

Testumgebung und Performancevergleich von Zigbee, Thread und Bluetooth Mesh Netzwerken

Bachelor Thesis

FACHBERICHT - ANKLIN, BOBST, HORATH
27. August 2020

Experte: Jürg M. Stettbacher

Fachcoach: Matthias Meier
Manuel Di Cerbo

Autoren: Raffael Anklin
Robin Bobst
Cyrill Horath

Studiengang: Elektro- und Informationstechnik

Semester: Frühlingssemester 2020

Abstract

Among the most popular low power mesh network protocols in free GHz ISM band the three mesh stacks Bluetooth Mesh, ZigBee and Thread are currently competing against each other. The assignment of this bachelor thesis was to build a consistent test framework for all three mesh-networks to benchmark them under realistic conditions. Due to better combability, the nRF52840 SoC from Nordic Semiconductors was the chosen microcontroller for all three network stacks. The benchmark is structured in two parts, a battery powered slave node and a master which is directly connected to a computer. The master node is responsible for controlling the measurement, whereas the slave nodes send benchmark messages to each other. These benchmark messages collect the necessary information to determine latency, RSSI, throughput and active radio time. For a better comparability an apartment house, an apartment and a labor environment were selected as different test benches. The Thread stack results the best in the different test benches. Because of its automatic routing it is able to adapt himself to the environment, as a result the latency of this stack is in every three benches similarly low. Bluetooth Mesh was able to reach the lowest latency with small payload. The ZigBee network stands out with its constant and low latency within one test bench. As a conclusion all of the three networks perform well in case of a home automation. Due to of their own assets and drawbacks it cannot be said this is the best mesh-stack. It depends on the application which mesh network performs the best.

Inhaltsverzeichnis

1 Einleitung	1
2 Übersicht	2
2.1 Ausgangslage und Aufgabenstellung	2
2.2 Vorarbeiten P5	2
2.3 Ziel der Arbeit	2
2.4 Abgrenzung	2
I Point to Point Testinfrastruktur	4
3 Messkonzept	5
3.1 Definition MAC-Layer	5
3.2 Anforderungen	5
3.3 Messgrößen	5
3.4 Konzept	6
3.5 Testszenarien	6
3.6 Messaufbau	7
4 Inbetriebnahme und Bedienung	8
5 Soft- und Firmware	12
5.1 Soft- und Firmware	12
5.2 Low Level Radio Driver	13
5.3 Broadcasting Collisions Probability	16
5.4 Zeitsynchronisation	16
5.5 Webserver	18
6 Schluss	24
6.1 Validierung	24
6.2 Verifizierung	24
6.3 Fazit	24
II Mesh Benchmark - Konzept und Umsetzung	25

7 Benchmark Konzept	26
7.1 Konzeptschema	26
7.2 Testumgebungen und Messaufbau	28
7.3 Ablauf Messvorgang	30
7.4 Vergleichswerte und Messgrößen	31
7.5 Messreihe	32
7.6 Allgemeine Benchmark Parameter	33
7.7 Traffic Generation	34
7.8 Messdatenerfassung und Auswertung	34
7.9 Messerwartung	36
8 Hardware Plattform	37
8.1 System on Chip (SoC)	37
8.2 Hardware Development Kits	37
8.3 Aufbau Testnode	37
8.4 Antenne und Strahlungscharakteristik	39
9 Soft- und Firmware	41
9.1 Konzept	41
9.2 Umsetzung	42
9.3 Shared Library	42
9.4 Benchmark Management	49
III Bluetooth Mesh	51
10 Einleitung und Grundlagen	52
11 Technische Grundlagen Bluetooth Mesh	55
11.1 Netzaufbau und Topologie	55
11.2 Protokoll Stack	55
11.3 Sicherheit	56
11.4 Bluetooth Mesh Software Development Kit	57
12 Umsetzung Benchmark	58
12.1 nRF Connect SDK	58
12.2 Benchmark und Stack Parameter	59
12.3 nRF SDK for Mesh	59

13 Analyse Bluetooth Mesh Stack	61
IV Thread	62
14 Einleitung	63
15 Technische Grundlagen Thread	65
15.1 Application Layer	65
15.2 Thread Protokoll Stack	66
15.3 Link und Physical Layer	66
15.4 Netzaufbau und Topologie	67
15.5 Sicherheit	71
15.6 Thread Software Development Kit	71
16 Umsetzung Benchmark	72
16.1 Openthread Stack Konfiguration	72
16.2 Openthread Stack Initialisierung	73
16.3 CoAP Benchmark Message	74
16.4 IPv6 Adressierung	75
17 Analyse Thread Stack	76
17.1 Vor- und Nachteile Openthread Stack	76
17.2 FreeRTOS Projekt	76
V Zigbee	77
18 Einleitung	78
19 Technische Grundlagen Zigbee	79
19.1 Netzaufbau und Topologie	79
19.2 Zigbee Protokoll Stack	80
20 Umsetzung Benchmark	84
20.1 Zigbee Software Development Kit	84
20.2 Stack Implementation	85
20.3 Mesh Node Firmware	87

21 Analyse Zigbee Stack	90
21.1 Stärken und Schwächen des Protokollstacks	90
21.2 Schwierigkeiten bei der Umsetzung	90
VI Mesh Benchmark - Resultate und Vergleich	92
22 Messresultate	93
22.1 Resultate	93
22.2 Validierung	96
22.3 Verifizierung	97
23 Vergleich Mesh Netzwerke	99
23.1 Vergleich der Messreihen	99
23.2 Vergleich Testumgebungen	103
23.3 Fazit	105
24 Schluss	110
24.1 Zielerreichung	110
24.2 Fazit	113
24.3 Ausblick	114
24.4 Schlusswort	114
Literatur	115
Abbildungsverzeichnis	117
Anhang	120
A Aufgabenstellung	120
B Pflichtenheft	125
C Bericht emv Messung Development Kits	144
D Messprotokolle Mesh Benchmark	157
E Paper	187
F Random Traffic Generation	195
G Bluetooth Mesh Stack Layers	196

1 Einleitung

Das *Internet der Dinge* und *Industrie 4.0* sind in der Technikszene die Schlagwörter der Stunde. Insbesondere im Heimbereich oder in der Industrie wird die intelligente Vernetzung von Sensoren und Akten immer wichtiger bei der Entwicklung neuer Produkte. Eine Beleuchtung, die nicht via Handy bedienbar ist, kann in der heutigen Zeit kaum mehr verkauft werden. Alles muss *Smart* und miteinander vernetzt sein und wenn möglich soll es keinen Zusatzaufwand und keine Zusatzkosten verursachen. Systeme mit geringem Installationsaufwand und geringen Hardwarekosten sind also bei den Konsumentinnen und Konsumenten sehr gefragt.

Die kabellose Vernetzung zu sogenannten *Wireless Sensor Networks*, kurz *WSN*, scheint hier ideal, um die Kommunikation gewährleisten zu können. Denn es sind keine bzw. nur sehr einfache Installationen nötig, da auf eine Verkabelung teilweise oder sogar ganz verzichtet werden kann. Durch den Einsatz von Systemen, die ein sogenanntes *Mesh Netzwerk* bilden, können auch grosse Gebiete auf einfachste Art und Weise abgedeckt werden. Systeme für solche Vernetzungen gibt es auf dem Markt einige, doch viele davon sind proprietär und daher oft nicht kompatibel mit Fremdsystemen. Die drei Protokolle *Bluetooth Mesh*, *Thread* sowie *Zigbee*, die die Schwerpunkte dieser Bachelor Thesis bilden, sind diesbezüglich anders. Sie wurden von der jeweiligen Organisation spezifiziert, standardisiert und veröffentlicht und können daher herstellerübergreifend eingesetzt werden.

Alle drei erwähnten Mesh Protokolle besitzen Stärken und Schwächen im Bezug auf deren Leistungsfähigkeit und sind teilweise einfacher oder eben aufwändiger in der Handhabung. Diese Stärken und Schwächen werden in der vorliegenden Arbeit aufgespürt und offengelegt. Dazu wird ein qualitativer Vergleich unter realen Anwendungsbedingungen vorgenommen. Ein Benchmark wird insbesondere zeigen, welches der drei Protokolle die geringste Latenzzeit und den geringsten Paketverlust aufweist. Als eigenständiger Teil dieser Arbeit werden zudem die beiden Protokollstandards der MAC Ebenen von *Bluetooth Mesh* mit *Bluetooth Low Energy* sowie von *Thread* und *ZigBee* mit dem *IEEE 802.15.4* Standard untersucht.

Der vorliegende Bericht ist in sechs Teile gegliedert. Nach einer kurzen Übersicht zum Thema und den Zielen dieser Bachelor Thesis wird im Teil I die Point to Point Testinfrastruktur auf MAC Ebene behandelt. Anschliessend wird im Teil II das Konzept und die Umsetzung des Mesh Benchmarks beschrieben. In den Teilen III, IV und V, welche jeweils individuell von Raffael Anklin (III), Robin Bobst (IV) und Cyrill Horath (V) verfasst wurden, wird auf die drei Protokollstacks *Bluetooth Mesh*, *Thread* und *ZigBee* detailliert eingegangen. Mit dem abschliessenden Teil VI werden sämtliche Resultate des Benchmarks aufgezeigt und die drei Mesh Protokolle verglichen.

2 Übersicht

2.1 Ausgangslage und Aufgabenstellung

Im 2.4GHz ISM-Band konkurrieren sich derzeit die drei weit verbreiteten Low Power Mesh Netzwerk Protokolle Bluetooth Mesh, Thread und Zigbee. Alle drei wurden für die kabellose Übertragung in sogenannten WSN (Wireless Sensor Networks) und für Smart Home Anwendungen konzipiert. Während Thread und Zigbee den *IEEE 802.15.4* Standard als Physical Layer benutzen, basiert der Bluetooth Mesh Stack auf dem *Bluetooth Low Energy (BLE)* Standard. Aufgrund der hohen Dichte an Protokollen und Systemen, die das 2.4GHz ISM-Band ebenso benutzen (z.B. Wifi), sind die Störeinflüsse auf die drei Mesh Protokolle nicht zu unterschätzen. Die Protokollstacks begegnen diesem und weiteren Problemen auf unterschiedliche Weise. Diese Unterschiede werden in dieser Arbeit aufgezeigt indem die Performance der Mesh Netzwerke unter unterschiedlichen Bedingungen getestet werden. So soll ein qualitativer Vergleich der drei Mesh Protokolle gemacht werden. Eine subjektive Bewertung der Mesh Stacks bezüglich deren Handhabung und Komplexität wird zudem vorgenommen. Die detaillierte Ausgangslage und Aufgabenstellung kann einerseits der Aufgabenstellung im Anhang A und andererseits dem Pflichtenheft im Anhang B entnommen werden.

2.2 Vorarbeiten P5

Im Rahmen des Projekt 5 mit dem Titel *Bluetooth-Mesh Plattform für IoT Anwendungen* wurde das Bluetooth Mesh Protokoll bereits vertieft betrachtet und dessen Vor- und Nachteile aufgezeigt. Basierend auf diesen Erkenntnissen und Erfahrungen sowie der oben beschriebenen Thematik wird in dieser Bachelor Thesis das Bluetooth-Mesh Protokoll mit den Alternativen Thread sowie Zigbee verglichen. Die Ergebnisse des P5 haben gezeigt, dass sogenannte Flooding Mesh Netze, wie zum Beispiel Bluetooth Mesh, einige Nachteile gegenüber gerouteten Mesh Netzen aufweisen. Ob und wie diese ins Gewicht fallen und ob Bluetooth Mesh auch Vorteile gegenüber Thread und Zigbee hat, konnte im Projekt 5 nicht gezeigt werden und ist nun ein wichtiger Bestandteil dieser Thesis.

2.3 Ziel der Arbeit

Hauptziel dieser Thesis ist ein objektiver Vergleich der drei gängigsten Low Power Mesh Netzwerke Bluetooth Mesh, Thread und Zigbee bezüglich deren Leistungsfähigkeit unter wechselnden Bedingungen. Es soll erkennbar werden, welches Protokoll in welchen Bereichen seine Stärken hat und wie es am besten eingesetzt werden kann. Weitere subjektive Aspekte wie beispielsweise die Komplexität des Stacks oder die Interoperabilität mit anderen Systemen sollen den Vergleich noch aussagekräftiger machen und die Praxistauglichkeit der Systeme aufzeigen.

Ein weiteres eigenständiges Ziel dieser Thesis ist, ein Instrument zu schaffen, um Signalmessungen auf den erwähnten Low Level Radio Treiber Ebenen *IEEE 802.15.4* und *BLE* durchführen zu können. Damit sollen beispielsweise geeignete Standorte für die Mesh Knoten ermittelt und ausserdem gezielte Störungen in die Mesh Benchmarks eingebracht werden können. Die ausführliche Zielesetzung dieser Arbeit ist im Pflichtenheft im Anhang B aufgeführt.

2.4 Abgrenzung

Die vorliegende Thesis beschränkt sich auf die in den nachfolgenden Abschnitten beschriebenen Testumgebungen und Testkriterien. Weitere Parameter können im Rahmen dieser Bachelor Thesis nicht betrachtet werden. Daher müssen folgende Abgrenzungen beachtet werden:

Der Performance Vergleich der Mesh Netzwerke soll unter möglichst realen Bedingungen durchgeführt werden. Dies schliesst in gewisser Hinsicht auch sogenannte Stresstests, bei welchen die Mesh Stacks an ihre Leistungsgrenzen gebracht werden sollen, ein. Diese sollen allerdings in der vorliegenden Arbeit nicht im Zentrum stehen.

Weiter sollen die Mesh Benchmarks nachvollziehbar und reproduzierbar sein. Da je nach Umsetzung und Implementation des jeweiligen Mesh Stacks jedoch grosse Unterschiede in der Performance entstehen können, sind die Resultate nur im Bezug auf unsere Umsetzung zu interpretieren. Messungen mit divergierenden Stack Implementationen können von den Resultaten, die in dieser Arbeit präsentiert werden, abweichen.

Einen Vergleich der drei Mesh wird nur auf dem SoC nRF52840 durchgeführt, wobei zusätzlich zu den Messungen die aktive Funkzeit aufgezeichnet wird. Diese Messung soll nur als Richtwert dienen, ein Vergleich der Leistungsaufnahme der verschiedenen Stacks wird nicht im Detail ausgeführt. Zudem wird auf eine Gegenüberstellung mit verschiedenen SoCs verzichtet. Weiter werden finanzielle Vergleiche gänzlich weggelassen.

Teil I

Point to Point Testinfrastruktur

3 Messkonzept

In den nachfolgenden Abschnitten wird auf die Konzeptionierung und Umsetzung der Point to Point (P2P) Testinfrastruktur eingegangen. Ziel dieser Infrastruktur ist, die Verbindungen auf den MAC-Layern *BLE* sowie *IEEE 802.15.4* auszumessen. Die Mac-Layer Verbindung beschreibt eine rudimentäre Kommunikation zwischen zwei Knoten, welche direkt auf der physikalischen Schicht stattfindet und keine Netzwerk- und Applikationsschichten daran beteiligt sind. Dazu wird zuerst das Messkonzept präsentiert und anschliessend die Inbetriebnahme und Bedienung der Messinfrastruktur erläutert. Schliesslich werden noch die technischen Einzelheiten hinsichtlich der Umsetzung dargelegt.

3.1 Definition MAC-Layer

Der MAC-Layer ist bei Protokollen für die drahtlose Kommunikation oftmals über Normen spezifiziert. Die eingesetzte Hardware-Plattform (siehe Abschnitt 8) unterstützt *BLE* sowie den *IEEE 802.15.4* Standard. Diese beiden Standards werden im Weiteren als Radio-Mode bezeichnet. Nebst dem Radio-Mode charakterisiert sich die MAC-Schicht über den verwendeten Kanal. Dabei handelt es sich um das Frequenzband, auf welchem gesendet wird.

3.2 Anforderungen

Die Anforderung an die Point-to-Point Testinfrastruktur ist, eine Verbindung zwischen zwei Funkmodulen anhand der folgenden Messwerte charakterisieren zu können:

- ***Signal to Noise Ratio (SNR)***
- ***Packetloss***

Das Ausmessen der obigen Eigenschaften soll pro Kanal möglich sein. Die folgenden Parameter müssen vom Benutzer eingestellt werden können, um eine Messung durchzuführen:

- ***Radio-Mode*** BLE-1Mbit oder IEEE 802.15.4
- ***Start-Channel***: Der Start-Channel definiert den Kanal, auf welchem die Messung gestartet werden soll. Dieser ist abhängig vom Radio-Mode.
- ***Stop-Channel***: Der Stop-Channel definiert den Kanal, bis zu welchem die Messung durchgeführt werden soll. Dieser ist ebenfalls abhängig vom Radio-Mode.
- ***Tx-Power***: Sendeleistung beim Versand der Testnachrichten.
- ***Collision-Avoidance (CA)***: Kollisionsvermeidung (nur bei IEEE 802.15.4 Radio Mode)
- ***Payload***: Anzahl der zu sendenden Bytes pro Paket.

3.3 Messgrößen

Wie bereits in Abschnitt 3.2 erwähnt, wird das Signal-Rausch-Verhältnis (SNR) und der Paketverlust gemessen. Die beiden Messgrößen können mit Hilfe der nRF-Plattform direkt bestimmt werden.

- ***SNR***: Gibt das Verhältnis zwischen der Signalleistung und Rauschleistung an. Als Rauschen werden jegliche Umwelteinflüsse (Störungen) bezeichnet. Zur Bestimmung des SNR muss die Rauschleistung und Signalleistung erfasst werden. Bei der Detektion der Rauschleistung muss der Signalbringer (Sender) deaktiviert sein. Das Messen der Signalleistung inkl. Rauschleistung erfolgt hingegen bei eingeschaltetem Sender.

- **Paketverlust:** Als Paketverlust wird das Verhältnis zwischen der Anzahl Pakete bezeichnet, die das Ziel nicht erreicht haben, zu der totalen Anzahl versendeten Pakete. Zur Bestimmung dieses Verhältnisses werden Pakete generiert und vom Sender zum Empfänger übermittelt. Dieser zählt die Anzahl empfangener Pakete und meldet den Zählerwert dem Sender zurück. Damit ist der Sender in der Lage, den Paketverlust zu berechnen.

3.4 Konzept

Zur Realisierung der Point to Point Testinfrastruktur wurde auf ein *One to Many* (1:n) Messprinzip gesetzt (auch als Multicast bezeichnet). Dies hat den Vorteil, dass mehrere Verbindungen gleichzeitig ausgemessen werden können. Der Nachteil dabei ist, dass so die Messpfade nur unidirektional charakterisiert werden (nur von Master zu Slave oder Uplink-Pfad). Die Abweichung zwischen Up- und Downlink Pfad kann durch die Wiederholung der Messung mit vertauschten Standorten verifiziert werden.

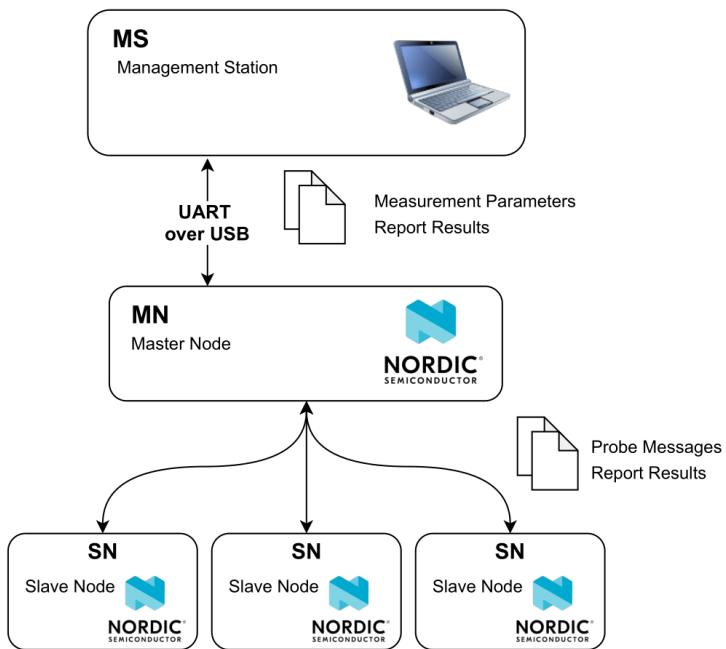


Abbildung 3.1: Konzeptschema P2P Testinfrastruktur

Das in Abbildung 3.1 gezeigte Konzept besteht aus einer Management Station, einem Master- und mehreren Slave Nodes. Über die Management Station können Messresultate angezeigt sowie Messparameter eingestellt werden. Als Ausgangspunkt zur Messung dient der MN (Master Node). Er koordiniert den Messablauf und sendet *Probe Packets* an alle SN (Slave Nodes). Diese protokollieren die Anzahl empfangener *Probe Packets* und melden ihre Messresultate an den MN zurück. Nach Empfang aller Resultate leitet der MN alle Messergebnisse an die MS weiter. Dort werden dem Benutzer die Messdaten dargestellt. Details zum Ablauf sind in Abschnitt 5.1 aufgeführt. Eine mögliche Anwendung der Testinfrastruktur wird im nachfolgenden Abschnitt 3.5 beschrieben.

3.5 Testszenarien

Die P2P Testinfrastruktur ist als einfach zu bedienendes Tool konzipiert. Es kann eingesetzt werden, um den Aufbau von sogenannten *Wireless Personal Area Networks (WPAN)* zu optimieren.

mieren. Mithilfe dieses Tools können zum Beispiel die Standorte der Teilnehmer eines Zigbee, Thread oder Bluetooth Mesh Netzwerks optimal gewählt werden.

Weiter kann die Testinfrastruktur als Störquelle in den Benchmarks von Mesh Netzwerken eingesetzt werden (siehe Abschnitt 7). Die Form der Störung kann dabei gemäss den einstellbaren Parametern aus Abschnitt 3.2 frei gewählt werden.

3.6 Messaufbau

Bei der verwendeten Hardware handelt es sich um einen nRF52840 Mikrocontroller (siehe auch Abschnitt 8). Als Master dient das Development-Kit (DK), welches über eine USB-Buchse verfügt. Als Slaves kommen sowohl DKs als auch die USB-Dongle Variante in Frage. Letztere sind einfacher unterzubringen und werden in diesem Messaufbau bevorzugt. Um die erhaltenen Daten auswerten zu können, geschieht die Visualisierung mittels einem Laptop oder PC. Dieser muss mit der USB Buchse des Masters verbunden werden und die entsprechende Auswertesoftware gestartet sein. Im nachfolgenden Abschnitt 4 wird die Inbetriebnahme und die Bedienung der P2P-Testinfrastruktur genauer beschrieben. Die Abbildung 3.2 zeigt den schematischen Messaufbau der P2P-Testinfrastruktur. Der Master kann durch den Einsatz eines Notebooks mobil gehalten werden, um so den optimalen Standort des Masters auszuloten. Die Slaves werden in der Wohnung oder im Haus an den möglichen Standorten der Mesh Nodes platziert. Danach wird die Messung gestartet.

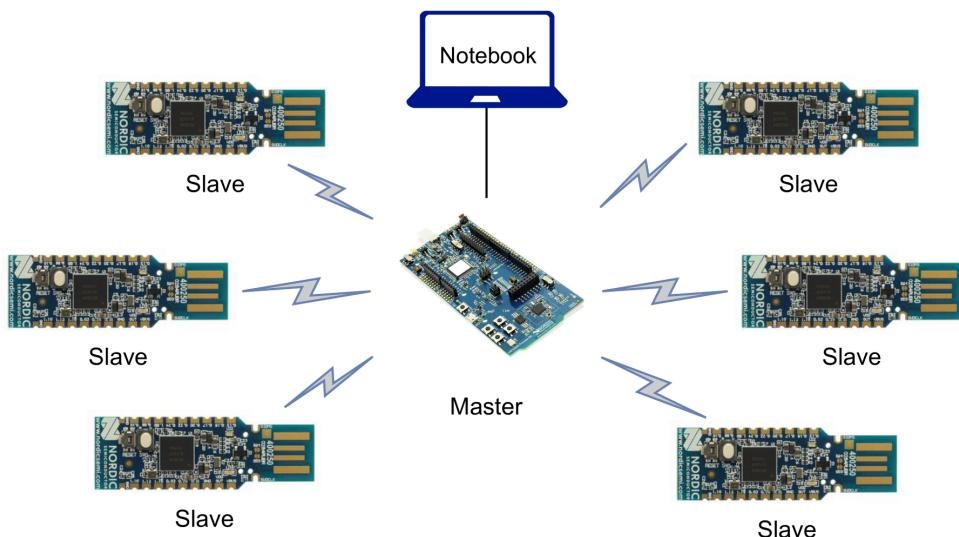


Abbildung 3.2: Messaufbau P2P Testinfrastruktur

4 Inbetriebnahme und Bedienung

Die P2P Testinfrastruktur kann von jedem interessierten Anwender eingesetzt werden. Zur Inbetriebnahme müssen jedoch folgende Hard- und Software Komponenten vorhanden und einsatzbereit sein:

- 1x nRF52840 DK (Master)
- 1x nRF52840 DK / Dongle (Slave)
- Laptop mit folgender Software:
 - Python 3.x
 - nrfutil (pip install nrfutil), zum Flashen der nRF52840.
 - Browser, zur Darstellung der Weboberfläche

Eine Anleitung und der Sourcecode sind im [Github-Repository](#)¹ zu diesem Projekt verfügbar. Master und Slaves sind mit der entsprechenden Firmware zu laden um diese für die Messung vorzubereiten. Anschliessend werden die Slaves im Raum platziert und der Master am Laptop eingesteckt. Nach erfolgtem Setup ist gemäss den nachfolgend beschriebenen Schritten vorzugehen.

Schritt 1 - Webserver Starten

Der Webserver wurde mit einer Windows Umgebung entwickelt und getestet. Die folgende Anleitung ist demzufolge nur für eine Windows Umgebung gültig. Für die Verwendung mit Linux oder MacOS sind unter folgendem [Link](#)² entsprechende Anleitungen verfügbar. Bevor der Webserver gestartet werden kann, muss der Sourcecode vom [Github-Repository](#)¹ geklont werden. Auf der Ordnerstruktur **P6_Software\P2P\Webserver\p2p_webserver** müssen folgende cmd Befehle ausgeführt werden.

```
Microsoft Windows [Version 10.0.18363.1016]
(c) 2019 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\GitHub\P6_Software\P2P\Webserver\p2p_webserver> .\myvenv\Scripts\activate
(myvenv) C:\Users\GitHub\P6_Software\P2P\Webserver\p2p_webserver> python .\manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
August 20, 2020 - 22:01:13
Django version 3.0.8, using settings 'p2p_site.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Sobald dies erfolgreich ausgeführt wurde, ist der Webserver gestartet und kann unter der Adresse <http://127.0.0.1:8000/> erreicht werden.

¹https://github.com/Rouben94/P6_Software[1]

²https://tutorial.djangogirls.org/en/django_installation/

Falls das Starten nicht geklappt hat, müssen möglicherweise Django und die benötigten Python Bibliotheken nachinstalliert werden. Dies wird wie folgt gemacht:

```
Microsoft Windows [Version 10.0.18363.1016]
(c) 2019 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\GitHub\P6_Software\P2P\Webserver> .\myvenv\Scripts\activate
(myenv) C:\Users\GitHub\P6_Software\P2P\Webserver> pip install Django==3.1
(myenv) C:\Users\GitHub\P6_Software\P2P\Webserver> pip install pyserial
(myenv) C:\Users\GitHub\P6_Software\P2P\Webserver> pip install plotly
(myenv) C:\Users\GitHub\P6_Software\P2P\Webserver> python manage.py makemigrations p2p
(myenv) C:\Users\GitHub\P6_Software\P2P\Webserver> python manage.py migrate p2p
(myenv) C:\Users\GitHub\P6_Software\P2P\Webserver> python .\manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
August 20, 2020 - 22:01:13
Django version 3.0.8, using settings 'p2p_site.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Schritt 2 - Verbindungsaufbau

Sobald der Webserver gestartet wurde, ist die Seite **Point to Point** zu sehen (siehe Abbildung 4.1). Wie in der Abbildung beschrieben, muss nun auf die Seite Connection gewechselt werden, damit die Verbindung zum Master aufgenommen werden kann. Es besteht die Möglichkeit das Hauptmenü auf der linken Seite auf und zu zuklappen (siehe Nummer 1 in Abbildung 4.1). Wenn das Hauptmenü aufgeklappt ist, kann das Github-Repository durch einen Klick auf die Schaltfläche **GitHub Repo** erreicht werden.

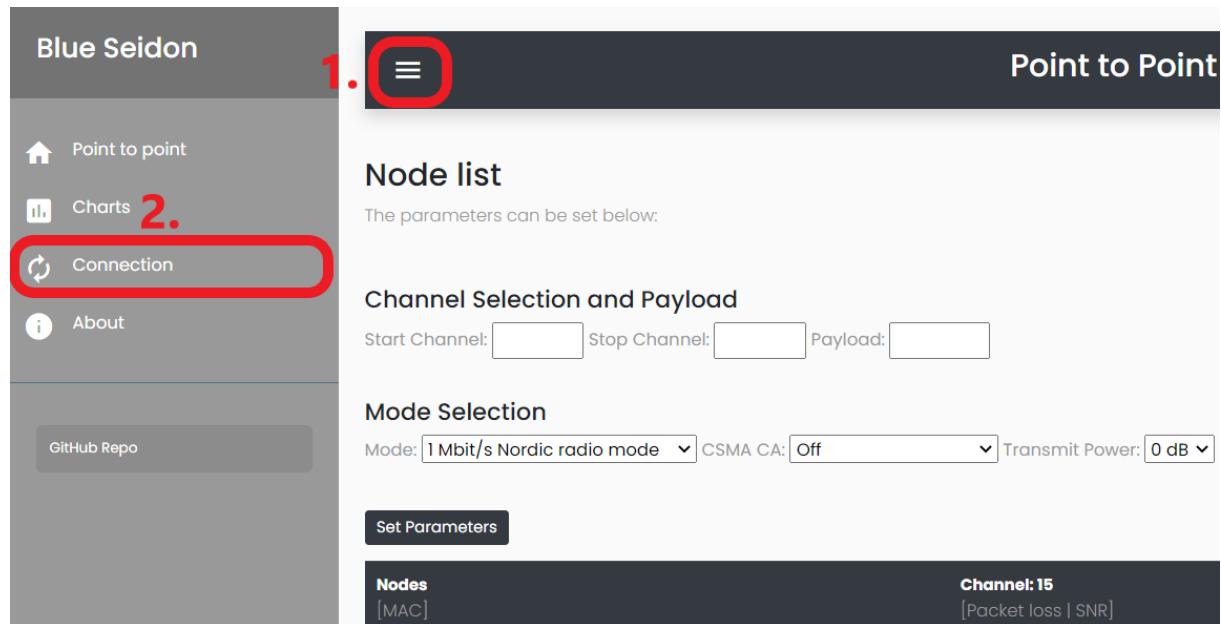


Abbildung 4.1: Webserver Nodelist View ausgeklappt

Auf der Seite **Connection** (siehe Abbildung 4.2) soll für die Kommunikation mit der Seriellen Schnittstelle der entsprechende COM-Port, an dem der Master registriert ist, ausgewählt werden (siehe Nummer 1 in Abbildung 4.2). Mit einem Klick auf das Feld **Connect** wird nun eine Verbindung hergestellt (siehe Nummer 2 in Abbildung 4.2). Falls der Master nicht angeschlossen oder auf einem anderen COM-Port registriert ist, wird dies mit einer Fehlermeldung mitgeteilt.

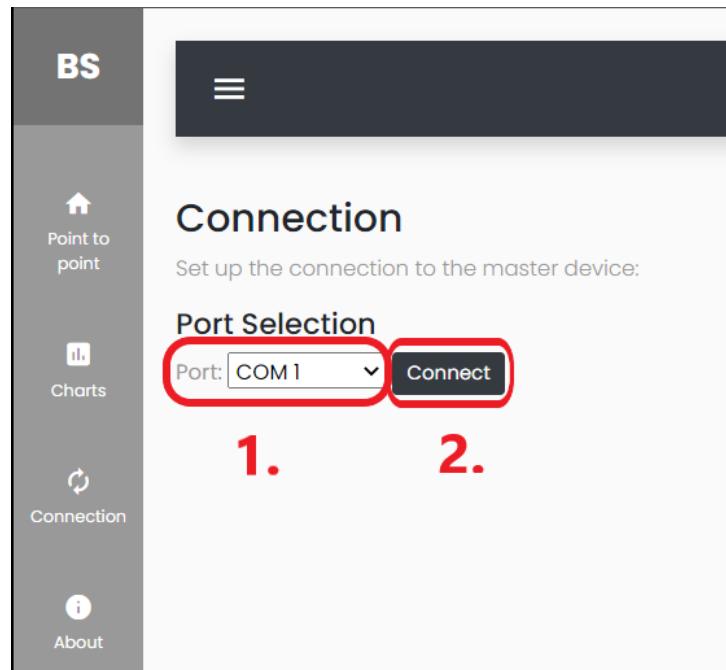


Abbildung 4.2: Webserver Connection View

Schritt 3 - Einstellen der Parameter

Nachdem die serielle Verbindung zum Master hergestellt ist, werden auf der Seite **Point to Point** die verfügbaren Slaves in der Tabelle aufgeführt. Die Tabelle aktualisiert sich automatisch. Wie in der Abbildung 4.3 ersichtlich, sind die Nodes anhand der MAC-Adresse identifizierbar. Die Tabelle zeigt die dazugehörigen Channels, welche zuvor eingestellt wurden. Pro Channel ist ein Wert für den Packetloss und den SNR zu sehen. Die Definition dieser Werte ist im Abschnitt 3.3 beschrieben.

Damit die Parameter der Nodes verändert werden können, besteht auf der gleichen Seite die Möglichkeit, die gewünschten Werte einzugeben und an den Master zu senden. Die Parameter können, wie in Abbildung 4.3 markiert, definiert werden. Folgende Vorgaben müssen bei der Eingabe eingehalten werden:

Radio Mode	IEEE 802.15.4 250 kBit	Alle Anderen Modi
Start Channel Feld	11	0
Stop Channel Feld	26	39
Payload Feld	0 - 120 bit	0 - 250 bit

Tabelle 4.1: Zulässige Parameter für die Eingabefelder in der Nodelist View

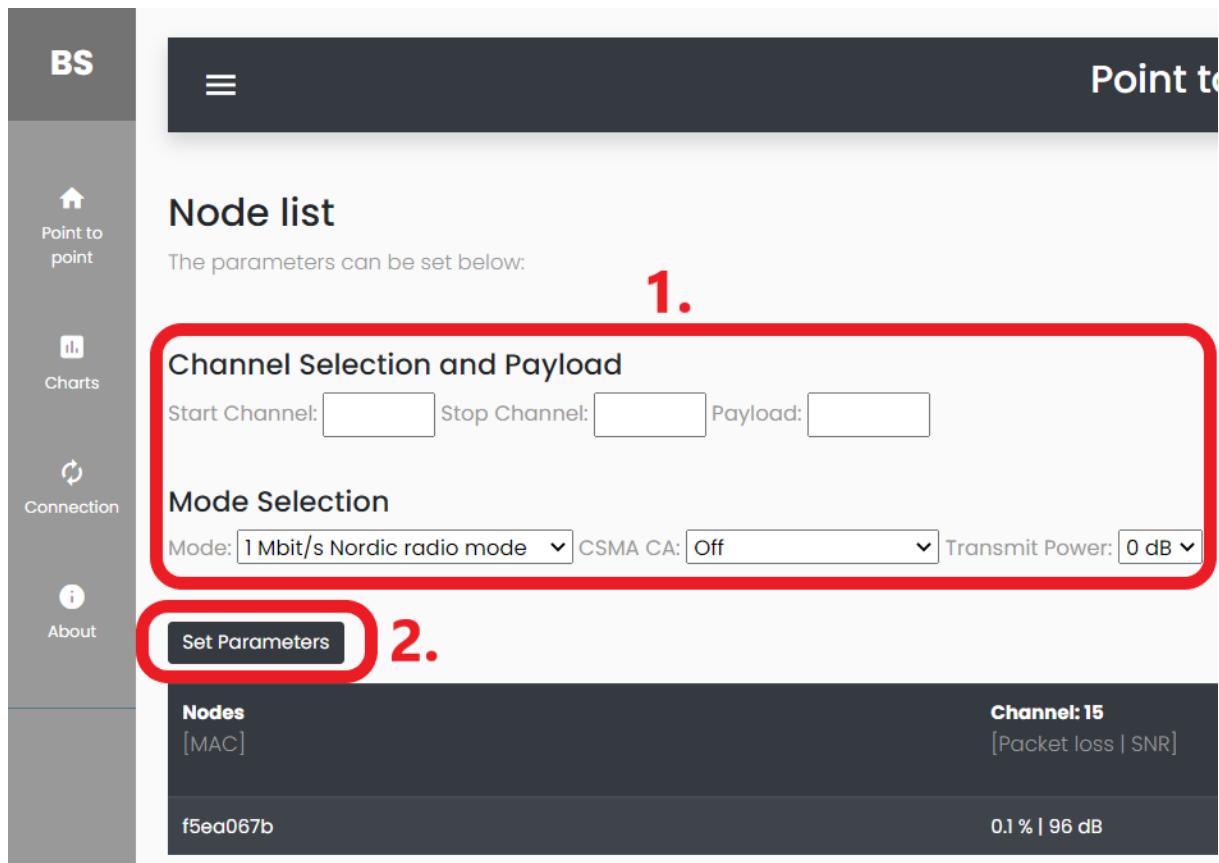


Abbildung 4.3: Webserver Nodelist View

Schritt 4 - Vergleich der Channels

Damit die Channels von einem Node besser miteinander verglichen werden können, ist auf der Seite **Chart** eine entsprechende Grafik verfügbar. Jeder Node wird in einem separaten aufklappbaren Fenster angezeigt. In Abbildung 4.4 mit der Nummer **1.** gekennzeichnet, ist ein solches Fenster zu sehen. Mit Klick auf die Schaltfläche mit der Nummer **1.** wird das Fenster aufgeklappt und alle eingestellten Channels werden nacheinander mit dem Packetloss in % aufgelistet.

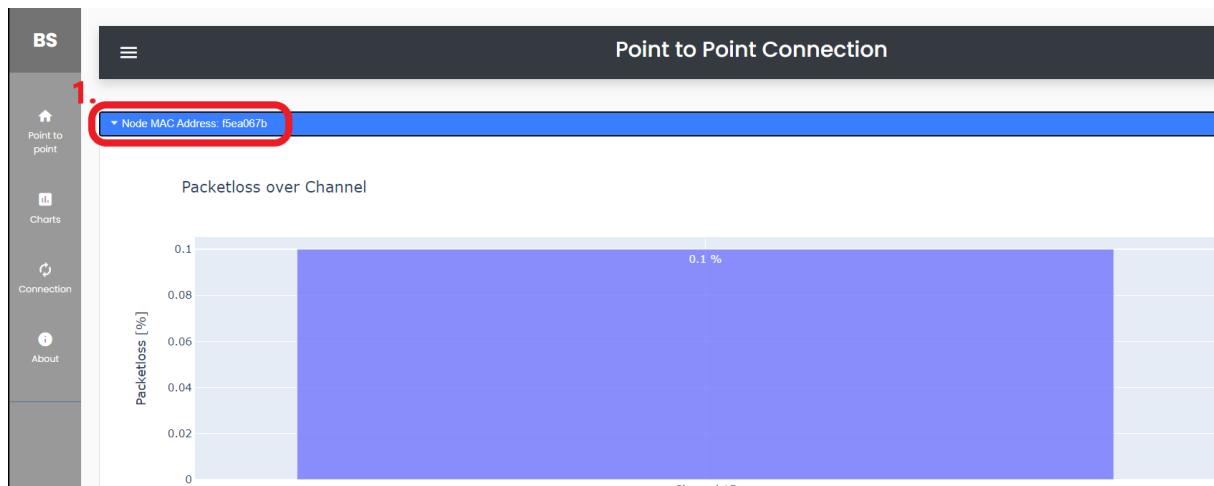


Abbildung 4.4: Webserver Chart View

5 Soft- und Firmware

In den nachfolgenden Abschnitten wird die Soft- und Firmware für die P2P Testinfrastruktur beschrieben. Der dazugehörige Sourcecode ist im [Github-Repository](#)³ zu diesem Projekt frei zugänglich und kann für die vertiefte Untersuchung konsultiert werden.

5.1 Soft- und Firmware

Die Firmware ist als zeitabhängige Schrittkette aufgebaut. Jeder Schritt ist über ein fixes Zeitfenster definiert. Damit alle Teilnehmer synchron die Schrittkette abarbeiten können, muss mit Hilfe einer Zeitsynchronisation auf den Master der exakte Startzeitpunkt kommuniziert werden. Unabhängig vom Zustand des aktiven Schrittes darf dieser sein Zeitfenster nicht überschreiten, ansonsten fällt die Schrittkette aus dem Takt.

