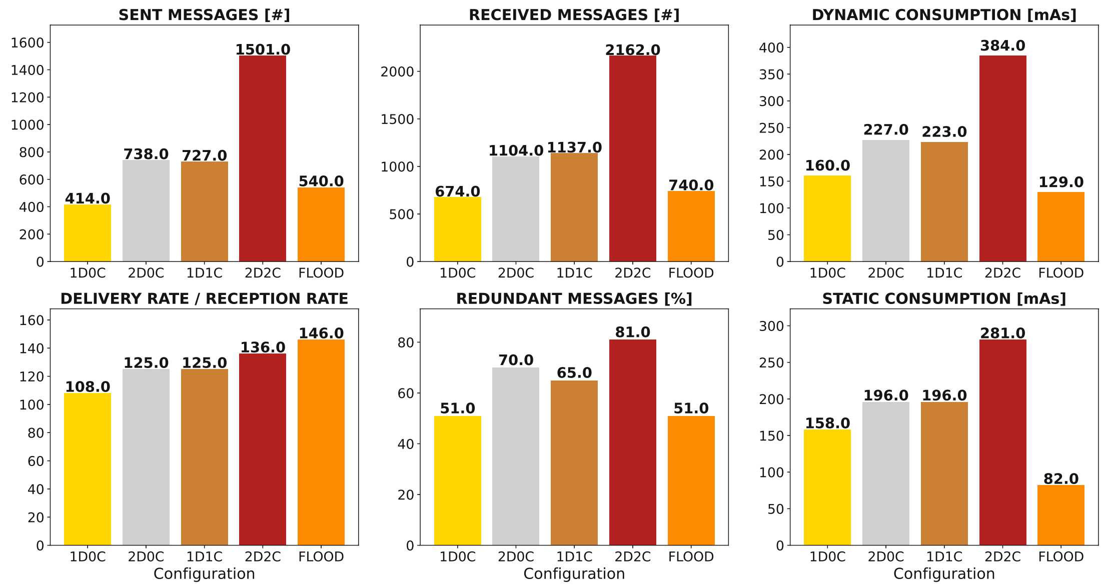

Figure 6.30: TI triangle benchmark routing engine comparison (+4dBm)

consumption, as more than double the energy is consumed for an 25% improvement of the delivery factor. Overall flooding offers a very high D/R-Factor with the lowest redundancy and current consumption making it a very good choice for an energy efficient routing engine.

Running the same benchmark setup on the TI LaunchXL equipped with the CC1352r1 chipset, yields the result shown in Figure 6.30.

It can be seen from the results that the overall behavior of both devices stays the same in regard to the hardware independent metrics discussed before. Slight differences in the bars are hereby caused by the external influences and could be mitigated more by increasing the number of benchmark runs. Important for this thesis though is that the network performances match up with the performance already discussed with the Nordic board. Due to this, the static analysis can be applied to any benchmark run and produce a valid comparison between the boards.
The comparison between the dynamic and the static analysis for both boards confirms the single benchmark comparison shown in section 6.2.
In the dynamic analysis of the Nordic board, it can be seen that flooding requires more current than 1D0C while this is not the case in the static analysis, as it is the case with 2D0C and 1D1C. Errors like these are to be expected in the dynamic analysis if more noise is introduced. Increasing the number or the length of the benchmarks in this case benefits the overall measurement system. Combining more of the same benchmarks into one, increases both the consumption in phase 1 and 3 but, due to the board spending most of its time in the RX-idle phase, it will more clearly increase the phase 2 consumption.

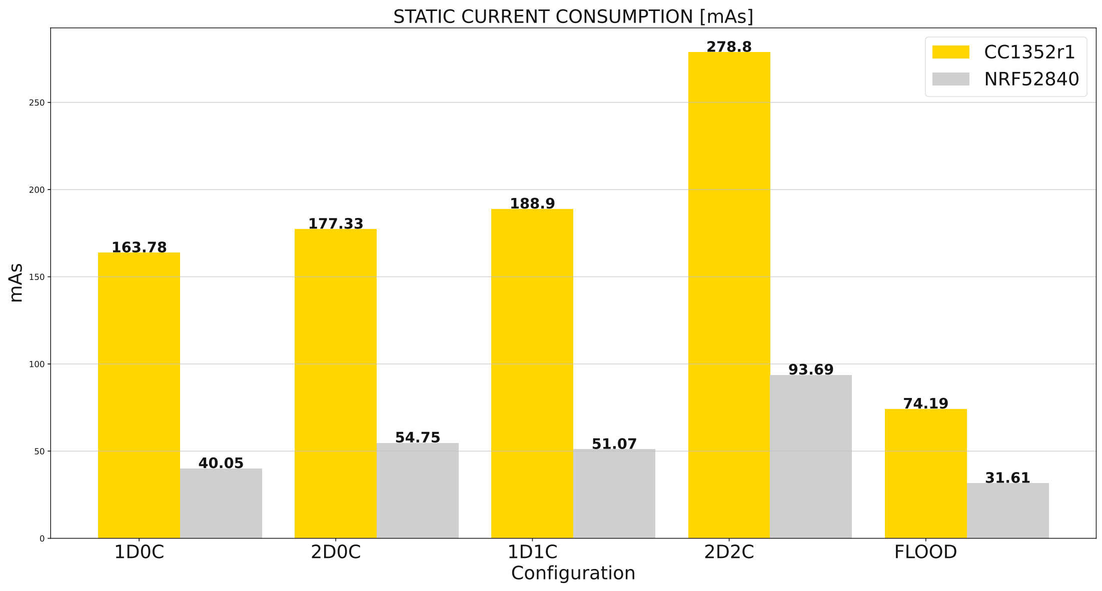

Figure 6.31: Static current consumption triangle benchmark at 4dBm transmit power

Increasing the spread between the two phase types, increases the Signal to Noise ratio (SNR) and makes the distinction between phases easier. Another possibility is to change the behavior of the benchmark overall, instead of triggering messages for the entire duration, the controller can issue messages for one half of the benchmark and keep the UUT dormant for the other half. This approach will increase the actual baseband region but keep the consumption caused by the routing itself at a constant level, which again aids the SNR
A less noisy board, like the CC1352r1, does not require this special handling, as evident in Figure 6.30. Since the goal of power supply design should be a stable output, it is therefore more advantageous to use the static analysis for routing engine evaluation. It offers a far more accurate representation of the actual message transmission, interlocks with the simulation based approach and does not require the handling of two systems in the field, the UUT and the measurement node.
For further analysis in this thesis, the current consumption is therefore always taken as the static consumption for both boards. Applying the static results to the benchmark run in Figure 6.29 yields the result shown in Figure 6.31.

Due to the higher number of messages, certain differences arise in comparison to the small static simulation in section 6.1. 1D1C consumes more power than 2D0C in the case of the TI board, but not in the NRF52840 DK. This presents the first benchmark indication that energy efficiency between two boards is not always related to hardware independent metrics. In the other cases, the ordering of the bars is the same as in the small-scale simulation.
Lowering the transmission power to 0 dBm yields the result shown in Figure 6.32.