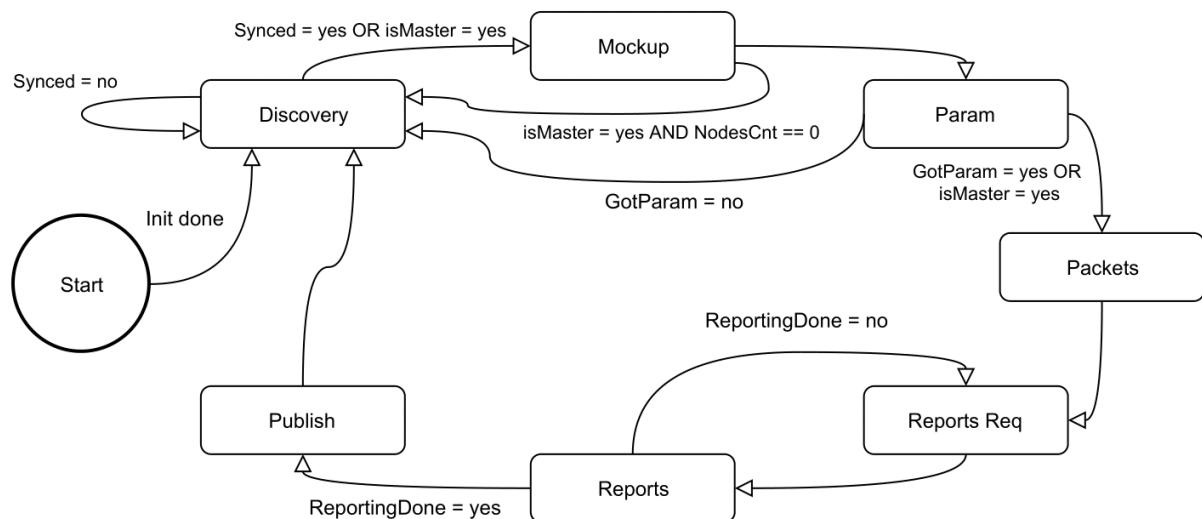


Abbildung 5.1: Schrittkette P2P Testinfrastruktur

Abbildung 5.1 zeigt die Schrittkette für den Master, sowie für die Slave-Nodes. Der Ablauf zeigt die Schritte, sowie die Bedingungen der Transitionen zwischen den Schritten. Jedoch muss beachtet werden, dass jeder Schritt immer während einer bestimmten Zeit aktiv bleibt. Somit ist ersichtlich dass der *Packets-State* nach Ablauf seines Zeitfensters immer zum *ReportsReq-State* führt. Dabei hängt der Wechsel nach dem *Discovery-State* von verschiedenen Bedingungen ab.

Auf die Funktion der Schritte wird nachfolgend kurz eingegangen:

- **Discovery:** Der Master verteilt die Zeitsynchronisation an die Slaves. Die Slaves synchronisieren sich auf das Signal des Masters auf, sofern sie in seiner Reichweite liegen. Hat ein Slave sich nicht synchronisieren können, verweilt dieser im *Discovery-State*. Der Master kann in jedem Fall zum nächsten Schritt voranschreiten.
- **Mockup:** Der Master wartet auf eine Antwort der einzelnen Slaves. Jeder Slave generiert einen zufälligen Timeslot, um sich beim Master anzumelden. Der Master führt alle Slaves, die sich gemeldet haben, in einer Liste auf. Hat sich kein Slave gemeldet, ist diese Liste leer und der Master wird zum *Discovery State* zurückkehren. Ist ein Slave bereits beim Master angemeldet, so muss er sich nicht erneut anmelden.
- **Param:** Der Master versendet die Packets-Parameter (Mode, StartChannel, StopChannel, etc.). Erhält ein Slave keine Daten, so kehrt er in den *Discovery-State* zurück.

³https://github.com/Rouben94/P6_Software[1]

- **Packets:** Der Master versendet *Probe Packets* mit den zuvor kommunizierten Einstellungen. Die Slaves empfangen diese Daten, zählen die Anzahl erfolgreich empfangener Pakete und erfassen weitere Messdaten.
- **Reports Req:** Nach dem Ausmessen beginnt der Master mit dem Einsammeln der einzelnen Slave Reports. Dazu sendet er an jeden Slave in seiner Liste (aus dem *Mockup-State*) Report-Anfragen, sogenannte *Report-Requests*. Die Slaves warten auf einen *Report Request* vom Master.
- **Reports:** Der Slave, welcher den Request erhalten hat, sendet seinen Report zum Master. Dies hat innerhalb einer gewissen Zeit zu erfolgen. Sendet ein Slave nach einer gewissen Anzahl von Versuchen innerhalb dieser begrenzten Zeit keine Reports, so wird er aus der Liste der gemeldeten Slaves entfernt. Sobald der Master alle Reports von allen Slaves empfangen hat, meldet er an alle Slaves über eine spezielle *Reports-Request-Message*, dass das Reporting beendet ist.
- **Publish:** Im *Publishing-State* hat der Master Zeit, die Daten an die übergeordnete Stelle zu übermitteln (USB-UART). Die Slaves können in diesem State Energie sparen. Alle Teilnehmer beginnen wieder von vorne, im *Discovery-State*, wo sie sich neu synchronisieren.

5.2 Low Level Radio Driver

Die Kommunikation über das Radio-Interface wurde mittels einem eigens entwickelten Radio-Driver speziell für die nRF52 / nRF53 SOCs ermöglicht. Dieser stellt die nötigen Funktionen für die P2P Testinfrastruktur zur Verfügung. Zum Beispiel kann die Mode- und Kanalwahl sowie das Senden und Empfangen von Daten mittels simplen Kommandos erfolgen. Bei der Entwicklung diente das *Radio Test Example* [2] als Vorlage. Der Radio-Driver steuert mithilfe der *NRF-HAL Radio Library* oder über direkten Zugriff auf die Peripherie-Register das Radio-Interface an.

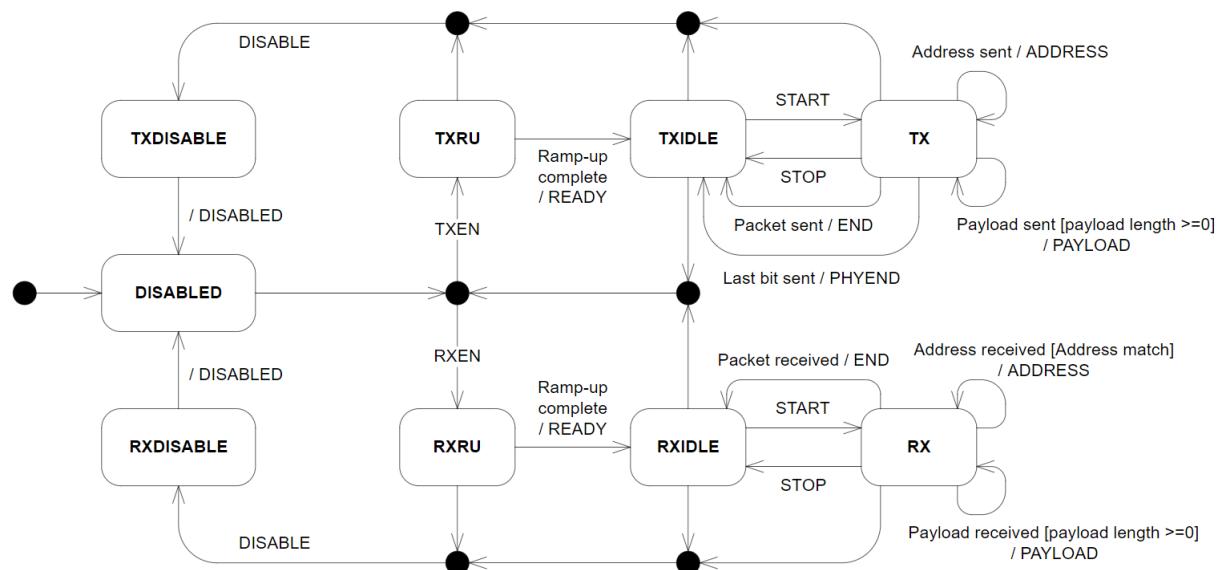


Abbildung 5.2: Radio States [3]

Die Radio Hardware der nRF52 und nRF53 SoCs verfügt über verschiedene Zustände (siehe Abbildung 5.2). Abhängig von der gewünschten Operation (Senden oder Empfangen) werden die States abgearbeitet. Um Energie zu sparen, wird nach Abschluss der gewünschten Tätigkeit immer in den Disabled-State gewechselt. Die Benachrichtigung des Radio Drivers durch die

Hardware erfolgt zudem mittels Interrupts, wodurch der Energieverbrauch zusätzlich minimiert werden kann.

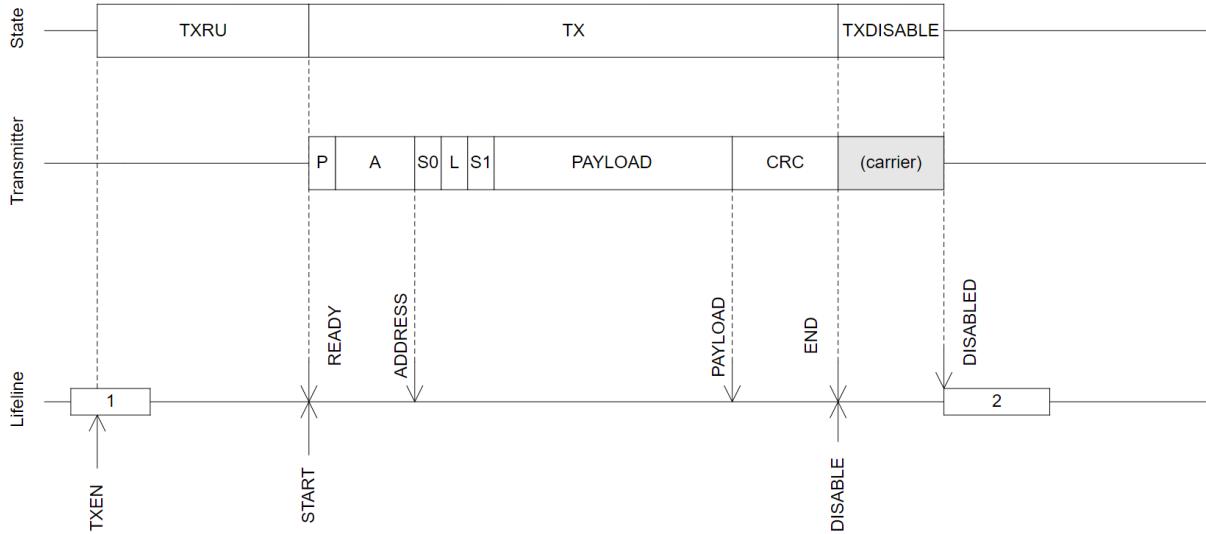


Abbildung 5.3: Sende-Ablauf mit Verknüpfungen [4]

Abbildung 5.3 zeigt den Ablauf beim Senden eines Pakets. Die zu sendenden Daten (Payload) müssen vorgängig im RAM vorliegen. Anschliessend wird der Packet Pointer des Radio-Interface auf die entsprechende Speicher-Adresse eingestellt. Zu beachten gilt, dass im ersten Byte die Länge der Payload angegeben werden muss. Das Längen-Feld wird vom Radio Driver automatisch ergänzt. Zusätzlich kann den Sendedaten ein Adressfeld mitgegeben werden. Nach der vollständigen Konfiguration des Radio-Interface wird das Senden durch den Befehl TXEN initiiert. Mithilfe von Verknüpfungen (Shorts) wird nach dem Ready-Event automatisch der Start-Task ausgelöst. Das Senden läuft also nach dem Initiieren vollautomatisch ab. Der Radio-Driver wartet auf das Auslösen des Disabled-Events. Danach ist das Senden erfolgreich abgeschlossen und der Funktionsaufruf kehrt zum Hauptprogramm zurück.

Das Senden mittels CCA wird im *IEEE802.15.4-Mode* unterstützt. Dazu wird vor dem Senden der Kanal abgehört, um Kollisionen zu vermeiden. Mittels dem RXEN Befehl wird der CCAStart-Task aktiv. Dieser führt abhängig vom konfigurierten CCA-Modus eine Prüfung des Kanals durch. Ist der Kanal nicht belegt, wird ein CCAIdle-Event generiert, welcher mittels Verknüpfung automatisch den TXEN-Task startet (siehe Abbildung 5.4). Sendet ein anderer Teilnehmer zum gleichen Zeitpunkt auf diesem Kanal, wird ein CCABusy-Event generiert, welcher ebenfalls den Disable-Task ausführt (siehe Abbildung 5.5). Somit enden beide Varianten (Idle und Busy) im Disabled-State, wobei bei Busy Variante keine Daten gesendet werden konnten. [4]

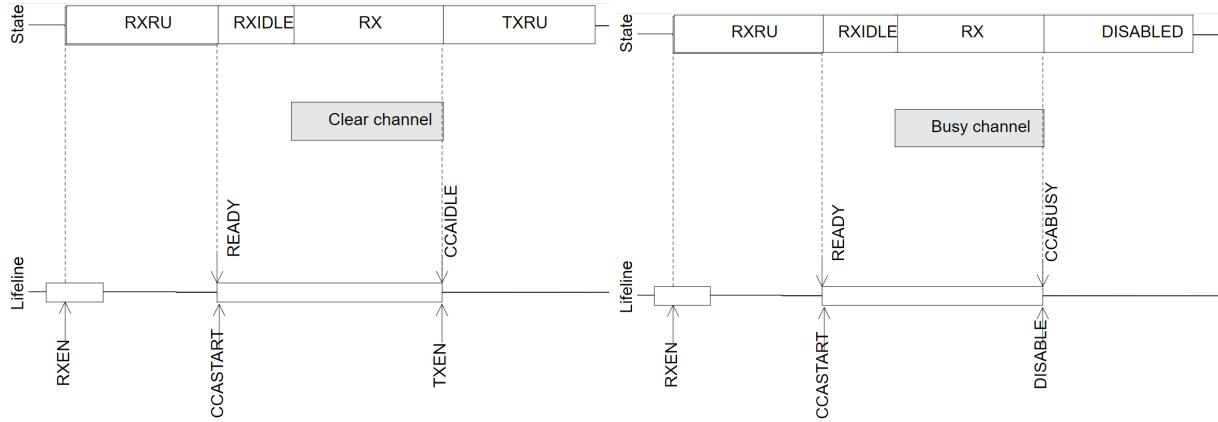


Abbildung 5.4: CCA Prüfung (Idle Event)[5]

Abbildung 5.5: CCA Prüfung (Busy Event)[5]

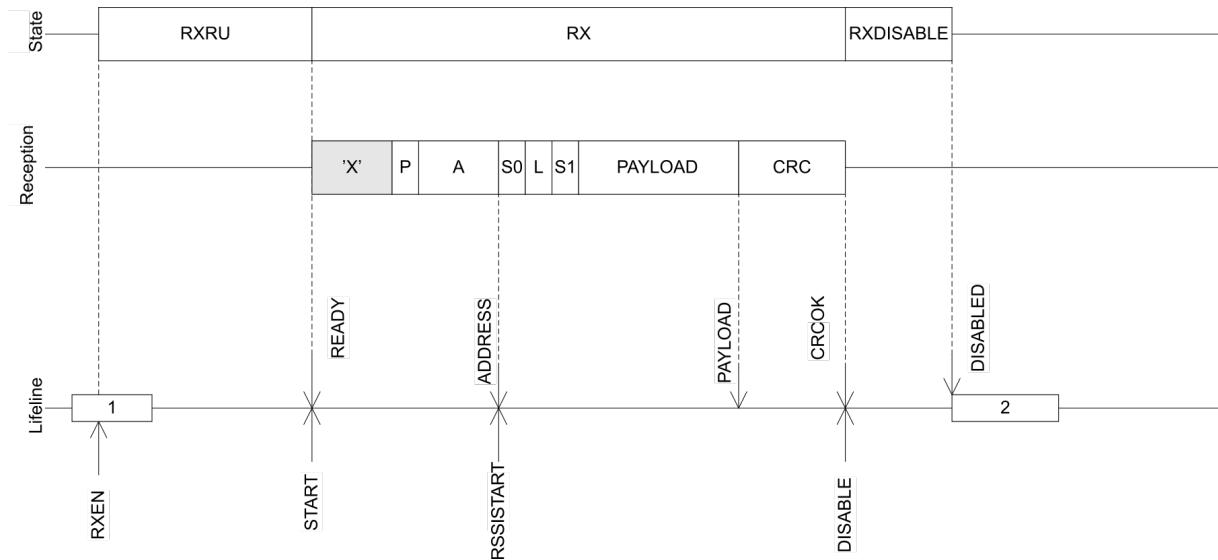


Abbildung 5.6: Empfangs-Ablauf mit Verknüpfungen [6]

Der Empfangsablauf eines Pakets wird in Abbildung 5.6 dargestellt. Grundsätzlich ist der Ablauf gleich aufgebaut wie beim Senden eines Pakets. Nach dem Initiieren mittels RXEN-Befehl fährt der Empfänger hoch und wartet auf den Empfang eines Pakets. Nach Eingang einer Präambel-Sequenz wird das Adressfeld geprüft. Dieses kann vorgängig so definiert werden, dass nur Pakete mit übereinstimmender Adresse empfangen werden. Bei allen BLE-Modes erfolgt die Prüfung auf Hardware-Ebene (ohne Interaktion der CPU). Beim IEEE802.15.4-Mode steht dieses Feature jedoch nicht zur Verfügung. Das Vergleichen des Adressfeldes muss daher der Radio-Driver selbst übernehmen.

Um die Signalstärke (RSSI) des eingehenden Pakets zu messen, wird eine Verknüpfung zwischen dem *Address-Event* und dem *RSSISTART-Task* aktiviert. Nach erfolgreicher Prüfung der CRC Checksumme wird mittels einer Verknüpfung des CRCOK-Event der Disable-Task ausgeführt. Beim Empfang eines Pakets wartet der Radio-Driver während eines gewissen Timeouts auf den Disabled-Event. Die Empfangsfunktion gibt die Restzeit des Timeouts in Millisekunden zurück, wodurch sich ein erfolgreicher Empfang prüfen lässt. Das Bufferhandling übernimmt der Radio-Driver. Er erwartet beim Senden und Empfangen von Daten eine vordefinierte *Radio-Packet-Structure*. Es gilt zu beachten, dass die Daten erst nach dem End-Event vollständig abgearbeitet

wurden, da der Zugriff der Hardware via DMA (Direct Memory Access) erfolgt. Ebenfalls wurde festgestellt, dass beim Wechsel zum oder vom *IEEE802.15.4-Mode* der Kanalwechsel vor dem Modewechsel erfolgen muss. Ansonsten werden keine Pakete empfangen. [6]

5.3 Broadcasting Collisions Probability

Kollisionen entstehen, wenn mehrere Teilnehmer gleichzeitig auf dem selben Kanal (Frequenz) senden. Dabei stören sich die beiden Sender und es besteht die Möglichkeit, dass der Empfänger keine Daten empfangen kann. Wenn sich mehrere Teilnehmer neu mit dem Netz verbinden möchten, kann dies ebenfalls zu Problemen führen. Leider können solche Störungen nicht immer vermieden werden. In der vorliegenden Anwendung kann ein solcher Fall im Mockup-State entstehend. Hier soll sich jeder Slave, welcher dem Master noch unbekannt ist, auf die Discovery-Anfrage des Masters melden. Die Lösung ist, dass jeder Slave einen zufälligen Zeitpunkt zum Antworten auswählt. Wie gross dieses Zeitfenster sein muss, in welchem ein Slave einen Zeitpunkt zufällig wählen darf, berechnet sich wie folgt:

Angenommen es gibt N Teilnehmer, welche t_a Sekunden brauchen, um eine Antwort zu senden. Zusätzlich existieren N_{ch} verschiedene Kanäle, auf welchen die Teilnehmer antworten können. Die Wahrscheinlichkeit, dass sich zwei Sender im Zeitfenster t_I Sekunden nicht überlappen P_{miss} , lässt sich gemäss Formel 5.1 bestimmen. [7]

$$P_{miss} = \left(1 - \frac{2 \cdot t_a}{N_{ch}} \cdot t_I\right)^{N-1} \quad (5.1)$$

Die folgenden Werte wurden zur Berechnung des Zeitfensters im Mockup-State verwendet:

- $N = 50$
- $t_a = 5ms$
- $N_{ch} = 3$
- $t_I = 1.2s$

Somit liegt die Wahrscheinlichkeit, dass alle 50 Nodes sich verfehlten bei 85%.

5.4 Zeitsynchronisation

Die Zeitsynchronisation wurde mittels einer Offset Kompensation gelöst. Der Zeitgeber (Master) sendet seine Zeit (Timestamp) über einen Broadcast an alle Teilnehmer in der Umgebung. Die Slaves vergleichen die empfangene Master-Zeit mit ihrer lokalen Zeit und errechnen den Unterschied (Offset) zwischen dem Master-Timestamp und Slave-Timestamp. Somit können sie ihre Zeit an die des Masters angleichen. Das Prinzip ist relativ simpel, besitzt jedoch einige Ungenauigkeiten. Die Verzögerungszeit zwischen dem Auslesen des Master-Timestamps bis zum Empfangen und Vergleichen mit dem Slave-Timestamp ist kritisch.

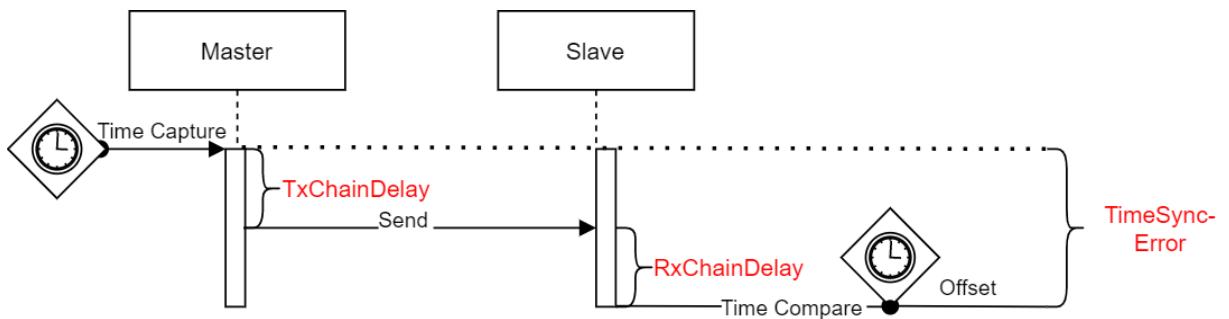


Abbildung 5.7: Zeitsynchronisation über Offset mit Fehler

Wie in Abbildung 5.7 ersichtlich ist, entsteht ab dem Auslesen der Zeit bis zum Senden eine Verzögerung die TxChainDelay sowie beim Empfänger die RxChainDelay. Das Ziel ist, diese Verzögerungen so gering wie möglich zu halten.

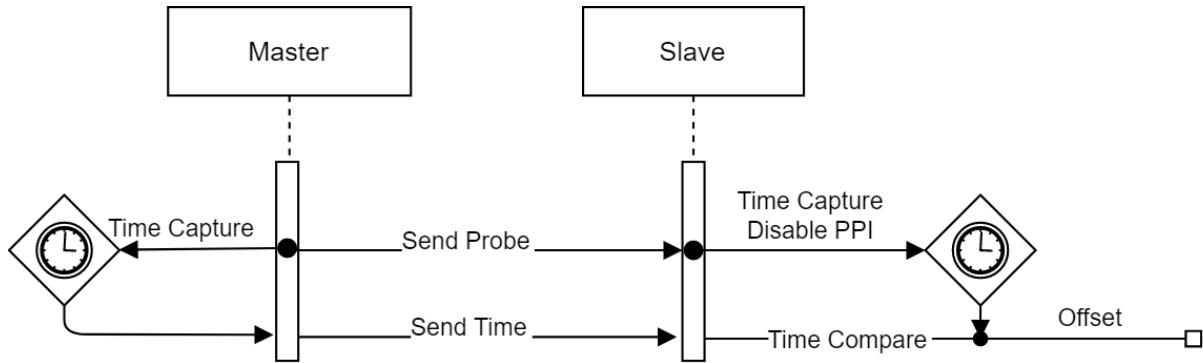


Abbildung 5.8: Zeitsynchronisation über Offset mit PPI

Die Lösung dieses Problems wird in Abbildung 5.8 gezeigt. Der nRF52840 SoC verfügt über ein PPI (Programmable peripheral interconnect) Modul. Mithilfe des PPI lassen sich verschiedene Events und Tasks direkt Verknüpfen, ohne dass die CPU dabei involviert ist. Dies führt zu einer minimalen Verzögerung (ca. $1/16\mu\text{s}$) zwischen Event und Task. Dazu wird der Event, wenn ein Packet erfolgreich gesendet wurde (Radio-End-Event), mit dem Task zum Erfassen des Synctimers (Synctimer-Capture-Task) verknüpft. Beim Slave wird der Event, wenn ein Packet erfolgreich Empfangen wurde (Radio-CRCOK-Event), mit dem Task zum Erfassen des Synctimers (Synctimer-Capture-Task) verknüpft. Nach dem Empfang muss der Slave diesen PPI sofort deaktivieren, da ein direkt anschliessendes Paket ebenfalls ein Erfassen des Timers auslöst. Der Master wird nach der Deaktivierung des PPI mit einem Folgepaket seine Master-Zeit dem Slave mitteilen, welche zum Zeitpunkt des Probe-Packets registriert wurde. Der Slave errechnet den Offset gemäss Formel 5.2 und die synchronisierte Zeit gemäss Formel 5.3. [8]

$$T_{Offset} = T_{Master} - T_{Slave} \quad (5.2)$$

$$T_{Sync} = T_{Slave} + T_{Offset} \quad (5.3)$$

Zur Verifizierung der Zeitsynchronisation wurde ein Oszilloskop an jeweils einen GPIO-Pin des Masters und an einen Pin des Slaves angeschlossen. Auf dem Master und Slave wurden nacheinander Timer-Interrupts registriert, welche relativ zur synchronisierten Zeit auf dem Master und Slave genau gleichzeitig auslösen. Mithilfe eines PPI-Kanals wurde der Interrupt-Event auf den GPIO-Pin verknüpft. Beim Auslösen des Interrupts wird der Zustand des GPIO-Pins invertiert. Dadurch wird die Abweichung präzise auf dem Oszilloskop ersichtlich.

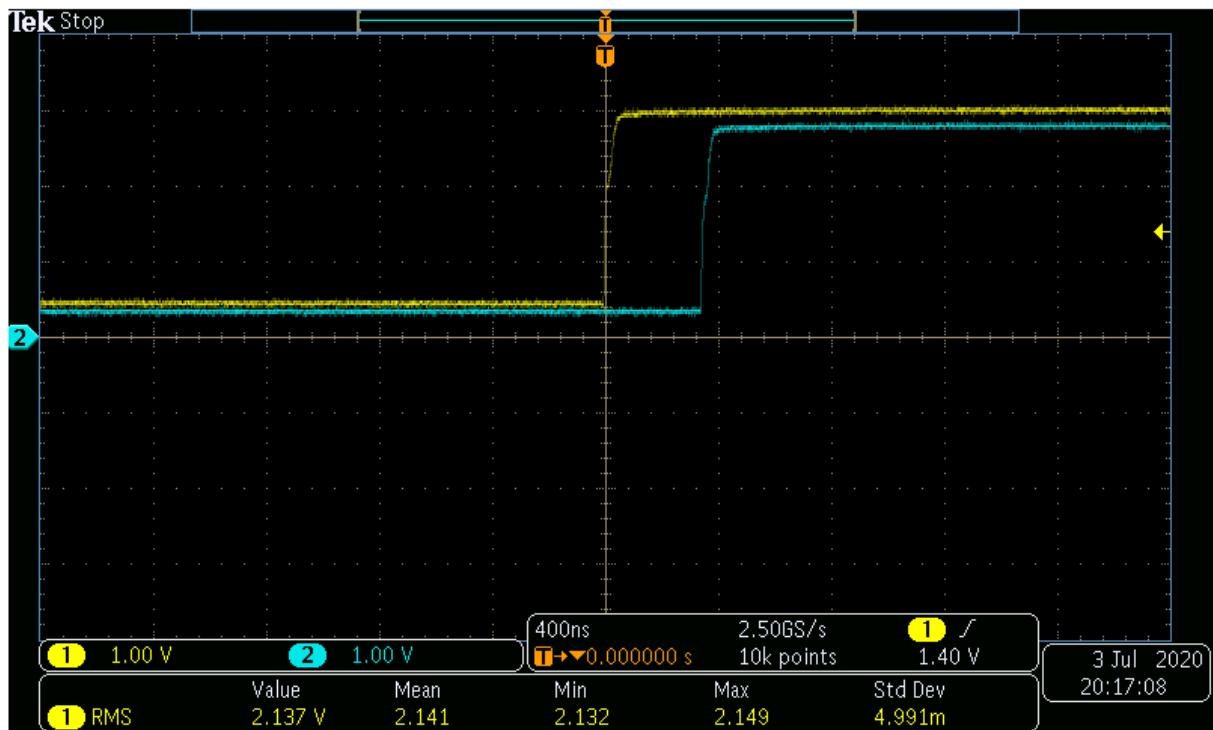


Abbildung 5.9: Verifizierung Zeitsynchronisation mittels Oszilloskop.

Abbildung 5.9 zeigt den Signalverlauf der GPIOs von Master (gelb) und Slave (blau). Anfänglich lagen die Abweichungen im Bereich von ca. $40\mu\text{s}$. Durch die Wiederholungen der Messung konnte dieser Fehler als systematisch identifiziert werden. Mithilfe einer Korrektur um $40\mu\text{s}$ konnte die Genauigkeit in den Bereich von 100ns bis 1200ns beschränkt werden. Eine Abweichung der Zeitsynchronisation von ca. $1\mu\text{s}$ sollte genügend genau sein, damit die Messungen als gute Grundlage für die State machine aus Abschnitt 5.1 dienen können.

5.5 Webserver

Damit die P2P-Testinfrastruktur visualisiert und mit einem schönen User Interface gesteuert werden kann, wurde ein Webserver aufgesetzt. Das Ziel ist, auf einem Rechner unter einer Web-Adresse einen Webserver zu erreichen, welche es ermöglicht alle Einstellungen über eine serielle Schnittstelle dem BMN mitzuteilen. Damit wird es möglich den Ablauf bequem über den Webserver zu steuern.

5.5.1 Django

Als Grundlage für den Webserver dient eine Django Instanz. Django ist ein open-source Webframework, welches auf dem Model-View-Controller Prinzip basiert und in der Programmiersprache Python geschrieben wurde. Das Framework hat sein eigenes Benennungssystem für alle Funktionen und Komponenten. Zum Beispiel werden HTTP-Antworten als Views bezeichnet. Ein grosser Vorteil von Django ist die Administrationsseite. Diese ist Bestandteil des Projekts und dient dazu die Models zu verwalten. Models sind Objekte, die in einer Datenbank des Webservers gespeichert werden. Die gängigsten Datenbanksysteme, wie z.B. SQLite sind in Django integriert und können aktiviert bzw. implementiert werden [9]. In der Tabelle 5.1 sind die wichtigsten Features von Django aufgelistet.

Einfacher Syntax (Python)	Model-View-Controller	Administrations Seite
Model-View-Controller	Eigener Webserver	Gängigste Datenbanken
HTTP Bibliothek	Python Unit Test Framework	Models für Datenbank

Tabelle 5.1: Features Django Webserver [9]

5.5.2 Software Grundlagen

Bei einer gewöhnlichen datengesteuerten Webseite wartet die Webanwendung auf eine HTTP-Anfrage des Browsers. Wird ein Anfrage empfangen, so bestimmt die Anwendung, auf Grundlage der URL, ob ein GET oder POST Event ausgelöst wird. Je nach Anfrage wird somit aus einer Datenbank gelesen oder es wird in die Datenbank geschrieben. Die Anwendung sendet danach eine Antwort an den Webbrower, wobei häufig eine HTML-Seite erzeugt wird, um die Antwort darzustellen. Diese Schritte sind in Django, wie in der Abbildung 5.10 aufgeführt, in separaten Dateien zusammengefasst. Dies werden unterhalb kurz beschrieben.

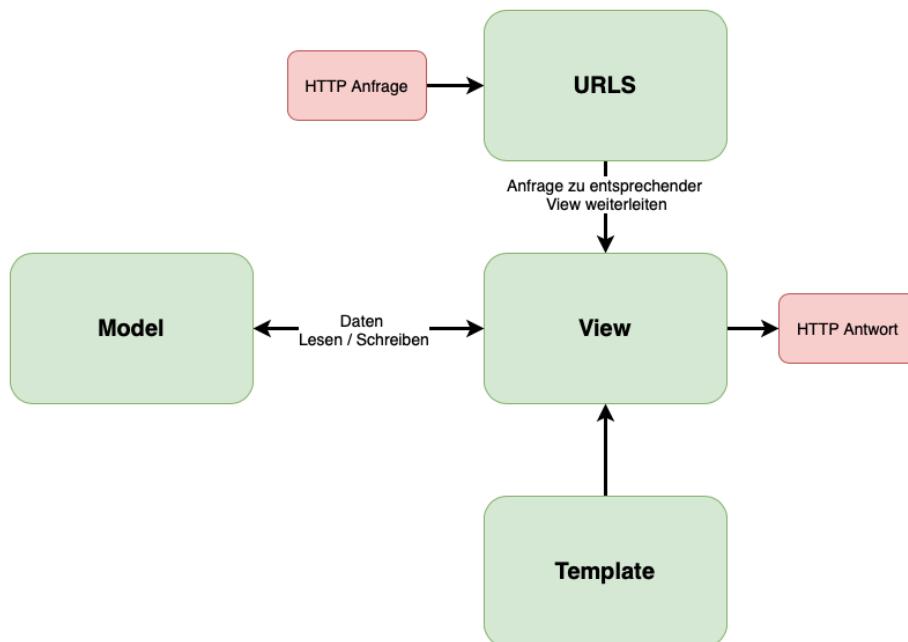


Abbildung 5.10: Django Funktionen [10]

URLs

Die URL Datei leitet die Anfrage an die entsprechende View weiter. Damit dies gelingt wird ein URL-Mapper eingesetzt. Dieser dient dazu, die Anfrage so zu interpretieren, damit die richtige View aufgerufen wird. Zusätzlich erkennt der URL-Mapper Zeichenketten und Ziffern in der URL und kann diese abgleichen und anschliessend als Daten an die View-Funktion weitergeben.

Views

Die View ist eine Request-Handler-Funktion, die HTTP-Anfragen empfängt und zurückgibt. Die View kann über die Model Funktion auf Daten in der Datenbank zugreifen. Über die Template-Funktion beschreibt die View schlussendlich, wie die HTTP-Antwort formatiert werden muss.

Models

Ein Model ist ein Python Objekt, welches die Struktur der Daten einer Anwendung definiert. Darüber hinaus stellt die Funktion die Mittel wie beispielsweise das Hinzufügen, Löschen oder

Ändern bereit, um die Daten in der Datenbank zu verwalten. Die Model Funktion ist somit die Schnittstelle zur Datenbank.

Templates

Das Template ist eine Textdatei, in der die Struktur oder das Layout einer zugehörigen HTML-Seite definiert wird. Für die Struktur werden Platzhalter zur Darstellung des eigentlichen Inhalts verwendet. Demnach kann die View-Funktion mit Hilfe vom Templates eine dynamische HTML-Seite erzeugen und diese mit Daten der Model-Funktion füllen.

5.5.3 Aufbau Software

Der Aufbau des Django Weberservers für die P2P-Testinfrastruktur ist in Abbildung 5.11 ersichtlich.

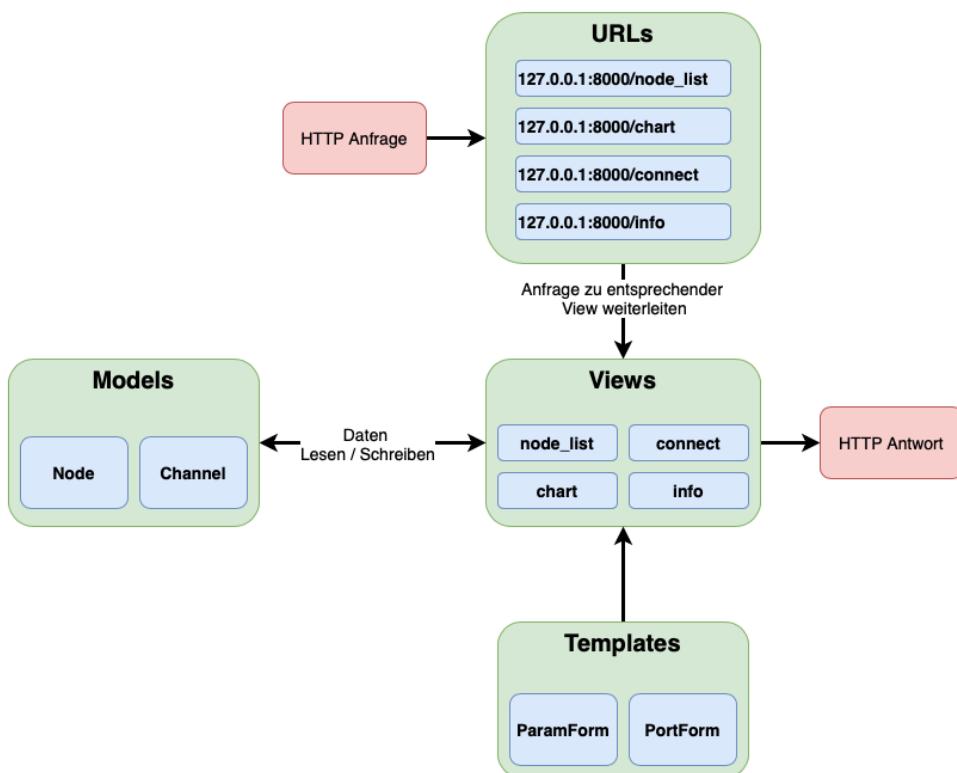


Abbildung 5.11: Funktionen P2P Webserver

URLs

In der folgenden Tabelle sind alle URLs aufgelistet, welche verwendet wurden, um auf die entsprechenden Views zu verweisen.

- `127.0.0.1:8000/node_list` referenziert auf die View `node_list`
- `127.0.0.1:8000/chart` referenziert auf die View `chart`
- `127.0.0.1:8000/connect` referenziert auf die View `connect`
- `127.0.0.1:8000/info` referenziert auf die View `info`

Views

Es wurden vier Views im P2P-Webserver erstellt, die die Schnittstellen zu den Forms, den Models und den HTML-Dateien bilden:

- `node_list`

- chart
- connect
- info

Die Bedienung der einzelnen Seiten wurde im Kapitel 4 bereits erläutert.

Models

Für die Speicherung der Daten vom P2P Master sind die folgenden zwei Objekte verfügbar:

- Node
- Channel

Diese zwei Objekte sind miteinander verknüpft. Wenn ein Objekt Node erstellt wird, können die dazugehörigen Channel Informationen ebenso erstellt werden. Dadurch ist eine klare Struktur gegeben. Über das Objekt Node können die verschiedenen Channel Informationen abgerufen werden.

Folgende Parameter können dem Objekt Node als Information mitgegeben werden:

- MAC Adresse

Folgende Parameter können dem Objekt Channel als Information mitgegeben werden:

- Channel Nummer
- Signal to noise ratio
- Packetloss
- Node ID

Templates

Als Template wurden zwei verschiedene Django Forms erstellt. Die eine Form ist eine Anordnung an Eingabefeldern, die frei bestimmt werden können. Die Dropdown Auswahlmöglichkeiten sind in der Tabelle 5.2 aufgelistet.

Auswahlfelder			
Mode	CSMA CA	Tx Power	Port
1 Mbit/s Nordic radio mode	Off	0 dB	Disconnect
2 Mbit/s Nordic radio mode	Ed Mode	1 dB	COM 1
1 Mbit/s BLE	Carrier Mode	2 dB	COM 2
2 Mbit/s BLE	Carrier and Ed Mode	3 dB	COM 3
Long range 125 kbit/s TX	Carrier or Ed Mode	4 dB	COM 4
Long range 500 kbit/s TX		5 dB	COM 5
IEEE 802.15.4-2006 250 kbit/s		6 dB	COM 6
		7 dB	COM 7
		8 dB	...

Tabelle 5.2: Tabelle Auswahlfelder

Für die Eingabe der Parameter, die auf dem P2P Master eingestellt werden können, ist die Form *ParamForm* mit folgenden Eingabefeldern verfügbar:

- Start Channel -> Integer Feld
- Stop Channel -> Integer Feld
- Size -> Integer Feld
- Mode -> Drop Down Auswahlfeld
- CSMA CA -> Drop Down Auswahlfeld
- Tx Power -> Drop Down Auswahlfeld

Für die Auswahl des Ports der Seriellen Verbindung zum Master ist die Form *ParamForm* mit folgenden Eingabefeldern verfügbar:

- Port -> Drop Down Auswahlfeld

In Abbildung 5.12 und 5.13 sind die Eingabe Forms ersichtlich. Die Daten werden mit einem Button, der ein HTTP-POST Ereignis auslöst, an die entsprechende View weitergeleitet.

Channel Selection and Payload

Start Channel: Stop Channel: Payload:

Mode Selection

Mode: CSMA CA: Transmit Power:

Abbildung 5.12: Parameter Form

Port Selection

Port:

Abbildung 5.13: Port Form

Serielle Kommunikation:

Die serielle Kommunikation wird mithilfe der Python Bibliothek *PySerial* aufgebaut. Auf der P2P-Webserver Seite Connect kann der Port ausgewählt werden, an dem der P2P-Master angeschlossen ist. Sobald die Verbindung zum Master besteht, wird ein separater Python Thread gestartet. In diesem Thread wird in einem while-loop das Serielle Interface permanent ausgelesen. Sobald der Master einen neuen Mess-Report schickt, wird dieser ausgewertet und die Objekte Node und Channel werden anhand des Reports erstellt. In Abbildung 5.14 ist dieser Ablauf ersichtlich.

Die erhaltenen Daten werden in der View *Node_List* in einer dynamischen Tabelle, die jede Sekunde aktualisiert wird, angezeigt. Zudem ist in der View *Chart* eine Visualisierung der Daten ersichtlich. Die Messergebnisse werden mit Hilfe der Bibliothek *PlotLy* dargestellt.

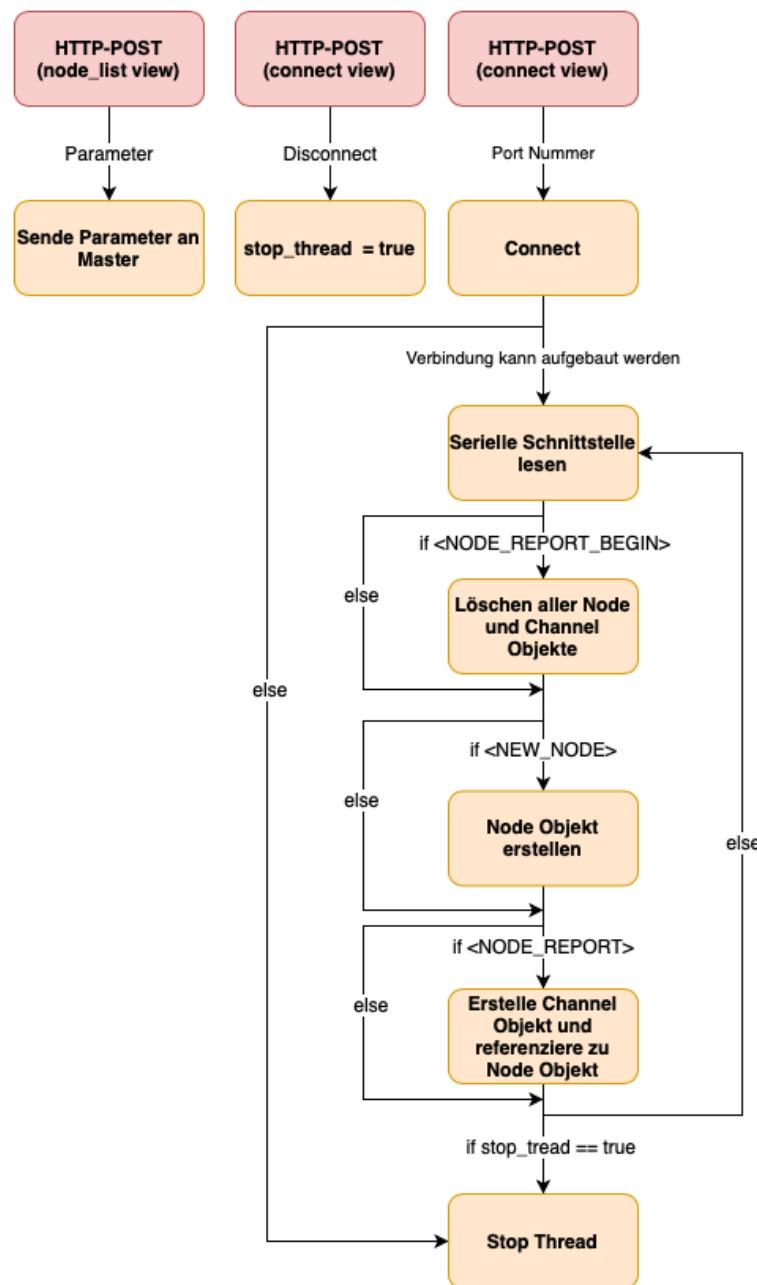


Abbildung 5.14: Ablauf Django Serial

6 Schluss

Abschliessend wird eine kurze Validierung und Verifizierung der Messresultate, sowie ein Fazit vorgenommen.

6.1 Validierung

Die vorgegebenen Ziele zur Erfassung des SNR und Paketverlusts konnten erreicht werden. Das Tool erfüllt mittels der grafischen Oberfläche seinen Zweck als universell einsetzbares Messwerkzeug. Das Erfassen des SNRs funktioniert jedoch nicht immer optimal. Durch Störeinflüsse, welche sich schnell ändern (WLAN etc.) kann es vorkommen, dass die Angabe des SNRs nicht korrekt erfolgt. Die Ergebnisse des Paketverlusts sehen allerdings valide aus. Das Messkonzept zeigt sich jedoch als überaus Funktionsfähig.

Die Wahl für Django als Webserver erwies sich als richtige Entscheidung. Das Projekt ist online sehr gut dokumentiert und dank der Programmierapache Python in wenigen Tagen verständlich erlernt. Zudem ist der Webserver sehr schnell eingerichtet und dank der guten Skalierbarkeit, erweiterbar auf die gewünschte Anwendung. Falls doch Probleme auftreten ist die Community des Django Projektes sehr gross und man erhält schnell Hilfe für ein Problem.

6.2 Verifizierung

Die Messwerte korrelieren mit denjenigen des Radio-Test-Examples, welches einen ähnlichen Zweck erfüllt [2]. Die Messresultate sind jedoch stark von äusseren Störeinflüssen abhängig. Somit wird eine exakte Verifizierung fast unmöglich. Zusätzlich müssten die Messwerte mit einem Messgerät kontrolliert werden. Da so ein Messgerät nicht vorhanden ist konnte die Verifizierung nur so, wie zu Beginn des Abschnittes erwähnt, durchgeführt werden.

6.3 Fazit

Firmware

Die Analyse des MAC-Layers und die Umsetzung der Testinfrastruktur hat viele Erkenntnisse im Bereich der hardwarenahen Programmierung gebracht. Durch die Erarbeitung der Firmware sind viele Module (Zeitsynchronisation, Statemachine, CLI, etc.) geschaffen worden, welche in weitere Programme integrierbar sind. Somit ergab sich eine optimale Grundlage, um in die Netzwerk-Stacks, welche auf dem MAC-Layer basieren, einzutauchen.

Webserver

Die Entwicklung eines Webservers für die Bedienung der P2P-Testinfrastruktur war eine sehr lehrreiche Erfahrung. Bis anhin waren nur die Programmiersprachen C und C++ bekannt. Das Erlernen und Anwenden von Python, HTML und Javascript für den Aufbau und die Darstellung des Webservers waren sehr intensiv aber auch lehrreich. Gewisse Funktionen vom Webserver könnten verbessert werden. Dazu gehören folgende:

- Das Aktualisieren der Tabellen und der Charts wird im Sekundentakt gemacht. Es wäre besser die Aktualisierung auszuführen sobald der Master Node meldet, dass er mit dem Report fertig ist.
- Das Aktualisieren der Charts geschieht, indem die ganze Seite neu geladen wird. Die einzelnen Charts neu zu zeichnen, ohne die Seite neu zu laden, wäre effektiver.
- Die Auswahl des COM-Ports ist fix hinterlegt. Eine dynamische Auswahl, die zeigt welche Geräte angeschlossen sind, wäre in der Bedienung komfortabler.

Teil II

Mesh Benchmark - Konzept und Umsetzung

7 Benchmark Konzept

Um die Performance der drei Mesh Stacks zu vergleichen, wurde ein einheitliches Benchmark Konzept erarbeitet. Dieses definiert die Mesh Parameter, Testumgebungen, den Ablauf sowie sämtliche Messgrößen und Messreihen. Nachfolgend wird detailliert auf dieses Konzept eingegangen.

7.1 Konzeptschema

Für den Vergleich der 3 Mesh Netzwerkstacks Bluetooth Mesh (BT Mesh), Thread und Zigbee wird ein vom Mesh Protokoll unabhängiges Testkonzept umgesetzt, welches in Abbildung 7.1 als Konzeptschema dargestellt ist. Die Benchmark Slave Nodes (BSN), in der Abbildung als Sensoren und Aktoren mit unterschiedlichen Funktionalitäten dargestellt, bilden zusammen mit dem Benchmark Master Node (BMN) das zu testende Mesh Netzwerk. Innerhalb des Netzwerks wird dessen Organisation vom jeweiligen Protokoll sichergestellt. Das Testnetzwerk bildet ein realitätsnahes Netzwerk nach. Eine Heimautomation in einem Einfamilienhaus oder einer Wohnung wird als Referenz angenommen, in welchem jeweils nur gewisse Nodes untereinander Applikationsdaten austauschen. Lichtschalter kommunizieren nur mit Lichtquellen und umgekehrt. Sie tauschen jedoch keine Daten mit Temperatursensoren aus. Diese unterschiedlichen Beziehungen innerhalb des Mesh Netzwerks sind in der Abbildung 7.1 angedeutet.

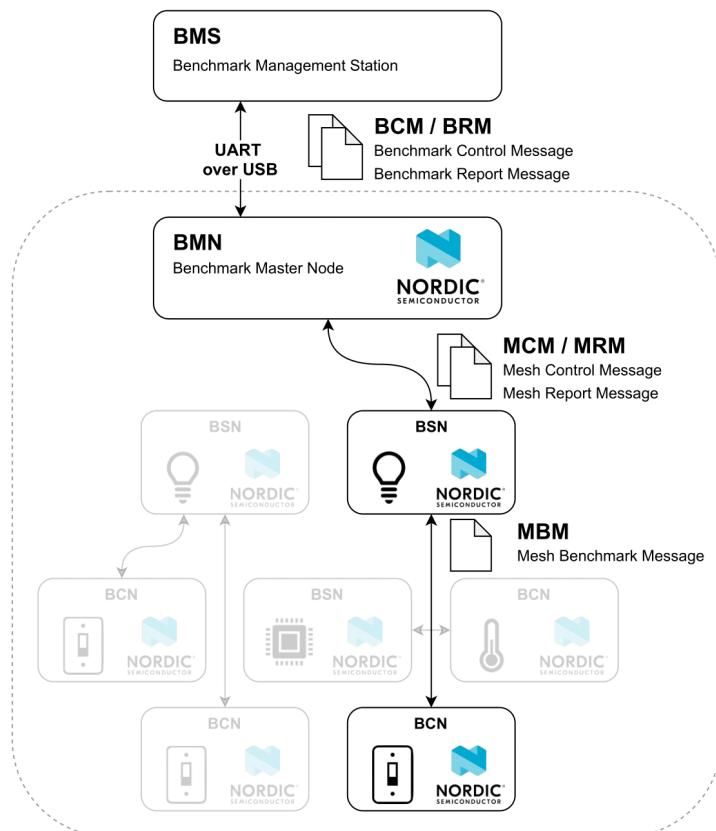


Abbildung 7.1: Konzeptschema für den Ablauf eines Mesh Benchmarks.

Die Benchmark Management Station (BMS), welche mit dem BMN via USB/UART kommuniziert, ist zuständig für die Verwaltung und Verarbeitung des Benchmarks. Während eines Benchmark Prozesses werden jedoch sämtliche Messungen unabhängig von der BMS durch-

geführt werden, damit allfällige Latenzzeiten der USB/UART Verbindung die Resultate nicht verfälschen.

7.1.1 Messages

In Abbildung 7.1 sind verschiedene Messages dargestellt. Dabei handelt es sich um die Nachrichten, die zwischen den einzelnen Teilen des Testaufbaus versendet werden und schliesslich einen Benchmark ausmachen. Die Messages besitzen folgende Funktionen:

Mesh Benchmark Message (MBM)

Die MBM ist jene Message, welche die eigentlichen Messdaten produziert und diese sogleich unter den BSN (Mesh Knoten) überträgt. Anhand dieser Messages werden die Parameter für den Vergleich der Protokolle gemäss Abschnitt 7.4 erfasst. Bei den MBM handelt es sich folglich um eine Sammlung von Messages, welche je nach gewünschtem Messwert bezüglich ihrer Form und Anzahl unterschiedlich ausfallen können.

Mesh Control Message (MCM)

Die MCM beinhaltet die Parameter für die Benchmarks, welche vom BMN an alle BSN übertragen werden. Ausserdem werden damit Kontrollbefehle für die Benchmarks wie beispielsweise *Start/Stop* sowie *Laufzeit, Wiederholrate usw.* übertragen.

Mesh Report Message (MRM)

Die MRM ist jene Message, welche die Messwerte von den BSN an den BMN übertragen. Gleichzeitig wird damit signalisiert, dass die Messung abgeschlossen ist und mögliche Fehler oder sonstige Status werden übermittelt.

Benchmark Control Message (BCM)

Die BCM beschreibt die Nachrichten, welche zur Steuerung eines Benchmarks durch das BMS dienen. Dabei handelt es sich um Befehle wie beispielsweise *Start/Stop*. Die BCM werden via serieller USB-UART Schnittstelle von der BMS auf den BMN übertragen.

Benchmark Report Message (BRM)

Die BRM beschreiben Nachrichten, welche den Status oder die Ergebnisse eines Benchmarks aus dem Mesh zurück an die BMS melden. Sie werden vom BMN initiiert und gelangen über die selbe USB-UART Schnittstelle, wie die BCM zum BMS. Die BRM wird erst nach Abschluss des Benchmarks initiiert. Zuvor werden die Messdaten auf dem BMN zwischengespeichert.

7.1.2 Nodes

Wie bereits angedeutet und in Abbildung 7.1 gezeigt, kann im Mesh Benchmark zwischen den folgenden 3 Node Typen unterschieden werden:

Benchmark Master Node (BMN)

Der Benchmark Master Node bildet den zentralen Zugriffspunkt zum Mesh Netzwerk für den Benchmark. Über ihn werden Control und Report Messages versendet und empfangen. Im Zigbee Mesh Protokoll fungiert er zugleich Coordinator (siehe Abschnitt 19.1).

Benchmark Client Node (BCN)

Der Benchmark Client Node repräsentiert beispielsweise einen Schalter in einer Lichtsteuerung. Im Benchmark Kontext ist er jene Instanz, die MBM versendet.

Benchmark Server Node (BSN)

Das Pendant zum BCN stellt der BSN dar. Er steht beispielsweise für eine Lichtquelle. Im Benchmark Kontext empfängt er die MBM, die vom BCN versendet werden.

7.2 Testumgebungen und Messaufbau

Unterschiedliche Testumgebungen sollen die Benchmarks und schliesslich den Vergleich der 3 Mesh Protokolle aussagekräftiger machen. Nachfolgende Umgebungen mit den entsprechenden Eigenschaften werden getestet. Die Abbildungen zu den Testumgebungen zeigen jeweils die Platzierung der Nodes sowie deren Funktion und Gruppenzugehörigkeit. Die Farbe Grün identifiziert den Node als Client Node während Blau für einen Server Nodes steht. Die Nummerierung zeigt, welcher Node zu welcher Adressgruppe gehört. Ein Client Node in Gruppe 1 sendet jeweils Nachrichten zu allen Server Nodes in der selben Gruppe.

Messumgebung 1: Labor

Der Laboraufbau ist ein Extremtest, welcher die Leistungsgrenzen der Protokollstacks ausloten soll. Dabei werden die Nodes auf einem Raster gemäss Abbildung 7.2 angeordnet. Die genauen Abmessungen sind dieser Abbildung zu entnehmen. Folgende Eigenschaften deckt diese Messung ab:

- Testaufbau unter Laborbedingungen auf engstem Raum.
- Ausgeglichene Anzahl Sensoren und Aktoren.
- Sehr hohe Node-Dichte.
- Geringe bis keine Störbeeinflussung durch die Umgebung zu erwarten.
- Die Mesh-Beziehungen werden künstlich bestimmt, sodass einfache Punkt zu Punkt Verbindungen mit oder ohne Hop entstehen.

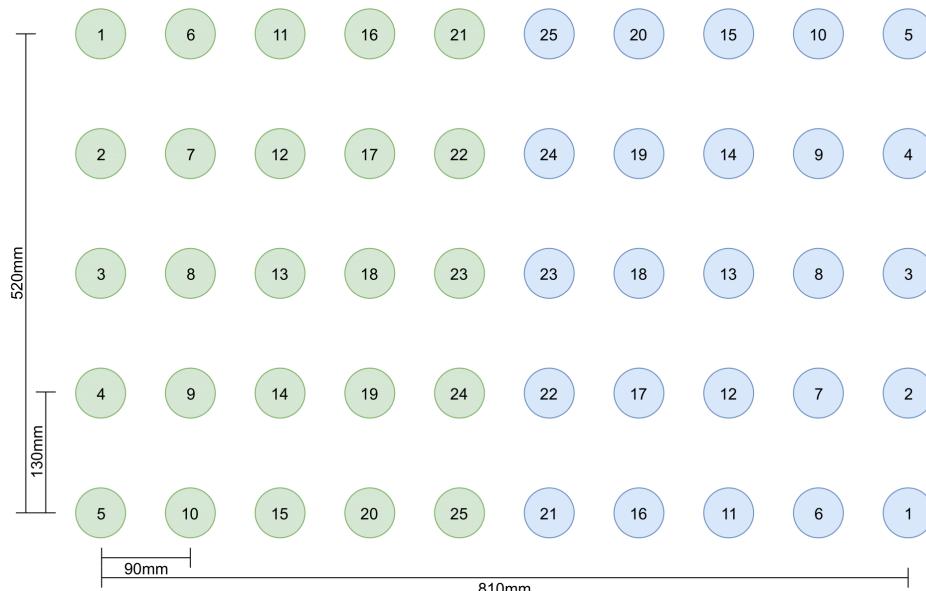


Abbildung 7.2: Anordnung Messumgebung 1: Labor

Messumgebung 2: Einfamilienhaus

Die Testgeräte werden in einem Einfamilienhaus installiert und repräsentieren damit eine flächendeckende Heim-Automatisierung. Folgende Eigenschaften deckt diese Messung ab:

- Einfamilienhaus über mehrere Etagen.
- Node-Dichte relativ gering.
- Kleine Beeinflussung durch Nachbarsysteme sind zu erwarten.
- Die Client/Server Beziehungen werden durch die bestehende Infrastruktur bestimmt.

Abbildung 7.3 zeigt die Platzierung der Nodes auf den 4 Etagen des Einfamilienhauses.



Abbildung 7.3: Anordnung Messumgebung 2: Einfamilienhaus

Messumgebung 3: Wohnung

Ebenfalls als Heim-Automatisierung gedacht, werden die Messungen in einer Wohnung durchgeführt. Folgende Eigenschaften deckt diese Messung ab:

- Wohnung über eine Etage in einem Mehrfamilienhaus
- Die Anzahl Client und Server Nodes ist ungefähr ausgeglichen.
- Node-Dichte höher als im Haus.
- Mögliche Störeinflüsse durch andere Systeme von Nachbarn sind zu erwarten.
- Die Mesh-Beziehungen werden durch die bestehende Infrastruktur bestimmt.

Bei der Wohnung handelt es sich um eine 3.5 Zimmer Wohnung mit einer Wohnfläche von 122 Quadratmetern. Die genauen Abmessungen sowie die Platzierung der Nodes ist in Abbildung 7.4 zu sehen.

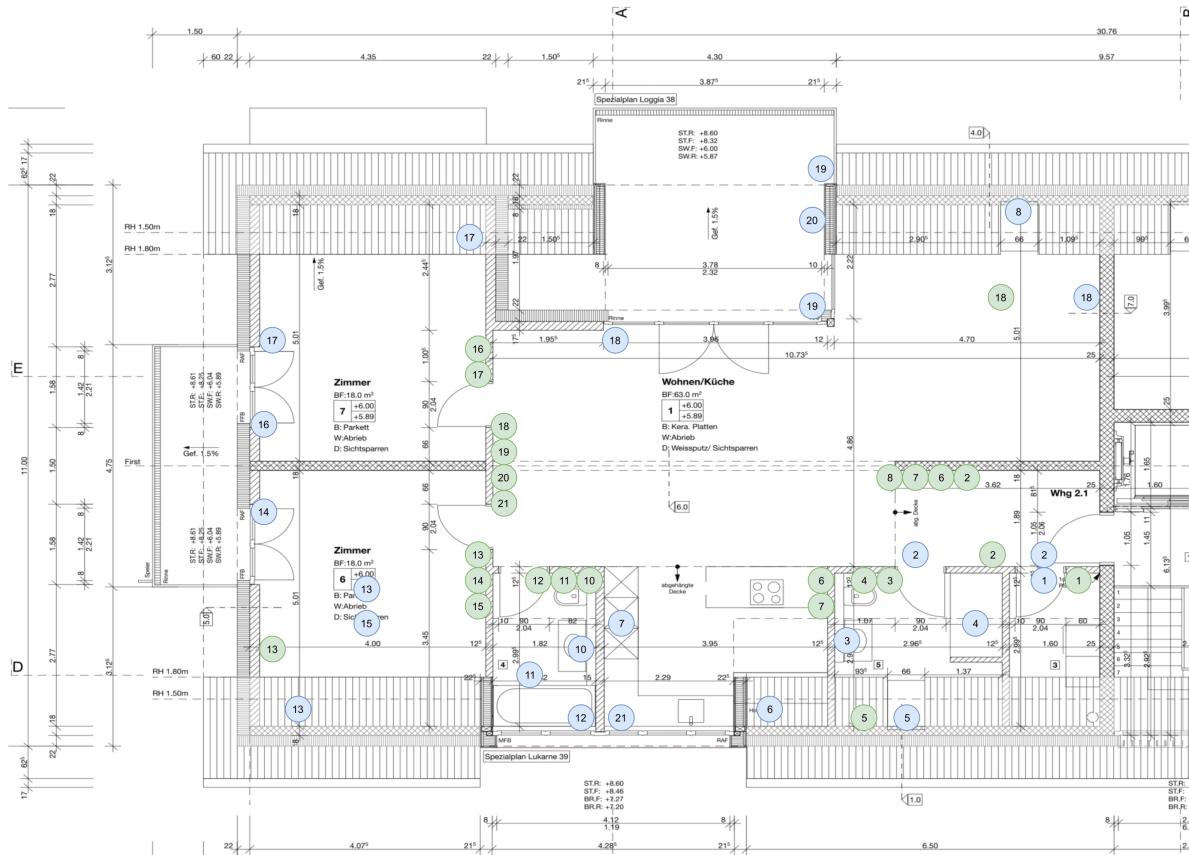


Abbildung 7.4: Anordnung Messumgebung 3: Wohnung

7.3 Ablauf Messvorgang

Ein Mesh Benchmark folgt einem klar definierten Ablauf. Abbildung 7.1 zeigt das Testkonzept in welchem auch der Ablauf eines Benchmarks bereits angedeutet ist. Folgende Schritte werden dabei in der entsprechenden Reihenfolge abgearbeitet.

1. Benchmark User-Init:

Im Config File auf dem BMS werden die gewünschten Parameter definiert, die Konfiguration verteilt und der Benchmark durch den Benutzer gestartet.

2. Benchmark Init BMN:

Die Parameter werden an den BMN übergeben, welcher diese wiederum an alle teilnehmenden BSN weiterleitet. Mit einem Startsignal vom BMN wird der Benchmark auf den BSN gestartet.

3. Benchmark Prozess:

Die BSN führen den Benchmark Prozess mit den definierten Parametern aus. Dies geschieht autonom und jeweils nur zwischen den entsprechenden BSN, die gemäss Benutzerkonfiguration in einer direkten Beziehung zueinander stehen. Die entstandenen Messdaten werden auf den BSN zwischengespeichert und nach Ablauf des Benchmarks ins Flash geschrieben.

4. Reporting:

Nach Ablauf der Benchmark Zeit werden die Messdaten an den BMN übertragen. Dies erfolgt gesteuert durch den BMN, welcher die Daten bei einem BSN nach dem anderen abfragt und direkt an das BMS weiterleitet.

5. Finish:

Der BMN kontrolliert, ob er die Daten von sämtlichen BSN korrekt auslesen konnte und bestätigt das Ende der Messung gegenüber dem BMS.

6. Auswertung:

Das BMS beendet den Benchmark Vorgang und speichert die Messdaten in seiner Datenbank ab. Von dort können die Daten gelesen und mit Excel ausgewertet werden.

7.4 Vergleichswerte und Messgrößen

Beim Vergleich der Mesh Protokolle wird zwischen Vergleichswerten und Messgrößen unterschieden. Vergleichswerte sind meist aus einer oder mehreren Messgrößen berechnete Werte, die einen aussagekräftigen Vergleich ermöglichen.

7.4.1 Vergleichswerte

Im Pflichtenheft zu dieser Arbeit (Anhang B) wurden die Testkriterien für den Vergleich bereits ausführlich behandelt. Folgende Vergleichswerte kommen nun in der Auswertung effektiv zum Tragen:

- **Latenzzeit:** Bestimmung der Latenzzeit eines Pakets das über das Mesh vom Client zum Server versendet wird.
- **Anzahl Hops:** Bestimmung der Anzahl Hops, über welche eine Nachricht übermittelt wird.
- **Datenrate:** Bestimmung der Datenrate in Bytes/s, welche zwischen zwei Teilnehmern erreicht wurde.
- **Paketverlust:** Pakete, welche ihr Ziel nicht erreichen konnten, werden gezählt, um so eine Paketverlustrate zu ermitteln.
- **Aktive Radio Zeit:** Zur Abschätzung des Energieverbrauchs der Nodes im Mesh Betrieb, wird die Aktivdauer der Radio Schnittstelle gemessen.

7.4.2 Messgrößen

Um die oben erwähnten Vergleichswerte zu berechnen, sind spezifische Messgrößen gefragt, die auf den Nodes direkt erfasst werden können. Die untenstehende Auflistung zeigt die auf den Mesh Nodes erfassten Messgrößen, deren Generierung und wie diese in der Auswertung verwendet werden.

Message Timestamp

Zur Bestimmung der Latenzzeit eines Pakets, das über das Mesh Netzwerk versendet wird, wird beim Senden sowie beim Empfangen ein Message Timestamp erfasst. Die unter den Nodes synchronisierte Zeit gibt Aufschluss darüber, wie lange das Paket unterwegs war. Die Berechnung der Latenzzeit wird jedoch erst bei der Analyse auf dem BMS durchgeführt.

Number of Hops

Die oben erwähnte Latenzzeit ist in einem Mesh Netzwerk abhängig vom Weg, den ein Paket bei der Übermittlung genommen hat. Mit jedem Hop nimmt die Latenzzeit zu. Deshalb wird beim Server Node die Anzahl Hops, die das Paket genommen hat, aus dem Message Header ausgelesen und für die Bestimmung der Latenzzeit abgespeichert.

Source-, Destination-, Group-Address

Aufgrund der veränderlichen Beziehungen von Client und Server Nodes ist die Zuordnung von Quell- und Zieladresse eines Pakets ebenfalls nicht statisch. Daher werden auf allen Nodes die

Quell-, Ziel- und Gruppenadressen der Pakete ausgelesen. Somit kann das Benchmark Paket eindeutig identifiziert und in der Auswertung erfasst werden. Die Adressinformationen werden aus dem Message Header des Mesh Pakets ausgelesen oder im Falle des Client Nodes mit der Benchmark Control Message übermittelt.

Message ID

Als Payload im Mesh Paket versendet, identifiziert die Message ID das Paket zusammen mit den Adressinformationen eindeutig. So kann der Paketverlust wie auch die Latenzzeit erfasst werden.

RSSI

Der RSSI (Received Signal Strength Indication) wird auf dem Server Node auf MAC Ebene erfasst und zeigt die Empfangsqualität der Übertragung. Dies kann bei der Analyse des Benchmarks als weiterer Indikator für den Paketverlust und die Latenzzeit verwendet werden.

Aktive Radio Zeit

Die Zeit, in der die Radio Schnittstelle aktiv ist, wird auf dem Node direkt gestoppt und gespeichert.

7.5 Messreihe

Um die Mesh Protokolle einem möglichst fairen Vergleich zu unterziehen, werden pro Messumgebung eine Reihe an Messungen durchgeführt. Die Tabelle 7.1 zeigt Total 8 Messreihen mit den entsprechenden Parametern.

#	Benchmark Parameter					Messaufbau		
	Msg. Gen.	Duration	Msg. Cnt.	Payload	Disturbance	Labor	Haus	Wohnung
1	Rand	600s	60	Small	No	x	x	x
2	Seq	600s	60	Small	No	x	x	x
3	Rand	600s	60	Large	No	x	x	x
4	Seq	600s	60	Large	No	x	x	x
5	Rand	600s	600	Small	No	x	x	x
6	Rand	600s	60	Small	Yes	x		
7	Seq	750s	10	Small	No	x		
8	Seq	750s	10	Large	No	x		

Tabelle 7.1: Parameter Benchmark Messreihe

Die Erläuterungen über die Bedeutung der Parameter ist in folgender Übersicht 7.2 zu finden.

Parameter:	Gültige Werte:	Bedeutung:
Msg. Gen	Seq/Rand	Nachrichten Generierung Sequentiell oder Pseudozufällig gemäss Abschnitt 7.7.
Duration	Ganze Zahlen	Dauer eines Benchmarks angegeben in Sekunden.
Msg. Count	Ganze Zahlen	Anzahl der Nachrichten die pro Client Node versendet werden.
Payload	Small/Large	Grosse oder Kleine Payload gemäss Definition in Tabelle 7.3.
Disturbance	Yes/No	Gewollte Störung durch P2P Testinfrastruktur auf dem selben Channel.

Tabelle 7.2: Bedeutung Benchmark Parameter

Die Messungen 1 bis 5 werden in allen drei Messumgebungen durchgeführt. So werden die Unterschiede der Topologien ersichtlich. Die Messungen 6 bis 8 hingegen werden nur im Laboraufbau durchgeführt, damit allfällige externe Störeinflüsse ausgeschlossen werden können und das Mesh Netzwerk unter Extrembedingungen getestet werden kann.

Bei der Messung mit dem Index 6 handelt es sich um ein Benchmark, welcher mit der P2P Testinfrastruktur gestört wird. So soll die Störiimmunität des Mesh Stacks getestet werden. Die Messungen 7 und 8 wurden aufgrund der Erfahrungen, die mit den Resultaten der Messungen 1 bis 6 gesammelt werden konnten, nachträglich ergänzt. Die Dichte der Nachrichten ist bei diesen Messungen deutlich tiefer damit die Netzbelaistung ebenfalls kleiner ausfällt.

7.6 Allgemeine Benchmark Parameter

Die oben erwähnten Messreihen haben nebst den variablen Parametern einige allgemeingültige Benchmark Parameter, die in Tabelle 7.3 aufgeführt sind. Diese sind für sämtliche Messungen innerhalb des jeweiligen Mesh Protokolls gültig. Die ersten vier Parameter *Group addressing mode*, *Application Layer*, *MAC Layer* sowie *Mesh Node Type* sind protokollabhängig und deshalb bei allen drei Mesh Stacks unterschiedlich. Da sich diese Parameter innerhalb des jeweiligen Protokolls unterschiedlich umsetzen lassen, sind die gewählten Werte resp. Funktionen hier aufgelistet. Deren Funktionsweise wird in separaten Abschnitten noch detailliert behandelt.

Beim *Group addressing mode* wird zwischen Broadcast bei BT Mesh, Multicast bei Thread und Unicast bei Zigbee unterschieden. Gemäß Definition im Pflichtenheft soll jeweils eine Gruppenadressierung stattfinden. Zigbee bietet zwar eine solche Möglichkeit eines Multicasts, jedoch wird dieser schliesslich als Broadcast umgesetzt, was eine deutliche Limitierung der Performance verursacht. Aufgrund dieser Erfahrung und der Erkenntnis, dass marktübliche Lichtsteuerungen⁴ nur selten ein solches Multicast Prinzip verwenden, wird die Adressierung im Benchmark gerichtet (Unicast) vorgenommen (siehe auch Abschnitt 20.2.6). Das bedeutet, die Nachrichten werden nicht an eine Gruppenadresse sondern direkt an die jeweilige Node Adresse gesendet. Bei mehreren Servern innerhalb einer Gruppe werden also auch mehrere Nachrichten versendet.

	BT Mesh	Thread	Zigbee
Group addressing mode	Broadcast	Multicast	Unicast
Application Layer	Models	CoAP [15.1]	ZCL [19.2.3]
MAC-Layer	BLE 1 Mbit	IEEE 802.15.4	IEEE 802.15.4
Mesh Node Type	Relay	FTD [??]	Zigbee-Router [19.1]
Mesh Node Cnt.	50	50	50
Client/Server Verhältnis	25/25	25/25	25/25
Message Ack (Appl. Layer)	No	No	No
Payload Size Small (Byte)	8	8	8
Payload Size Large (Byte)	32	50	50

Tabelle 7.3: Allgemeine Benchmark Parameter

In den Punkten *Mesh Node Cnt.*, *Client/Server Verhältnis*, *Message Ack* sowie *Payload Size Small* unterscheiden sich die Benchmarks der drei Protokolle nicht. Sämtliche Benchmarks werden mit 50 Mesh Nodes durchgeführt. Davon sind jeweils gleich viele Client Nodes wie Server Nodes (siehe Abschnitt 7.1.2). Die Benchmark Nachrichten werden auf dem Application Layer *unacknowledged* versendet. Wenn also ein Paket sein Ziel nicht erreicht, wird dies vom Sender nicht registriert und das Paket wird nicht erneut übertragen.

Die *Payload Size Small* mit der Grösse von 8 Byte soll eine kleine Nachricht repräsentieren, wie sie beispielsweise beim Dimmen einer Lampe versendet wird. Mit der *Payload Size Large* hingegen soll eine vergleichsweise grosse Payload simuliert werden, die über das Mesh übertragen wird. Die beiden Protokollstacks Thread und Zigbee erlauben dank *IEEE 802.15.4* eine totale

⁴Analyse mittels Sniffer Trace einer IKEA Tradfri Lichtsteuerung.

Framegrösse von 127 Byte. Abzüglich der Grösse der Header bieten die Protokolle noch Platz für eine Payload von ca. 50 Byte. Je nach Implementation der Protokolle variiert die Grösse der Header und die Payload kann entsprechend erhöht werden. BT Mesh hingegen beginnt bereits bei einer Payload Grösse von 8 Byte mit der Fragmentierung. Somit werden bei den 32 Byte für die *Payload Size Large* bereits 4 Frames versendet. Diverse Tests haben ergeben, dass hier der BT Mesh Stack an seine Leistungsgrenze stösst. Aus diesem Grund wäre eine weitere Erhöhung der Payload nicht sinnvoll.

7.7 Traffic Generation

Die in Tabelle 7.2 erwähnte Message Generation ist im Benchmark verantwortlich für die Verteilung der Benchmark Nachrichten über die Nodes in Abhängigkeit der Zeit. So wird bestimmt, wie der Traffic innerhalb des Netzes sowie auf den einzelnen Nodes generiert wird. Dabei werden die folgenden zwei Modi unterschieden:

Random

Im Modus *Random* werden die Zeitpunkte für den Versand einer Nachricht sogenannt *pseudozufällig* generiert. Dies bedeutet, dass jedem Client Node eine Liste von einmalig zufällig generierten Werten zugewiesen wird. Aus dieser Liste werden nur so viele Werte gelesen, wie es Nachrichten zu versenden gilt. Diese werden schliesslich nach Grösse sortiert und über den gesamten Benchmark Zeitraum gelegt, woraus nun die Zeitpunkte für den Versand der Nachrichten entstehen. Eine solche Methode erlaubt, die Zeitpunkte reproduzierbar zu machen und identisch auf alle 3 Mesh Stacks anzuwenden.

Sequentiell

Mit der *Random* Methode kann die Belastung eines Nodes sowie des gesamten Netzes zwischenzeitlich stark ansteigen. Um dies zu verhindern, kann der Traffic Generation Modus *Sequentiell* gewählt werden. Dabei werden die Nachrichten in regelmässigen Zeitschlitten gemäss Formel 7.1 pro Node versendet. Die Reihenfolge der Nodes bleibt jeweils die selbe.

$$T_{Msg_send} = \frac{T_{Bench_Duration}}{(Msg_Cnt \cdot 25)} \cdot NodeID \cdot i_{Msg} \quad (7.1)$$

T_{Msg_send}	Zeitpunkt zu welchem die Nachricht gesendet wird.
$T_{Bench_Duration}$	Benchmark Dauer (siehe Tabelle 7.1)
Msg_Cnt	Anzahl Nachrichten die versendet werden.
$NodeID$	Identifikationsnummer des Nodes.
i_{Msg}	Message Index

7.8 Messdatenerfassung und Auswertung

Für die Messdatenerfassung und Auswertung wird keine dedizierte Hardware verwendet. Die eigentlichen Messwerte, die in Abschnitt 7.4 erläutert sind, werden auf den Mesh Nodes direkt erfasst und erst im RAM und später im Flash des Microcontrollers zwischengespeichert. Von dort werden sie mittels BRM (Benchmark Report Message) an den BMN übertragen. Auf den Mesh Nodes inkl. BMN findet noch keine Auswertung der Daten statt. Diese wird an das BMS ausgelagert. Das BMS empfängt die BRM via CLI (Command Line Interface) und speichert die Messwerte in einem Excel CSV File ab. Hierfür kommt ein Python Skript zum Einsatz (siehe Abschnitt 9.4.3). Mittels einem weiteren Python Skript und manueller Bearbeitung in Excel können die so erfassten Daten ausgewertet werden.

7.8.1 Identifizierung Benchmark Slave Node

Die verschiedenen Benchmark Slave Nodes (BSN) werden mithilfe ihrer MAC-Adresse identifiziert. Jedem nRF52840 SoC wird während der Produktion eine 48-bit lange Adresse eingebrannt. Dabei handelt es sich um zufällig generierte Werte. Zur Identifizierung im Benchmark werden nur die letzten 32-bit dieser MAC-Adresse verwendet. Auf dem nRF-52840-Dongle ist die Adresse auf der Rückseite angegeben (siehe Abbildung 7.5) und kann dadurch einfach abgelesen werden.



Abbildung 7.5: MAC Adresse nRF52840-Dongle

7.8.2 Konfiguration Benchmark Slave Node

Zur Konfiguration der BSN dient ein Excel-File. Darin sind alle Nodes mit ihren Konfigurations-Parametern aufgelistet. Die folgenden Angaben sind in einem Konfigurations-Eintrag enthalten.:

- **Nummer:** Gibt die Nummer des zu konfigurierenden Teilnehmers an.
- **Firmware:** Verweis auf das Firmware-File, mit welchem der Teilnehmer geflasht wird.
- **Dev ID:** Gibt die letzten 32bit der MAC-Adresse des zu konfigurierenden Teilnehmers an (siehe Abschnitt 7.8.1).
- **Group ID:** Index der Gruppe, zu der der Teilnehmer gehört. (siehe Abschnitt 7.2)
- **Node ID:** Die Node ID wird nur bei Clients benötigt. Dieses Feld gibt an, welche pseudofälligen Werte im Traffic Generation Modus Random (siehe Abschnitt 7.7) benutzt werden.
- **Ack:** Versand von Acknowledged Messages ein oder ausschalten.
- **Add Payload Len:** Gibt die Anzahl Bytes an, welche zusätzlich zu jeder Nachricht angehängt werden, um die Gesamtgrösse des Pakets zu verändern.
- **Traffic Gen Mode:** Gibt die Art an, wie der Traffic generiert wird (siehe Abschnitt 7.7).
- **DST_Node_1-3:** Gibt die Nummer des Servers an, welcher im Unicast Addressing Mode als Ziel verwendet wird. Es können Total drei Einträge definiert werden.

- **DST_MAC_1-3:** Gibt die MAC-Adresse des Servers an, welcher im Unicast Addressing Mode als Ziel verwendet wird. Diese Felder sind von den Feldern *DST_Node_1-3* abhängig.

Mithilfe eines CLI Befehls können die einzelnen Teilnehmer konfiguriert werden (siehe Abschnitt 9.3.7). Mithilfe des Python Scripts aus Abschnitt 9.4.2, wurde dieser Vorgang automatisiert.

7.8.3 Starten einer Messung

Beim Starten einer Messung über das CLI müssen die folgenden Parameter dem Startbefehl mitgegeben werden:

- **BenchmarkTime** Die Zeit wie lange der Benchmark dauert, in Sekunden.
- **BenchmarkMsgCnt** Die Anzahl Nachrichten, welche pro Client versendet werden sollen.

Die Eingabe des Befehls erfolgt gemäss Abschnitt 9.3.7.

7.8.4 Einholen der Messdaten

Mithilfe eines CLI-Befehls aus Abschnitt 9.3.7 können die Messdaten eingeholt werden. Durch das Python Script aus Abschnitt 9.4.3 wurde dieser Vorgang automatisiert.

7.9 Messerwartung

Wie in der Übersicht 2.2 bereits erwähnt, konnte im Rahmen der Vorarbeiten zu dieser Thesis der BT Mesh Stack bereits vertieft untersucht und erste Erfahrungen damit gesammelt werden. Bei den beiden anderen Stacks, Thread und Zigbee, musste dies erst noch erfolgen. Bei diesen Vorarbeiten konnten bereits deutliche Unterschiede zwischen den Stacks beobachtet werden, hauptsächlich verursacht durch das Routing bei Thread und Zigbee sowie das Flooding-Mesh Prinzip bei BT Mesh. Aufgrund dieser Erfahrungen wird erwartet, dass die beiden auf dem *IEEE 802.15.4* Standard aufbauenden Protokolle klare Vorteile haben werden, da die Nachrichtenzustellung durch das Routing effizienter realisiert werden kann. Hingegen könnte BT Mesh Vorteile haben wenn das Netz weniger stark belastet wird.

Das menschliche Auge ist in der Lage eine Verzögerung festzustellen, wenn die Latenz vom Knopfdruck bis das Licht angeht mehr als 200ms beträgt. In einer modernen Hausautomation darf keine solche Verzögerung wahrgenommen werden. Daher wird erwartet, dass die Latenzzeiten im Schnitt unter 200ms bleiben werden [11].