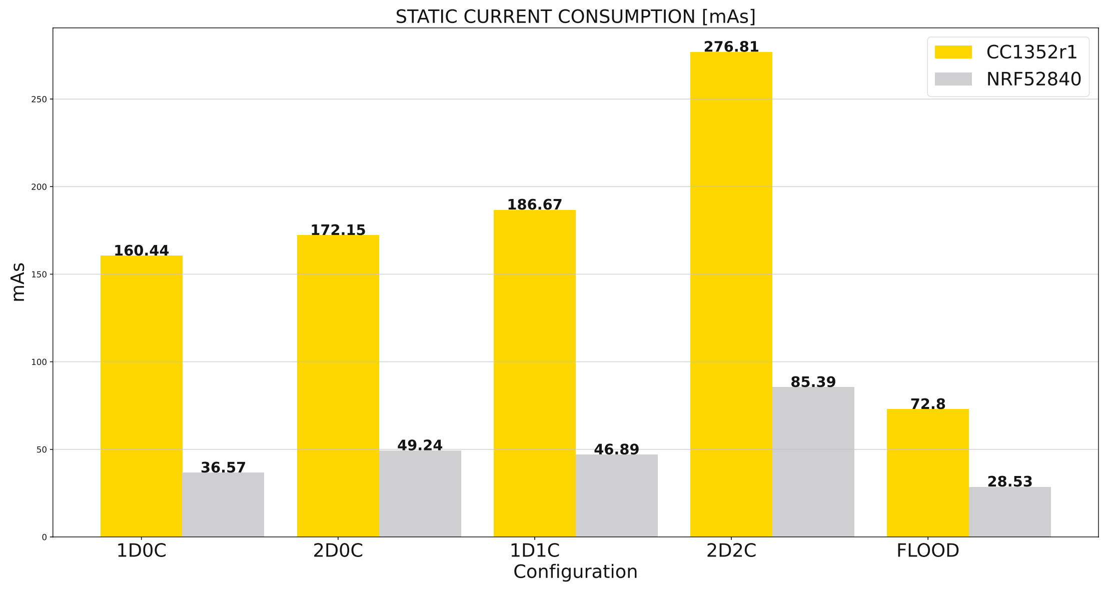

Figure 6.32: Static current consumption triangle benchmark at 0dBm transmit power

From both these analyses, it can be seen that the consumption of the TI board, when confronted with the same benchmark, only drops 1.75% on average. The difference is hereby much larger for configurations with a lower number of messages. For 1D0C the gain of the reduction is a 2% consumption reduction, while for 2D2C it is only 0.7%. The reason for this is the software dominance of the CC1352r1. Increasing the number of messages highlights constant consumption contributors in particular. While the current consumption during the TX phase does drop as well in the case of the CC1352r1, their influence is miniscule in comparison to all the summed up constant consumption phases like the disable or the software phase. The transmission dominated NRF52840 on the other hand, consumes on average 9.1% less current. The drop is thereby constant across all routing configurations, no matter their number of messages. This highlights the influence of constant phases onto the energy efficiency. While the NRF52840 also features a constant software/OS phase, their duration is very short and its influence is mitigated by the TX phase.
For a system developer, the knowledge of system behavior, as given by the static analysis, is therefore very important when designing an energy efficient system. Simply lowering the transmission power will not always decrease the current consumption by the same amount. In addition to this, a reduction in transmission power brings with it a lower device range, which in turn can decrease the reception rate between nodes. The triangle topology hereby even presents a best-case for mesh-routing, as multiple delivery routes are available. A less ideal case is presented by the line topology.

6.2.2 Line Topology

If the nodes are arranged in the line topology presented in chapter 2, the result of the benchmark is changed to the one shown in Figure 6.33. As discussed in the previous chapter, the dynamic analysis is hereby replaced by the static analysis of both boards, combining the hardware information with the hardware independent metrics into one chart.

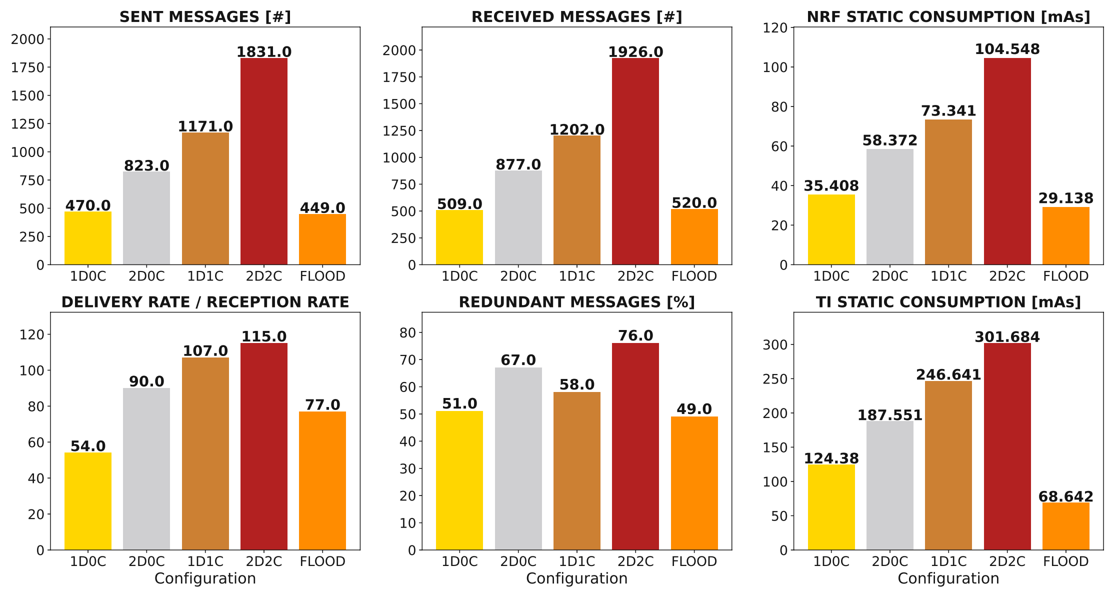

Figure 6.33: Line benchmark routing engine comparison (+4dBm)

Since the line topology presents a difficult case for lossy networks, the D/R-Factor is lower for all routing engines when comparing them to the triangle benchmark. Certain engines, though, dropped more than others. 1D0C and flood routing practically half, while 1D1C and 2D2C show a 15% difference. The reason for this difference is the introduction of the additional control message type in MPL. In the triangle setup, data messages alone, with only little help from the control messages, were able to deliver the messages to all nodes in the network. In the line topology, these redundant paths are missing, so although the reception rate between the nodes is kept on the same levels as in the triangle case, there is a lot more effort needed to maintain this singular route. The introduction of the control message type aims at exactly this repair functionality. The D/R-Factor from 2D0C and 1D1C increases by 19%, while it stays the same in the triangle benchmark, but this positive effect on the one hand introduces an 31% increase in current consumption on the other.
An interesting behavior is also shown by the D/R-Factor of 1D0C and flooding. While it is also the case in the triangle benchmark that flooding performed better than 1D0C, the line setup presents a certain shortcoming of MPL when directly analyzing the raw input of the benchmark.