Robin: Durchlesen und Ergänzen.

8 Hardware Plattform

Für den Benchmark und den Vergleich der drei Mesh Protokolle sowie die P2P Testinfrastruktur wird eine einheitliche Hardwareplattform verwendet. Dies macht ein Vergleich der Testresultate möglich. Im Zentrum steht dabei der nRF52840 SoC von Nordic Semiconductor, welcher eine vollumfängliche Plattform für *Bluetooth* und *IEEE 802.15.4* darstellt. Damit die Hardware einfach nutzbar ist, werden Entwicklungsboards vom Chip Hersteller eingesetzt, welche vergleichsweise günstig und gut verfügbar sind. Nachfolgend wird auf die Hardware und deren Einsatz genauer eingegangen.

8.1 System on Chip (SoC)

Der nRF52840 ist ein Highend System-on-Chip, welcher gleichermaßen *Bluetooth 5* und den *IEEE 802.15.4* Wireless Personal Area Network (WPAN) Standard unterstützt, auf Wunsch sogar als Multiprotocol Lösung. Entscheidend für die vorliegende Anwendung ist, dass der SoC von den drei Organisationen *Bluetooth SIG*, *OpenThread* und der *Zigbee Alliance* offiziell für das jeweilige Protokoll zertifiziert ist. Die Eckdaten des nRF52840 SoC sind in Tabelle 8.1 kurz zusammengefasst.

Microcontroller	32-bit ARM Cortex-M4F
Clock frequency	64 MHz
Flash	1MB
RAM	256kB
Frequency Band	2.4GHz
Wireless protocols	IEEE 802.15.4 (Thread, Zigbee), Bluetooth 5.2
Operating voltage	3V
TX only run current	4.8mA

Tabelle 8.1: Eigenschaften nRF52840 SoC [12][13]

8.2 Hardware Development Kits

Der Hersteller Nordic Semiconductor bietet gleich zwei Entwicklungsboards für den SoC an. Einerseits handelt es sich um das umfangreiche nRF52840-DK (pca10056) (siehe Abbildung 8.1), welches zusammen mit den Software Development Kits von Nordic Semiconductor (siehe Abschnitt 9.1) eine komplette Entwicklungsumgebung bietet. Auf dem Board ist nebst diverser Peripherie auch ein Segger J-Link Debugger integriert. Dieser erleichtert das flashen und debuggen enorm. Im Benchmark Betrieb fungiert ein nRF52840-DK als Benchmark Master Node wobei die USB-UART Schnittstelle als Interface zur Benchmark Management Station verwendet wird. Als einfacher Alternative zum vollumfänglichen Development Kit werden als Benchmark Slave Nodes 7.1.2 die einfacheren und günstigeren nRF52840-Dongle (pca10059) (siehe Abbildung 8.2) verwendet. Offensichtlich als USB-Dongle entworfen, können die Dongles flexibel eingesetzt werden. Die fehlende Peripherie macht die Hardware jedoch unpassend für die Entwicklung neuer Firmware. Beide Typen von Entwicklungsboards besitzen eine integrierte PCB Antenne, wenn auch in unterschiedlichen Ausführungen. Im Abschnitt 8.4 wird näher auf die Ausführung der Antennen und deren Strahlungscharakteristik eingegangen.

8.3 Aufbau Testnode

Der Aufbau des Testnodes wie er in Abbildung 8.3 abgebildet ist, wurde so einfach wie möglich gehalten. Ein externes Batteriepaket wird eingesetzt, damit der Benchmark Slave Node unab-

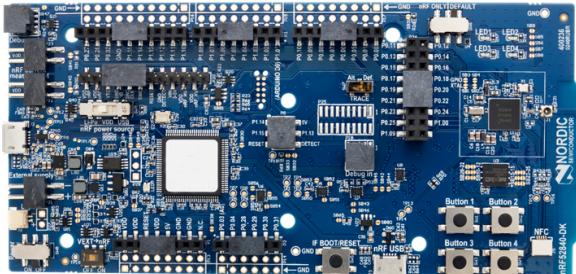


Abbildung 8.1: PCA10056 [14]

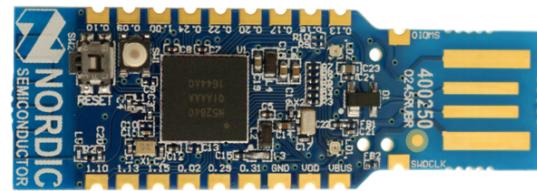


Abbildung 8.2: PCA10059 [15]

hängig von einer externen Stromversorgung betrieben werden kann. So kann der Einsatzort frei gewählt und auch kurzfristig geändert werden. Das Batteriepaket besteht aus zwei Standard Mignon Einwegzellen vom Typ AA mit je einer Spannung von 1.5V. Als Serieschaltung betrieben wird die Speisespannung von 3V gemäss der Vorgabe in Tabelle 8.1 erreicht. Mit fortschreitender Entladung der Zellen sinkt die Spannung. Da der SoC einen Betriebsspannungsbereich von 1.7V bis 5.5V hat, stellt die sinkende Spannung jedoch kein Problem dar. Die Speisung des nRF52840-Dongle erfolgt über einen 2-poligen JST Header, der gemäss Userguide [16] mit dem VDD OUT und GND Anschluss verbunden ist. Am VDD OUT Anschluss ist ein Betriebsspannungsbereich von 1.8V bis 3.6V zulässig. Das Batteriegehäuse verfügt außerdem über einen Schalter, um den Testnode nach Belieben ein- und auszuschalten. Mit den verwendeten Batterien, welche eine Kapazität von 2900mAh besitzen, können die Nodes im Mesh Benchmark Betrieb über mehrere Tage betrieben werden [17]. Eine Messung des Stromverbrauches im Benchmark Betrieb hat gezeigt, dass dieser bei 3V Batteriespannung durchschnittlich 15mA beträgt. Dieser relativ hohe Wert wird einerseits verursacht durch die LED's, die geschaltet werden, und andererseits durch das Radio Interface welches im Leerlauf permanent im Empfangsmodus ist, um allfällige Start Messages empfangen zu können. Dieser Verbrauch ermöglicht eine Betriebsdauer von rund 190 Stunden, was mehr als ausreichend ist für diese Verwendung.



Abbildung 8.3: Testnode: nRF52840-Dongle inkl. Batteriepack

Die Befestigung des Dongles am Batteriepaket erfolgt so, dass die PCB Antenne welche sich am Ende des PCB's befindet, möglichst frei liegt und sich dadurch die bestmöglichen Abstrahlungseigenschaften ergeben. Ein weiterer Vorteil der in Abbildung 8.3 dargestellten Montage ist die Zugänglichkeit des USB Ports. So können die Benchmark Nodes auf einfache Arte und Weise mit einer neuen Firmware geflasht werden.

8.4 Antenne und Strahlungscharakteristik

Wie bereits im Abschnitt 8.2 erwähnt, besitzen die beiden eingesetzten Development Kits jeweils eine PCB Antenne. Beim nRF52840-DK ist diese als einfache IFA (Inverted-F Antenna) gemäss Abbildung 8.4 ausgeführt, währenddessen der Dongle mangels Platz auf dem PCB über eine sogenannte MIFA (Meandered Inverted-F Antenna) gemäss Abbildung 8.5 verfügt.

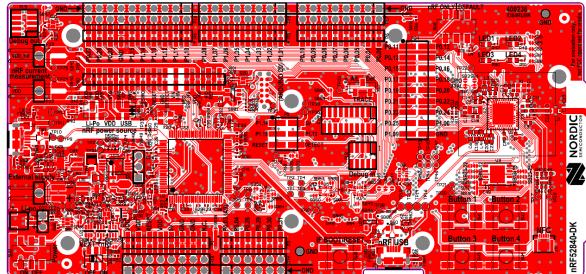


Abbildung 8.4: PCA10056 Copperplane [18]

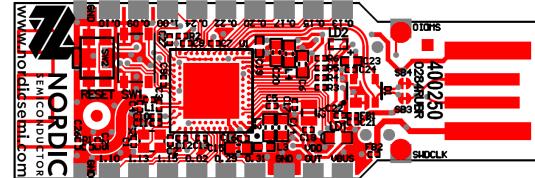


Abbildung 8.5: PCA10059 Copperplane [19]

Erste Versuche zu Beginn dieser Projektarbeit haben gezeigt, dass sich die Eigenschaften der beiden Antennen deutlich unterscheiden. Beispielweise konnte festgestellt werden, dass die Antenne des pca10056 eine deutlich höhere Reichweite erzielt als jene des pca10059. Um diese Vermutungen zu prüfen, wurde eine Spektrumsanalyse der gestrahlten Emmission an den beiden Entwicklungsboards durchgeführt. Diese Messungen wurden im Rahmen eines Projektes für das Modul emv durchgeführt und dort in einem Bericht dokumentiert. Nachfolgend werden die wichtigsten Erkenntnisse daraus nochmals wiedergegeben. Der ganze Bericht ist im Anhang C zu finden.

Bei den durchgeführten Messungen lag das Hauptaugenmerk auf der Erfassung der Abstrahlcharakteristik in dreidimensionaler Richtung. Die absoluten Signalpegel konnten dabei nicht korrekt erfasst werden, waren jedoch auch nicht von grosser Bedeutung. Die in den beiden Abbildungen 8.6 und 8.7 dargestellten Abstrahlcharakteristiken der jeweiligen Development Kits zeigen deutliche Unterschiede auf. Während das pca10056 eine sehr gleichmässige Verteilung der Sendeleistung über die gesamte Halbkugeloberfläche ausweist, zeigt das pca10059 deutliche Schwachstellen in Form von Dellen in der Halbkugel auf. Weiter unterscheidet sich der Maximalwert in Z-Richtung optisch zwar nur minimal, in Wirklichkeit bedeutet dies jedoch eine zweibis vierfach höhere Dämpfung und somit eine deutlich kleinere Reichweite.

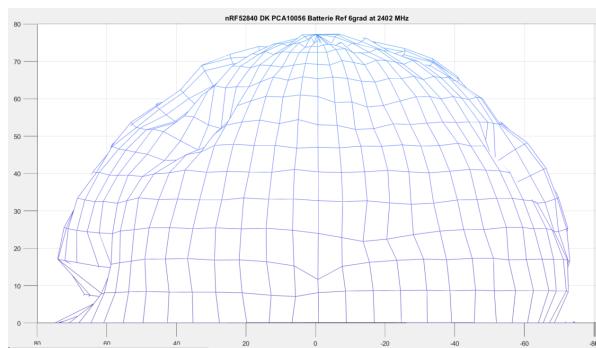


Abbildung 8.6: Abstrahlung in Z-Richtung
PCA10056

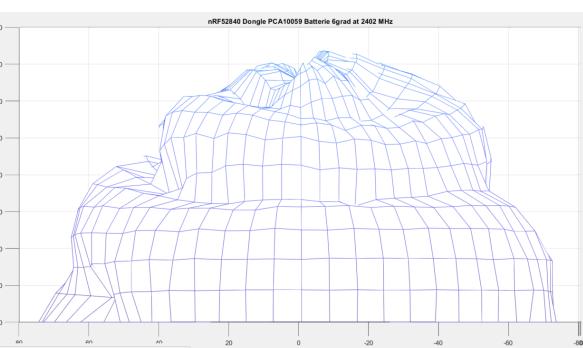


Abbildung 8.7: Abstrahlung in Z-Richtung
PCA10059

Die gemessenen Abstrahleigenschaften der beiden Antennen bestätigen schliesslich unsere Erfahrungen, die wir mit den Boards bereits gemacht haben. Die MIFA Antenne des pca10059 weist einige klare Nachteile gegenüber der IFA Antenne des pca10056 auf. Bei eingeschränkten

Platzverhältnissen, wie sie auf dem pca10059 und bei vielen Low-Power Mesh Anwendungen bestehen, ist eine MIFA eine solide Alternative.

Die Erkenntnisse aus den vorliegenden Messungen waren hilfreich bei der Planung und Durchführung der Mesh Benchmarks. Beim Aufbau der Testnodes 8.3 flossen diese beispielsweise ein um die Ausrichtung der Antenne zu optimieren. Weiter halfen die Ergebnisse bei der Interpretation der weiteren Messungen, da auch dort klar aufgezeigt werden konnte, dass die Positionierung und Ausrichtung der Antenne von entscheidender Bedeutung für die Performance ist.

9 Soft- und Firmware

Im folgenden Abschnitt wird das Konzept und die Umsetzung der Software für den Mesh Benchmark dokumentiert. Dabei stand die Erfüllung der Anforderungen an die Software im Zentrum. Diese wurden in der Aufgabenstellung (siehe Anhang A) sowie im Pflichtenheft (siehe Anhang B) zu dieser Arbeit vorgängig definiert.

9.1 Konzept

Um das Benchmark Konzept, welches bereits im Abschnitt 7 erläutert wurde, umsetzen zu können, soll die Soft- und Firmware so aufgebaut sein, dass sie von allen Mesh-Stacks genutzt werden kann. Zur Umsetzung der drei Mesh-Stacks auf der *NRF-Plattform* wurden folgende *Software Development Kits (SDK's)* eingesetzt.

nRF Connect SDK	Die <i>nRF Connect SDK</i> baut auf dem Zephyr-RTOS auf und unterstützt alle drei Mesh-Protokolle (Bluetooth-Mesh, Openthread und Zigbee). Der Einsatz der SDK in einem End-Produkt wird jedoch noch nicht empfohlen. [20]
nRF5 SDK for Thread and Zigbee	Die <i>nRF5 SDK for Thread and Zigbee</i> ist eine Software Bibliothek für die Entwicklung von Thread und Zigbee zertifizierten Produkten. [21]
nRF5 SDK for Mesh	Die <i>nRF5 SDK for Mesh</i> ist eine Software Bibliothek für die Entwicklung von Bluetooth Mesh Lösungen. [22]

Tabelle 9.1: Eingesetzte SDK's für die Umsetzung der Benchmark Firmware

Die Messungen sollten trotz der verschiedenen Bibliotheken möglichst einheitlich sein, um die Vergleichbarkeit nicht zu beeinträchtigen. Daher wurde für das Benchmark Management eine eigene Kommunikationsebene entwickelt. Diverse Module der P2P-Testinfrastruktur konnten dafür direkt oder indirekt verwendet werden.

9.2 Umsetzung

Die Soft- und Firmware für die Mesh Benchmarks setzt sich aus mehreren Modulen zusammen. Tabelle 9.2 zeigt eine Übersicht der wichtigsten Module und deren Verwendung in den 3 Benchmark Teilen: BLE Mesh, Thread und Zigbee. Gemeinsame Module auf Firmware Ebene sind im Ordner SharedLib im Github Repository⁵ zusammengefasst.

Jene Python Scripts für das Benchmark Management stehen im gleichnamigen Ordner ebenso auf dem Github Repository⁶ zur Verfügung. Die Stack bezogenen Implementationen der Benchmark Firmware werden in den individuellen Teilen III, IV und V behandelt.

Library	Funktion	Modul	Referenz
SharedLib	Statemachine	bm_statemachine.c	9.3.1
	Timesync	bm_timesync.c	9.3.2
	Control	bm_control.c	9.3.3
	Report	bm_report.c	9.3.4
	Logging	bm_log.c	9.3.5
	Flash Save	bm_flash_save.c	9.3.6
	CLI	bm_cli.c	9.3.7
	Low Layer Radio	bm_radio.c	9.3.9
	Random and Sequential Traffic Generator	bm_rand.c	9.3.8
Benchmark Management	Flash	Flasher.py	9.4.1
	Configurator	Configurator.py	9.4.2
	Benchmark and Reporter	Benchmark_and_Reporte.py	9.4.3
BT Mesh	Stack Init and Configuration	bm_blemesh.c	12
	Model Handling	bm_blemesh_model_handler.c	12
	Stack Configuration	bm_config.h	12
Thread	Stack-Handling	bm_ot.c	16.1
	Stack Configuration	bm_config.h	16.1
Zigbee	Stack Handling	bm_zigbee.c	20.3
	Stack Configuration	bm_config.h	20.3

Tabelle 9.2: Übersicht Shared Lib und Benchmark Management Module

9.3 Shared Library

Die *Shared Library* ist die geteilte Bibliothek zwischen allen Mesh-Stacks. Sie enthält alle relevanten Module, um eine Messung zu verwalten und zu steuern. Der jeweilige Mesh-Stack ist lediglich für das Senden und Empfangen von Nachrichten sowie für die Erfassung der Messdaten zuständig.

Abbildung 9.1 zeigt vereinfacht die Schnittstellen zwischen der Shared-Library und den Mesh-Stacks. Die Bibliothek ist spezifisch auf die nRF52840 SoC's zugeschnitten. Damit die *Shared-Lib*

⁶https://github.com/Rouben94/P6_Software [23]

in die jeweilige SDK integrierbar ist, müssen die enthaltenen Module möglichst ohne weitere Abhängigkeiten auskommen. Sofern Treiber oder externe Bibliotheken für Module benötigt werden, müssen diese von allen SDKs vergleichbar zur Verfügung gestellt werden. Im folgenden Abschnitt wird auf die einzelnen Module der *Shared-Lib* eingegangen.

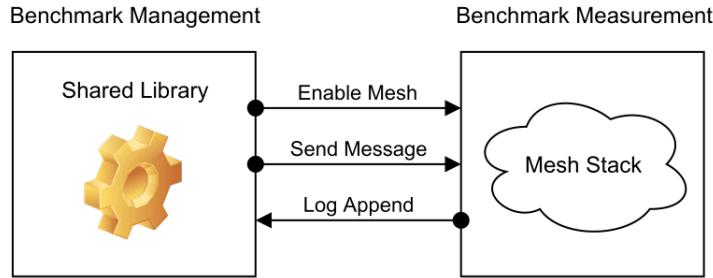


Abbildung 9.1: Vereinfachte Anbindung der *Shared Library* an die Mesh-Stacks

9.3.1 Statemachine

Die State machine dient dazu, den Ablauf, welcher in Abschnitt 7.3 bereits beschrieben wurde, abzuarbeiten. Sowohl der Master als auch die Server und Clients arbeiten den in Abbildung 9.2 dargestellten Ablauf ab.

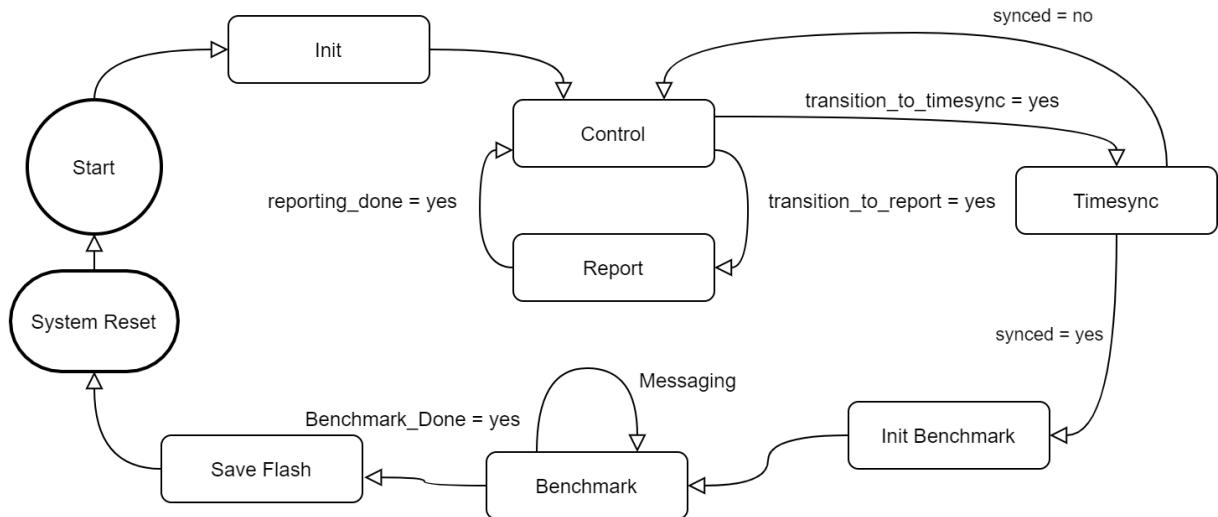


Abbildung 9.2: Flowgraph der State machine

Die Funktionen der einzelnen Schritte werden nachfolgend grob beschrieben.

Init

Dieser Schritt initialisiert alle notwendige Funktionen (Radio, Timers, usw.). Ebenfalls rekonstruiert er die *Log-Daten*, indem er diese aus dem Flash ins RAM lädt. Nach der erfolgreichen Initialisierung wechselt die State machine in den Control-State.

Control

Im Control-State wartet jeder Teilnehmer auf eingehende Befehle des Benutzers. Der Master erhält über das Command Line Interface (CLI) den auszuführenden Befehl. Diesen wandelt der Master in eine Nachricht (Control-Message) um und leitet sie über das Radio-Interface an die Slaves weiter. Alle Slaves (Clients und Servers) erwarten eine Control-Nachricht über das Radio-Interface. Damit alle Clients und Servers die Nachricht empfangen können, wiederholt jeder

Slave jede Control-Message. Somit wird die Mesh-Fähigkeit der Verwaltungsschicht (Benchmark Management) sichergestellt. Folgende Transitionen sind vom Control-State zugelassen:

Transition to Timesync: Wird ein Benchmark gestartet erfolgt dies durch einen Wechsel in den Timesync-State. Jeder Teilnehmer wechselt nach der Wiederholung der Nachricht in den entsprechenden Schritt.

Transition to Report: Das Reporting wird über den Report-State abgehandelt. Nur der Teilnehmer, von welchem Reports angefordert wurden, wechselt in den Report-State.

Timesync

Der Master verteilt die Zeitsynchronisation an die Slaves. Die Slaves synchronisieren sich auf das Signal des Masters auf, sofern sie in seiner Reichweite liegen. Sobald ein Slave synchronisiert ist, beginnt er ebenfalls mit der Verteilung der Zeitsynchronisation. Dadurch können Slaves, welche nicht über einen Hop vom Master erreicht werden, sich ebenfalls synchronisieren. Konnte ein Slave keine Zeitsynchronisation durchführen, wechselt dieser in den *Control-State* zurück und bringt einen Fehler mittels der roten-LED zur Anzeige. Der Master wird immer zum nächsten Schritt voranschreiten.

Init Benchmark

Im Init Benchmark State werden die Mesh Stacks auf allen Teilnehmern initialisiert und anschliessend gestartet. Das Netzwerk hat nun Zeit sich aufzubauen, falls dies notwendig ist. Vor dem Initialisieren des Mesh-Stacks löscht jeder Node die Log-Daten aus dem Flash und dem RAM. Beim Client wird zusätzlich der Sendezeitpunkt der Benchmark-Nachrichten berechnet und vorbereitet. Hat sich der Slave erfolgreich initialisiert und mit dem Netzwerk verbunden, leuchtet das grüne Status-LED auf. Der Wechsel zum Benchmark State wird unabhängig vom Verbindungsstatus nach einer vordefinierten Verzögerungszeit ausgelöst.

Benchmark

Die Statemachine wartet in diesem Schritt auf die Auslösung von Events. Beim Benchmark-Server überlässt sie dem Mesh-Stack das Loggen der empfangenen Nachrichten, welcher über einen eigenen Event-Handler verfügt. Beim Client plant die Statemachine den Versand von Nachrichten mittels Timer-Interrupts. Löst ein solcher Timer-Interrupt aus, wird der Mesh-Stack unmittelbar benachrichtigt. Die Erfassung der Logdaten erfolgt ebenfalls im Stack. Der Status des einzelnen Slaves (Licht Ein / Aus) ist über die grüne RGB-LED (Benchmark-Client) respektive die blaue RGB-LED (Benchmark-Server) erkennbar. Nach dem ablaufen der eingestellten Benchmark-Zeit löst der letzte Timer-Interrupt aus und die Abarbeitung des Mesh-Stacks wird unterbrochen. Es folgt der Wechsel zum letzten Schritt.

Save Flash

Die erfassten Benchmark Logdaten werden vom RAM ins Flash geschrieben. Die Speicherung der Log-Daten im Flash ist notwendig, damit diese auch nach einem Neustart persistent bleiben. Anschliessend wird der Mesh-Stack durch einen Reset des Mikrocontrollers heruntergefahren und neu gestartet. Dies ist wiederum notwendig um sicherzustellen, dass der Stack komplett angehalten wurde und somit das Radio-Interface dem Benchmark Management zur Verfügung steht. Zudem soll ein neuer Benchmark ohne Vorbelastung gestartet werden können. Nach dem Reset beginnt die Statemachine automatisch wieder von vorne im Init-State.

Reports

Das Einholen von Reports ist nur über eine direkte Verbindung vom Master zum Slave möglich. Die Übertragung der Log-Einträge erfolgt über ein abgesichertes Handshake-Verfahren. Der Master sendet eine Anfrage und verlangt beim Slave den *n-ten* Log-Eintrag. Der Slave antwortet auf die Anfrage mit dem entsprechenden Log-Eintrag. Hat der Master diesen erhalten, so verlangt er den nächst höheren Eintrag. Falls der aktuelle Log-Eintrag nicht angekommen ist, verlangt der

Master den selben Einträge erneut. Ist die Übertragung zu oft fehlgeschlagen, wird das Reporting abgebrochen. Master wie auch Slave wechseln nach erfolgreichem oder abgebrochenem Reporting in den Control-State.

9.3.2 Timesync

Die Zeitsynchronisation wurde identisch wie im Point to Point Teil gemäss Abschnitt 5.4 umgesetzt. Durch die Wiederholung der Master-Timestamps bei jedem Hop nimmt der Synchronisationsfehler stetig zu. Angenommen pro Hop entsteht ein maximaler Fehler von $1.2\mu s$, so wäre der maximale Fehler nach sieben Hops bereits $8.4\mu s$ gross. Dieser Fehler ist noch immer vernachlässigbar klein verglichen mit dem Clock-Drift, der zusätzlich vorhanden ist. Die Genauigkeit des Quarz-Oszillators auf dem nRF52840 liegt bei $\pm 10\text{ppm}$ [13]. Das ergibt pro Sekunde ein maximalen Clock-Drift von $20\mu s$.

Um die Funktionen von Master und Slave zu unterscheiden, wurde eine Subscribe- sowie eine Publish-Funktion definiert. Der Master verteilt den Timestamp über die Publish-Funktion. Der Slave reagiert auf Timesync-Pakete mittels Subscribe Funktion. Erhält der Slave ein Timesync-Paket, wiederholt dieser das Frame mittels Publish-Funktion. Vor dem Weiterleiten mittels Publish muss ein zufälliger Timeslot abgewartet werden, damit es nicht zu Kollisionen zwischen den Timesync-Paketen kommt. Die maximale Backoff-Zeit wird gemäss Abschnitt 5.3 berechnet. Dementsprechend muss das Zeitfenster zur Synchronisation genügend lang sein, damit sich alle Teilnehmer synchronisieren können.

Ein Timesync-Paket setzt sich aus folgenden Feldern zusammen:

- *uint64_t LastTxTimestamp*: Gibt den Zeitstempel (Master-Zeit) an, welcher zum Ver sandzeitpunkt des Probepakets erfasst wurde.
- *uint64_t NextState_TS_us*: Definiert den Zeitpunkt bei dem zum nächsten Schritt gewechselt werden kann.
- *uint32_t MAC_Address LSB*: Identifiziert den Time-Masters. Diese MAC Adresse muss mit jener Adresse des zuvor empfangenen Pakets übereinstimmen.
- *uint8_t seq*: Aufsteigende Nummer um jedes Times-Sync Paket zu identifizieren. Der Slave prüft ob das eingehende Paket das direkt Folgende des zuvor erhaltenen Pakets ist.

9.3.3 Control

Im Control State befinden sich alle Teilnehmer in Bereitschaft wie bereits in Abschnitt 9.3.1 beschrieben. Analog zu den im Abschnitt 9.3.2 definierten Publish- und Subscribe-Funktionen wurden solche auch für die Verteilung der Control-Messages definiert. Eine Control-Message beinhaltet die folgenden Felder:

- *uint32_t MACAddressDst*: Control-Messages erfüllen den Zweck, den Zustand eines Teilnehmers oder aller Teilnehmer zu ändern. Dazu wurde dieses Feld als Ziel MAC-Adresse festgelegt. Die MAC-Adresse 0xFFFFFFFF ist als Broadcast definiert.
- *uint8_t depth*: Depth steht für die Anzahl Relays, die die Nachricht bereits passiert hat. Der Master sendet immer mit der Depth 0. Wird ein Paket relayed (wiederholt) wird das Depth-Feld inkrementiert. Empfängt der Node ein Paket mit einer Depth, die höher ist als jene des zuletzt empfangenen Pakets, so wird das Paket nicht erneut gesendet. Die zuletzt empfangene Depth ist nur eine gewisse Zeit gültig, um auf Veränderungen im Netzaufbau reagieren zu können. Abbildung 9.3 zeigt die Verbreitung einer Control Message mithilfe des Depth-Feldes.

⁶https://github.com/Rouben94/P6_Software/blob/master/SharedLib/bm_timesync.c [24]

- **div. Control-Parameter:** Die restlichen Felder sind Parameter welche den Zustand des Nodes verändern oder Parameter übergeben (siehe Abschnitt 9.3.7).

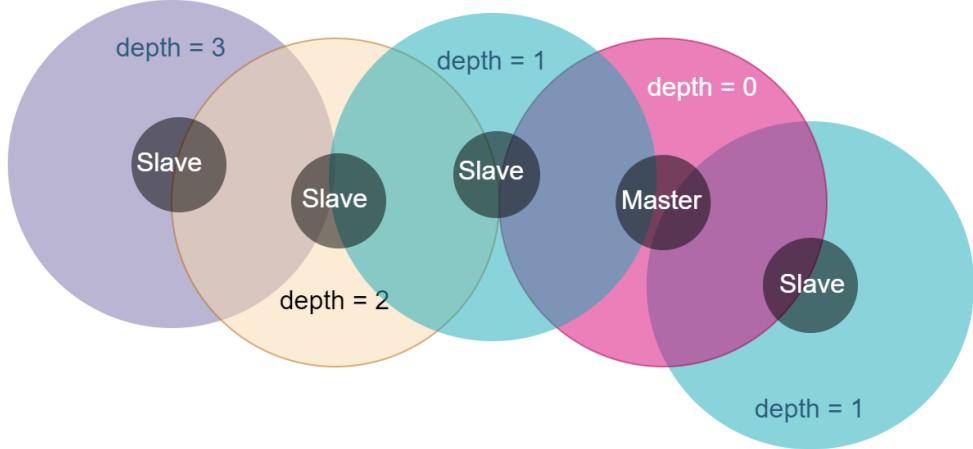


Abbildung 9.3: Konzept des Depth-Feld beim Versand einer Control Message

Ein Teilnehmer signalisiert über aufblinken der grünen-LED seine Bereitschaft, weitere Control Messages verarbeiten zu können. Blinkt die blaue-LED auf, zeigt dies an, dass der Teilnehmer eine Zustands- oder Parameteränderung erhalten hat.

Das Abfragen von Control-Messages erfolgt durch Pollen. Dies hat den Ursprung, dass nicht bei allen SDK's der Interrupt des Radio-Interfaces frei ist.

9.3.4 Report

Wie bereits in Abschnitt 9.3.1 beschrieben, versendet der Master Anfragen an einen Slave, um dessen Reports einzuholen. Dazu wird eine Control-Message vom Master initiiert, um den Slave in den Reporting Zustand zu versetzen. Diese Nachricht wird zwar an die weiteren Teilnehmern weitergeleitet, jedoch funktioniert das anschliessende Reporting nur über eine direkte Verbindung.

Das Reporting teilt sich in eine Subscribe-Funktion und eine Publish-Funktion auf. Der Master registriert sich auf Reports und sendet Report Requests an den Slave. Der Slave erwartet in der Publish-Funktion Report-Requests des Masters. Erhält ein Slave einen Request, so antwortet er dem Master mit dem verlangten Eintrag.

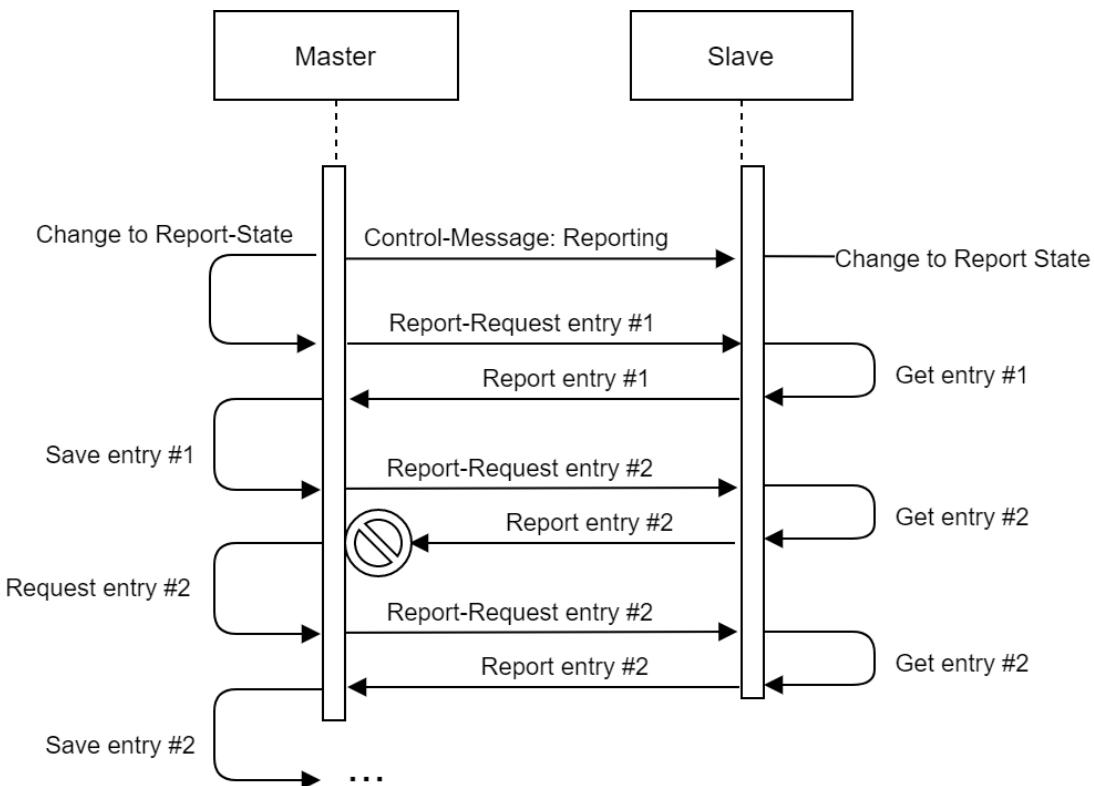


Abbildung 9.4: Ablauf des Reportings mit Handshake zwischen Master und Slave.

Abbildung 9.4 zeigt den Ablauf des Reportings anhand eines Beispiels. Mithilfe des Handshake-Verfahrens (siehe Abschnitt 9.3.1) wird der Report-Entry 2 nochmals angefordert, da dieser nicht erfolgreich beim Master angekommen ist.

Sobald der Report als ungültig erkannt wird, wird das Reporting beidseitig beendet. Dies geschieht durch die Prüfung des Zeitstempel-Feldes des jeweiligen Report Eintrages auf den Wert Null. Falls keine Verbindung zustande kommt, brechen Master sowie Slave nach einer bestimmten Anzahl von Versuchen das Reporting ab.

9.3.5 Logging

Das Logging-Modul ist für die Verwaltung der Log-Daten im RAM sowie im FLASH zuständig. Es definiert das Format eines Log-Eintrages, welcher bereits in Abschnitt 7.4.2 vorgestellt wurde. Die Länge eines Eintrages erreicht eine Grösse von 28 Byte. Sämtliche Einträge werden durch das Modul als Array mit 3000 Log-Einträgen gespeichert. Diese konstante Grösse für das Array kommt folgendermassen zustande. Pro Client können maximal 1000 Nachrichten verschickt werden. Ein Server soll mit maximal drei Clients in der selben Gruppe sein. Damit er alle Einträge aufnehmen kann, wurde die Länge des Arrays auf 3000 festgelegt. Somit müssen 84kByte RAM und Flash für das Log zur Verfügung stehen. Die im Ablauf benötigten Funktionen des Moduls lassen sich folgendermassen beschreiben.

- **bm_log_clear_ram()** wird genutzt, um alle Log-Einträge aus dem RAM zu entfernen.
- **bm_log_append_ram()** fügt einen Log-Eintrag dem RAM hinzu. Das Modul verwaltet selbst den nächsten freien Index.
- **bm_log_clear_flash()** bewirkt ein Löschen des gesamten Logs im FLASH.
- **bm_log_save_to_flash()** schreibt den gesamten Log vom RAM in das FLASH.

- `bm_log_load_from_flash()` lädt den gesamten Log vom FLASH in das RAM.
- `bm_log_init()` initialisiert das Logging-Modul.

Die Abarbeitung des Flash Moduls wurde bei Zephyr direkt in diesem Log-Modul integriert. Handelt es sich um eine nRF-SDK, findet das Handling des Flashes in einem separaten Modul statt (siehe Abschnitt 9.3.6).

Zephyr organisiert die Flash-Bereiche mithilfe eines Device-Tree-Source-Files (DTS). Darin werden verschiedene Partitionen definiert. Die Partition, in welcher die Daten gespeichert werden, ist vom Bootloader für ein DFU-Upgrade reserviert. Sofern kein DFU-Upgrade im Gange ist, kann der Bereich beliebig genutzt werden. Die Partition muss mindestens 84kByte zur Verfügung stellen. Der Zugriff auf das Flash ist über das Zephyr OS geregelt, welches einen Flash-Treiber zur Verfügung stellt. Dieser kann eine Pages-Grösse von 4kByte löschen.

9.3.6 Flash Save

Das Modul Flash Save wird zur Speicherung von Daten im FLASH benutzt. Dies geschieht mit Hilfe der Flash-Device-Storage (FDS) API, welche in der *nRF SDK for Mesh* sowie in der *nRF5 SDK for Thread and Zigbee* angeboten wird. Sie erlaubt die Speicherung von Daten im Flash und implementiert ein *Basic-Wearleveling*.

Das FDS-Modul beginnt, falls nicht anders vorgesehen, direkt unterhalb der Bootloader-Adresse mit der Speicherung der Daten im Flash. Mittels der Konfiguration die im File `sdk-config.h` definiert ist, werden die Länge des Flash-Bereichs und weitere Einstellungen für das FDS-Modul festgelegt. Die abzuspeichernden Daten werden in Records unterteilt. Zur Abspeicherung der Log-Daten wurde nur ein Record verwendet [25].

9.3.7 CLI

Mit diesem Modul werden Ein- und Ausgaben über das Command-Line-Interface (CLI) ermöglicht. Die einfachste Funktionalität ist die Ausgabe über einen Log-Befehl. Dieser ist mittels `bm_cli_log("Hello")` aufrufbar. Das Kommando wird auf die jeweilige Umgebung (SDK) angepasst. Bei den *nRF SDK's* werden Logs mittels dem `NRF_LOG()` Befehl und bei *Zephyr* mittels `printk()` ausgegeben.

Die zentrale Funktionalität des Moduls bezieht sich auf die Verarbeitung von CLI-Befehlen. Dafür werden verschiedene CLI-Befehlssätze angeboten, welche im Sub-Modul `bm_cli_cmds.c` definiert sind:

- **`setNodeSettings`:** Mithilfe dieses Befehlssatzes können die Einstellungen eines Slaves (Server/Clients) geändert werden. Eine Eingabe muss folgendes Format haben: `setNodeSettings <MAC in Integer format> <GroupNumber> <Node Id> <Ack> <AdditionalPayloadSize> <BenchmarkTrafficGenMode> <DST_MAC_1> <DST_MAC_2> <DST_MAC_3>`. Die Bedeutung der einzelnen Felder ist in Abschnitt 7.8.2 beschrieben.
- **`getNodeReport`:** Mithilfe dieses Befehlssatzes können die Reports eines Slaves (Server/Clients) abgefragt werden. Eine Eingabe muss folgendes Format haben: `getNodeReport <MAC in Integer format>`.
- **`startBM`:** Mithilfe dieses Befehlssatzes wird ein Benchmark gestartet. Eine Eingabe muss folgendes Format haben: `startBM <BenchmarkTime (seconds)> <BenchmarkPackets Count>`. Das erste Feld gibt die Benchmarkzeit in Sekunden an. Beim zweiten Feld handelt es sich um die Anzahl Nachrichten, welche versendet werden sollen.

Erhält der Master eine der oben erwähnten Befehlssätze löst, dieser eine entsprechende Control-Message aus. Die zugewiesenen Parameter werden ebenfalls mithilfe des Moduls abgespeichert und initialisiert.

9.3.8 Traffic Generator

Im folgenden Modul werden die Sende-Zeitwerte von Nachrichten generiert. Das Konzept dazu wurde bereits in Abschnitt 7.7 erläutert. Dabei wird zwischen zufälligen und sequentiellen Werten unterschieden.

Wird eine Messung mit sequentiellen Werten gestartet, erstellt dieses Modul, abhängig von den Benchmark-Parametern, eine Abfolge gemäss der bereits erwähnten Formel 7.1.