Flooding, in practice, is a control-less routing engine except for the broadcast storm logic. This means, that if a message has not been seen, it is forwarded immediately. The network performance practically falls with the reception rate between the nodes. MPL in comparison, is based on the Trickle algorithm, messages are not transmitted immediately. Instead, the Trickle timer is started and the message is only retransmitted if this timer elapses. If the node, during the timer run time, receives the same message again due to a retransmission of another node, the message currently waiting for the timer elapse will never be transmitted if the redundancy factor is set to 1. E.g. together with the direct message transmission, the client will start its own retransmission Trickle timer. If this timer elapses before the timer running on the second node, the second node will never retransmit the message, therefore starving the 3rd node.
According to the MPL specification, the configuration needs to be the same across the network. In the triangle benchmark, all D/R-Factors are greater than 100%, meaning that every routing engine is able to outperform the reception rate of the network with a reasonable redundancy rate. If the MPL redundancy constant is now increased across the network, a much higher number of messages are retransmitted. A K equal to 2 in the triangle benchmark would mean that every message of the MPL configuration is transmitted, and no Trickle cancellations take place, which has a great impact on the number of messages sent.
A developer therefore needs to know the layout of his network and find the MPL settings that are needed for his network, in order to meet the desired D/R-Factor. This process would need a lot of effort in the dynamic analysis, requiring a high number of benchmark runs with a high number of possible configurations. Here the simulation of the network comes into play. In the virtual environment, any number of nodes can be placed, and their performance can be evaluated, even with simulated interference sources. Missing from this though is the actual current consumption of the hardware being used. A fact that can be changed via the presented methodologies in this thesis, which are summarized in the following chapter.

CHAPTER 7

Discussion

7.1 How is energy-consumed by the hardware?

The first research question that needs to be answered is how energy is consumed by a specific hardware during the routing operation. It is shown in section 6.1, that two chipsets running the same software/OS combination behave very differently when analyzing the current consumption over time. The current consumption during the actual radio transmission is thereby dictated by the amplifier and antenna setup of the chip. CPUs like the NRF52840 feature a very powerful radio stage, while the one built into the CC1352r1 features a lower output power. Most hardware enables the developer to reduce or increase the transmission power of the radio. In general, a reduction of the transmission power will reduce the current consumption of the radio during the transmission stage, i.e. when the message is transmitted. It is important to note that this change in transmission power is limited by the hardware itself, for the Nordic board it can be chosen between +8 dBm and -20 dBm, on the TI board the maximum power is given as +5 dBm in the datasheet, but the minimum is never given. Looking at the static analysis of the board, it is determined that the minimum transmission power with the minimum current is -16 dBm. Lowering further starts to increase the current consumption as the hardware appears to fall back into the known range.
In addition to this current consumption, devices need processing time in order to handle external inputs, assemble messages, execute the routing engine and handle radio mode switchovers. Most of these processes can be configured by the user directly in software and knowledge of the consumption pattern can be used to tune the energy consumption of the whole system. In addition to these controllable influences, that are part of the observable software/OS, both boards show inherent differences in behavior when entering their own proprietary software stacks or in hardware itself. The TI board, in regard to energy conservation, suffers from a long disable phase that shows up every time an event involves higher OS levels or the application itself. Other message types like the reception

of a control message or an already known data message do not feature this pattern. While this makes the transmission and the reception of a new data message rather expensive for MPL, the flood based routing engine achieves a higher energy efficiency, as this phase is used to retransmit a new message. Interruption of this disable phase by the other hardware near processes like radio setup, transmission and mode switch does not prolong the duration. The actual gain of this behavior can be seen in Figure 7.1. In fact, the bar chart in Figure 7.1, even shows that the length of the disable phase and therefore the current consumption is cut short by the additional hardware interaction, further advancing the advantage of flooding on this hardware.

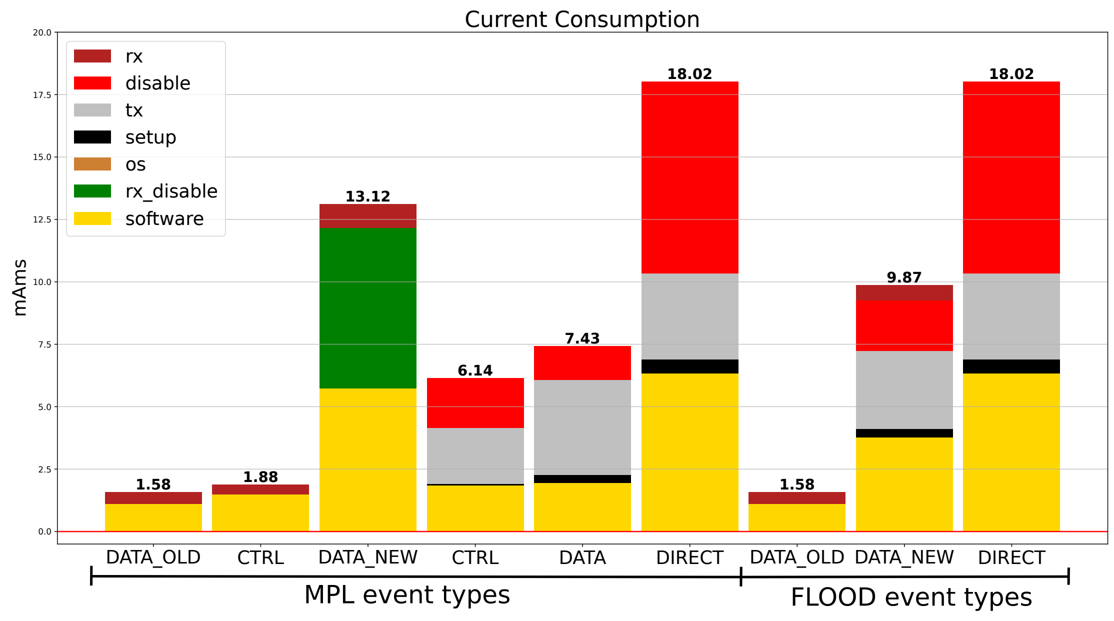

Figure 7.1: Comparison between CC1352r1 routing events

While this disable phase is present at all times on the CC1352r1, independent of the external trigger event of the message and the behavior of the software, the NRF52840 shows a different behavior. After the reception of a message there is no disable phase. Even when the message is handled by the upper layers of the OS, the consumption drops back to RX-idle directly after all computation has finished. The behavior of the direct message transmission on the other hand is different. When the message, as in the benchmark case, is triggered via the UART, no disable phase is observed. If the message is triggered by an external interrupt (button press on the board), the NRF52840 features a very dominant disable phase with an increase from RX-idle of about 0.7 mA and a length ranging from 160-200 ms. Due to the length of this phase, the current consumption of the chipset is dominated by this phase, even when transmitting with the maximum transmission power. When looking at Figure 7.2, a complete direct message with +4 dBm transmission power consumes 15.61 mAms, the disable phase when using buttons as events will consume between 112 and 140 mAms, an overall current consumption increase

of nearly 10x.
It is behavior such as this, that is very difficult to catch or to evaluate during a simulation. Even if the simulation engine features an energy model or some sort of emulation, it would need inherent knowledge of how the software, OS, proprietary software stack and the hardware interact when confronted with different events.