Wird eine Messung mit zufälligen Werten gestartet, erstellt dieses Modul, abhängig von den Benchmark-Parametern, ein neu generiertes Set oder verwendet ein bereits vorhandenes Set an zufälligen Werten. Es existieren 25 verschiedene vordefinierte Sets, welche vorgängig über die Webseite [random.org](https://www.random.org/)⁷ erzeugt wurden. Die Einstellungen, welche zur Erzeugung der Werte verwendet wurden, sind im Anhang F zu finden.

Wurde ein Set festgelegt, werden diese Werte mittels eines *Bubble-Sort* Algorithmus in aufsteigender Reihenfolge sortiert. Anschliessend müssen die Werte auf den Zeitbereich des Benchmarks skaliert werden. Da es sich bei den zufällig generierten Werten um *uInt16* Datentypen handelt, liegt der Wertebereich zwischen 0-65535. Die Werte werden gemäss Formel 9.1 berechnet.

$$T_{Msg_send} = \frac{V_{Rand}}{UINT16MAX} \cdot T_{Benchmark} \quad (9.1)$$

T_{Msg_send}	Zeitpunkt zu welchem die Nachricht gesendet wird.
$T_{Benchmark}$	Benchmark Dauer (siehe 7.1)
V_{Rand}	Random Value aus dem Set
$UINT16MAX$	Maximaler Wert des UInt16 (65535)

9.3.9 Low Level Radio

Der Low Level Radio Treiber soll möglichst unabhängig direkt mit dem Radio-Interface operieren. Er stellt die für die Verwaltung der Messung notwendigen Funktionen zur Verfügung. Senden oder Empfangen von Daten wird mittels einfachen Aufrufen realisiert.

Das Radio-Interface wird beinahe identisch zur P2P-Testinfrastruktur bedient (siehe Abschnitt 5.2). Lediglich auf die Verwendung von Interrupts musste verzichtet werden, da die meisten Radio-Driver der Mesh-Stacks diese bereits belegt haben. Insbesondere beim Zephyr findet die Initialisierung des Radio-Drivers vor dem Aufrufen der State machine statt. Somit wurden die Events durch Pollen abgefragt.

9.4 Benchmark Management

Um die Verwaltung einer Messung zu automatisieren stehen untenstehende Python-Scripts zur Verfügung. Die Scripts werden auf der BMS (Benchmark Management Station) ausgeführt. Die Konfiguration der Teilnehmer wird mittels einem Excel-File durchgeführt (siehe Abschnitt 7.8.2).

⁷<https://www.random.org/>

9.4.1 Flasher

Dieses Script dient dazu, alle nRF52-Dongles mit der entsprechenden Firmware zu flashen. Hierzu liest das Script die Konfigurationsdatei aus. Mithilfe der dort vorhanden Informationen fordert das Script den Benutzer dazu auf, den Dongle mit der entsprechenden Nummer via USB Buchse mit dem PC zu verbinden. Sobald der Dongle eingesteckt wurde, beginnt das Script mit dem Flashen. Dies geschieht mit Hilfe des nrfutil-Tools.

9.4.2 Configurator

Mithilfe des Configurator-Scripts kann eine voreingestellte Konfiguration auf die einzelnen Teilnehmer verteilt werden. Das Script arbeitet die Konfigurations-Einträge der Reihe nach ab. Mittels dem an der BMS angeschlossenen Masters versendet das Script den entsprechenden Konfigurationsbefehl.

9.4.3 Benchmark and Reporter

Über dieses Script wird entweder eine Messung gestartet oder die Resultate einer vorhergehenden Messung eingeholt. Wird ein neuer Benchmark initiiert, muss der Benutzer die Benchmark-Parameter angeben. Anschliessend startet das Script den Vorgang über den angeschlossenen Master. Während der Messung zeigt das Script die Meldungen des Masters an.

Nach Abschluss der Messung können die Reports mithilfe des Scripts eingeholt werden. Dazu fordert das Script den Benutzer auf, sich in die Nähe des ersten Slaves zu begeben. Nach der Bestätigung durch den Benutzer beginnt das Script damit, die Daten des Slaves einzusammeln. Nach dem Einsammeln aller Daten der Slaves schreibt das Script die Werte in ein CSV-File.

Teil III

Bluetooth Mesh

Raffael Anklin

10 Einleitung und Grundlagen

Bluetooth Mesh ist ein auf dem Bluetooth-Standard aufbauendes Mesh-Netzwerk. Der Standard wurde im Jahr 2017 von der Bluetooth-SIG vorgestellt. Das Ziel des Standards ist es die Reichweite und das Einsatzgebiet von Bluetooth-Geräten zu erweitern. Somit sollen in Zukunft Lichtschalter und Lampen, sowie Sensoren und Aktoren im Heimbereich, Industriebereich und diversen Anwendungsbereichen mittels Bluetooth-Technologie verbunden werden.



Abbildung 10.1: Parkhaus mit Bluetooth-Mesh [26]

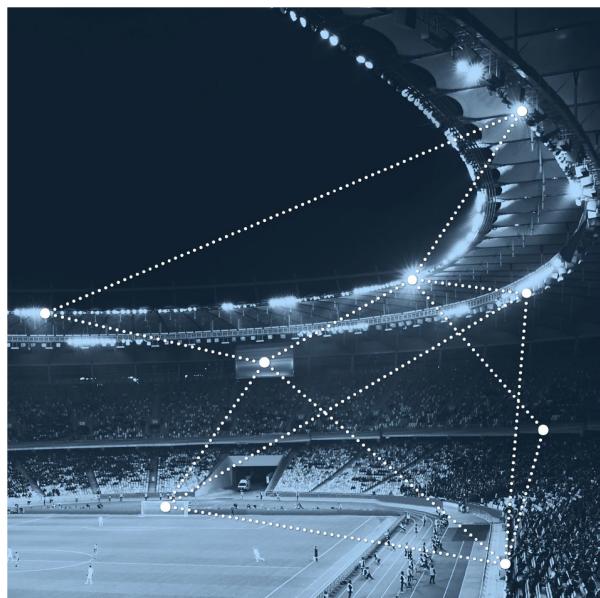


Abbildung 10.2: Stadion mit Bluetooth-Mesh [26]

Die Technologie baut auf den weit verbreitetem BLE-Standard auf, welcher in einer viel zahl von Endgeräten verbaut ist. Ab BLE-Version 4.0 könnten sich die Geräte zu einem Mesh-Netzwerk verbinden. Im Anschluss wird der Netzaufbau kurz erklärt und der Leser mit dem Funktionsprinzip des Mesh-Stacks vertraut gemacht.

Provisioning

Das Einbinden neuer Teilnehmer in ein Bluetooth-Mesh Netzwerk findet über den Provisioning-Prozess statt. Die Rolle des Provisioners kann ein Smartphone, Laptop oder ein berechtigter Mesh-Teilnehmer übernehmen. Ähnlich wie beim anmelden bei einem WLAN-Netzwerk teilt der Provisioner dem neuen Teilnehmer die Zugangsdaten (Netzwerkschlüssel, etc.) mit. Abschliessend beginnt das neue Gerät im Netzwerk als Node zu operieren.

Managed-Flooding

Bluetooth-Mesh basiert auf dem Managed-Flooding Prinzip. Einfach erklärt wiederholen alle Nodes, abhängig von verschiedenen Bedingungen, jede empfangende Nachricht (Relaying). So mit gelangen die Daten über Zwischenstationen (Hops) zum Ziel. Nachrichten welche bereits wiederholt wurden werden nicht erneut gesendet. Zusätzlich wird ein Time-To-Live Zähler zu

jeder Nachricht mitgegeben. Dieser gibt an, wie oft die Mitteilung erneut gesendet wird.

Models

Um die Funktion von Nodes (Lichtschalter, Lampe, Temperatursensor, etc.) zu unterscheiden, werden sogenannte Models definiert. Jedes Model spezifiziert Zustände (Licht EIN / Licht AUS), welche als States bezeichnet werden. Weiterhin sind Models in drei Kategorien eingeteilt: Server-, Client- und Control-Models. Diese Einteilung schreibt vor, ob die Zustände des Nodes zur Veränderung Angeboten werden (Server-Model z.B. Lampe), Zustände verändert werden (Client-Model z.B. Schalter) oder beides möglich ist (Control-Model z.B. Pumpe). Mithilfe dieser Normen lassen sich Geräte von unterschiedlichen Hersteller vereinen, sofern keine Hersteller spezifischen Models verwendet wurden.

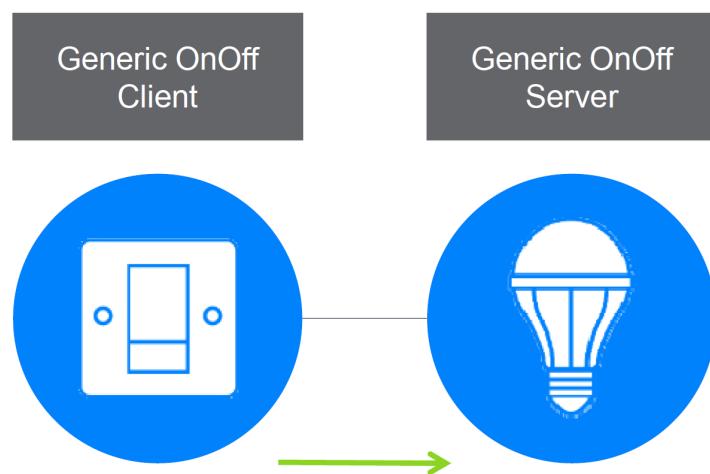


Abbildung 10.3: Client - Server Prinzip Bluetooth Mesh [26]

Adressierung

Zur Identifizierung im Netzwerk besitzt jeder Node eine Unicast-Address (einzigartige Adresse). Um mehrere Nodes zu einer Gruppe zusammenzufassen, werden Group-Addresses vergeben. Die Beziehungen zwischen Nodes werden durch Publishen (Veröffentlichen) oder Subscriben (Abonnieren) auf Adressen geregelt. Im Anwendungsfall Publisht der Client (Schalter) an die Adresse auf welche einer oder mehrere Server (Lampe/Lampen) Subscriben. Wie in Abbildung 10.4 ist das Unterscheiden von Bereichen mittels Gruppen möglich. Ein Node kann gleichzeitig an mehrere Adressen Publishen und Subscriben, was zu beliebig komplexen Aufbauten führt.

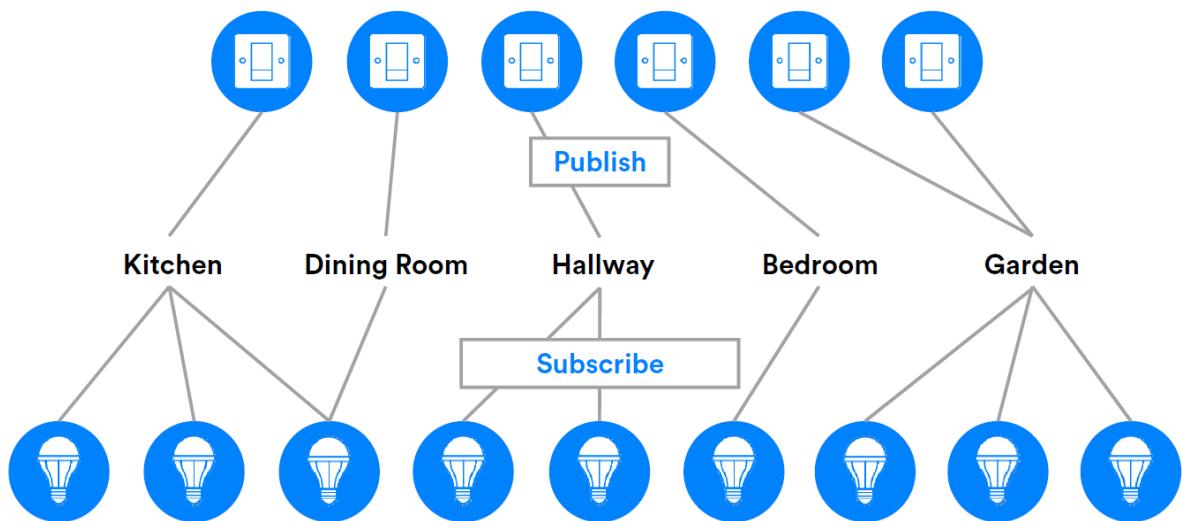


Abbildung 10.4: Publish - Subscribe Prinzip Bluetooth Mesh [26]

11 Technische Grundlagen Bluetooth Mesh

11.1 Netzaufbau und Topologie

Nebst dem funktionellen Aspekt übernehmen Nodes unterschiedliche Rollen im Netzaufbau. Ein Node wird als Relay-Node bezeichnet, wenn dieser Nachrichten an weitere Teilnehmer weiterleitet. Ein Friend-Node dient als Zugangspunkt für einen Low-Power-Node. Der Low-Power-Node wird dort eingesetzt wo keine konstante Stromversorgung zur Verfügung steht. Dieser geht eine Beziehung mit einem Friend-Node ein. Der Friend-Node speichert alle Nachrichten der LPNs, welche mit ihm eine Beziehung pflegen. In einem festen Zeitintervall fragt der LPN die verpassten Nachrichten beim Friend-Node ab. Dadurch kann der LPN zwischen Abfragen inaktiv bleiben um Energie zu sparen. Um die Interoperabilität zwischen inkompatiblen Bluetooth-Mesh Geräten und einem Mesh-Netzwerk zu ermöglichen existieren Proxy-Nodes. Ein Proxy-Node dient als Schnittstelle in das Netzwerk und erlaubt das Interagieren über Bluetooth-GATT mit dem Mesh. Dies ermöglicht das steuern des Netzwerks über das Smartphone. Die in Abbildung 11.1 gezeigte Topologie zeigt die verschiedenen Node-Typen an ihrem Einsatzort.

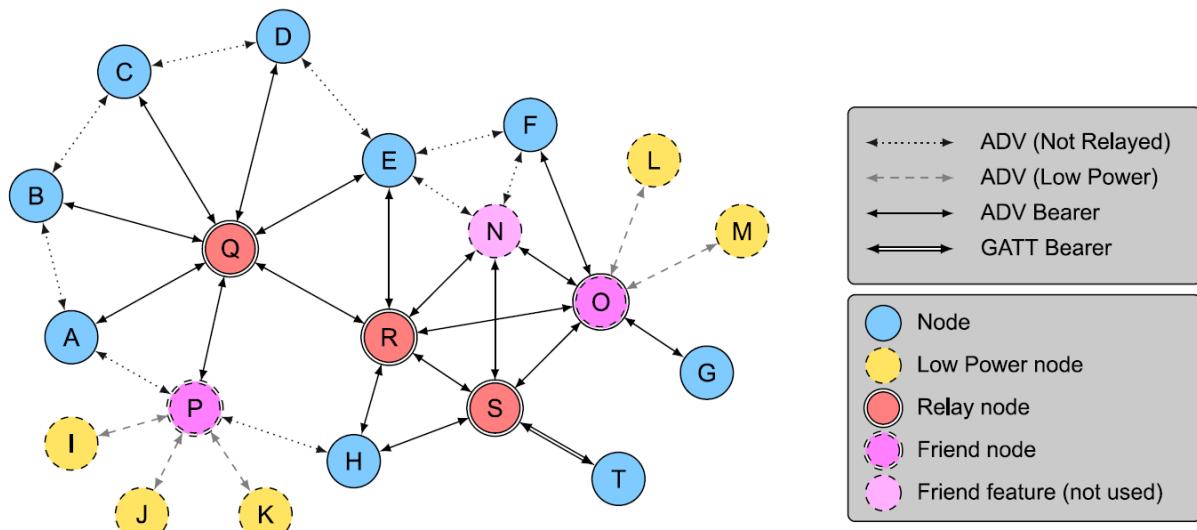


Abbildung 11.1: Beispielhafte Topologie eines Bluetooth-Mesh Netzwerks [27]

11.2 Protokoll Stack

In diesem Abschnitt wird die Architektur des Mesh-Stacks genauer untersucht. Der Grundsätzliche Aufbau wird Abbildung 11.2 veranschaulicht. Eine Grafik zur Veranschaulichung des Message-Flows durch die einzelnen Schichten ist im Anhang G ersichtlich.

Wie in Kapitel 10 bereits erwähnt basiert der Stack auf Bluetooth Low Energy. Der BLE-Layer dient zur Grundlegenden Schicht des Stacks. Der Zugriff für Mesh-Traffic erfolgt über das GAP-Profil, der für Proxy Traffic über das GATT-Profil. Der Bearer-Layer regelt den Zugriff auf den BLE-Stack. Es existieren verschiedene Bearers. Der GATT-Bearer ermöglicht Geräten ohne GAP Zugriff auf das Netzwerk. Der Advertising-Bearer wird für den Mesh-Traffic benutzt.

Der Network-Layer hat folgende Aufgaben zu erfüllen:

- Ver- und Entschlüsselung der Network-PDU
- Filtern von nicht relevanten Nachrichten (Adressauflösung)
- Relaying von Paketen mittels TTL-Field

Zudem bedient dieser Layer verschiedene Bearers und ist dafür verantwortlich das alle relevanten Pakete zur entsprechenden Stelle weitergeleitet werden.

Über dem Network Layer befindet sich der Transport-Layer, welcher in einen Lower- und Upper-Transport-Layer aufgeteilt ist. Der Lower-Transport-Layer handelt das Acknowledgement einzelner Segmente von eingehenden Nachrichten ab. Zudem beeinhaltet er das Herz-Stück des Friend-Features, nämlich die Friend-Queue. In diesem Buffer werden alle Nachrichten der Low-Power-Nodes aufbewahrt. Ebenfalls führt diese Schicht das Segmentieren und Zusammensetzen der Pakete (SAR) durch. Pakete werden in Fragmente aufgeteilt, welche lediglich 8-10 Byte an Applikations-Daten beherbergen können.

Der Upper-Transport-Layer dient hauptsächlich zur Ver- und Entschlüsselung von Paketen auf Transport-Ebene. Zudem werden Anfragen wie Friend-Requests über diese Schicht verarbeitet.

Der Access-Layer führt die Zuordnung der Nachrichten an die Models durch. Diese Schicht regelt den Zugriff der Applikation auf die unterliegenden Schichten. Zudem werden Nachrichten auf allfällige Replay-Messages geprüft, bevor sie an die Applikation weitergegeben werden.

Die zweitletzte Schicht bildet der Foundation-Models-Layer. Seine Aufgabe ist es die Models zu Verwalten, welches über das Config-Server-Model geschieht. Zusätzlich gehört ein Health-Model zu den Foundation-Models, welches den Zustand des Teilnehmers überwacht. Beide Models müssen zwingend auf jedem Node vorhanden sein. [27] [26]

Als letzte Schicht dienen die Models. Dessen Funktionalität ist rein von der Applikation abhängig (siehe Abschnitt 10). Daher kann diese Schicht als Application-Layer aufgefasst werden.

11.3 Sicherheit

Bleutooth-Mesh Nachrichten werden zweifach Verschlüsselt. Die Network-PDU ist mit dem zugehörigen Network-Key gesichert. Wer im Besitz des Netzwerk-Schlüssels gelangt, kann Nachrichten bis zum Upper-Transport-Layer auslesen. Für Teilnehmer welche die Nachricht nur weiterleiten (Relayen) müssen, reicht dieser Einblick aus. Auf Applikationsebene findet die Verschlüsselung zusätzlich über einen Application-Key statt. Nur Teilnehmer mit dem richtigen Application-Key können an Informationen der Models gelangen und diese Steuern.

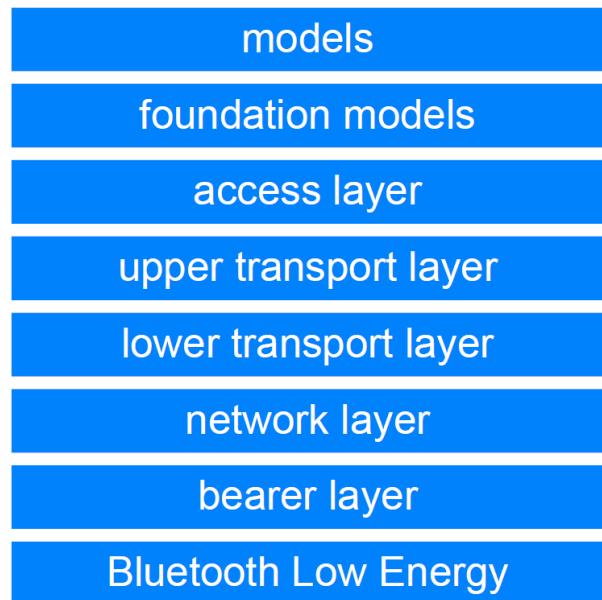


Abbildung 11.2: Bluetooth-Mesh Stack [26]

Zusätzlich zur Verschlüsselung führt jeder Teilnehmer eine fortlaufenden Nummerierung der Nachrichten durch (SEQ-Field). Damit lassen sich sogenannte Replay-Attacks verhindern.

Um während des Provisioning-Ablauf die Authentifizierung des neuen Teilnehmers sicher zu stellen, wird ein Device-Key verwendet. Dieser dient einmalig dazu das neue Gerät einzubinden und zu konfigurieren.

11.4 Bluetooth Mesh Software Development Kit

Die Umsetzung des Bluetooth-Mesh-Stacks ist für die nRF-Platform mittels den folgenden zwei Bibliotheken möglich.

- **nRF Connect SDK:** Auf dem Zephyr-RTOS aufbauende SDK. Zephyr Implementation des Mesh-Stacks, welche von Bluetooth SIG anerkannt ist. Von Nordic nicht zur Verwendung in Endprodukten empfohlen. Unterstützt neuere SOCs wie den nRF5340. [20]
- **nRF SDK for Mesh:** Proprietäre SDK Library von Nordic Semiconductor für Bluetooth-Mesh. Ist zum Einsatz in Endprodukten freigegeben. [22]

Da der neuere SOC nur von der nRF Connect SDK unterstützt wird, sowie die Zephyr Implementation auf weitere SOCs andere Hersteller anwendbar ist, wurde die Implementation mit dieser Bibliothek vorangetrieben. Als Nebenprodukt wurde die Entwicklung mit der nRF SDK for Mesh ins Auge gefasst um die beiden SDKs miteinander zu vergleichen. Zu diesem Zeitpunkt ist ein direkter Vergleich leider noch nicht möglich, da die Funktionalität der nRF SDK for Mesh Variante noch nicht vollständig umgesetzt wurde. Die Umsetzung des Benchmarks wird im folgenden Abschnitt dokumentiert.

12 Umsetzung Benchmark

Die Umsetzung des Benchmarks, wurde gemäss den Anforderungen des Benchamrk-Konzepts (siehe Abschnitt 7) umgesetzt. Zudem wurden die Schnittstellen gemäss dem Software Konzept (siehe Abschnitt 9) an die Shared-Library eingebaut. Im Anschluss wird auf die Umsetzung mittels der SDKs eingegangen, sowie die einzelnen Module dazu Beschrieben.

12.1 nRF Connect SDK

Das aufsetzen der nRF Connect SDK, sowie das Installieren der IDE ist online⁸ Dokumentiert. Mithilfe von Beispielen⁹ und der Bluetooth-Mesh-API¹⁰ wurde die Umsetzung bewerkstelligt. Dabei gliedert sich das Programm in zwei Module welche Nachfolgend beschrieben werden.

12.1.1 Stack Initialisierung und Konfiguration

Das Modul *bm_blemesh.c*¹¹ dient zur Initialisierung des Stacks. Dazu wird die Funktion *bm_blemesh_enable()* aufgerufen. Diese führt die notwendigen Schritte aus um den Mesh-Stack hochzufahren. Nach erfolgreicher Initialisierung wird mit dem Provisioning weitergefahren.

Self Provisioning

Um ein vollautomatischen Testablauf zu gewährleisten, wurden die Sicherheitsschlüssel direkt im Code vorgegeben. Mittels der eigens dafür vorgesehenen API *bt_mesh_provision* kann sich der Teilnehmer selbst ins Netzwerk einbinden. Dies ist nur zu Testzwecken empfohlen. Die Einbindung der Teilnehmer ins Netz erfolgt somit zuverlässig und unkompliziert.

Self Configuring

Nach der erfolgreichen Einbindung ins Netz, beginnt der Node mit seiner Konfiguration. Ob es sich dabei um einen Client oder Server handelt muss zur Laufzeit bereits bekannt sein. Dies wurde zum Zeitpunkt der Kompilierung mittels der *bm_config.h* Datei festgelegt. Um Testnachrichten zu senden oder zu Empfangen wurde das Generic ON/OFF Server- resp. Client-Model eingesetzt. Des weiteren werden die Models initialisiert, an den Applikations-Key gebunden und die Publish- oder Subsrcibe-Adressen auf die jeweilige Group-Address konfiguriert. Diese Werte sind entweder aus der vorhergehenden Konfiguration bekannt oder wurden fest vorgegeben. Der Teilnehmer ist nun bereit Nachrichten zu Senden oder zu Empfangen.

12.1.2 Model Handler

Das Modul *bm_blemesh_model_handler.c*¹² beherbergt alle notwendigen Model-Handler, welche zum Senden oder Empfangen von Nachrichten aufgerufen werden. In den jeweiligen Handlern soll ein Log-Eintrag mit den entsprechenden Informationen aufgefüllt werden.

¹⁰https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/getting_started.html

¹⁰https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/samples.html

¹⁰https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/zephyr/reference/bluetooth/mesh.html

¹¹https://github.com/Rouben94/P6_Software/blob/master/Bluetooth/Zephyr_Mesh/src/bm_blemesh.c

¹²https://github.com/Rouben94/P6_Software/blob/master/Bluetooth/Zephyr_Mesh/src/bm_blemesh_model_handler.c

Message Sending

Um eine Nachricht zu senden wird eine universelle Schnittstelle der Shared-Library zur Verfügung gestellt. Diese kann mittels dem Aufrufen der Funktion `bm_send_message()` eine Nachricht versenden. Anschliessend wird der entsprechende Handler des Generic ON/OFF Clients benachrichtigt. Dieser wechselt den Zustand des grünen RGB-LEDs, speichert eine Log-Eintrag und schickt die Nachricht ab. Dieser wird, gemäss der Konfiguration des Nodes, acknowledged oder unacknowledged und mit der zusätzlichen anzahl Bytes gesendet.

Message Receiving

Sobald eine Nachricht vom Stack empfangen wurde, wird der Handler des Generic ON/OFF Servers benachrichtigt. Im Handler-Kontext sind viele notwendige Informationen für das Log bereits enthalten. Das erfassen des Zeitstempels und abfragen der Nachrichtenlänge wird über externe Funktionsaufrufe und Hilfsvariablen bewerkstelligt. Schlussendlich wird die blaue RGB-LED geschaltet, welches den erfolgreichen Empfang einer Nachricht anzeigen.

12.2 Benchmark und Stack Parameter

Die Tabelle 12.1 soll einen Überblick der verwendeten Paramter des Stacks geben.

Stack Init Time	10000 ms	Zeit die benötigt wird um den Stack für den Benchmark zu initialisieren.
Node Type	Relay-Node	Alle Nodes werden als Relay-Nodes konfiguriert
Default Group ID	0xC000	Zu diesem Wert wird der Index für die zugewiesene Gruppe addiert um damit die Gruppenzugehörigkeit festzulegen.

Tabelle 12.1: Bluetooth Mesh Benchmark und Stack Parameter

Ein detaillierte Konfiguration des Zephyr Kconfig kann übder den Source-Code¹³ abgefragt werden.

12.3 nRF SDK for Mesh

Die Umsetzung mittels der *nRF SDK for Mesh* wurde noch nicht vollumfänglich abgeschlossen. Es liegen zurzeit Probleme beim Zugriff auf den Flash-Treiber vor, welche noch zu beheben sind. Die Umsetzung des Benchmarks wurde jedoch abgeschlossen und es können Nachrichten versandt sowie empfangen werden. Das Loggen der Informationen funktioniert ebenfalls.

Das aufsetzen der *nRF SDK for Mesh*, sowie das Installieren der IDE ist online¹⁴ Dokumentiert. Mithilfe von Beispielen¹⁵ und der Bluetooth-Mesh-API¹⁶ wurde die Umsetzung bewerkstelligt. Dabei gliedert sich das Programm in zwei Module, welche Nachfolgend beschrieben werden.

¹³https://github.com/Rouben94/P6_Software/blob/master/Bluetooth/Zephyr_Mesh/prj.conf

Die Umsetzung wurde ähnlich realisiert wie mit der *nRF Connect SDK*. Im Anschluss sind die Unterschiede aufgezeigt.

12.3.1 Stack Initialisierung und Konfiguration

Das Modul *bm_ble_mesh.c*¹⁷ sowie das Modul *bm_mesh_stack.c*¹⁸ dienen zur Initialisierung des Stacks. Dazu wird die Funktion *bm_blemesh_init()* aufgerufen. Diese führt die notwendigen Schritte aus um den Mesh-Stack hochzufahren. Nach erfolgreicher Initialisierung wird mit dem Provisioning weitergefahren.

Self Provisioning and Configuring

Das Self Provisioning sowie das Self Configuring werden bei der *nRF SDK for Mesh* mittels dem Device-State-Manager durchgeführt. Dieser verwaltet alle notwendigen Keys sowie Address-Listen, welche zur Konfiguration gebraucht werden. Durch analysieren des Config-Server-Models, welches normalerweise die Konfiguration des Nodes vornimmt, wurde der notwendige Ablauf zur Konfiguration rekonstruiert.

12.3.2 Model Handler

Die Model-Handler sind im Modul *bm_ble_mesh.c*¹⁹ integriert. Das Empfangen und Senden von Nachrichten wird ähnlich wie bei der *nRF Connect SDK* abgehandelt.

¹⁶https://infocenter.nordicsemi.com/topic/com.nordic.infocenter.meshsdk.v4.2.0/md_doc_getting_started_getting_started.html

¹⁶https://infocenter.nordicsemi.com/topic/com.nordic.infocenter.meshsdk.v4.2.0/md_examples_lig ht_switch_README.html

¹⁶<https://infocenter.nordicsemi.com/topic/com.nordic.infocenter.meshsdk.v4.2.0/modules.html>

¹⁸https://github.com/Rouben94/P6_Software/blob/master/Bluetooth/nRF_SDK_Mesh/src/bm_ble_mesh.c

¹⁸https://github.com/Rouben94/P6_Software/blob/master/Bluetooth/nRF_SDK_Mesh/src/bm_mesh_stack.c

13 Analyse Bluetooth Mesh Stack

Bleutooth-Mesh ist ein relativ neuer Stack mit wenig Anwendungsbeispielen. Er wurde entwickelt um mit der Konkurrenz wie Z-Wave oder Zigbee mithalten zu können. Im folgenden werden die Grenzen des Stacks aufgezeigt.

13.0.1 Grenzen des Stacks

Die Topologie eines Netzwerks beeinträchtigt stark seine Performance. Ist die Node-Dichte sehr hoch, steigt die Wahrscheinlichkeit, dass es zu Kollisionen der einzelnen Nachrichten kommt. Der BLE-MAC-Layer verfügt im Gegensatz zum IEEE 802.15.4 Layer über keine Collision Avoidance. Somit ist es im Interesse aller Teilnehmer, wenn die Radio-Sende-Zeiten so kurz wie möglich gehalten werden. Deshalb schreibt der Stack lediglich 29 Bytes als maximale Network-PDU vor. Durch die Geschwindigkeit des MAC-Layers von 1-2Mbit/s sinkt die Wahrscheinlichkeit von Kollisionen der Pakete weiter. Das versenden von längeren Nachrichten, welche stark segmentiert werden müssen, erhöhen die Gesamtbelastung des Netzwerks. [27]

Der Mesh-Stack schreibt eine maximale Grösse von 32767 Teilnehmern vor. Dies soll die theoretische sowie praktische Grenze darstellen. Ein Gebäude der Firma Silvair wurde mit über 1000 Bluetooth-Mesh-Nodes ausgestattet. Laut dem Betreiber laufe das System einwandfrei. Dabei handelt es sich um ein Office-Gebäude mit vielen anderen Bluetooth-Geräten wie Smartphones, Mäuse und Headsets sowie WLAN-Access-Points. [28]

Ein Bluetooth-Mesh Paket wird immer auf drei Kanälen (37,38,39) gesendet (BLE-GAP Advertising). Dadurch erhöht sich die Störfestigkeit. Jedoch verringert sich der Durchsatz mit dem Faktor drei, da ein Paket dreimal solange braucht um gesendet zu werden.

13.0.2 Bluetooth Mesh Implementation in Zephyr

Die Implementation des Bluetooth-Mesh-Stacks in Zephyr weist einige Mängel auf. Es liegen Probleme mit dem HCI-Treiber (Host Controller Interface), welcher das Radio-Interface ansteuert, vor. Diese äussern sich dadurch, dass nach jedem Advertising (Senden) eines Mesh-Pakets der aktive Thread für 10ms pausiert wird. Dadurch erhält die Abarbeitung des Stacks eine gravierende Verzögerung, bis wieder in den Scanner (Empfangs) Betrieb gewechselt werden kann. Vermutlich führt diese Lücke ebenfalls zum verkürzen der Aktive-Radio-Time, welche gemessen wird um den Energieverbrauch abzuschätzen. Sie gibt an wie lange das Radio-Interface aktiv genutzt wurde.

Zudem wurde einen Problematik beim Segmentieren von Nachrichten in der Bluetooth-Mesh Implementation in Zephyr festgestellt. Beim Empfangen von segmentierten Nachrichten, werden diese zum Teil mehrmals empfangen. Ob dies Ebenfalls einen Einfluss auf das Messresultat hatte ist unklar. Ein Online²⁰ Beitrag verweist ebenfalls auf diese Problematik.

²⁰<https://github.com/zephyrproject-rtos/zephyr/issues/16886>

Teil IV

Thread

Autor: Robin Bobst

14 Einleitung

In diesem Teil der Arbeit werden die Eigenschaften und Besonderheiten des Thread Mesh Stacks erläutert und es wird auf die Umsetzung des Benchmarks auf der Stack Ebene eingegangen. Dieser Teil soll als eigenständiger Teil betrachtet werden, in dem ausschliesslich Thread behandelt wird.

Thread ist ein auf IPv6 basiertes Netzwerkprotokoll, das speziell für Internet of Things (IoT)-Anwendungen entwickelt wurde. Die einzelnen Teilnehmer im Netzwerk verbinden sich zu einem Mesh-Netzwerk. Wie in Abbildung 14.1 ersichtlich, verwendet Thread für eine effiziente Kommunikation mit IPv6-Paketen das Kommunikationsprotokoll 6LoWPAN (IPv6 over Low Power Personal Area Network). 6LoWPAN wendet ein Header-Kompressionsverfahren an, welches es ermöglicht, die IPv6-Pakete über den Standard IEEE-802.15.4 zu übermitteln. Dank diesem Standard ist es machbar, die mit Thread entwickelten Geräte so energieeffizient zu gestalten, dass ein Batteriebetrieb realisierbar ist. In Tabelle 14.1 sind die wichtigsten Merkmale von Thread aufgelistet. Im Juli 2014 wurde die Thread Working Group ins Leben gerufen, bei der folgende Firmen Bestandteil der Gruppe sind: Nest Labs, Samsung, ARRM Holdings, Qualcomm, NXP Semiconductors, Silicon Labs, Big Ass Solutions, Somfy, OSRAM Tyco International und Yale. Ab August 2018 trat auch Apple der Arbeitsgruppe bei, um das Protokoll populär zu machen. [29, Kapitel 1]

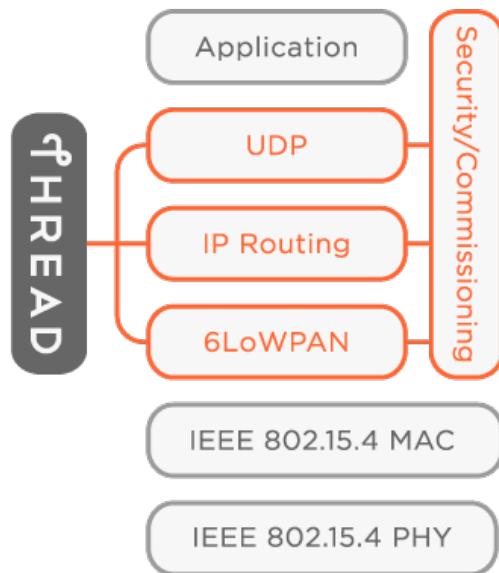


Abbildung 14.1: Thread Protokoll Layer [30]

Netzwerk		Applikation	
Merkmal	Beschreibung	Merkmal	Beschreibung
IEEE 802.15.4	Protokoll	IPv6	IP-Kommunikation
MAC Security	Verschlüsselte Übertragung	UDP	UDP-Sockets
6LoWPAN	Effiziente IPv6 Paket Übertragung	CoAP	Client und Server
Mesh Routing	Many-to-many Kommunikation	DHCPv6	Client und Server

Boarder Router		Weitere Merkmale	
Web - UI	Für Netzwerk Management	Periodic parent search	Endgerät wechselt zu besserem Parent
Externer Kommissioner	Externes Gerät für Neuaufnahme	Jam Detection	Signal Stau verhindern
NAT64	Kommunikation mit IPv4	Child Supervision	Endgerät überprüfen
wpantund	Interface Treiber	Inform previous parent on reattach	Endgerät Management

Tabelle 14.1: Merkmale Thread [31]

15 Technische Grundlagen Thread

In Abbildung 15.1 ist zu sehen, wie das Thread Protokoll aufgebaut ist. In den folgenden Unterkapiteln wird auf die verschiedenen Layer des Stacks eingegangen.

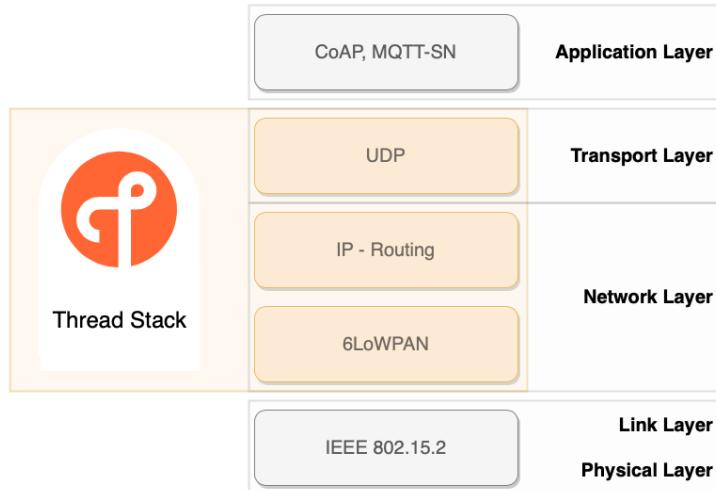


Abbildung 15.1: Übersicht Thread Protokoll

15.1 Application Layer

Da das Thread Protokoll auf einer UDP Kommunikation basiert, ist die Wahl des Applikation Layers dem Entwickler freigestellt. Grundsätzlich ist jedes Protokoll einsetzbar, das auf einer UDP-Kommunikation basiert. Da Thread das Netzwerkmanagement über das Constrained Application Protocol (CoAP) abfertigt, wird CoAP in diesem Kapitel etwas näher erläutert.

15.1.1 CoAP

Das Constrained Application Protocol (CoAP) ist ein spezialisiertes Web-Übertragungsprotokoll, das dafür entwickelt wurde, eingeschränkten Geräten die Teilnahme am IoT zu ermöglichen. CoAP ist durch seinen niedrigen Stromverbrauch und geringen Netzwerk-Overhead dafür ausgelegt, bei einem Netzwerk mit kleiner Bandbreite und hoher Auslastung zu funktionieren. UDP wird als grundlegendes Netzwerkprotokoll verwendet. Dies bedeutet, dass es sich beim CoAP um ein Client-Server-IoT Protokoll handelt, welches die gleichen Methoden wie HTTP verwendet. Das Protokoll wird für Machine-to-Machine (M2M)-Anwendungen zur intelligenten Energie- und Gebäudeautomatisierung verwendet. Dank diesen Eigenschaften ist es möglich, CoAP bei stromsparenden Modulen einzusetzen, während TCP-basierte Protokolle nicht in der Lage sind, Informationen auszutauschen. Das Protokoll wurde von der Internet Engineering Task Force (IETF) entworfen, welche CoAP in IETF RFC 75852 spezifiziert. In Tabelle 15.1 sind einige Features aufgelistet. [32]

Features		
Web-Protokoll M2M	Geringer Overhead	Proxy- und Caching Fähigkeiten
Asynchroner Nachrichtenaustausch	URI Uniform Resource Identifier	User Datagram Protocol (UDP)

Tabelle 15.1: Features CoAP

15.2 Thread Protokoll Stack

Der Thread Stack besteht im Grunde aus dem Transport und Network Layer (Abbildung 15.1). Dank dieser Schicht ist es möglich, eine geroutete IPv6-Verbindung mit verschiedenen IoT fähigen Geräten aufzubauen. Nachfolgend wird erläutert, wie der Stack aufgebaut ist.