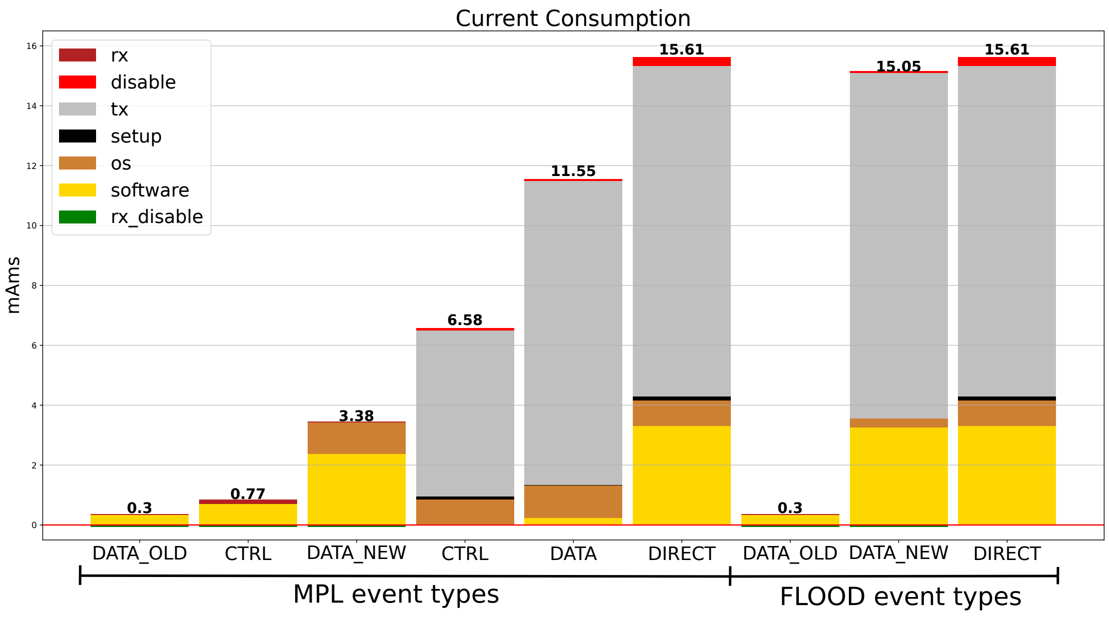

Figure 7.2: Comparison between NRF52840 routing events

When comparing the other phases of Figure 7.1 and Figure 7.2, certain other differences appear. The CC1352r1 has a clock speed of 48 MHz, the NRF52840 on the other hand features a 64 MHz clock. Since more time is needed to compute the same instructions, more current is consumed longer. In regard to the efficiency of a certain routing engine, this introduces a fixed difference between the chipsets. While the transmission power can be tweaked in order to consume less, or even consume a "negative" amount, i.e. less current is consumed in comparison to RX-idle, the software phase always introduces a positive consumption.
If the developer has insight into the behavior of the chipset, as it is given by the static analysis, current consumption of the routing and message exchange itself can be optimized in order to consume the least amount of energy during operation. How many mAms are consumed by a certain message type, and how the change of one of these variables benefits the system, is dependent on the consumption pattern type of the device. The CC1352r1 and the NRF52840 hereby present two opposites.
During message transmission, the faster NRF52840 has a short software/os phase in combination with a powerful radio stage, making it a transmission dominated chipset. Changes in transmission power, or the number of bytes that need to be transmitted, heavily influence the current consumption of a message transmission. At +4 dBm and 20

bytes payload, the NRF52840 consumes 15.61 mAms. Reducing the transmission power alone to -20 dBm decreases this value by more than 100% to -0.43 mAms, the chipset even saves energy by transmitting in comparison to staying in RX-idle.
The TI CC1352r1 on the other hand consumes most of its current (77% in case of +4 dBm at 20 bytes payload) during computation and in the disable phase. This board is therefore characterized as software dominant. Since all consumption values are summed up by the analysis, a reduction in transmission power should still reduce the overall current consumption of the system, but in practice, the minute changes introduced by the power reduction are easily negated by a slight change in the disable phase length. At +4 dBm with 20 bytes payload, a current consumption of 18.02 mAms is observed, at the minimum it is 9.91 mAms, a difference of 47.3%. As evident in Figure 7.3 though, the consumption of the disable phase varies a lot as its length is not constant. Due to this, the current in the -12 dBm measurement decreases 21.8% but at the higher -8 dBm it decreases by 27% from the +4 dBm case.

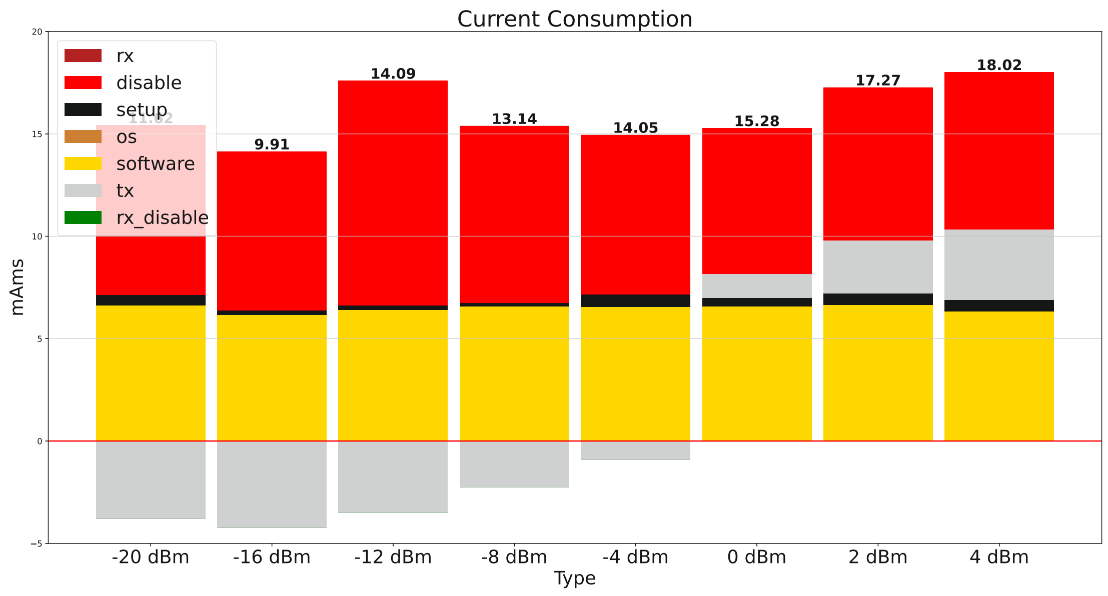

Figure 7.3: Comparison between CC1352r1 direct message events at varying transmission powers

In order to evaluate the hardware in the case of the CC1352r1, it is therefore necessary to repeat the static analysis of the message types featuring a disable phase a number of times. For further analysis, the average of these results is then used in order to achieve an accurate estimation.
Depending on the length of the data that needs to be transmitted, i.e. all bytes added by the routing engine, IEEE 802.15.4 and 6LoWPAN together with the payload, a board might switch between being software and transmission dominated. Instead of evaluating this by a change in payload size and subsequent measurements, the static analysis enables

the developer to find an estimation via calculation.
This calculation is relevant since in mesh use cases, radio duty cycling, i.e. shutting the radio off completely, is not always feasible for all nodes. Such protocols would require nodes to know when they have to listen and transmit or repeat messages in such a way that the messages fall into at least 1 listening interval of the neighbors. In the case presented in this thesis, the RX stage is not switched off and the energy that is left in the system is gradually decreasing with the RX-idle current consumption of the node. Studies on other routing protocols, as for example by Muh Ahyar et al.[AYES12] on AODV variants, talk about a negative effect of additional routing overhead. The results of this thesis suggest that an increased overhead of the right type of messages in combination with an optimized transmission power to payload length might even profit the overall network lifetime.
Looking at a small 2 node network of NRF52840 nodes deploying 1D0C MPL, a single message transmission will come with 1 direct message, 1 new data message receive, 1 data message retransmissions and 1 old message receive event. At 20 bytes payload and -20 dBm, this sums up to a current consumption of 0.06 mAms. Choosing a routing protocol with a higher overhead like 2D0C consumes -2.8288 mAms, even though more messages are transmitted, the overall network has consumed less energy than what it would have just idling. Of course, a reduction in transmission power brings with it a reduction in range, too. If lowering the transmission power below 0 dBm is feasible for the present network arrangement, it is beneficial to the network lifetime that its nodes are transmission dominated. If this cannot be accomplished, additional nodes cannot be placed in order to increase the range and the transmission power is above or equal 0 dBm it is beneficial to the network lifetime for its nodes to be software dominated.
At +4 dBm the NRF52840 consumes, during direct message transmission, 11.05 mAms during the TX phase and 4.56 mAms in the other stages. For both values to break even, the consumption of the transmission stage would need to be reduced by 58.7%. Together with the observed 2.4 ms transmission time, with 0.03 ms per byte at 250 kbps, the overall 80 bytes would need to be reduced to 46. Since this means a reduction of 34 bytes, longer than our actual payload, the number of bytes needed by the routing and addressing itself would need to be reduced as well.
When reducing the transmission power to -16 dBm on the CC1352r1, the overall network would profit from a transmission dominated hardware behavior. A direct message, in this configuration, consumes -4.234 mAms during transmission and 14.14 mAms for all other phases. In this case, the number of bytes would need to be increased in order to consume at least -14.14 mAms in the TX phase. Using the same calculation as before, with 0.03 ms per byte, this would mean a minimum payload length of 268 bytes. As with the Nordic board, this value is not practically achievable. While adding more bytes to a payload is easy, the underlying networking layers will need to fragment the overall message into multiple message transmissions, which would in turn increase the current consumption.

7.2 Are hardware-independent metrics enough?

When evaluating the energy efficiency of a routing engine in simulation, a number of different metrics are used. A study on multi-hop routing energy efficiency, takes a look at maximizing the minimum node lifespan and minimizing the overall energy consumption of the system [EV05]. Another study by Banur Doko et al. also regards the network lifetime [BJBJ19] and an AODV-centered study directly compares the routing overhead of a protocol to the energy efficiency [AYES12]. At their core, hardware independent metrics often correlate or directly depend on the number of messages that are sent over the network. The study by Muh Ahyar et al.[AYES12] directly lists the occurrence of routing overhead as a negative behavior in regard to energy efficiency. While the energy models used in the other two studies inherently feature the number of messages as a positive energy model, i.e. more messages, more current consumed.
In the last section, it is shown that in fact, on real hardware this behavior might not be true, or may vary between different routing engines and their message types. Due to factors like the disable phase, flooding offers a combination of 3 message types that is cheaper than the ones used by MPL while exchanging the same amount of information. The right combination between payload length and transmission power is even able to reverse the behavior altogether, presenting energy savings when a higher number of messages is transmitted. Using the results of the dynamic analysis, Figure 7.4 shows the correlation between the number of sent messages and the current consumption at varying transmission powers and 20 bytes payload for the CC1352r1. While the results in the dynamic analysis included the RX-idle consumption in order to highlight the function of the dynamic measurement setup, the analysis in this case subtracts this consumption in order to highlight the correlation effect between the metrics and directly show possible energy savings via a negative number.

In Figure 7.4, the orange trendline denotes the maximum transmission power of +4 dBm, while the blue line denotes the transmission power with the lowest current consumption (-16 dBm). Since the CC1352r1 is software dominant at 20 bytes payload, it can be seen in the graph that both the minimum and the maximum as well as all transmission powers between them, correlate to the number of messages sent. Blue thereby already shows a flattening effect, indicating that this correlation drops with the lower transmission power. In addition to this flattening effect, the scatter plot shows 3 flyers over all power settings. These flyers denote the flood based routing engine. As discussed in the previous section, the 3 flood event types, in combination, consume less energy than the 4 MPL event types. This behavior is highlighted here in this scatterplot where it shows that these benchmark runs in fact, even with more message exchange between the nodes, do not correlate to the MPL trend for the CC1352r1.
In comparison to the results for the CC1352r1, Figure 7.5 shows the correlation between sent messages and the current consumption for the NRF52840.

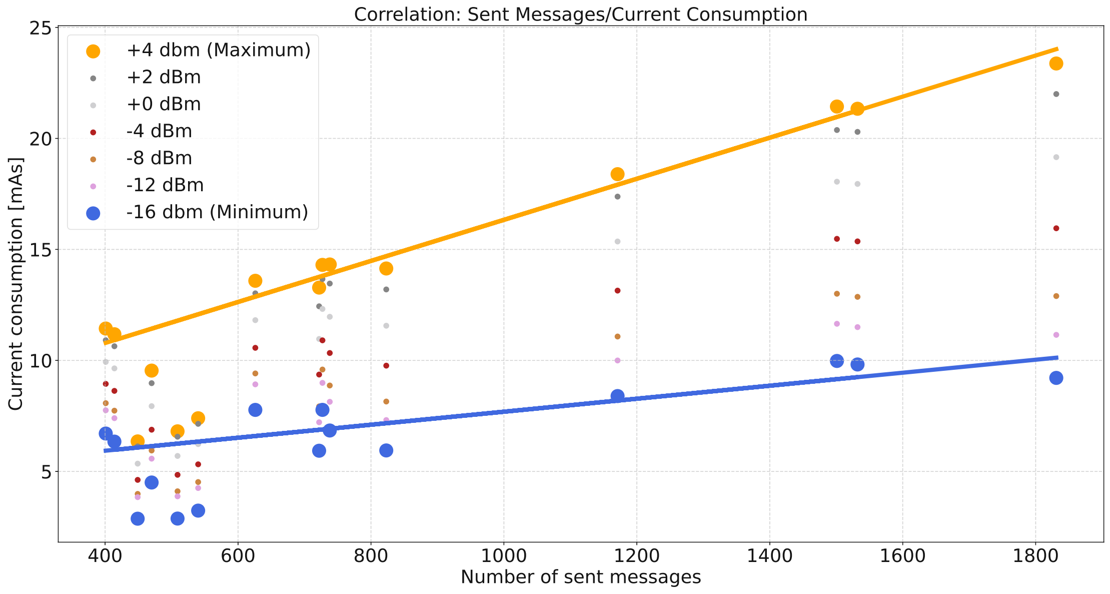

Figure 7.4: Correlation between number of sent messages and the current consumption, CC1352r1