15.2.1 UDP

Das Thread Protokoll schreibt vor, dass ein User Datagram Protocol (UDP) nach dem Standard [33] implementiert werden muss. UDP ist ein minimales und verbindungsloses Netzwerkprotokoll, das ein Versand von Datagrammen innerhalb IP-basierten Netzwerken ermöglicht. Das Protokoll verwendet Ports, um die Nachrichten an die richtigen Empfänger zu versenden. Zusätzlich besteht die Möglichkeit, eine Prüfsumme mit der Nachricht zu versenden, um fehlerhafte Nachrichten zu identifizieren. [29, Seite 6-2]

15.2.2 IP Routing

Das Thread Routing-Protokoll ist ein einfaches Distanz-Vektor-Routing-Protokoll. Das Ziel ist, die Routing-Information, die mit einer Nachricht versendet werden kann, zu erhöhen. Aus diesem Grund ist die Anzahl Router im Netzwerk limitiert. Alle Router versenden periodisch ihre Link-Kosten und die Verbindungsqualitäten zu direkt erreichbaren Routern im ganzen Netz. Die endgültigen Routing-Kosten zu einem Ziel, sind demnach die Kosten von allen Routern zum Ziel plus die Kosten zum direkten Nachbarn. Mit dem Trickle-Algorithmus [34] wird die Rate festgelegt, mit der ein Router seine Informationen ins Netz sendet. [29, Seite 5-23]

15.2.3 6LoWPAN

6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) ist ein drahtloses Low-Power-Mesh-Netzwerk, bei dem alle Knoten über IPv6-Adressen kommunizieren. Ein Thread Gerät muss das Protokoll implementiert haben. 6LoWPAN dient als Anpassungsschicht zwischen dem MAC Layer und dem Netzwerk Layer. Das Protokoll fragmentiert und setzt die IPv6-Pakete wieder zusammen, damit die Grösse der Payload vom IEEE 802.15.4 Protokoll übereinstimmt. 6LoWPAN adaptiert somit die IPv6-Pakete auf die IEEE 802.15.4 Pakete, die von der Radioschnittstelle gesendet werden. [35] In Tabelle 15.2 sind einige Features aufgelistet. [29, Seite 3-10]

Features		
Offene IP Standards	Unterstützt Schlafende Endgeräte	Skalierbares Mesh-Routing
End-Zu-End IP Kommunikation	Selbst Heilendes Netzwerk	Offener Standard (RFC 6282)

Tabelle 15.2: Features 6LoWPAN

15.3 Link und Physical Layer

Das Thread Protokoll verwendet im Link und Physical Layer das IEEE 802.15.4 Protokoll. [36] [29, Seite 3-2]

15.4 Netzaufbau und Topologie

In den nachfolgenden Unterkapiteln wird erklärt, wie das ganze Thread Netzwerk aufgebaut und verwaltet wird.

15.4.1 Node Rollen

Wie im Bild 15.2 ersichtlich, wird das Thread Netzwerk in zwei verschiedene Rollen aufgeteilt.
[29, Seite 1-4]

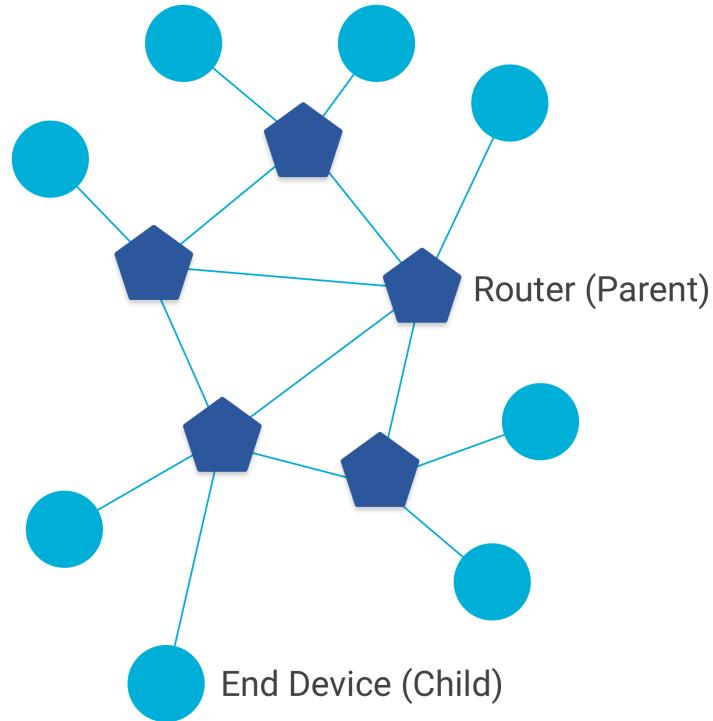


Abbildung 15.2: Thread Device Roles [37]

Router:

Ein Gerät mit der Router Rolle leitet alle Pakete für Netzwerkgeräte weiter und hält aus diesem Grund seinen Transceiver jederzeit aktiviert. Weiter bietet der Router sichere Commissioning-Dienste für Geräte an, die versuchen, dem Netzwerk beizutreten. [29, Seite 1-4]

Endgerät:

Ein Gerät mit der Rolle Endgerät kommuniziert hingegen nur mit einem einzelnen Router und kann seinen Transceiver deaktivieren, falls dieser nicht benötigt wird. Das Endgerät kann deshalb seine Leistung reduzieren und trotzdem voll funktionsfähig im Netzwerk kommunizieren. Da der Transceiver sporadisch abgeschaltet wird, leitet das Endgerät keine Pakete für andere Netzwerkgeräte weiter. [29, Seite 1-4]

15.4.2 Node Typen

Ausserdem können die Thread Geräte, wie in Abbildung 15.3 zu sehen ist, in zwei verschiedene Node Typen unterteilt werden: [29, Seite 1-4]

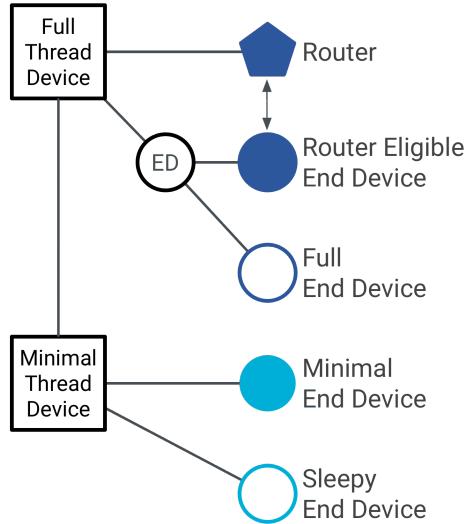


Abbildung 15.3: Thread Device Types [38]

Full Thread Device (FTD):

Ein Full Thread Device (FTD) hat sein on-chip-radio immer aktiviert. Es erhält und leitet alle Multicast Nachrichten weiter und verwaltet die IPv6-Adresszuordnungen. Ein FTD kann als Router oder als Endgerät arbeiten. Es gibt drei Arten von FTDs: [29, Seite 1-4]

- **Router**
- **Router Eligible End Device (REED)** - Kann Router oder Endgerät sein.
- **Full End Device (FED)** - Kann nur ein Endgerät sein.

Minimal Thread Device (MTD):

Ein Minimal Thread Device leitet keine Multicast Nachrichten weiter. Das Gerät kommuniziert nur über einen Router mit dem Netzwerk. Ein MTD kann nur als Endgerät arbeiten. [29, Seite 1-4]

- **Minimal End Device (MED)** - Transceiver ist immer eingeschaltet und muss keine Nachrichten vom Router abfragen.
- **Sleepy End Device (SED)** - Transceiver ist ausgeschaltet, schaltet sich nur ein, um Nachrichten vom Router abzufragen.

Leader:

Der Thread Leader ist ein Router, der für die Verwaltung aller Router im Netzwerk verantwortlich ist. Der Leader wird dynamisch selbst vom Netzwerk gewählt und hat Zugriff auf das gesamte Thread Netzwerk. Er verteilt auf ganzer Netzwerksebene Konfigurationsinformationen. In jedem Netzwerk gibt es immer nur einen Leader. [29, Seite 1-4]

Border Router:

Ein Border Router ist ein Gerät, das Informationen zwischen dem Thread Netzwerk und einem nicht Thread Netzwerk weiterleitet. Mit dem Border Router ist es möglich, das Thread Netzwerk über das Internet zu erreichen. Es besteht zudem die Möglichkeit, das Thread Netzwerk über den Border Router zu konfigurieren. So können z.B. mit einer Bluetooth Verbindung einzelne Knoten hinzugefügt oder entfernt werden. [29, Seite 1-4]

15.4.3 IPv6 Adressierung

Die Thread Knoten kommunizieren in einem IPv6 Adressraum miteinander. In diesem Kapitel wird dies etwas näher erläutert. [29, Kapitel 5]

Bereiche:

Wie in Abbildung 15.4 ersichtlich, gibt es für eine Unicast Nachricht drei verschiedene Adressräume:

- **Link-Lokal** - Beinhaltet alle Geräte, die direkt (ohne Hop) erreichbar sind.
- **Mesh-Lokal** - Beinhaltet alle Geräte innerhalb desselben Thread Netzwerkes.
- **Global** - Beinhaltet jegliche Geräte, insbesondere solche die über einen Border Router erreichbar sind.

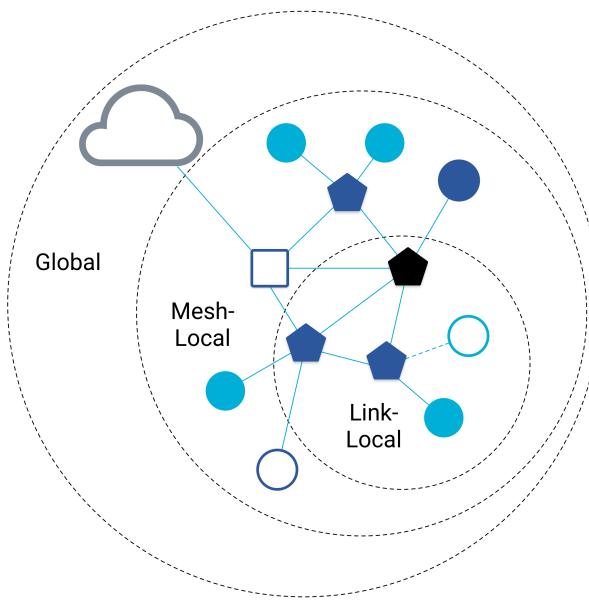


Abbildung 15.4: IPv6 Adressräume [39]

Unicast:

Ein Thread Gerät beinhaltet mehrere IPv6 Adressen, die jede eine andere Funktion hat. Nachfolgend werden alle Adressen kurz aufgelistet:

- **Link-Lokale Adresse** - Adresse ist nur mit einer einzigen Funkübertragung erreichbar (keine Hops). Diese Adresse wird vom Stack verwendet, um direkt benachbarte Knoten zu finden, Routing Informationen auszutauschen und die Verbindungen zu konfigurieren. Präfix: fe80::/16
- **Mesh-Lokale EID** - Adresse wird nach dem Eintritt in das Netzwerk zufällig gewählt und ist eindeutig. Mit dieser Adresse sind alle Knoten auch über mehrere Hops innerhalb des Thread Netzwerkes erreichbar. Diese Adresse wird für Anwendungen innerhalb des Thread Netzwerkes, also ohne Border-Router verwendet. Präfix: fd00::/8
- **Routing Locator** - Diese Adresse lokalisiert die Position des Knoten im Netzwerk. Das bedeutet, die Adresse verändert sich, wenn sich die Topologie im Netzwerk ändert. Der Routing Locator wird für das Routing verwendet.
- **Anycast Locator** - Wird verwendet, falls der Routing Locator nicht verfügbar ist.
- **Globale Unicast Adresse** - Wird als Globale IPv6 Adresse für Anwendungen mit Border Router verwendet. (Öffentliche IPv6 Adresse) Präfix: 2000::/3

Multicast:

Wenn Nachrichten an mehrere Geräte gleichzeitig gesendet werden müssen, wird eine Multicast IPv6 Adresse verwendet. Die Adressen können unter Einhaltung der zwei Präfixe für Multicast und Unicast selbst definiert werden. Es gibt jedoch vier Multicast Adressen, die vordefiniert sind:

IPv6 Adresse	Bereich	Verwendung
ff02::1	Link Lokal	Alle FTDs und MEDs
ff02::2	Link Lokal	Alle FTDs
ff03::1	Mesh Lokal	Alle FTDs und MEDs
ff03::2	Mesh Lokal	Alle FTDs

Tabelle 15.3: Reservierte Multicast Adressen

Anycast:

Falls der Routing Lacator nicht verfügbar ist, werden über vordefinierte Anycast Adressen gewisse Funktionen reserviert. In Tabelle 15.4 sind die Funktionen mit der dazugehörigen reservierten Adresse aufgelistet.

ALOC16	Type
0xfc00	Leader
0xfc01 - 0xfc0f	DCHPv6 Agent
0xfc10 - 0xfc2f	Service
0xfc30 - 0xfc37	Commissioner
0xfc40 - 0xfc4e	Neighbor Discovery Agent
0xfc38 - 0xfc3f 0xfc4f - 0xfcff	Reserviert

Tabelle 15.4: Reservierte Anycast Adressen

15.4.4 Netzwerk Aufbau

Das Thread Netzwerk wird mit drei verschiedenen einzigartigen Identifikatoren definiert [29, Seite 8-18]:

- 2-byte Personal Area Network ID (PAN ID)
- 8-byte Extended Personal Area Network ID (XPAN ID)
- Netzwerkname

Wenn ein Thread Knoten hochfährt und in einem Netz aufgenommen werden will oder ein neues Netz kreieren will, scannt das Gerät aktiv nach einem IEEE 802.15.4 Netzwerk in seiner Reichweite:

1. Das Gerät sendet eine 802.15.4 Beacon-Anfrage von einem spezifischen Channel aus.
2. Im Gegenzug senden alle Router oder router-fähigen Endgeräte in der Umgebung ihre PAN-ID, XPAN-ID sowie den Netzwerknamen in einem Beacon zurück.
3. Schritt 1 und 2 werden für alle Channels wiederholt.

Wenn der Knoten alle Netze in seiner Reichweite gefunden hat, kann sich das Gerät entweder mit einem existierenden Netz verbinden oder eröffnet ein neues Netz.

15.5 Sicherheit

Das Thread Netzwerk wurde so entwickelt, dass während dem Hinzufügen von neuen Geräten und während dem Betrieb ein hohes Mass an Sicherheit gewährleistet ist. Wenn sich ein neues Gerät dem Netzwerk anschliessen möchte, muss dieses mit einem Schlüssel-Vereinbarungs-Mechanismus authentifiziert und autorisiert werden. Sobald das Gerät im Netz aufgenommen wurde, wird jegliche Kommunikation mit einem Netzwerkschlüssel gesichert.

Authentifizierung

Zur Authentifizierung eines neuen Gerätes, dass dem Netzwerk beitreten möchte, wird ein J-Pake(Juggling Password Authenticated Key Exchange) mit elliptischer Kurve verwendet. Mit einem Diffie-Hellmann Schlüsselaustausch, der eine elliptische Kurve für die Berechnung verwendet, wird ein Schlüssel mit dem Netz und dem Gerät, das dem Netzwerk beitreten möchte, festgelegt. Mit dieser Methode wird das Gerät authentifiziert und erhält die nötigen Schlüssel, um dem Netz hinzugefügt zu werden. [29, Seite 1-4]

Netzwerk Schlüssel

Das Thread-Netzwerk wird mit einem netzwerkweiten Schlüssel geschützt. Von diesem Schlüssel werden weitere Schlüssel abgeleitet, um den MAC-Layer mit den IEEE 802.15.4 Nachrichten zu schützen. Dadurch wird das Netzwerk vom Abhören und Unterbrechen von Außenstehenden geschützt.[29, Seite 1-5]

15.6 Thread Software Development Kit

In dieser Arbeit wird OpenThread verwendet. OpenThread ist eine open-sorce Implementation von Thread, die von Google umgesetzt und weiterentwickelt wird. Das Software Development Kit von Nordic Semiconductors beinhaltet eine vorkompilierte Bibliothek des Openthread Stacks. Zuzüglich der Thread Bibliothek befinden sich auch die Normale SDK von Nordic Semiconductor in dem Kit. Alle Module, die für den SoC NRF52840 verfügbar sind, sind ebenfalls darin enthalten. Dank der guten Dokumentation und Community-Foren wurde die Firmware auf der Thread SDK von Nordic aufgebaut.

IDE

Für die Erstellung der Firmware wurde die IDE Segger Embedded Systems verwendet. Die Firma Nordic Semiconductor empfiehlt mit dem Gebrauch der SDK for Thread and ZigBee das Arbeiten mit Segger. Da alle Beispiel Anwendungen und Dokumentationen mit Segger gemacht wurden, war dies die beste Lösung.

16 Umsetzung Benchmark

In diesem Kapitel wird die Thread Stack spezifische Benchmark Umsetzung beschrieben. Das gesamte Projekt ist im [Github-Repository](#)²¹ unter dem Ordner **P6_Software/Openthread/Thread_SDK/custom/P6_Project/ot_benchmark/** verfügbar. Ausserdem sind im Ordner **P6_Software/Openthread/Thread_SDK/custom/benchmark_hex/** die vorkompliierten .zip Dateien abgespeichert, um diese auf den nRF52840 SoC zu flashen.

16.1 Openthread Stack Konfiguration

In nachfolgender Tabelle sind alle Parameter aufgelistet, die verwendet werden, um den Openthread Stack mit CoAP Funktion zu konfigurieren.

Stack Init Time	30000 ms	Zeit, die benötigt wird, um den Stack für den Benchmark zu initialisieren. Reserve Zeit, um das Routing des Netzwerkes durchzuführen.
IEEE Channel	15	Thread Kanal, der verwendet wird, um das Mesh aufzubauen.
Netzwerk Name	OpenThread	Netzwerkname
PAN ID	0xABCD	Personal Area Network ID
Netzwerkschlüssel	0x00112233445566778899AABBCCDDEEFF	Wird für das Precommissioning verwendet.
Radio Modus	radio-on-when-idle	Das Radio Interface wird nie abgeschaltet.
Node Type	FTD	Alle Nodes werden als Full Thread Devices konfiguriert und habe entweder eine Parent oder eine Child Rolle.
Comissioning	Auto-Comissioning	Alle Thread Gerät befinden sich nach der Initialisierung im gleichen Netzwerk.
BSP LED 0	BSP_LED_THREAD_CONNECTION	Zeigt den Verbindungsstatus an.
BSP LED 1	BSP_LED_THREAD_COMMISSIONING	Zeigt den Commissioning Status an.
BSP LED 2	BSP_LED_ALERT	Zeigt an, ob ein Fehler anliegt.
CoAP Port	0x1633	CoAP Port für Nachrichtenübertragung.
URI Path	bm_probe	URI konfiguration für die Identifizierung der CoAP Benchmark Message
Group Address	ff03::Aktuelle Gruppe	Beim Initialisieren wird sogleich die eingestellte Gruppen Multicast Adresse konfiguriert.

Tabelle 16.1: Parameter zu Konfiguration des Stacks

Autocomissioning

Alle Thread Knoten werden mit den Parametern von Tabelle 16.1 initialisiert. Da die Parameter fest programmiert wurden, verbinden sich alle Knoten direkt in das Netzwerk namens OpenThread. Somit entfällt ein Comissioning-Prozess, was das Aufstarten extrem beschleunigt. Trotzdem ist ein Zeitfenster von 30s hinterlegt, bevor der Benchmark startet, da die einzelnen Knoten zuerst ihre Rolle bestimmen müssen und den Routing-Prozess durchlaufen müssen.

²¹ https://github.com/Rouben94/P6_Software[1]

Channel Wahl

Die Openthread API stellt eine automatische Channel Wahl namens Channel Manager zur Verfügung. Der Channel Manager führt ein Qualitätscheck aus und vergleicht die verfügbaren Channels. Nachdem der Channel mit der besten Qualität ausgewählt wurde, wechseln alle Knoten den Channel zu dem vom Channel Manager vorgegebenen Channel. Leider konnte die Funktion nicht ausreichend geprüft werden und der Channelwechsel funktionierte nicht immer bei allen Nodes. Aus diesem Grund wurde der Channel Manager deaktiviert und ein spezifischer Channel ausgewählt, auf dem der Benchmark läuft. Wie in Abbildung 16.1 ersichtlich ist Channel 15 auf der Frequenz 2425MHz am wenigsten von anderen Quellen wie W-Lan oder BLE belastet. Somit wurde Channel 15 als Standard Channel ausgewählt.

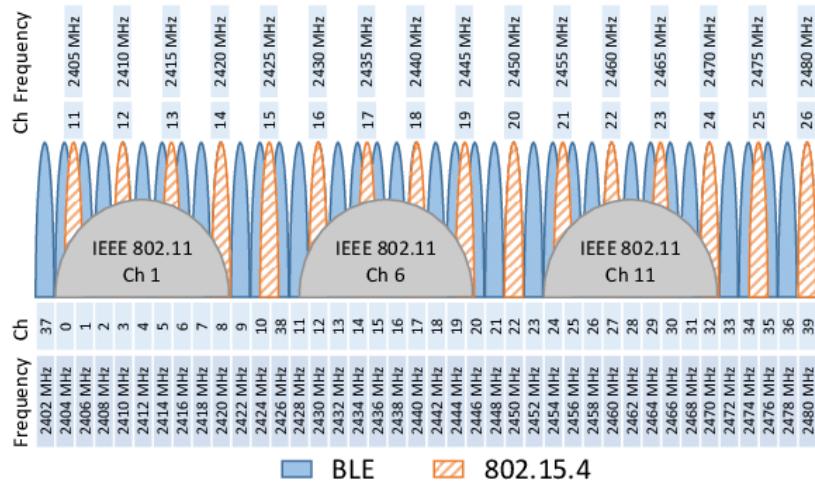


Abbildung 16.1: Channel Bandbreite zand_fig_nodate

16.2 Openthread Stack Initialisierung

Der Openthread Stack wird, wie in Abbildung 16.2 ersichtlich, mit den bereits erläuterten Parametern von Kapitel 16.1 initialisiert:

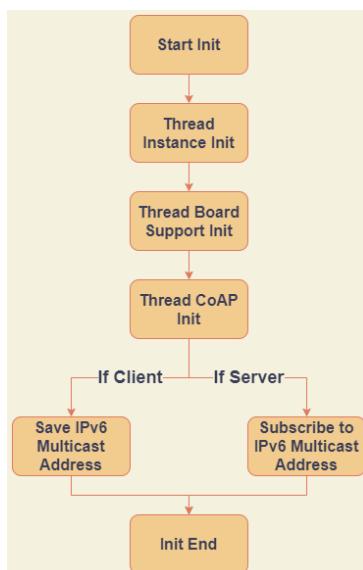


Abbildung 16.2: Initialisierung Thread Stack

16.3 CoAP Benchmark Message

Um die Benchmark Message vom Client an den Server zu senden, wurde ein CoAP-Protokoll implementiert. Das Protokoll ist im Kapitel 15.1.1 näher erläutert. Nachfolgend wird beschrieben, welche Funktionen implementiert wurden.

Default Handler

Der Default-Handler ist eine Callbackfunktion, die jede CoAP Applikation beinhalten muss. Bei der Initialisierung von CoAP muss eine Callbackfunktion mitgegeben werden. Diese wird aufgerufen, falls die CoAP Nachricht nicht zugeordnet werden kann. Falls letzteres der Fall sein sollte, bedeutet dies zwar, dass die Adresse der Nachricht stimmt, jedoch der Uniform Resource Identifier (URI) der Nachricht nicht registriert wird. Der URI wird verwendet, um die Nachricht zu identifizieren.

Probe Message Send

Nur Clientseitig implementiert ist die Funktion `bm_coap_probe_message_send`. Mit dieser Funktion wird eine neue Nachricht generiert und an die entsprechende Multicast Adresse gesendet. Zusätzlich zu der Adresse wird ein CoAP Port benötigt. Dieser ist standartmäßig auf die Portadresse `0x1633` festgelegt. Folgende Parameter werden mit der Nachricht gesendet:

- **URI** - `bm_probe`
- **Message ID** - Die aktuelle Message ID, um die Nachricht identifizieren zu können. (1 byte)
- **Source Address** - Die Adresse des Clients, damit in der Auswertung die Gruppenbeziehungen ausgewertet werden können. (2 byte)
- **Payload** - Zusätzliche Payload, die mit zufälligen Daten gefüllt wird (1 - 1021 byte)

Bevor die Nachricht gesendet wird, werden folgende Parameter ausgelesen und ins RAM gespeichert:

Parameter	Erklärung
Source Address	EID Adresse vom Client
Destination Address	Zieladresse, an den die Nachricht gesendet wird.
Group Address	Aktuelle Gruppenadresse, die initialisiert ist.
Message ID	Aktuelle Message ID
Network Time	Aktuelle Netzwerk Zeit, die mit der Zeitsynchronisation verfügbar ist.
Payload Length	Zusätzliche Payload, die mitgegeben werden kann.

Tabelle 16.2: Messparameter Client Thread

Probe Message Handler

Um die Benchmark Message auf dem Server zu erhalten, wird eine Callbackfunktion mit dem entsprechenden URI (*bm_probe*) initialisiert. Wird die Funktion *bm_probe_message_handler* nach Eingang einer validen CoAP Nachricht aufgerufen, so werden folgende Parameter entweder aus der Nachricht ausgelesen oder vom System abgerufen und ins RAM gespeichert:

Parameter	Erklärung	In der Payload mitgegeben
Source Address	EID Adresse vom Client	X
Destination Address	EID Adresse vom Server	
Group Address	Aktuelle Gruppenadresse, die initialisiert ist.	
Message ID	Aktuelle Message ID	X
Network Time	Aktuelle Netzwerk Zeit, die mit der Zeitsynchronisation verfügbar ist.	
Payload Length	Zusätzliche Payload, die mitgegeben werden kann.	
RSSI	RSSI Durchschnittswert über alle Hops	
Number od Hops	Anzahl Hops die die Nachricht genommen hat	

Tabelle 16.3: Messparameter Server Thread

16.4 IPv6 Adressierung

Die einzelnen Knoten im Benchmark Test sollten in Gruppen konfiguriert werden, da dies der Praxis einer Heimautomation entspricht. Dort können verschiedene Schalter mehrere Leuchten ein- und ausschalten. Hierfür bietet sich die Multicast Adressierung von Thread an, da mehrere Nachrichten gleichzeitig versendet werden können, wie im Paragraph ?? beschrieben. Um diese Gruppenadressierung mit Multicast Adressen umzusetzen, wurden 25 verschiedene Adressen vordefiniert:

Gruppen Nummber	Multicast Adresse
1	ff03::3
2	ff03::4
3	ff03::5
4	ff03::6
...	...
25	ff03::27

Tabelle 16.4: Gruppenadressierung Thread

17 Analyse Thread Stack

In diesem Kapitel wird die persönliche Erfahrung mit dem Mesh-Stack zu Tage gelegt, die während des ganzen Entwicklungsprozesses entstanden ist.

17.1 Vor- und Nachteile Openthread Stack

Nachfolgend werden die wichtigsten Vor- und Nachteile aufgezeigt, die sich als persönliche Erfahrungen beschreiben lassen.

Vorteile

Der Openthread Stack von Google Nest ist auf der Webseite [Openthread](https://openthread.io/)²² sehr gut beschrieben. Es gibt eine kurze Einleitung wie der Stack funktioniert, ohne dabei zu grob ins Detail zu gehen. Weiter sind fantastische Tutorials auf der Webseite verfügbar, mit denen man beginnt ein Thread Netzwerk zu simulieren. Danach wird man Schritt für Schritt angeleitet, wie man mit einem SoC ein reales Netzwerk aufbaut. Zum Schluss, wenn man das minimale Netzwerk erfolgreich aufgebaut hat, wird man auf die API verwiesen. Diese ist auch sehr ausführlich dokumentiert und man kann alle Funktionen, die benötigt werden, mit einer Suchfunktion herausfinden.

Nachteile

Ein grosser Nachteil des Projekts von Google Nest ist die Beschreibung der API. Die Dokumentation ist zwar lückenlos, jedoch wäre es sehr praktisch einige Beispiele der wichtigsten Funktionen zu haben. Meistens ist es schwierig, die API Funktionen zu implementieren, da mehrere Parameter eingestellt werden müssen. Dies könnte Google Nest in einem kurzen Beispiel gut ersichtlich machen. Ein weiterer Nachteil ist das Auslesen der Message ID. Dies ist nicht immer zuverlässig möglich. Dadurch musste mit der Payload der Benchmark Message eine Message ID mitgegeben werden. Aus diesem Grund wurde die Messung mit Acknowledgement nicht umgesetzt, da die Message ID nicht mit der Acknowledge Nachricht mitgegeben werden konnte. Daraus resultiert, dass die Nachrichten nicht identifiziert werden können.

17.2 FreeRTOS Projekt

Auf dem [Github-Repository](https://github.com/Rouben94/P6_Software)²³ ist ein Branch namens openthread_freertos verfügbar. Das Openthread Projekt wurde zuerst auf Basis von FreeRTOS entwickelt und ist grundsätzlich voll funktionsfähig. Es gab ein grosses Problem, das bis zum Schluss nicht behoben werden konnte, weshalb das Projekt in die Shared Lib 9.3 implementiert werden musste. Das Problem war, dass sich ab ca. 10 Knoten Netzwerkgrösse der SoC nRF52840 nach einiger Zeit in einen Hardfault verfing. Das Problem trat sporadisch und sehr zufällig auf. Es wird vermutet, dass dies auf die RAM Benutzung des SoCs zurückzuführen ist. Laut Google Nest vergrössert sich das RAM mit steigender Anzahl Knoten im Netzwerk. Da das RAM schon zu 50% mit dem Projekt belegt wird, wäre es möglich, dass dies der Grund ist für den sporadisch auftretenden Hardfault des SoCs.

²²<https://openthread.io/>

²³https://github.com/Rouben94/P6_Software[23]

Teil V

Zigbee

Autor: Cyrill Horath

18 Einleitung

In diesem Teil der Arbeit werden die Eigenschaften und Besonderheiten des Zigbee Mesh Stacks erläutert und es wird auf die Umsetzung des Benchmarks auf der Stack Ebene eingegangen. Es handelt sich um einen eigenständigen Teil in dem ausschliesslich der Zigbee Protokollstack behandelt wird.

Zigbee ist ein auf dem *IEEE 802.15.4* Standard aufbauendes, drahtloses Low Power Mesh Netzwerk Protokoll. Es nutzt das im *IEEE 802.15.4* Standard definierte ISM-Funkfrequenzband 2.4GHz und weitere Sub-GHz Bänder je nach Region und Bedarf. Die im Jahre 2002 gegründete Zigbee Allianz ist Herausgeber der Zigbee Protokoll Spezifikation. Unterdessen ist der Protokollstack bereits in der Version 3.0 spezifiziert. Im Zuge der Verbreitung von Technologien in der Heim-Automatisierung erhielt auch Zigbee immer mehr Aufmerksamkeit und wuchs bis heute zudem am weitesten verbreiteten Mesh Netzwerk Protokoll in diesen Gebiet heran. Besonders in Systemen für die Steuerung von Beleuchtungen wie zum Beispiel Phillips Hue oder Ikea Tradfri kommt Zigbee verbreitet zum Einsatz.

Die Spezifikationen innerhalb des Zigbee Protokollstacks sind weitreichend. Von der MAC Ebene über die Netzwerkschicht bis hin zur Applikationsebene gibt es klare Vorgaben, wie ein Zigbee Produkt aufgebaut sein soll (siehe Abbildung 18.1). Mit der *Zigbee Cluster Library* werden sogar spezifische Anwendungen vordefiniert wie beispielsweise die Steuerung einer Lichtquelle mit Dimmfunktion. Diese Spezifikationen ermöglichen die Interoperabilität von Systemen mit der gleichen Funktion von unterschiedlichen Herstellern.

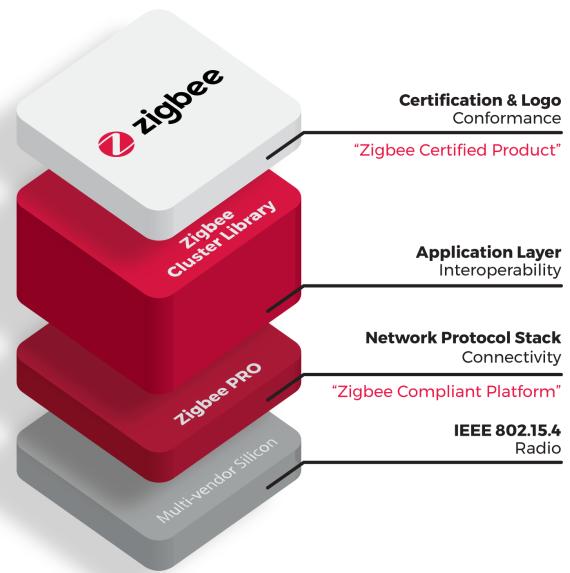


Abbildung 18.1: Zigbee Protokollstack

19 Technische Grundlagen Zigbee

19.1 Netzaufbau und Topologie

Zigbee ist nicht gleich Zigbee. Obschon Zigbee von der zentralen Stelle der Zigbee Alliance spezifiziert wurde, gibt es eine Vielzahl an Versionen und Umsetzungsvarianten. In den Spezifikationen der Zigbee Alliance wird zwischen zwei sogenannten Stackprofilen *ZigBee* und *ZigBee PRO* unterschieden. Während *ZigBee*-Netzwerke eine Baumstruktur bilden und der Koordinator dabei einen Single-Point-of-Failure darstellt, haben *ZigBee PRO*-Netzwerke geroutete Mesh Funktionalitäten mit Routing Tabellen und Wegentdeckung. Der Koordinator bildet dabei nicht länger einen Single-Point-of-Failure, da sich das Routing dynamisch anpassen kann. Abbildung 19.1 zeigt die Unterschiede eines Baumnetzwerks im Stackprofil *ZigBee* links, und eines Meshnetzwerks im Stackprofil *ZigBee PRO* rechts. In der vorliegenden Arbeit wurde ausschliesslich das *ZigBee PRO*-Stackprofil verwendet, womit vollwertige Meshnetzwerke möglich sind.

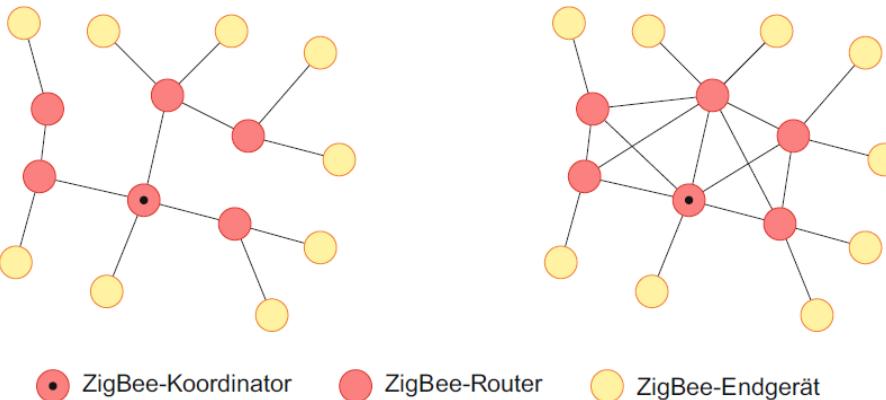


Abbildung 19.1: Zigbee Baumnetzwerk links und Meshnetzwerk rechts [40, S. 221]

Wie in Abbildung 19.1 bereits angedeutet, kann innerhalb eines Zigbee Meshnetzwerkes zwischen drei Node Rollen unterschieden werden. Diese besitzen unterschiedliche Aufgaben und Eigenschaften und sind wie nachfolgend beschrieben, spezifiziert:

Zigbee Koordinator

Als zentrale Einheit übernimmt der *Zigbee Koordinator*, Aufgaben wie den Aufbau und die Verwaltung eines WPAN (Wireless Personal Area Network) inkl. der Definition der wichtigsten Parameter, wie der PAN-ID, den Sicherheitsschlüsseln sowie der Wahl des IEEE Channels. In einem Zigbee-Netzwerk gibt es genau ein Gerät das die Rolle des *Zigbee Koordinators* einnimmt. Welches diese ist, wird vom Anwender respektive Entwickler bestimmt. Wenn dieses Gerät das Netzwerk verlässt oder kurzzeitig ausser Betrieb ist, kann das Netzwerk weiter bestehen und wie bisher betrieben werden. Für die Aufnahme von zusätzlichen Nodes oder um beispielsweise die Sicherheitsschlüssel zu aktualisieren, muss der Koordinator wieder mit dem Netz verbunden werden. Jeder *Zigbee-Koordinator* besitzt gleichzeitig auch die Rolle eines *Zigbee-Router*. [40]

Zigbee Router

Zigbee-Router bilden das eigentliche Meshnetzwerk wie es in Abbildung 19.1 schematisch dargestellt ist. Sie übernehmen die Aufgabe des Routings und leiten Pakete innerhalb des Netzwerkes weiter. Durch Wegentdeckungsanfragen, werden Routing-Tabellen aufgebaut und fortlaufend aktualisiert. Diese Routing-Tabellen sind entscheidend für den gerouteten Versand von Datenpaketen. *Zigbee-Router* sind zudem potentielle Zugriffspunkte zum Netzwerk für *Zigbee End-Devices*. [40]

Zigbee End-Device

Die einfachste Rolle in einem Zigbee Netzwerk ist jene des *Zigbee End-Device*. *Zigbee End-Devices* stehen in einer Parent-Child Beziehung mit einem *Zigbee-Router*. Diese Kommunikation findet entweder periodisch oder ausgelöst durch einen Userinput statt. Ankommende Pakete werden jeweils vom Parent-Node gespeichert bis das *Zigbee End-Device* diese abruft. *Zigbee End-Devices* besitzen außerdem keine Routing Funktionen und gelten deshalb als sehr energiesparend. Ausgeführt als *Sleepy-End-Device* können CPU und RAM des entsprechenden Nodes ganz oder teilweise heruntergefahren und durch periodische Interrupts geweckt werden. Dadurch können noch längere Batteriestandzeiten erreicht werden. In der Anwendung werden beispielsweise Lichtschalter als *Sleepy-End-Device* ausgeführt die keine drahtgebundene Energieversorgung besitzen. [40]

19.2 Zigbee Protokoll Stack

Die Architektur des Zigbee Stacks besteht aus vier Layern, dem Physical Layer (PHY), dem Media Access Control Layer (MAC), dem Network Layer (NWK) und dem Application Layer (APL). Abbildung 19.2 zeigt den Aufbau des Protokoll Stacks. Jede der Schichten ist mit bestimmten Aufgaben betraut und stellt der darüber liegenden Schicht die notwendigen Daten und Dienste bereit. Nachfolgend wird auf die vier Schichten des Zigbee Stacks eingegangen und deren Aufgabe und Funktionsweise kurz erläutert.

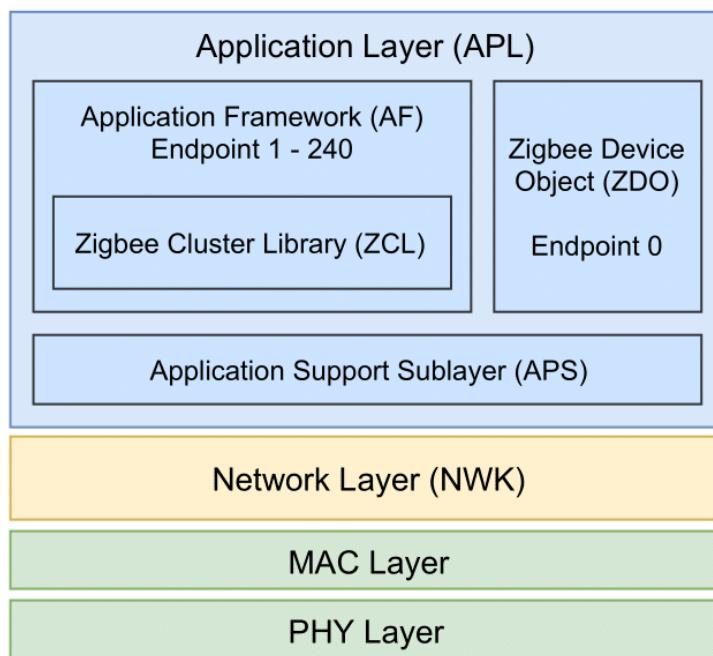


Abbildung 19.2: Architektur des Zigbee Protokoll Stacks

19.2.1 MAC und PHY Layer

Der MAC wie auch der PHY Layer werden im Zigbee Protokoll Stack gebildet durch den *IEEE 802.15.4* Standard für *Wireless Personal Area Networks (WPAN)*. Während beispielsweise Wifi oder Bluetooth, die auf dem selben 2.4 GHz ISM-Funkfrequenzband betrieben werden können, für hohe Datenübertragungsraten konzipiert wurden, ist dieser Standard für kleinere Datens Mengen optimiert. Durch die Vermeidung von unnötigen Steuerinformationen, kann der *IEEE 802.15.4* Standard auf einfacherer Hardware realisiert und mit kleinstem Energieaufwand betrieben werden. Dies ist ideal für sogenannte *Wireless Sensor Networks (WSN)*. Zigbee ist nur

eines von vielen Protokollen die diesen Standard benutzen. MAC und PHY Layer sind für die physikalische Datenübertragung von einem Node zum Anderen zuständig. Dazu besitzt jedes Funkmodul eine einmalige 48-Bit MAC Adresse mit welcher das Gerät eindeutig identifiziert und adressiert werden kann. [41]

19.2.2 Network Layer (NWK)

Der NWK Layer ist im Zigbee Stack verantwortlich für den Aufbau sowie das Management der Netzwerkfunktionen und das Routing innerhalb dieses Netzwerkes. Im NWK Layer wird das eigentliche Mesh gebildet und unterhalten. Dazu gehören die beiden Hauptaufgaben *Netzaufbau und Adressierung* sowie das *Routing*.