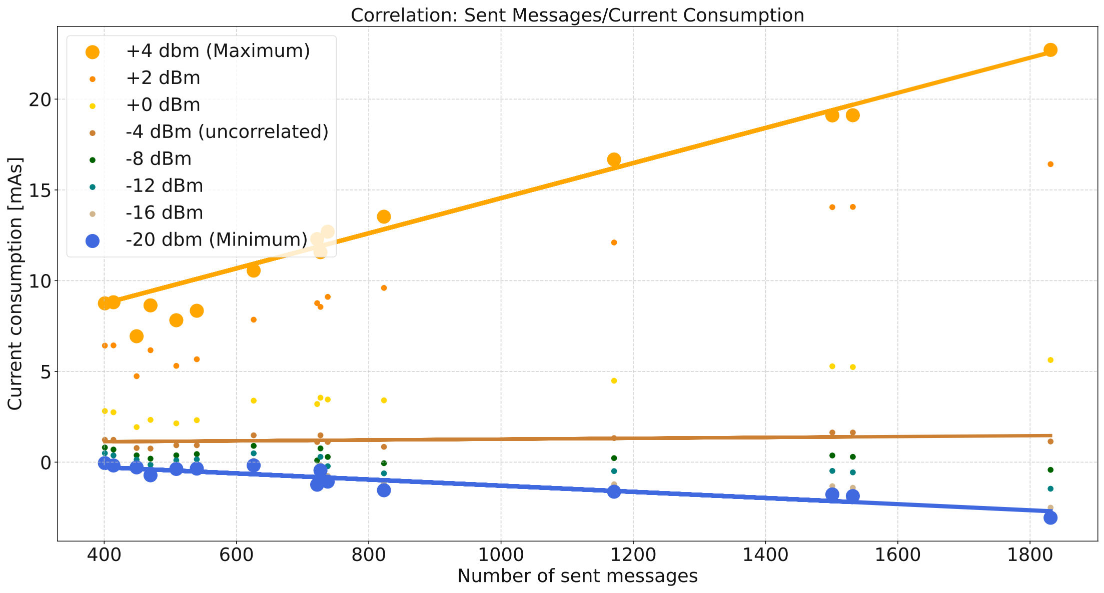

Figure 7.5: Correlation between number of sent messages and the current consumption, NRF52840

The scatterplot in Figure 7.5 hereby highlights 3 main trendlines that show that hardware-independent metrics in fact are not enough to determine the energy efficiency of the two used routing engines. Shown in orange is again the maximum transmission power, which,

as expected for a transmission dominated board, correlates well with the number of sent messages. Again you can see 3 flyers denoting the flood based routing engines. While flood events are much closer to the MPL consumption in the NRF52840, the engine still eliminates the need of 1 event type, which is the reason for this uncorrelated behavior.
Lowering the transmission power until -4 dBm presents the first change in behavior. At this transmission power, the current consumption of the average exchange, per direct message transmitted in the network, cancels out. Meaning that the current consumption of the routing engines is completely unrelated to the number of sent messages, as an increase in the number of sent messages does in fact not increase the current consumption, as shown in Figure 7.5.
Lower than -4 dBm, going down to the minimum transmission power, shows another type of correlation. A negative correlation means that an increase in routing overhead even improves the node lifetime. I.e. in comparison to an idle board, less current is consumed. Even though the number of messages stays the same for all these data streams, the transmission setting of the hardware is able to display 3 types of correlation. Above -4 dBm, sent messages and current consumption correlate. Below they correlate as well, but an increase in sent messages is now a more energy efficient system consuming less current than when the board is idle. At exactly -4 dBm the two metrics do not correlate at all. Correlation between hardware-independent metrics and the actual energy efficiency of the system has therefore to be seen as non-constant and is dependent on the hardware, the combination of payload length with hardware consumption type and on the events required by the routing engine. At the same payload, for example, flooding and MPL are by default offset in the favor of flooding, as the three required event types consume less current than the ones used in MPL.

7.3 How to introduce hardware behavior into considerations?

This thesis presents two types of analysis that can be used to introduce the actual hardware current consumption and behavior into the energy efficiency considerations.
The first one is the static analysis, hereby the current consumption of the used nodes is captured in a laboratory setup. As a result, the current consumption of the routing events present in the network is captured and the individual stages and their influences are known. The results of this, can then be used in a variety of ways.

- Hardware classification
  Based on the consumption type of the device and the required transmission power, the developer can try to adjust the current consumption by increasing or decreasing the payload and the number of messages in the system. Transmission powers lower than 0 dBm will benefit from longer payloads and a higher number of messages, ones greater or equal than 0 from a reduction in size. The effects that these changes have on the overall energy consumption are varying, dependent on the hardware

consumption type. If the transmission phase consumes more than the rest of the phases, the CPU is classified as transmission dominated. Meaning, that it benefits a lot from payload/transmission power changes. If the comparison goes the other way, the board is software dominated at the given payload length, changes in the factors will not benefit the system in the same amount as in the other type.

- Event cost estimation
  Certain events, e.g. the reception of a new data message, can be expensive in comparison to other events used by a routing engine. The CC1352r1 for example has an expensive new message reception event. Even before running a simulation/a dynamic analysis, this offers a first overview over which types of events need to be minimized in order to build an energy efficient system. If for example the control message consumes less energy than the data message retransmission, a routing configuration favoring control over data can offer a higher energy efficiency.
  Differences in routing engines themselves also show up, combining the 3 routing event types with the event types needed by MPL for a single message between two nodes, shows that a rearrangement of certain internal routing events can lower the current consumption. Combining the message reception and the retransmission into one step in the case of flooding, offers a consumption advantage over splitting it into two separate routing events like it is done in MPL.

- Payload length tuning
  If the consumption type of the hardware in combination with the transmission power appears as the wrong type for the target application, the results of the static analysis can be used to, in theory, change the type. If the transmission speed and the number of bytes is known, as in the case presented in the last section, a developer can use the results of the static analysis to calculate the break even value. Dependent on the behavior of the system and the number of bytes added upon the payload due to the operating system/network/routing engine, a developer can then fine tune the behavior of the device to suite his needs.

While these analyses can be carried out without a network, the main goal of the evaluation is to determine the behavior in a real-world use case. To do this in a system, at execution time, the dynamic analysis is presented. Here, the measurement setup is distributed together with the nodes into two benchmark topologies. Through this process, the nodes experience real-world interferences and challenges. The measurement node thereby directly provides the current consumption of the individual nodes together with the influence of the node's power supply, or additional CPU behavior missing from the static analysis. The effect of these influences can then be calculated by applying the static analysis to the routing events encoded in the produced statistics file.
When looking at the statistics file alone, it can be generated by either the dynamic analysis, capturing the radio interference's present at the node location, or via a simulation. Running these numbers through the payload length tuning process of the static analysis will show how the hardware-independent metrics will correlate with the hardware behavior.

The developer can then tune the behavior of his system in accordance with the network requirements.

7.4 What is energy efficiency in mesh-multicast routing?

The results of the previous sections show that an energy efficient system needs to combine the behavior of the routing engine with the behavior of the hardware.
When evaluating the routing engines in a hardware independent way, the dynamic analysis in this thesis offers two metrics: The D/R-Factor and the percentage of redundant messages. While energy efficiency is an important factor in guaranteeing the longevity of the system, the application implemented by the network needs to perform as well. Performance in this case is measured by these two metrics, i.e. how good a routing engine is in communicating successfully in the adverse reception conditions of the dynamic analysis. A high D/R-Factor is desirable for the system. A value greater than 100 thereby denotes that a higher number of actual payload messages is delivered in the system than the overall reception rate would suggest.
Since topologies in real-world applications must not be homogeneous, like it is the case in the triangle topology, it is also important to measure the network performance in the line topology which presents especially bad conditions for message delivery in a lossy environment. The need for analyzing both topologies is thereby best shown when looking at the flooding approach, while MPL routing engines, once control messages are introduced, perform comparably in both conditions, with only slight losses due to the adverse nature of the line, the performance of flooding is cut in half.
The percentage of redundant messages offers a normalizing effect onto the results of the D/R-Factor. A high delivery rate can of course be reached by introducing a high number of duplicates or additional messages in the system. A good performing system is thereby a combination of a high D/R-Factor with an in comparison low redundancy, as more of the used messages actually benefit the system. Here additional message types, like the control messages in MPL come into play, while 2D0C and 1D1C present nearly identical numbers for the D/R-Factor, the redundancy of the 1D1C is lower. The positive effect of the network repair functionality is especially visible in the results of the line benchmark, while a high number of additional messages is sent by 1D1C, the redundancy is low since these messages actually benefit the system. While the pure data retransmission approach not only misses this functionality, but additionally suffers from a starvation effect in this case.
As presented in the previous section, though, this additional redundancy might not be a disadvantage in regard to energy efficiency. A system with a negative correlation between the number of sent messages and the current consumption can profit from a higher number of messages. The search for an energy efficient system therefore needs to also include the behavior of the hardware into the considerations.
Overall, the following definition for an energy-efficient mesh-multicast routing engine is proposed: "An energy efficient routing system has a high delivery/reception factor with an in comparison low percentage of redundancy running and is running on hardware that

balances the current consumption of the occurring event types with the payload length and the transmission power used."
While the actual selection of a suitable routing engine remains to be a discussion of trade-offs, the given definition gives a system that performs well under adverse network condition and includes the measured hardware behavior as presented in this thesis. Depending on the target network, the developer needs to include the triangle, the line or both benchmarks into his considerations. Just looking at the triangle benchmark, the flood based routing offers a very high D/R-Factor with a comparably low redundancy and current consumption. In the line, the winner is the 1D1C routing with a high D/R-Factor and a low redundancy. Both of these recommendations are of course only valid for the used +4 dBm transmission power and 20 bytes payload, values for which both boards display a positive correlation between the number of sent messages and the current consumption.
If a lower transmission power of -4 dBm is applicable for the system, the winner in the case of the CC1352r1 LaunchXL node is still 1D1C or flooding, while for the NRF52840, 2D2C with its higher redundancy but nearly identical current consumption now provides a much better candidate when looking at both benchmarks, as evident in Figure 7.6 for the triangle benchmark.

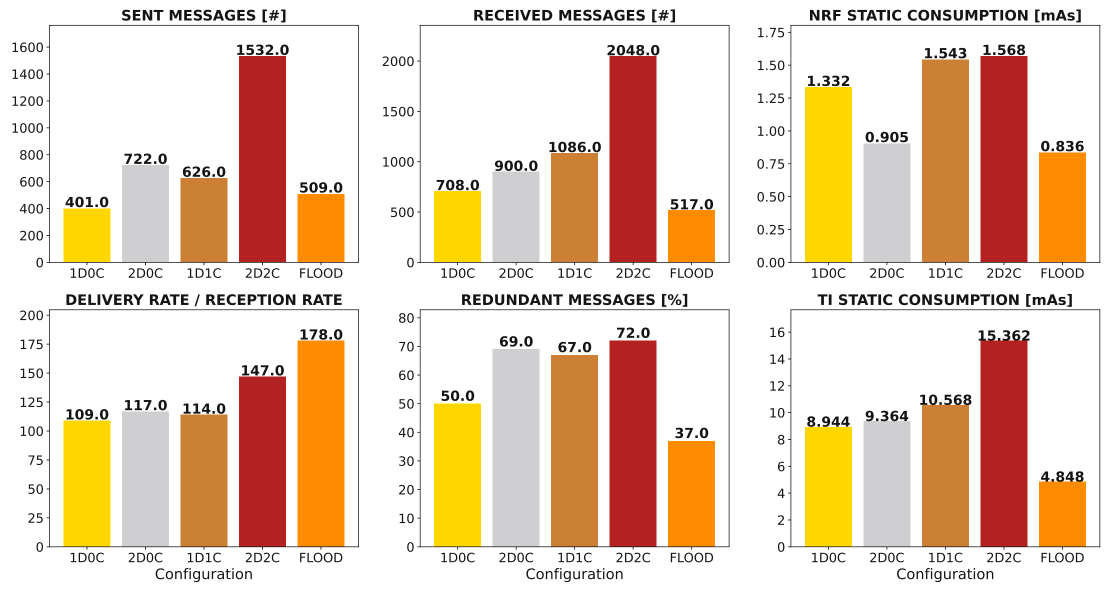

Figure 7.6: Metrics comparison at -4 dBm transmission power and 20 bytes payload

CHAPTER 8

Conclusion

The processes presented in this thesis enable developers to determine the energy efficiency of a mesh routing system running on real hardware. Two measurement setups, forming two different analyses approaches are discussed in detail. The static analysis features a laboratory setup and produces high accuracy, high resolution measurements, enabling the identification of current consumption contributors. The dynamic analysis on the other hand features a distributed measurement setup and nodes are deployed in two benchmark topologies. The result of this analysis is a lower resolution current consumption measurement that includes the behavior of the complete deployed node, including its power supply or additional periphery. Together with this data stream, the number of sent, received or dropped messages and their types are captured in order to form a hardware-independent statistic.
Applying the results of the static analysis on the statistics generated by benchmark runs shows that hardware-independent metrics are not always enough to determine if a routing engine is energy efficient. Depending on the consumption type of the hardware, the payload length and the transmission power of the radio, the same routing engine can correlate with the number of sent messages positive and negative. At certain configurations, it is even possible to create a system, that does not correlate with any hardware-independent metrics.
The following definition is therefore proposed for energy efficiency in the mesh-multicast context:
"An energy efficient routing system has a high delivery/reception factor with an in comparison low percentage of redundancy and is running on hardware that balances the current consumption of the occurring event types with the payload length and the transmission power used."