Netzaufbau und Adressierung

Wie unter 19.1 bereits erwähnt, ist der Koordinator verantwortlich für den Aufbau des Zigbee Netzwerks und der Wahl von entsprechend geeigneten Parametern. Dazu gehört beispielsweise eine 16-Bit PAN-ID sowie die Wahl eines möglichst störungsfreien Funkkanals. Beim Beitritt eines neuen Funkmoduls, wird diesem durch den Koordinator eine im Netzwerk einmalige 16-Bit *Short-Address* zugewiesen. Anhand dieser kann das Funkmodul nun im Netzwerk adressiert werden und es selbst kann damit Routing Funktionen wahrnehmen. Die im MAC Layer definierte 48-Bit MAC Adresse wird im NWK Layer zu einer statischen 64-Bit *Long-Address* gewandelt. Diese kann im Zigbee Stack ebenfalls für die Adressierung verwendet werden. In einer *Address-Table* sind die statischen *Long-Addresses* und die dynamischen *Short-Addresses* einander eindeutig zugewiesen. Für die Adressierung im NWK Layer und für das Routing wird ausschliesslich die *Short-Address* verwendet.

Routing

Innerhalb von Zigbee Mesh Netzwerken, welche das *ZigBee PRO* Stackprofil verwenden, werden durch jeden *Zigbee-Router*, Routingtabellen erstellt. Falls sich das Netzwerk verändert, werden auch diese Routing-Tabellen nachgeführt. In den Routing-Tabellen ist die *Short-Address* des Zielenodes sowie die zugehörige *Next-Hop-Address* zum Ziel hinterlegt. Eine zentrale Einheit die das Routing übernimmt gibt es dementsprechend nicht. Enthält die Routingtabelle veraltete Einträge oder sind für das entsprechende Ziel noch keine Informationen vorhanden, muss ein *Route Discovery* durchgeführt werden. Hierbei handelt es sich um eine Broadcast Nachricht welche an alle Router gesendet wird. Die Router in unmittelbarer Umgebung empfangen die Nachricht und leiten sie, wiederum als Broadcast, an alle Router in ihrer Reichweite weiter. Dabei werden die Wegkosten jeweils addiert um diese, sobald die Nachricht beim Zielenode angekommen ist, dem Absender mitzuteilen. So wird der Weg mit den geringsten totalen Wegkosten ermittelt und in der Routingtabelle abgelegt.

19.2.3 Application Layer (APL)

Der Zigbee Application Layer kann in drei Teile unterteilt werden, den Application Support Sublayer, das Zigbee Device Object und das Application Framework mit der Zigbee Cluster Library in der die eigentliche Anwendung definiert ist.

Application Support Sublayer (APS)

Wie der Name andeutet ist der APS Layer für die Anwendungsunterstützung zuständig und als Zwischenschicht im Application Layer eingebettet. Zu den Aufgaben des APS Layers gehören das *Binding*, das *Group-Management*, die Datenübertragung inkl. Fragmentierung sowie die Erstellung und der Versand von APS-Frames. Ausserdem bietet der APS Layer eine Möglichkeit der Empfangsbestätigung auf Applikationsebene. Anders als die Empfangsbestätigung auf MAC Ebene ist diese nicht nur auf Funkmodule in unmittelbarer Reichweite beschränkt.

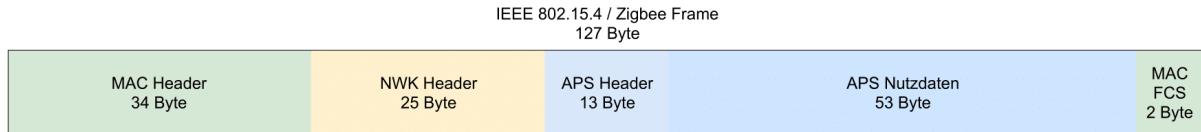


Abbildung 19.3: Zigbee Frame Struktur bei aktivierten Sicherheitsfunktionen [40, S. 286]

Die Fragmentierung von APS Nutzdaten basiert darauf, dass die Grösse von Frames durch den PHY Layer auf 127 Byte beschränkt ist. Abzüglich sämtlicher Header auf MAC, NWK sowie APS Ebene und sonstigem Overhead des Protokolls, beispielsweise für Sicherheitsfunktionen (siehe Abschnitt 19.2.4), reduziert sich die nutzbare APS-Payload Grösse auf 53 Byte. In Abbildung 19.3 ist die Struktur eines kompletten Zigbee Frames dargestellt. Sollen nun grössere Daten übertragen werden, führt der APS Layer eine Fragmentierung durch [40, S. 279 - 299].

Application Framework (AF) mit Zigbee Cluster Library (ZCL)

Das Application Framework bildet den Bereich des APL in dem die eigentliche Anwendung abläuft. Für die Adressierung stehen dem Anwender dabei 240 sogenannte Endpoints zur Verfügung. Endpoints können mit dem Prinzip von Ports im TCP/IP Modell verglichen werden. Sie dienen dazu unterschiedliche Anwendungen auf dem selben Node zu adressieren. Innerhalb des AF können Anwendungen nun prinzipiell frei umgesetzt werden. Um den Anwendern jedoch eine herstellerunabhängige Plattform bieten zu können, wurde durch die Zigbee Alliance die *Zigbee Cluster Library (ZCL)* spezifiziert. Anwendungen wie beispielsweise ein Lichtschalter oder eine Lampe, werden durch sogenannte Cluster detailliert beschrieben. Dabei handelt es sich um eine Sammlung von Kommandos und Attributen für den jeweiligen Anwendungszweck. [42]

Zigbee Device Object (ZDO)

Das Zigbee Device Object ist ein eigenständiges Anwendungsobjekt welches immer mit dem Endpoint 0 addressiert wird. Es setzt die Funktionalitäten gemäss Definition der Zigbee Rollen (Koordinator, Router, End-Device) um, und benutzt dafür Funktionen der NWK sowie APS Schicht. Beispielsweise ist es zuständig für das Netzwerkmanagement, Knotenmanagement und auch für die Implementation von Sicherheitsfunktionen.

19.2.4 Sicherheit

Die drahtlose Übertragung in einem WPAN ist vom Grundsatz her anfälliger für Angriffe oder Manipulationen wie eine drahtgebundene Kommunikation. Deshalb ist die Implementation von Sicherheitsfeatures in einem *Low Power Mesh Network* von grosser Bedeutung. Zigbee verwendet ein *CCM*²⁴ Verfahren auf MAC wie auch auf NWK und APS Ebene. Dieses ist vom *IEEE 802.15.4* Standard für die Verschlüsselung und Authentifizierung spezifiziert. *CCM* ist ein Verfahren für eine mehrmalige Blockverschlüsselung welches mit einem einzigen Schlüssel auskommt. Das Verfahren nutzt den *AES*²⁵-Verschlüsselungsalgorithmus als Blockverschlüsselung.

Sicherheitsstufen

Im *CCM* Verfahren können 8 verschiedene Sicherheitsstufen gemäss Tabelle 19.1 definiert werden. Die Sicherheitsstufe 0 deaktiviert sämtliche Sicherheitsfunktionen. Die Stufen 1 bis 3 fügen dem MAC Frame eine *MIC*²⁶ Prüfsumme von steigender Grösse hinzu, wodurch die Grösse des Frames zunimmt. Erst ab Stufe 4 werden die Nutzdaten des MAC-Frames verschlüsselt und die Stufen 5 bis 7 mit einer *MIC* Prüfsumme ergänzt. [40, S. 334]

²⁴Counter mode encryption and Cipher block chaining Message authentication code

²⁵Advanced Encryption Standard

²⁶Message Integrity Code

Sicherheitsstufe:	Verschlüsselung:	Authentifikation:
0	Nein	Nein
1	Nein	32-Bit <i>MIC</i>
2	Nein	64-Bit <i>MIC</i>
3	Nein	128-Bit <i>MIC</i>
4	Ja	Nein
5	Ja	32-Bit <i>MIC</i>
6	Ja	64-Bit <i>MIC</i>
7	Ja	128-Bit <i>MIC</i>

Tabelle 19.1: Sicherheitsstufen in Zigbee [40, S. 334]

Schlüssel

Für das *CCM* Verfahren in Zigbee Netzwerken, werden zwei verschiedene Schlüsseltypen eingesetzt, der Netzwerkschlüssel und sogenannte Linksschlüssel. In einem Zigbee Netzwerk mit normalem Sicherheitsmodus²⁷ wird einem Node die Rolle des *Trustcenters* zugeordnet. Üblicherweise übernimmt diese Rolle der *Zigbee-Koordinator*. Der Trustcenter-Node ist zuständig für die Verteilung der Sicherheitsschlüssel. Beim Beitritt eines Nodes zum Netzwerk, überträgt das *Trustcenter* den Netzwerkschlüssel über einen unverschlüsselten Kanal. Dieser Netzwerkschlüssel wird nun für die Kommunikation zwischen Node und *Trustcenter* benutzt. Um Angriffe zu verhindern, wird der Netzwerkschlüssel durch das *Trustcenter* regelmässig erneuert. Dazu werden sogenannte Schlüsselwechsel-Kommandoframes versendet. Für die Verschlüsselung von End-zu-End Verbindungen auf APS Ebene werden die Linksschlüssel eingesetzt, welche nur über die bereits gesicherte Verbindung vom Trustcenter zum Node übertragen werden. Diese Linksschlüssel sind nur den beteiligten Nodes bekannt und werden üblicherweise durch das *Trustcenter* ausgestellt. Eine Alternative dazu wäre, dass die Linksschlüssel auf den entsprechenden Funkmodulen bereits vorinstalliert sind.

²⁷Die Alternative zum Normalen Sicherheitsmodus wäre der Hohe Sicherheitsmodus. [40, S. 338]

20 Umsetzung Benchmark

Die Umsetzung des Benchmarks geschah im Rahmen der Anforderungen des Benchmark Konzepts gemäss Abschnitt 7. Wie im Abschnitt 9 bereits ausführlich behandelt wurde, sind innerhalb der Shared Library (siehe Abschnitt 9.3) klare Schnittstellen und Grenzen für die Mesh Protokollstacks definiert. In den nachfolgenden Abschnitten wird auf die Umsetzung des Zigbee Stacks innerhalb dieser Grenzen eingegangen.

Das gesamte Zigbee Softwareprojekt ist im [Github-Repository²⁸](https://github.com/Rouben94/P6_Software) zu diesem Projekt unter dem Ordner **P6_Software/Zigbee/zigbee_benchmark**/ abgelegt. Darin enthalten ist je ein Projektordner für den *Benchmark-Master*, *Benchmark-Server* und *Benchmark-Client*. Außerdem sind im Ordner **P6_Software/Zigbee/zigbee_benchmark/hex/** die vorkompilierten .hex Dateien abgelegt, welche direkt auf die nRF52840 SoC geflasht werden können.

20.1 Zigbee Software Development Kit

Für die Umsetzung von Zigbee Applikationen auf dem nRF52840 SoC stellt Nordic Semiconductor mit der *nRF5 SDK for Thread and Zigbee²⁹* ein eigenes Software Development Kit (SDK) zur Verfügung. Da Thread und Zigbee den selben *IEEE 802.15.4* MAC Layer benutzen, teilen sich die beiden Protokollstacks eine gemeinsame SDK. Innerhalb der SDK nutzen Thread und Zigbee jedoch unterschiedliche Ressourcen.



Abbildung 20.1: nRF5 SDK for Thread and Zigbee Plattform Design Referenz für Zigbee Applikationen [21]

Für Zigbee Applikationen kommt ein Plattform Design gemäss Abbildung 20.1 zum Einsatz. Als eigentlicher Zigbee Stack wird der *ZBOSS* Zigbee Stack in der Version 3.3.0 von DSR³⁰ implementiert. Dieser steht als vorkompilierte Library zur Verfügung. Es handelt sich dabei um einen weit verbreiteten *Zigbee PRO* Stack welcher die neuste Version der Zigbee Core Specification rev22 umsetzt. *ZBOSS* verwendet dabei ein kooperatives Multitasking Prinzip mit einem Scheduler welche die Tasks verwaltet.

²⁸https://github.com/Rouben94/P6_Software[23]

²⁹<https://www.nordicsemi.com/Software-and-tools/Software/nRF5-SDK-for-Thread-and-Zigbee>[21]

³⁰<https://dsr-iot.com/> [43]

Unter dem *ZBOSS* Stack verwendet die SDK den *nRF IEEE 802.15.4 radio driver* für die Ansteuerung der Funkschnittstelle. Über dem *ZBOSS* Stack befindet sich die Applikationsebene in welcher Applikationen gemäss den ZCL Spezifikationen³¹ umgesetzt werden können. *ZBOSS* stellt die dafür notwendigen Funktionen zur Verfügung.

20.2 Stack Implementation

Der Zigbee Stack wurde mit Hilfe der *nRF5 SDK for Thread and Zigbee* wie oben beschrieben implementiert. Dazu mussten einige Punkte bei der Konzeptionierung beachtet werden. Diese Einzelheiten werden in den nächsten Abschnitten erläutert.

20.2.1 Topologie

Das Zigbee Netzwerk für den Benchmark wurde mit Hilfe des *ZigBee PRO* Stackprofils als vollständiges Mesh Netzwerk aufgebaut (siehe Abschnitt 19.1). Um dabei Resultate erhalten zu können die mit jenen der beiden anderen Mesh Protokolle vergleichbar sind, wurden ausschliesslich *Zigbee-Router* konfiguriert. Total ergibt dies ein Mesh Netzwerk mit 50 *Zigbee-Routern* und einem *Zigbee-Koordinator* welcher gleichzeitig ebenso als *Zigbee-Router* fungiert. Auf den Einsatz von *Zigbee End-Devices* wurde verzichtet da diese für die Performance des Netzes, im gewählten Messkonzept (siehe Abschnitt 7) irrelevant sind.

20.2.2 Wahl des Funkkanals

Wie schon einige Male erwähnt, ist die Konkurrenz im 2.4GHz ISM Band gross. Aus diesem Grund ist die Wahl des Funkkanals entscheidend für die Performance jedes Protokolls. Zigbee respektive der *IEEE 802.15.4* Standard bietet die Möglichkeit den besten Funkkanal je nach Belastung automatisch zu bestimmen und auch während dem Betrieb zu ändern. Dies bringt für den Benchmark jedoch einige Probleme, in Form von inkonstanten Bedingungen, mit sich. Deshalb wurde der Funkkanal für das Zigbee Netzwerk fix hinterlegt.

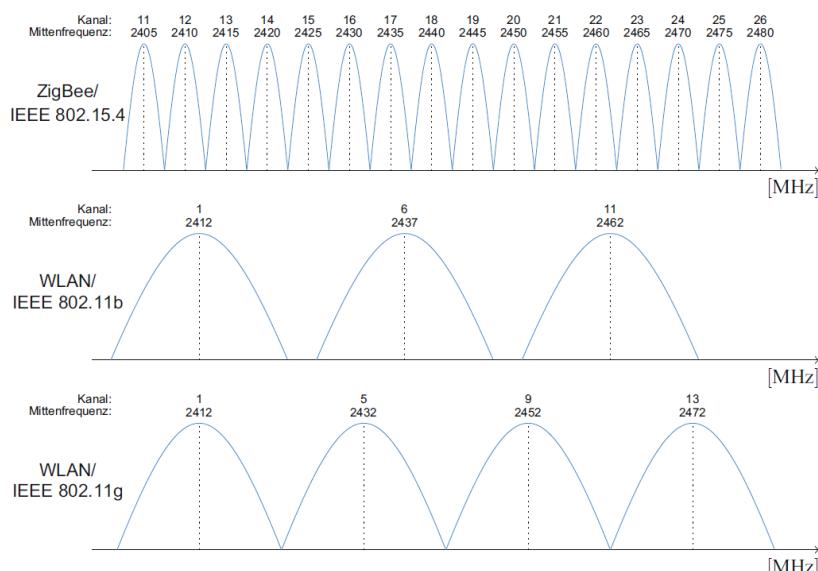


Abbildung 20.2: Konkurrenz Zigbee/IEEE und WLAN Funkkanäle [44]

³¹ <https://zigbeealliance.org/solution/zigbee> [42]

Abbildung 20.2 zeigt wie die Kanäle von Zigbee/IEEE 802.15.4 mit den WLAN Kanälen in Konkurrenz stehen. Da in den Messumgebungen gemäss Abschnitt 7.2 eine störende Beeinflussung durch WLAN Netze zu erwarten ist, wurde Kanal 15 mit einer Mittenfrequenz von 2.425 GHz für die Benchmarks gewählt. Dieser Kanal liegt genau zwischen den potentiell störenden WLAN Kanälen.

20.2.3 ZCL Level Cluster

Der Performancevergleich der Mesh Protokolle soll gemäss Benchmark Konzept (siehe Abschnitt 7) auf Applikationsebene durchgeführt werden. Wie im Abschnitt 19.2.3 beschrieben, wird diese gebildet durch die *Zigbee Cluster Library (ZCL)*. Für den Versand von Benchmark Nachrichten wurde deshalb der *ZCL Level control cluster* implementiert. Dieser definiert Attribute und Funktionen zur Steuerung eines Gerätes, welches einen Level Wert annehmen kann. Beispielsweise die Helligkeit einer dimmbaren Lampe. Die verwendete SDK stellt den *ZCL Level control cluster* standardmäßig zur Verfügung.^[42]

20.2.4 Endpoint Handler

Der Empfang und die Verarbeitung von *ZCL* Nachrichten wird in der *nRF5 SDK for Thread and Zigbee* durch einen Callback Handler realisiert. Dieser wird beim Start der Anwendung einmalig registriert und ausgelöst durch ein Callback-Ereignis der Radio Schnittstelle. Im Zigbee Benchmark wurde ein *Costum Endpoint Handler* implementiert, mit welchem die Benchmark Nachrichten ausgewertet werden können. Nachrichten welche an den *Benchmark-Server-Endpoint* (siehe Abschnitt 19.2.3) adressiert sind, werden in diesem Handler verarbeitet. Die Registration des Handlers erfolgt durch den entsprechenden API Aufruf (siehe Abschnitt sub-subsec:ZigbeeStackInitundKonfiguration).

20.2.5 APS Frame

Während eines Benchmark Vorganges werden mit jeder Benchmark Nachricht die notwendigen Messgrößen gemäss Abschnitt 7.4 generiert. Dazu muss der *Server Node*, also der Empfänger einer Nachricht, folgende Werte aus den Headern des Pakets auslesen:

- **Source Address:** *Short-Address* des Senders
 - **Message ID:** Sequenznummer der empfangenen Benchmark Nachricht.

Diese Daten können dem APS Header des Pakets, welcher in Abbildung 20.3 dargestellt ist, entnommen werden. Die Message ID wird als *uint_16t*-Wert im *Manufacturer Specific* Feld des erweiterten APS Header übermittelt. Damit können 65535 Benchmark Nachrichten eindeutig identifiziert werden. Die eigentliche Sequenznummer des ZCL Pakets ist nur ein *uint_8t*-Wert und kann deshalb nur bis zu einem Maximalwert von 255 inkrementiert werden. Die Identifikation der Benchmark Nachrichten wäre somit aufwendiger.

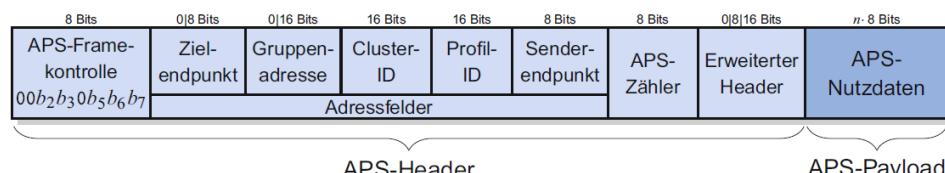


Abbildung 20.3: Aufbau APS Datenframe [44]

20.2.6 Adressierung

Die Adressierung der Nodes innerhalb des Zigbee Mesh Benchmarks kann entweder Unicast oder Multicast erfolgen. Eine Multicast Adressierung wird mit Hilfe des *ZCL Groups cluster* umgesetzt. Dazu wird der Cluster auf den entsprechenden Nodes implementiert und diese werden mittels *ZCL Add Group Request*-Befehl einer gewissen Gruppe hinzugefügt. Eine solche Adressierung kann durch die Angabe einer *Group Number*, in der Konfiguration der Benchmark Nodes gemäss Abschnitt 9.3.7, definiert werden. Funktionstests zu Beginn der Benchmark Messungen haben allerdings gezeigt, dass diese Multicast Adressierung auf diesem Wege nicht praktikabel ist (siehe Abschnitt 21.2). Deshalb wurden die Zigbee Benchmark Messungen im Unicast Modus durchgeführt. Dabei werden die Benchmark Nachrichten direkt an die jeweiligen Empfänger gesendet. Die Unicast Adressierung erfolgt durch Angabe der MAC Adressen der Empfänger bei der Konfiguration der Client Nodes (siehe Abschnitt 9.3.7). Durch Abfrage der *Adress-Table* werden schliesslich die *Short-Addresses* der Empfänger ermittelt.

20.3 Mesh Node Firmware

Die Firmware für die Benchmark Nodes die in Abschnitt 9 bereits behandelt wurde, wird durch ein C- und ein H-Modul ergänzt, die spezifisch für die Implementation des Zigbee Stack zuständig sind. Im Modul *bm_config.h* werden Konstanten für den Benchmark wie auch die Funktion des Stacks definiert. Das Modul *bm_zigbee.c* hingegen beinhaltet je nach Funktion des Nodes (Master, Client oder Server) die entsprechenden Funktionen für die Initialisierung und Konfiguration des Stacks sowie das Benchmark Handling. Darin ist der Versand und Empfang der Benchmark Nachrichten vereint. Der gesamte Sourcecode ist im [Github-Repository](#)³² zu diesem Projekt einsehbar.

20.3.1 Stack Initialisierung und Konfiguration

Die Stack-Init-Routine wird durch die Benchmark State machine gemäss Abschnitt 9.3.1 im State *Init Benchmark* aufgerufen. Diese Routine ist durch die Funktionen *bm_zigbee_init()* sowie *bm_zigbee_enable()* im Modul *bm_zigbee.c* abgebildet. Hier wird der Zigbee Stack in der Rolle als *Zigbee-Koordinator* resp. *Zigbee-Server* initialisiert und hochgefahren. Zudem wird auf dem *Benchmark-Server* der Endpoint Handler gemäss Abschnitt 20.2.4 mit dem folgenden API Aufruf registriert:

```
ZB_AF_SET_ENDPOINT_HANDLER(BENCHMARK_SERVER_ENDPOINT,
bm_ep_handler)
```

Nach dem Start des Stacks wird der *ZBOSS Mainloop* im Benchmark State abgearbeitet. Damit wird der *ZBOSS Scheduler* aktiviert und der Stack wird ausgeführt.

Der *Zigbee-Koordinator* welcher auf dem Benchmark Master Node aufgesetzt ist, startet nun das *Network Formation*, sofern keine persistenten Netzwerk Parameter gespeichert sind. Dies ist nur beim Erststart des *Benchmark-Masters* der Fall. Im *Network Formation* wird das Zigbee Netzwerk auf dem gewählten Kanal gebildet und die entsprechenden Netzwerkschlüssel generiert. Ist beim Start eine persistente Konfiguration im Flash vorhanden, wird diese geladen und direkt mit dem *Zigbee Commissioning* gestartet. Ansonsten wird das erfolgreiche Beenden des *Network Formation* abgewartet und erst danach das *Zigbee Commissioning* gestartet.

Im *Zigbee Commissioning* State melden sich sämtliche Teilnehmer beim *Zigbee-Koordinator* an und erhalten von diesem die nötigen Netzwerkparameter. Unter anderem wird jedem Node

³²https://github.com/Rouben94/P6_Software[23]

eine 16-Bit *Short-Address* zugewiesen. Der Start des *Commissioning* wird bei den Benchmark Teilnehmern um eine zufällige Zeit zwischen 0 und 30 Sekunden verzögert. So kann sichergestellt werden, dass der *Zigbee-Koordinator* nicht von Anfragen überhäuft wird. Nach erfolgreichem *Commissioning* ist das Zigbee Netzwerk bereit für die erste Messung.

Die gesamte *Benchmark Init* Phase wie sie im Abschnitt 9.3.1 definiert wurde, ist auf die Dauer von 60 Sekunden limitiert. Diese Zeit wird benötigt um das *Commissioning* abzuschliessen. Danach wird automatisch in den *Benchmark State* gewechselt.

20.3.2 Benchmark Handling

Während der Master am Benchmark Handling nicht beteiligt ist, sind im Modul *bm_zigbee.c* für den Benchmark Client und den Benchmark Server unterschiedliche Funktionen implementiert. Im [Github-Repository](#)³³ sind dafür die drei Projektdateien, *benchmark_master*, *benchmark_server* und *benchmark_client*, hinterlegt.

Benchmark Client

Der Benchmark Client ist für den Versand von Benchmark Nachrichten zuständig. Dazu wird durch die Benchmark Statemachine, mittels universeller Schnittstelle in der SharedLib (siehe Abschnitt 9.3) die Funktion *bm_send_message()* aufgerufen. Diese löst wiederum einen Call-back aus in welchem die Benchmark Nachricht konstruiert und schliesslich versendet wird. Die Nachricht ist einem *ZCL Move to Level Request* nachempfunden. Allerdings kann die Payload der Nachricht beliebig erhöht werden.

Noch vor dem Versand der Benchmark Nachricht werden die Message Infos in der Funktion *bm_read_message_info()* ausgelesen und ein Log-Eintrag wird erstellt. Dabei handelt es sich um die Messgrößen gemäss Abschnitt 7.4. Zudem wird der Zustand des grünen RGB-LEDs gewechselt.

Benchmark Server

Der Benchmark Server wartet im *Benchmark State* auf ankommende Nachrichten. Sobald der Benchmark Server ein Paket empfängt, wird der Benchmark Endpoint Handler (*bm_ep_handler()*) signalisiert. Dieser entscheidet anhand der *ZCL Cluster ID* ob es sich bei der Nachricht um eine Benchmark Nachricht handelt oder nicht. Falls es eine Benchmark Nachricht ist, wird die Funktion *bm_receive_message()* aufgerufen und die Referenz des Paket Buffers (*bufid*) übergeben. Mit Hilfe der *bufid* werden nun die Message Infos ausgelesen und daraus ein Log Eintrag erstellt. Zum Schluss wird der Status des blauen RGB-LED's gewechselt womit der erfolgreiche Empfang einer Nachricht signalisiert wird.

20.3.3 Benchmark und Stack Parameter

Für die Umsetzung des Zigbee Protokoll Stacks sowie des Benchmark Frameworks für den Zigbee Benchmark, mussten einige Parameter gesetzt oder neu definiert werden. Die Tabelle 20.1 zeigt die wichtigsten Benchmark und Stack Parameter.

³³https://github.com/Rouben94/P6_Software[23]

Stack Init Time	60000 ms	Zeit die benötigt wird um den Stack für den Benchmark zu initialisieren. Reserve Zeit um das Routing des Netzwerkes durchzuführen.
Stack Startup max Delay	30000 ms	Maximale Verzögerungszeit für den Startup des Zigbee Stacks auf allen Client und Server Nodes. Wird benötigt um das Netz sauber aufzubauen zu können.
Network Formation Delay	5000 ms	Zeit die zur Stack Startup Delay Time addiert wird um das Network Formation des Koordinators abzuwarten.
IEEE Channel	15	IEEE Kanal der verwendet wird um das Zigbee Mesh Netzwerk aufzubauen.
Node Type	Zigbee-Router	Alle Nodes werden als Zigbee-Router konfiguriert
Client Endpoint	1	Nummer für den Client Endpoint im ZCL Level Cluster
Server Endpoint	10	Nummer für den Server Endpoint im ZCL Level Cluster
Default Group ID	0xB331	Zu diesem Wert wird der Index für die zugewiesene Gruppe addiert um damit die Gruppenzugehörigkeit festzulegen.
BSP LED 0	ZIGBEE_NETWORK_STATE_LED	Zeigt den Status der Netzwerkverbindung an.
BSP LED 1	ERROR_LED	Zeigt an, ob ein Fehler in der Statemachine vorliegt.
BSP LED 2	CLIENT_LED	Wird beim Versand einer Benchmark Nachricht ein oder ausgeschaltet.
BSP LED 3	BULB_LED	Imitiert die Lichtquelle. Wird abwechselnd ein oder ausgeschaltet wenn ein Paket empfangen wird.

Tabelle 20.1: Zigbee Benchmark und Stack Parameter

21 Analyse Zigbee Stack

Für die Umsetzung des Benchmark Konzepts war eine vertiefte Auseinandersetzung mit dem Zigbee Protokoll nötig. Dabei konnten grosse Erfahrungen mit dem Stack sowie der verwendeten SDK gesammelt werden. Aufgrund dieser Erfahrungen wird nachfolgend eine Analyse des Zigbee Protokollstacks sowie der eingesetzten SDK gemacht und einige Schwierigkeiten und Hindernisse bei der Softwareentwicklung aufgezeigt.

21.1 Stärken und Schwächen des Protokollstacks

Folgende Stärken und Schwächen weist der *ZBOSS Zigbee Stack* auf. Stack Implementationen von anderen Herstellern können nicht bewertet werden.

Stärken:

- Die **Zigbee Cluster Library (ZCL)** erleichtert die Umsetzung eines Produktes, welches mit anderen Zigbee Geräten konform sein soll erheblich.
- Die **Adressierung mit Short-, Long- und Group-Adressen** ist übersichtlich und schlank gehalten. Die vereinfacht das Handling für Entwickler und Anwender.
- Ein simpler **Scheduler und Callback** Mechanismus vereinfacht das Nachrichten Handling.
- Der **IEEE 802.15.4 Standard** für die MAC Ebene bietet dem Entwickler eine solide Basis um die gewünschte Anwendung darauf umsetzen zu können.

Schwächen:

- Bei der **Gruppen Adressierung** zeigt der Zigbee Stack deutliche Schwächen da für Broadcast Nachrichten ein kleinerer Packetbuffer verwendet wird.
- Wenn **Zigbee Cluster Library (ZCL)** Funktionalitäten nicht zwingend benötigt werden, wird die Applikationsentwicklung umständlich. Der Stack ist zu sehr auf vordefinierte Anwendungen ausgerichtet.
- Die Spezifikationen des Stacks sind frei verfügbar. Leider implementieren die Hersteller den Stack schliesslich im **Closed-Source** Verfahren weshalb keine Community entstehen kann die den Stack verbessert.
- Das **Commissioning** von neuen Teilnehmern hat in unserer Anwendung leider nicht immer reibungslos funktioniert. Bei vielen gleichzeitigen Anmeldeanfragen kam der **Zigbee-Koordinator** gerne ins Straucheln und der Vorgang musste neu gestartet werden.

21.2 Schwierigkeiten bei der Umsetzung

Einige Probleme haben die Entwicklung der Zigbee Benchmark Firmware schwieriger gemacht als erhofft. Die folgenden 5 Punkte haben die Arbeiten merklich beeinflusst.

Software Development Kit

Da die *nRF5 SDK for Thread and Zigbee* wohl bald von der *nRF Connect SDK* abgelöst wird, scheint der Hersteller Nordic Semiconductor nicht wirklich Interesse daran zu haben, die „alte“ SDK zu unterhalten. Dies macht sich in der Dokumentation bemerkbar, denn gewisse API's sind nicht oder nur sehr schlecht dokumentiert. Da es sich beim *ZBOSS Zigbee Stack* um eine vorkompilierte Library handelt ist auch die Interpretation des Codes schwierig bis unmöglich. In einigen Fällen verweist die Dokumentation sogar auf Funktionen und Beispiele die in der aktuellsten Version der SDK nicht mehr verfügbar sind.

Custom APS Data

Die *nRF5 SDK for Thread and Zigbee* bietet die Möglichkeit sogenannte *Custom APS Data* zu

versenden. Dabei handelt es sich um eine Low Level API für den Versand von benutzerdefinierten Anwendungsdaten ohne dass die ZCL benutzt werden muss. Während der Umsetzung des Benchmarks hat sich gezeigt, dass der Empfang und die Auswertung dieser Daten nur sehr schwierig umsetzbar ist. Die entsprechende Dokumentation innerhalb der SDK war schlicht nicht ausreichend detailliert. Im Benchmark Kontext hätte diese Funktion den Versand simpler Applikationsdaten vereinfacht.

Anzahl Hops

Für die Auswertung der Messergebnisse in den Abschnitten 22 und 23 wurden unterschiedliche Messgrößen erfasst. Unter anderem wäre auch die Anzahl Hops die eine Benchmark Nachricht innerhalb des Mesh Netzwerkes passiert hat, von grosser Wichtigkeit. Nur so kann die Latenzzeit unabhängig von gewählten Weg definiert und mit den Latenzzeiten der anderen Protokolle verglichen werden. Leider bietet die *nRF5 SDK for Thread and Zigbee* keine Möglichkeit diesen Wert zu bestimmen, da das Radius-Feld aus dem NWK-Header nicht ausgelesen werden kann. Das Radius-Feld wäre gemäss neuster [Zigbee Specification³⁴](#) ein 8-Bit Feld im NWK-Header, welches beim passieren eines Hops dekrementiert wird [45]. Das Paket wird verworfen sobald das Feld den Wert 0 erreicht. Der folgende Beitrag aus dem Supportforum von Nordic Semiconductor bestätigt dieses Problem: [Zigbee - Read number of hops \(radius\)³⁵](#)

Group addressing mode

Diverse Tests mit implementiertem *ZCL groups cluster* haben gezeigt, dass Zigbee bei der Adressierung von Gruppenadressen schnell überfordert ist. Selbst bei durchschnittlichem Verkehrsaufkommen im Netzwerk, konnte grosser Packetverlust von über 90 Prozent festgestellt werden. Getestet wurde dies mit den Messreihen 1 und 2 gemäss Abschnitt 7.5. Eine Gruppenadressierung löst in Zigbee eine Broadcast Nachricht aus, welche von allen Teilnehmern weitergeleitet wird. Dies verursacht ein relativ grosses Verkehrsaufkommen im Netz. Es wird vermutet, dass der Paketbuffer für Broadcast Nachrichten zu klein ist und dieser zu langsam abgearbeitet wird. Dadurch überläuft der Paketbuffer zu schnell und es entstehen die erwähnten Probleme. Es konnte festgestellt werden, dass nach ca. 15 versendeten Nachrichten, keine weiteren Nachrichten zugestellt werden können, bis sich der Stack nach einiger Zeit wieder erholt hatte.

nRF Connect SDK

Nordic Semiconductor stellt nebst der in dieser Arbeit verwendeten *nRF5 SDK for Thread and Zigbee* noch eine weitere SDK für die Entwicklung von Zigbee Produkten zur Verfügung. Die *nRF Connect SDK* unterstützt seit der Version v1.3.0 ebenfalls den Zigbee Protokollstack. Im Verlauf dieser Arbeit wurde auch diese SDK testweise eingesetzt. Leider hat sich erst spät herausgestellt, dass die *nRF Connect SDK* noch einige Kinderkrankheiten bei der Verwendung mit Zigbee besitzt. So werden Nachrichten teilweise erst mit einer Verzögerung von 200ms versendet. In den [Release Notes³⁶](#) zu der entsprechenden Version ist dies festgehalten.

³⁴ <https://zigbeealliance.org/solution/zigbee> [45]

³⁵ <https://devzone.nordicsemi.com/f/nordic-q-a/63815/zigbee---read-number-of-hops-radius> [46]

³⁶ https://developer.nordicsemi.com/nRF_Connect_SDK/doc/1.3.0/nrf/doc/release-notes-1.3.0.html

Teil VI

Mesh Benchmark - Resultate und Vergleich

22 Messresultate

Bei der Durchführung der Mesh Benchmark Messungen wurde für jede Messung unter den entsprechenden Bedingungen ein Messprotokoll resp. eine Auswertung erstellt. Die acht unterschiedlichen Bedingungen sind in Tabelle 22.1 nochmals zusammengefasst sind. Diese detaillierten Auswertungen sind im Anhang D dem Bericht angefügt. Nachfolgend wird exemplarisch eine dieser Auswertungen erläutert, um aufzuzeigen, was diese darstellen und wie diese gelesen werden können. Eine Interpretation der Messresultate im Rahmen eines Vergleichs erfolgt anschliessend im Abschnitt 23.

#	Msg. Gen.	Duration	Msg. Cnt.	Payload	Disturbance
1	Rand	600s	60	Small	No
2	Seq	600s	60	Small	No
3	Rand	600s	60	Large	No
4	Seq	600s	60	Large	No
5	Rand	600s	600	Small	No
6	Rand	600s	60	Small	Yes
7	Seq	750s	10	Small	No
8	Seq	750s	10	Large	No

Tabelle 22.1: Messbedingungen Mesh Benchmark

22.1 Resultate

Die Messresultate im Anhang D wurden mit den Messindizes 1-8 gemäss Tabelle 22.1 sowie der entsprechenden Messumgebung (z.B. Wohnung) versehen. So können die Messungen eindeutig identifiziert werden. Folgende Einschränkungen müssen dabei jedoch beachtet werden: Gemäss den Erläuterungen im Abschnitt 7.5 wurden die Messungen 6 - 8 nur im Labor-Messaufbau durchgeführt. Von diesen Messungen sind dementsprechend auch nur diese drei Auswertungen vorhanden. Weiter musste bei der Durchführung der Messung 5 festgestellt werden, dass die Resultate unbrauchbar waren. Aus diesem Grund musste die Auswertung dieser Messung gestrichen werden. Im Abschnitt 22.2 wird nochmals darauf eingegangen.

Die Abbildungen 22.1 bis 22.6 zeigen die Messresultate der Messung 2 in der Messumgebung *Wohnung*. Sie stehen exemplarisch für die Messergebnisse aller Messreihen. In Abbildung 22.1 ist die prozentuale Verteilung der Latenzzeiten pro Hop dargestellt. In sämtlichen Grafiken werden die Ergebnisse von BT Mesh in blau, jene von Thread in grün und jene von Zigbee in rot dargestellt. Dadurch ist ein direkter Vergleich der Protokolle möglich. Abbildung 22.1 zeigt folglich, wie viele Nachrichten das Ziel mit einer bestimmten Latenzzeit erreicht haben. Nachrichten, die das Ziel nicht erreicht haben, also Pakete die verloren gegangen sind, werden dabei nicht berücksichtigt. Diese werden jedoch als Paketloss aufgezeichnet. Im gezeigten Beispiel haben rund 76 Prozent der Nachrichten, die im BT Mesh Test versendet wurden, das Ziel mit einer maximalen Latenzzeit von 10 Millisekunden erreicht. Die weitere Verteilung geht bis auf Latenzzeiten von über 300 Millisekunden, wobei die Prozentzahl der Nachrichten in diesem Bereich sehr tief ist. Die Werte für Thread und Zigbee zeigen hingegen eine deutlich kleinere Verteilung der Latenzzeiten.

Aus den in Abbildung 22.1 aufgezeigten Latenzzeiten wurde der Durchschnitt über alle Messreihen gebildet und in Abbildung 22.2 dargestellt. Es handelt sich dabei wiederum um die Latenzzeit pro Hop. Im Falle von Zigbee ist dies erwähnenswert, da hier die Anzahl Hops nicht ausgelesen werden konnte (siehe Abschnitt 21.2) und die Resultate somit mit Vorsicht interpretiert werden müssen. Mehr dazu in der Validierung im Abschnitt 22.2.