List of Figures

List of Tables

Acronyms

6LoWPAN	IPv6 over Low power Wireless Personal Area Network. 1, 22, 29, 86
ADC	Analog Digital Converter. 7, 25, 26, 41
AODV	Ad-hoc On-demand Distance Vector. 16, 87, 88
BMRF	Bidirectional Multicast RPL Forwarding. 15
CSMA	Carrier Sense Multiple Access. 52, 56
CSV	Comma-separated values. 42, 44
D/R-Factor	Delivery/Reception rate factor. 77, 78, 81, 82, 92, 93
DOA	Destination Advertisement Object. 15
DODAG	Destination Oriented Directed Acyclic Graph. 15, 16
ESMRF	Enhanced SMRF. 15
HBA	Home and Building Automation. 10
I2C	Inter-Integrated Circuit. 26, 27, 41
IMAODV	Improved MAODV. 16
IoT	Internet of Things. vii, ix, 1, 3–5, 15, 16, 22, 23, 72
JSON	Java Script Object Notation. 37, 39
MAODV	Multicast Ad-hoc On-demand Distance Vector. 16
MPL	Multicast Protocol for Low-Power and Lossy Networks. 16–22, 29–36, 51–53, 59, 60, 63, 65, 66, 74, 77, 81, 82, 84, 87, 88, 90–92

OS Operating System. 2, 5, 19, 22, 23, 29, 30, 33, 38, 49, 50, 52, 53, 55–57, 59–61, 63, 65, 80, 83–85

OSI Open Systems Interconnection. 1

PEM Power Measurement System. 7, 8

PHY Physical Layer. 5, 24–26

RECOUP Reliable Group Communication Protocol. 15

REMI Reliable and Secure Multicast Routing Protocol for IoT. 15

RF Radio Frequency. 5, 7, 23, 25–27, 39

RPL Routing Protocol for Low power and Lossy Networks. 15, 16

SEM Self Energy Meter. 6, 7

SMRF Stateless Multicast Forwarding with RPL. 15

SNR Signal to Noise ratio. 79

SPI Serial Peripheral Interface. 28, 41

TCP Transmission Control Protocol. 16

UART Universal Asynchronous Receiver-Transmitter. 9, 10, 37–39, 43, 50, 84

UDP User Datagram Protocol. 10, 16, 30, 31, 34, 37–39, 43, 50

UUT Unit under Test. 5–10, 27, 43, 79

WSN Wireless Sensor Networks. vii, ix, 1, 2, 5, 10, 15, 16, 22, 23

Bibliography

[AEkEB13] Sherine M Abd El-kader and Basma M Mohammad El-Basioni. Precision farming solution in Egypt using the wireless sensor network technology. *Egyptian Informatics Journal*, 14(3):221–233, 2013.

[AS20] Amal Antony and Sarika S. A review on iot operating systems. *International Journal of Computer Applications*, 176:33–40, 05 2020.

[AYES12] Muh. Ahyar, Mohammad Yani, Elfisa, and Riri Fitri Sari. Comparison of Energy Efficiency and Routing Packet Overhead in Single and Multi Path Routing Protocols over S-MAC for Wireless Sensor Network. In *2012 Sixth UKSim/AMSS European Symposium on Computer Modeling and Simulation*, pages 406–411, 2012.

[BJBJ19] Đoko Banđur, Branimir Jakšić, Miloš Banđur, and Srđan Jović. An analysis of energy efficiency in Wireless Sensor Networks (WSNs) applied in smart agriculture. *Computers and Electronics in Agriculture*, 156:500–507, 2019.

[CGR11] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[EV05] Sinem Coleri Ergen and Pravin Varaiya. On multi-hop routing for energy efficiency. *IEEE Communications Letters*, 9(10):880–881, 2005.

[FBV$^+$13] Carlos Fernández, Diego Bouvier, Jorge Villaverde, Leonardo Steinfeld, and Julián Oreggioni. Low-power self-energy meter for wireless sensor network. In *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, pages 315–317. IEEE, 2013.

[GMGS15] Hemant Ghayvat, Subhas Mukhopadhyay, Xiang Gui, and Nagender Suryadevara. WSN-and IOT-based smart homes and their extension to smart buildings. *sensors*, 15(5):10350–10379, 2015.

[JASK18] Farhana Javed, Muhamamd Khalil Afzal, Muhammad Sharif, and Byung-Seo Kim. Internet of Things (IoT) Operating Systems Support, Networking

Technologies, Applications, and Challenges: A Comparative Review. *IEEE Communications Surveys Tutorials*, 20(3):2062–2100, 2018.

[KV15] Manmeet Kaur and Amandeep Kaur Virk. An improved multicast AODV routing protocol for VANETs. *International Journal of Computer Applications*, 975:8887, 2015.

[LCVdPDS20] Guus Leenders, Gilles Callebaut, Liesbet Van der Perre, and Lieven De Strycker. An Experimental Evaluation of Energy Trade-Offs in Narrowband IoT. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–5. IEEE, 2020.

[LLC$^+$17] Guillermo Gastón Lorente, Bart Lemmens, Matthias Carlier, An Braeken, and Kris Steenhaut. BMRF: Bidirectional multicast RPL forwarding. *Ad Hoc Networks*, 54:69–84, 2017.

[LNJY18] Meryem Lamkimel, Najib Naja, Abdellah Jamali, and Aymen Yahyaoui. The Internet of Things: Overview of the essential elements and the new enabling technology 6LoWPAN. In *2018 IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD)*, pages 142–147. IEEE, 2018.

[MKH$^+$07] Gabriel Montenegro, Nandakishore Kushalnagar, Jonathan Hui, David Culler, et al. Transmission of IPv6 packets over IEEE 802154 networks. *Internet proposed standard RFC*, 4944:130, 2007.

[Nora] Nordic Semiconductors. NRF52840 DK. `https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dk`. [Online; accessed December 05, 2021].

[Norb] Nordic Semiconductors. NRF52840 Dongle. `https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dongle`. [Online; accessed December 05, 2021].

[OP12] George Oikonomou and Iain Phillips. Stateless multicast forwarding with RPL in 6LowPAN sensor networks. In *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 272–277. IEEE, 2012.

[PBLS14] Albert Pötsch, Achim Berger, Christian Leitner, and Andreas Springer. A power measurement system for accurate energy profiling of embedded wireless systems. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–4. IEEE, 2014.

[RK18] Philipp Raich and Wolfgang Kastner. The need for efficient multicast routing in low-power IPv6 mesh networks. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–4. IEEE, 2018.

[Tex] Texas Instruments. CC1352r1 LaunchXL. `https://www.ti.com/tool/LAUNCHXL-CC1352R1`. [Online; accessed December 05, 2021].

[Wol20] Wolfgang Ewald. INA226 module. `https://wolles-elektronikkiste.de/ina226`, 2020. [Online; accessed December 05, 2021].