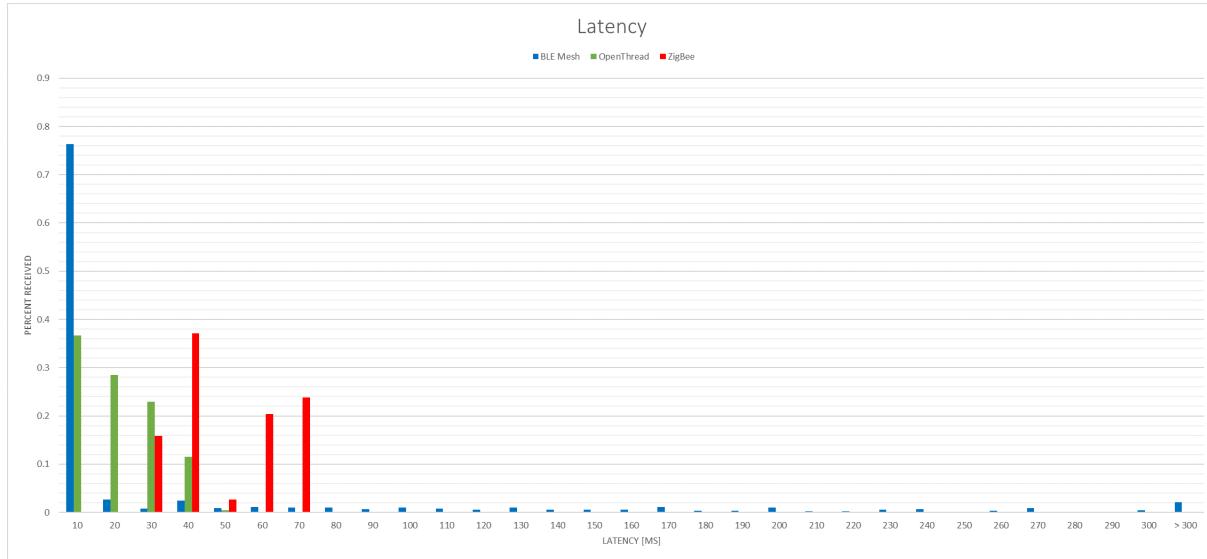


Abbildung 22.1: Messung 2 Wohnung: Verteilung der Latenzzeiten pro Hop

Der durchschnittliche Datendurchsatz, der in Abbildung 22.3 aufgezeigt wird, muss ebenfalls mit Vorsicht betrachtet werden. Denn auch hier werden die Werte pro Hop für die Berechnung verwendet. Die präsentierten Werte werden aus der Paketgrösse (Small oder Large) gemäss den Definitionen in Abschnitt 7.6 und der Latenzzeit des übertragenen Pakets (siehe Gleichung 22.1), errechnet. Dabei werden die Werte für den Durchsatz pro empfangenes Paket berechnet und davon schliesslich der Mittelwert gebildet. Die oben erwähnten Ausreisser bei BT Mesh bewirken nun einen unerwartet hohen Durchsatz im Vergleich zu jenem bei Thread, das konstant tiefe Latenzzeiten aufweist.

$$TP = \frac{S_{\text{packet}} \cdot 8}{t_{\text{lat}}} \quad (22.1)$$

TP Throughput (kBit/s)
 S_{packet} Packetsize (Byte)
 t_{lat} Latency time (ms)

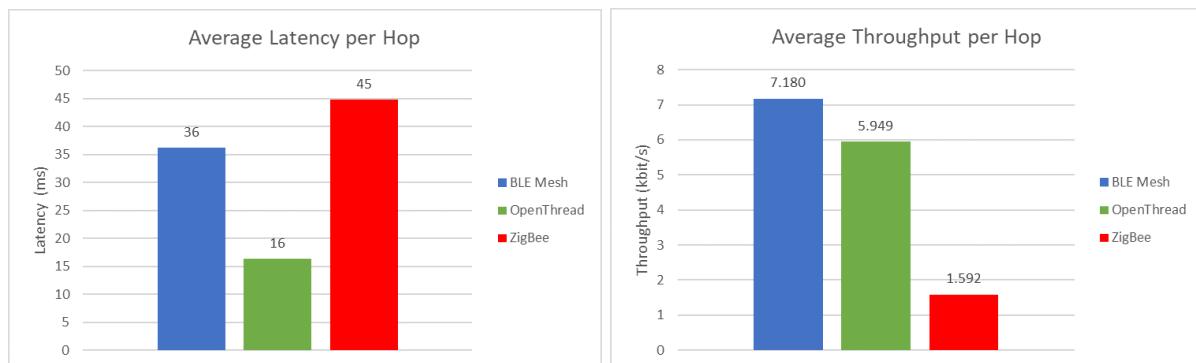


Abbildung 22.2: Messung 2 Wohnung: Durchschnittliche Latenzzeit pro Hop

Abbildung 22.3: Messung 2 Wohnung: Durchschnittlicher Durchsatz pro Hop

In Abbildung 22.4 wird der Paketverlust im Verhältnis zur gesamten Anzahl Nachrichten, die während dem Benchmark versendet wurden, in Prozenten aufgezeigt. Die Paketverluste von

einzelnen Client-Server Beziehungen werden nicht separat ausgewertet. Wiederum im Beispiel von BT Mesh sind in dieser Messung 2.07 % der Pakete nicht am Ziel angekommen.

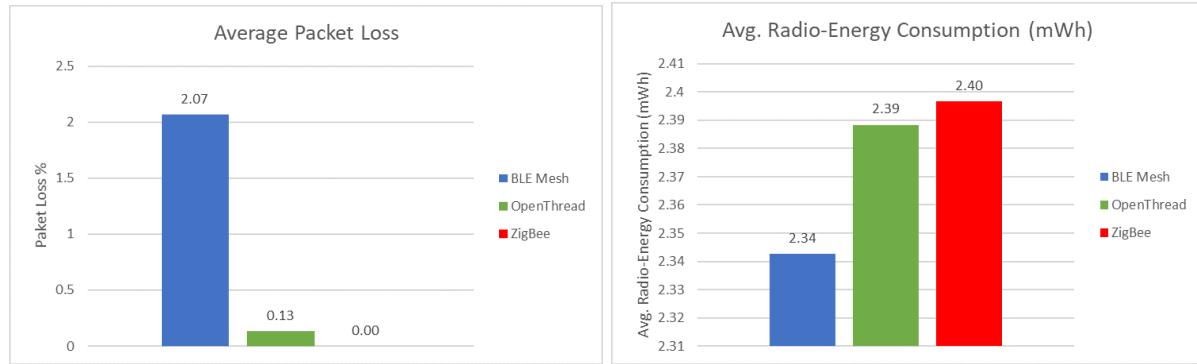


Abbildung 22.4: Messung 2 Wohnung:
Durchschnittlicher Paketverlust

Abbildung 22.5: Messung 2 Wohnung:
Durchschnittlicher Energieverbrauch

Mit dem Diagramm in Abbildung 22.5 wird schliesslich noch der durchschnittliche Energieverbrauch der Protokolle dargestellt. Dieser wird abgeleitet aus der *Active Radio Time* (siehe Abschnitt 7.4.1), welche direkt auf dem nRF52840 SoC mit der entsprechenden API ausgelesen werden kann. Die *Active Radio Time* wurde schliesslich mit dem Strombedarf des SoC's bei definierter Speisespannung von 3V verrechnet. Gemäss den Angaben in Tabelle 8.1 aus Abschnitt 8.1 beträgt der Strombedarf bei aktivem Funkmodul im Sendemodus 4.8mA. Eine solche Berechnung erlaubt einen qualitativen Vergleich des Energiebedarfs unter den drei Protokollen, da die Umsetzung auf der gleichen Hardware erfolgte. Die Werte in Abbildung 22.5 sind jedoch keine quantitativen Verbrauchswerte und können deshalb nur im Kontext des Vergleichs verwendet werden. Der Strombedarf sämtlicher Peripherie wurde nicht berücksichtigt, da dieser prinzipiell bei allen Protokollen identisch und in diesem Fall daher nicht interessant ist.

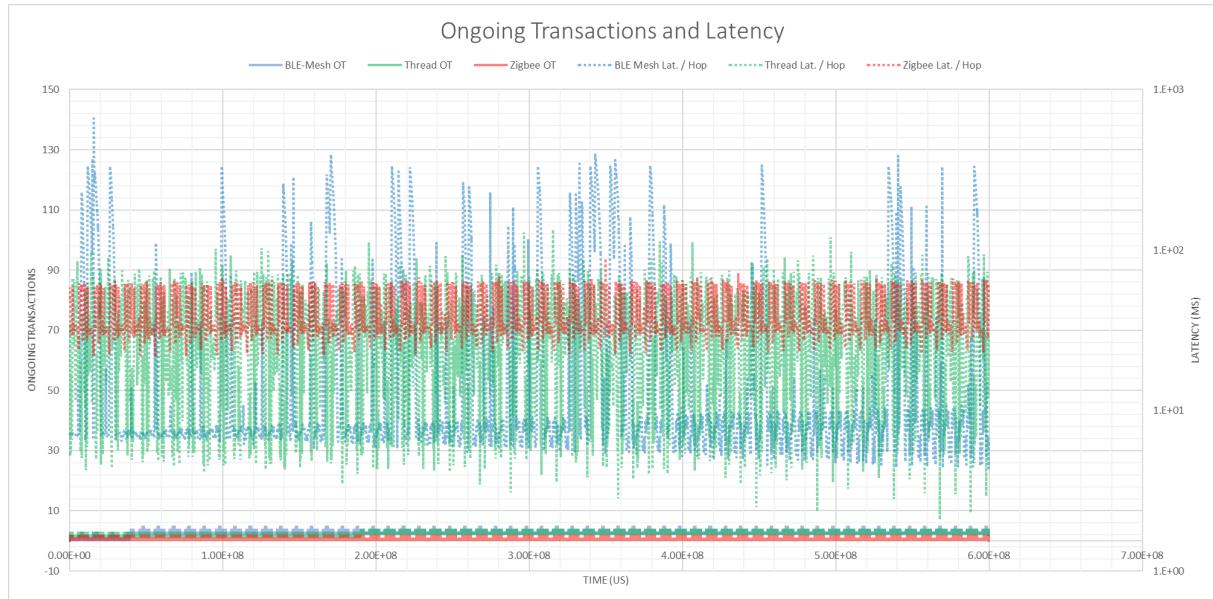


Abbildung 22.6: Messung 2 Wohnung: Ongoing Transactions und Verlauf der Latenzeiten über die Messdauer.

Die letzte Grafik gemäss Abbildung 22.6 zeigt den Verlauf der *Ongoing Transactions* sowie der Latenzeiten über die Gesamtdauer einer Messung. In diesem Fall beträgt die Dauer 600 Sekunden. Die Grafik soll einen Eindruck darüber vermitteln, wie die Stacks damit umgehen,

wenn vielen Nachrichten zur selben Zeit versendet werden. Die *Ongoing Transactions*, welche als durchgezogene Linien dargestellt sind, zeigen zu welchem Zeitpunkt wie viele Nachrichten im Übermittlungsprozess sind. Da die Nachrichten in diesem Beispiel sequentiell versendet werden, gibt es nur sehr geringe Ausschläge, welche im unteren Bildrand der Abbildung 22.6 zu sehen sind. Die logarithmische Darstellung der Latenzzeiten als gestrichelte Linie bestätigt die Beobachtungen, die in Abbildung 22.1 bereits gemacht wurden. Zigbee und Thread weisen einen ziemlich regelmässigen Verlauf der Latenzzeiten auf. BT Mesh hingegen zeigt starke Ausreisser.

22.2 Validierung

Die durchgeföhrten Messungen haben aussagekräftige Resultate geliefert, welche jedoch stark vom gewählten Vorgehen abhängig sind. Dieses Vorgehen wird nun kritisch überprüft und allfällige Mängel im Konzept sowie in der Umsetzung werden aufgezeigt. Zudem werden systematische Messfehler deklariert.

Large Payload

Die Messungen 3, 4 und 8 gemäss Tabelle 22.1, welche mit einer grossen Payload durchgeföhrten wurden, sind nur für Thread und Zigbee aussagekräftig. Der BT Mesh Stack liefert bei diesen Messungen keine brauchbaren Resultate. Eine Recherche zu diesem Problem hat ergeben, dass durch die Fragmentierung einer 32 Byte Payload die sichere und schnelle Übertragung der Daten nicht mehr gewährleistet werden kann. Dieses Problem wird in Abschnitt 13.0.2 genauer erläutert.

Zigbee Latency

Wie bereits im Abschnitt 21.2 erwähnt, konnte bei Zigbee die Anzahl Hops, die ein Paket passiert hat nicht ausgewertet werden. Der Forumsbeitrag [Zigbee - Read number of hops \(radius\)](#)³⁷ bestätigt, dass der entsprechende Wert mit der verwendeten SDK nicht ausgelesen werden kann. Die Folge davon ist, dass die Latenzzeit nur als Total über die gesamte Strecke erfasst werden kann. In der Auswertung verschafft dies fälschlicherweise BT Mesh und Thread einen Vorteil gegenüber Zigbee. Die Auswertung der totalen Latenzzeit bei allen drei Protokollen könnte das Problem lösen. Dies würde jedoch dem Messkonzept widersprechen und wurde deshalb nicht für alle Messungen umgesetzt. Die Abbildungen 22.7 und 22.8 zeigen den Unterschied am Beispiel der oben analysierten Messung im Abschnitt 22.1. Während in Abbildung 22.7 die Latenzzeit pro Hop dargestellt ist, zeigt die Abbildung 22.8 die totale Latenzzeit. Der Unterschied ist vor allem bei Thread deutlich erkennbar. Bereits in Abbildung 22.1 ist diese Problematik zu erahnen. Denn die Statistik der Latenzzeiten von Zigbee zeigt zwei Hauptäulen bei 40ms und 70ms. Dies deutet darauf hin, dass die Pakete mit 70ms Latenzzeit einen Hop passiert haben und jene bei 40ms nicht.

Nachrichten Dichte

Bei der Definition der Messreihen 7.5 wurden zu Beginn nur die Messungen 1 bis 6 spezifiziert. Die Messreihen 7 und 8 kamen erst nachträglich hinzu, als festgestellt werden musste, dass die Dichte der Nachrichten für den BT Mesh Stack zu hoch gewählt wurde. Der BT Mesh Stack war überfordert beim Empfang von zufällig generierten Nachrichten, wie im Abschnitt 7.7 beschrieben ist. Thread und Zigbee zeigten indes keine Mühe mit der Dichte von 60 Nachrichten in 600 Sekunden pro Node. Die Resultate der Messungen 7 und 8 haben schliesslich gezeigt, dass die Reduktion der Nachrichtendichte einen positiven Einfluss auf die Performance von BT Mesh hat.

Die Messung Nummer 5 wurde aufgrund der selben Problematik aus den Auswertungen gestrichen. Hier wäre die Dichte der Nachrichten um das zehnfache grösser gewesen als bei der

³⁷ <https://devzone.nordicsemi.com/f/nordic-q-a/63815/zigbee---read-number-of-hops-radius>

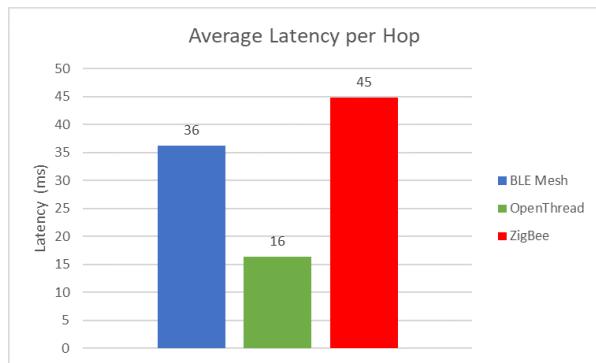


Abbildung 22.7: Durchschnittliche Latenzzeit pro Hop

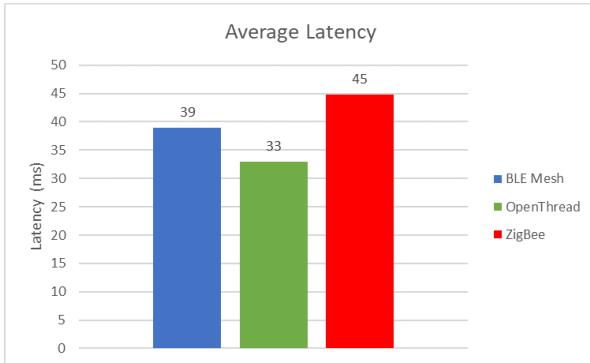


Abbildung 22.8: Durchschnittliche Latenzzeit ohne Berücksichtigung der Hops.

vergleichbaren Messung mit dem Index 1. Eine Auswertung der Messdaten wäre daher sinnlos gewesen.

Group addressing mode

Der *Group addressing mode* wurde im Abschnitt 7.6 für die drei Protokolle unterschiedlich definiert. Erste Tests vor den eigentlichen Benchmarks haben gezeigt, dass eine Multicast Adressierung bei Zigbee unbrauchbar ist (siehe Abschnitt 21.2). Deshalb hat man sich entschieden, bei Zigbee eine Unicast Adressierung umzusetzen. Während den Benchmarks musste schliesslich festgestellt werden, dass Zigbee durch diese Änderung einen wesentlichen Vorteil erlangt hat. Besonders auf die Paketverlustrate hat sich die Unicast Adressierung positiv ausgewirkt, denn Unicast Nachrichten verursachen deutlich weniger Verkehr innerhalb des Netzwerkes als Broadcast Nachrichten.

Auch in der Messung Nr. 5 hätte sich die Unicast Adressierung für Zigbee positiv ausgewirkt, da die Netzbelastrung deutlich geringer gewesen wäre.

Durchschnittswerte in den Resultaten

In den Resultaten 22.1 wurden sämtliche Durchschnittswerte als Mittelwerte einschliesslich allfälliger Ausreisser aus den Messwerten gebildet. Dadurch wurden gewisse Resultate deutlich verfälscht. In einer verbesserten Auswertung müsste die Ursache für die einzelnen Ausreisser genauer geklärt werden und diese allenfalls gestrichen werden. Zudem müsste für eine bessere Auswertung der Median bestimmt werden, um allfällige Ausreisser zu kompensieren. Dies wurde leider zu spät entdeckt und konnte daher nicht umgesetzt werden.

22.3 Verifizierung

Eine Verifizierung der Messresultate konnte nur anhand des Referenzberichts *AN1142: Mesh Network Performance Comparison*³⁸ von *Silicon Labs* gemacht werden. Dieser ist auf der Webseite von *Silicon Labs* einsehbar.

Der Vergleich der Messergebnisse hat gezeigt, dass die Größenordnung der Resultate mit jenen aus dem Bericht von Silicon Labs übereinstimmt. Selbst die Ausreisser in der Latenzzeit bei BT Mesh liegen im selben Rahmen. Zudem kann die klare Abhängigkeit der Resultate von der Größe der Payload bestätigt werden. Einige Unterschiede sind jedoch in der Verteilung der Latenzzeiten erkennbar. Im Referenzbericht ist diese in einem Bereich zwischen 20ms und 60ms

³⁸<https://www.silabs.com/documents/public/application-notes/an1142%2Dmesh%2Dnetwork%2Dperformance%2Dcomparison.pdf> [11]

ziemlich regelmässig, während in den Ergebnissen dieser Thesis die Verteilung unregelmässiger daherkommt.

23 Vergleich Mesh Netzwerke

Der Vergleich der Mesh-Netzwerke basiert auf den Messergebnissen die im Abschnitt 22 präsentiert wurden. Die Messungen unterscheiden sich aufgrund der unterschiedlichen Messparameter, sowie dem Messaufbau (Wohnung/Labor/Haus). Um einen Vergleich ziehen zu können, wird daher auf diese beiden Hauptmerkmale abgestellt. Der Vergleich der Mesh Netzwerke, inkl. den wichtigsten Eigenschaften des Benchmarks, wurde in einem eigenständigen Paper im Anhang E zusammengefasst. Nachfolgend wird ein ausführlicher Vergleich der Mesh Netzwerke, bezüglich den oben genannten Hauptmerkmalen, präsentiert. Im ersten Abschnitt 23.1 werden die Messreihen und im zweiten Abschnitt 23.2 die Testumgebungen verglichen. Innerhalb dieser Abschnitte werden die Vergleichswerte gemäss Abschnitt 7.4.1 betrachtet.

23.1 Vergleich der Messreihen

Dieser Vergleich bezieht sich auf eine gleichbleibende Testumgebung jedoch mit unterschiedlichen Messreihen. Als Referenz wurde die Umgebung des Labors ausgewählt, da dort Messergebnisse aller Messreihen vorliegen (siehe Anhang D). Im Anschluss soll gezeigt werden, wie sich die einzelnen Mesh-Netzwerke bei ändernden Messreihen verhalten.

Zusammengefasst lassen sich die Messreihen folgendermassen unterscheiden (siehe auch Tabelle 22.1).

- Die ***Payload*** wurde zwischen den Messreihen 1&3, 2&4 und 7&8 von 8 Byte auf 32 Byte (BT Mesh), resp. 50 Byte (Thread, Zigbee) erhöht.
- Der ***Traffic Generation Mode*** wurde zwischen den Messreihen 1&2 und 3&4 von Random auf Sequentiell geändert.
- Die ***Message-Dichte*** wurde zwischen den Messreihen 2&7 und 4&8 von 2.5 M/s (Messages/Sekunde auf 0.33 M/s gesenkt).
- Der ***Disturbance*-Vergleich kann zwischen der Messung 2 und der Messung 6 erfolgen. Bei Messung 6 ist die Disturbance eingeschaltet.**

23.1.1 Latenzzeit

Durch die in Abbildung 23.1 dargestellten Latenzzeiten wird ersichtlich, dass Bluetooth-Mesh am anfälligsten auf die Erhöhung der Payload reagiert. Eine detaillierte Analyse zu diesem Phänomen wird in Abschnitt 23.3.3 durchgeführt. Die Verzögerung bleibt bei Zigbee und Thread trotz Erhöhung der Paketlänge stabil.

Eine Änderung des *Traffic Generation Mode* von Random auf Sequentiell bewirkt bei Bluetooth-Mesh eine drastische Abnahme der Latenz. Bei Zigbee ist zwischen der Messreihe 1 und 2 ebenfalls eine Abnahme zu verzeichnen, welche jedoch nicht zwischen der Reihe 3 und 4 feststellbar ist. Somit muss ein anderer Einfluss für die Abnahme verantwortlich sein, welcher im Abschnitt 23.3.2 behandelt wird. Die Änderung des *Traffic Generation Mode* führt bei Thread ebenfalls zu einer Verbesserung der Latenzzeiten. Es zeigt sich, dass die Stacks mit gleichmässigem Nachrichtenaufkommen besser zurecht kommen, als mit zufällig generierter Abfolge der Nachrichten.

Das Senken der Message Dichte bewirkt bei Bluetooth-Mesh eine markante Abnahme der Latenzzeit. Bei Thread und Zigbee bleibt die Latenz nahezu konstant.

Die Einbringung von Störungen hat lediglich bei Bluetooth-Mesh einen negativen Einfluss, wodurch sich die Latenzzeit erhöht. Dank CSMA/CA des IEEE 802.15.4 MAC Layers, werden die Messungen von Thread und Zigbee nicht beeinflusst.

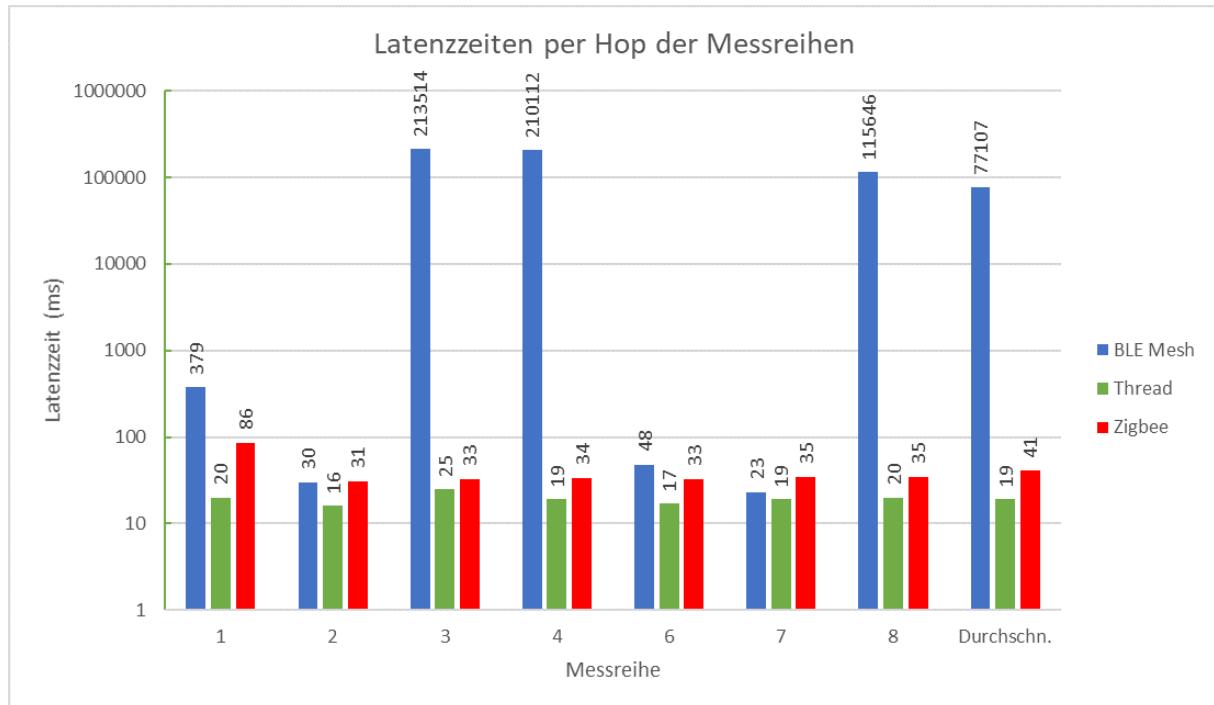


Abbildung 23.1: Durchschnittliche Latenzzeit per Hop der einzelnen Messreihen im Vergleich

23.1.2 Durchsatz

Die in Abbildung 23.2 dargestellten Durchsatzraten zeigen, dass sich ähnliche Vergleichsresultate ergeben wie bei der Latenzzeit zuvor. Bluetooth-Mesh reagiert am anfälligsten auf die Payload-Erhöhung. Wie bereits in Abschnitt 23.1.1 dargelegt wird eine detaillierte Analyse zu diesem Phänomen in Abschnitt 23.3.3 durchgeführt. Der Durchsatz steigt bei Zigbee und Thread durch die Erhöhung der Paketlänge stark an. Dies wird in Abschnitt 23.3.1 für Thread und in Abschnitt 23.3.2 für Zigbee genauer untersucht.

Eine Änderung des *Traffic Generation Mode* von Random auf Sequentiell bewirkt bei Bluetooth-Mesh eine drastische Zunahme des Durchsatzes. Bei Thread ist ebenfalls eine Steigerung der Übertragungskapazität feststellbar. Bei Zigbee hingegen bewirkt diese Änderung keine merkliche Verbesserung des Durchsatzes.

Die Senkung der Message-Dichte führt bei Bluetooth-Mesh zu einer Verbesserung des Durchsatzes. Bei Thread und Zigbee kann keine Abhängigkeit festgestellt werden.

Das Einbringen von Störungen hat bei allen Netzwerken einen leicht negativen Einfluss, wodurch der Durchsatz sinkt.

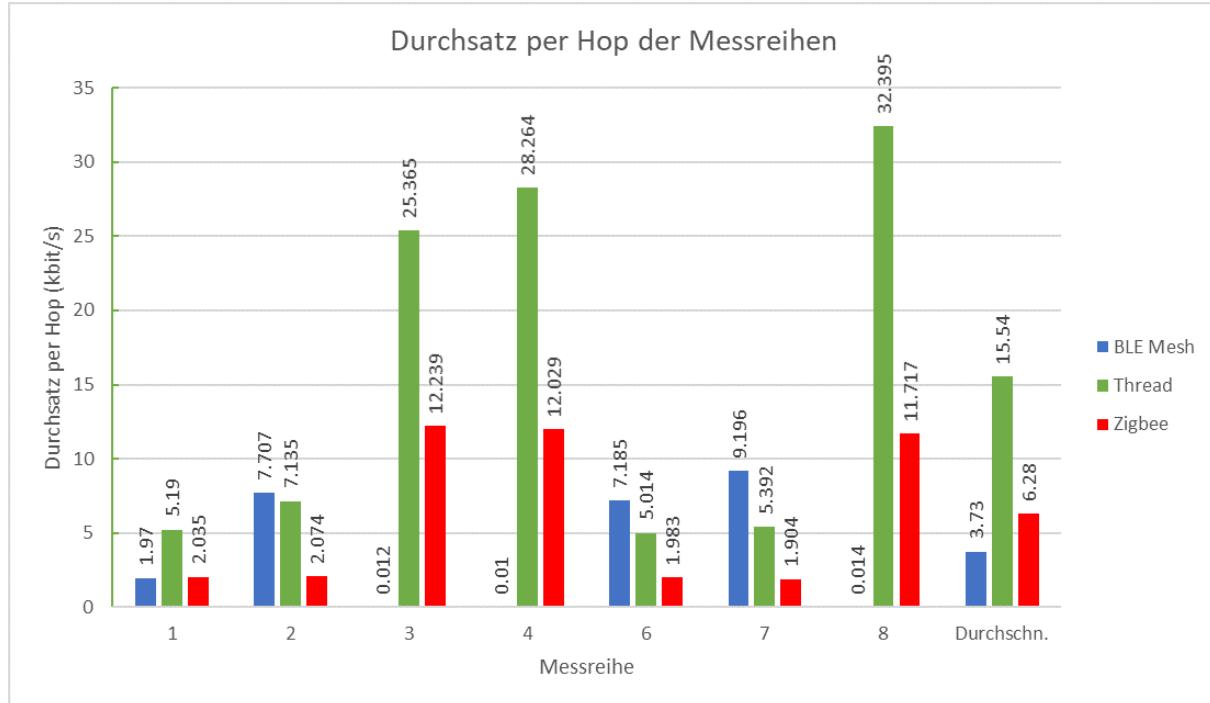


Abbildung 23.2: Durchschnittlicher Durchsatz per Hop der einzelnen Messreihen im Vergleich

23.1.3 Paketverlust

Bei der Betrachtung des Paketverlustes, zeigt sich Bluetooth-Mesh sehr anfällig auf die Erhöhung der Paketlänge. Die Abbildung 23.3 zeigt dies deutlich. In Abschnitt 23.3.3 wird diese Problematik genauer untersucht. Zigbee und Thread zeigen sich über alle Messreihen hinweg, sehr konstant und zuverlässig. Das CSMA/CA des IEEE 802.15.4 MAC Layers erweist sich auch hier als sehr nützlich.

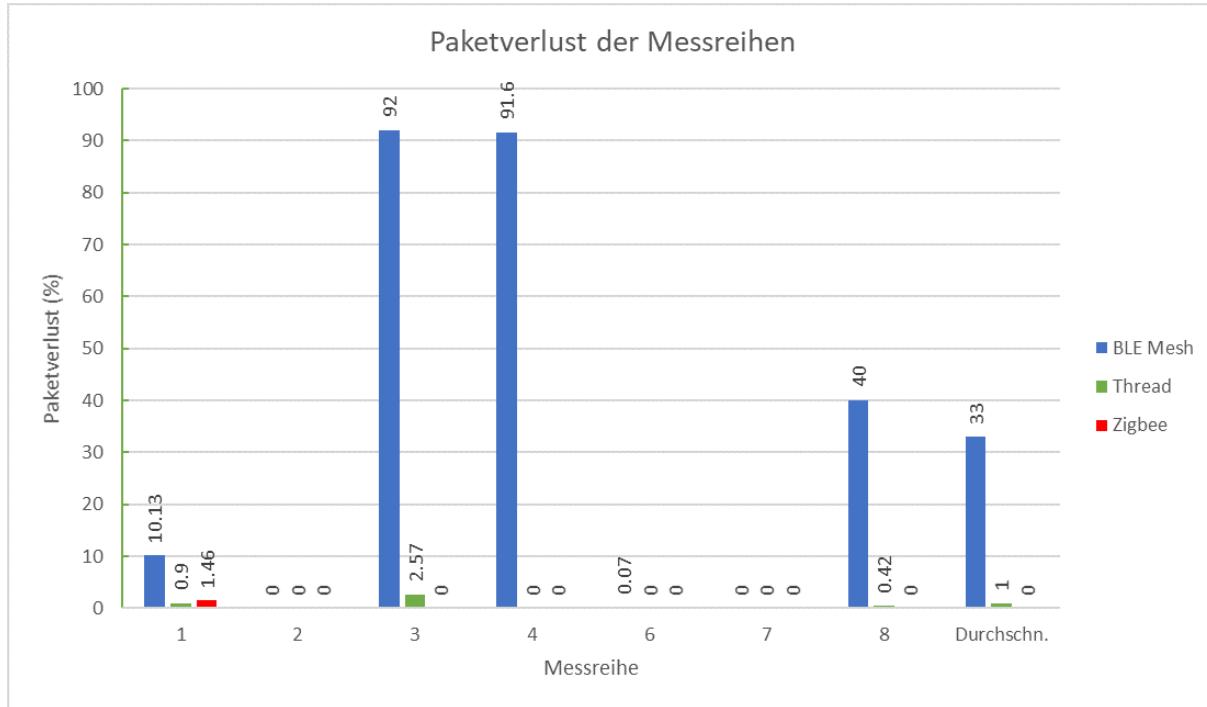


Abbildung 23.3: Durchschnittlicher Paketverlust der einzelnen Messreihen im Vergleich

23.1.4 Energieverbrauch

Durch Abbildung 23.4 wird ersichtlich, dass Bluetooth-Mesh den etwas geringeren Energiebedarf aufweist als Thread und Zigbee. Ansonsten zeigen alle Netzwerke ähnliche Resultate unabhängig der Messreihen. Die Unterschiede zwischen den Protokollen fallen sehr gering aus. Es kann also kein wesentlicher Unterschied bezüglich Energiebedarf festgestellt werden. Messung 7 und 8 dauerten länger als die restlichen Messungen, daher ist Ihr Energieverbrauch höher.

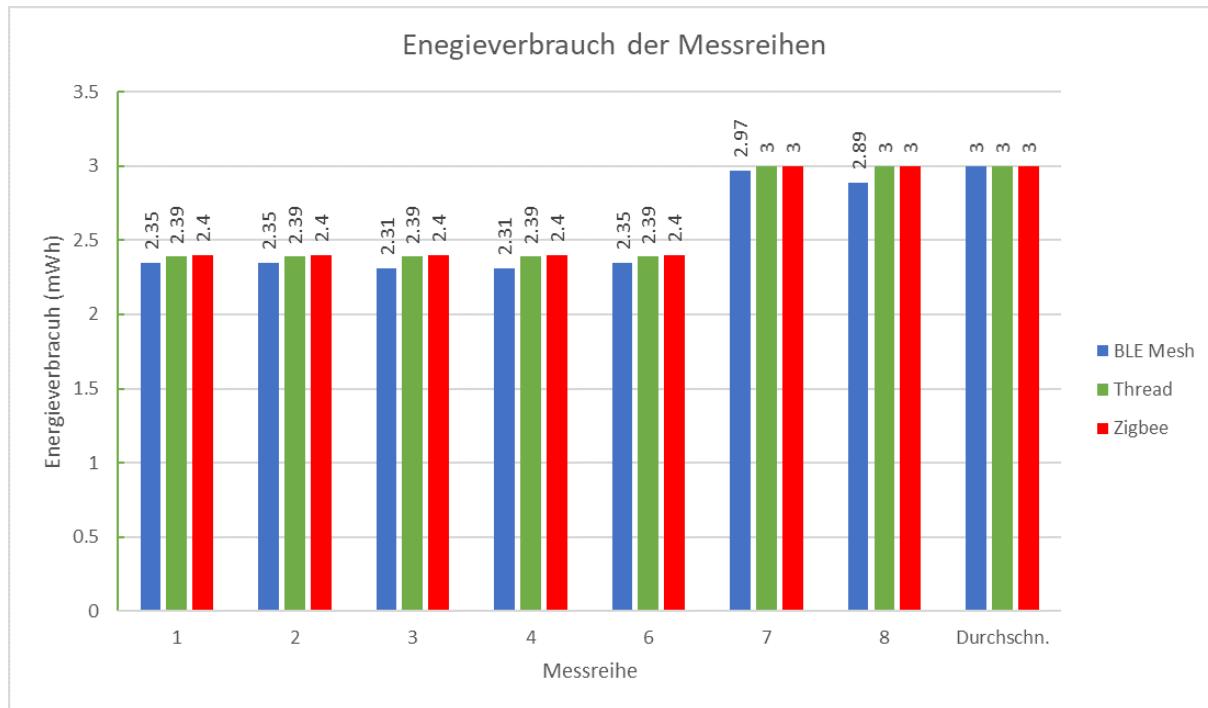


Abbildung 23.4: Durchschnittlicher Energiebedarf der einzelnen Messreihen im Vergleich

23.2 Vergleich Testumgebungen

Der nachfolgende Vergleich bezieht sich auf eine gleichbleibende Messreihe, welche in unterschiedlichen Testumgebungen durchgeführt wurde. Als Referenz wurde die Messreihe 2 ausgewählt, da diese für alle Testumgebung repräsentative Resultate aufweist (siehe Anhang D). Im Anschluss wird gezeigt, wie sich die Resultate der einzelnen Mesh-Netzwerke bei wechselnden Testumgebungen verändern.

23.2.1 Latenzzeit

Abbildung 23.5 zeigt, dass die Testumgebung Labor bei allen Netzwerken die geringsten Latenzzeiten verursacht. Bluetooth-Mesh erfährt bei flächenmäßig weiter Verteilung der Teilnehmer, wie sie im Haus anzutreffen ist, die grösste Steigerung der Latenzzeit. Zigbee weist in der Wohnung eine kaum relevante Erhöhung der Verzögerung auf und Thread zeichnet sich als unabhängig bezüglich der Testumgebung aus.

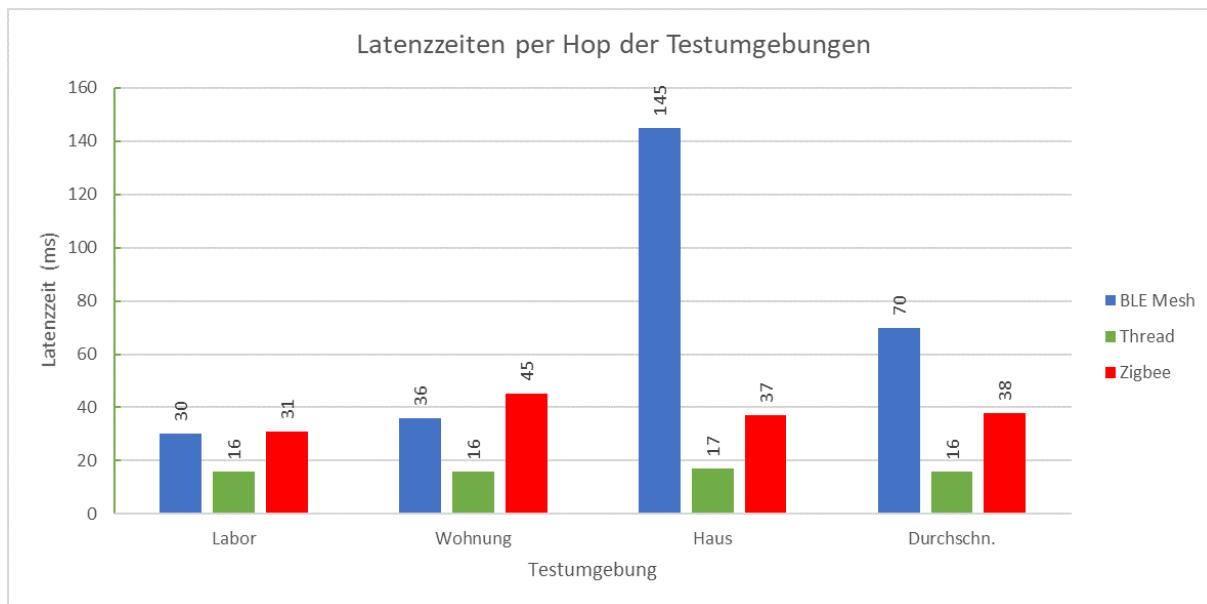


Abbildung 23.5: Durchschnittliche Latenzzeit per Hop der einzelnen Testumgebungen im Vergleich

23.2.2 Durchsatz

Ähnlich zu den Betrachtungen der Latenzzeit zeigt Abbildung 23.6 den höchsten Durchsatz im Testaufbau Labor. Bluetooth-Mesh erfährt hingegen, den grössten Einfluss durch die Änderung der Topologie.

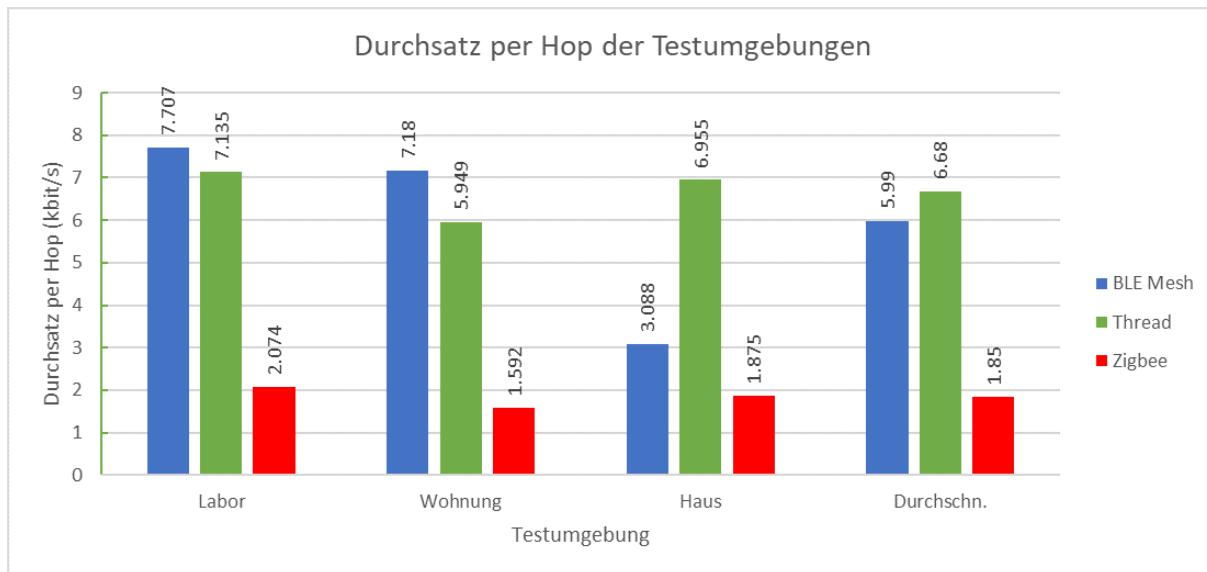


Abbildung 23.6: Durchschnittlicher Durchsatz per Hop der einzelnen Testumgebungen im Vergleich

23.2.3 Paketverlust

Abbildung 23.6 zeigt, dass im Labor-Aufbau alle Netzwerke zuverlässig und konstant arbeiten. In der Wohnung erlebt Bluetooth-Mesh einen geringen Anstieg des Paketverlusts während die Änderung bei Thread kaum relevant ist. Zigbee zeigt gar keine Veränderung der Paketverlustrate. Bei starker Verzweigung des Netzwerkes, wie dies im Haus der Fall ist, weisen alle Netzwerke einen Anstieg der Verlustrate auf.

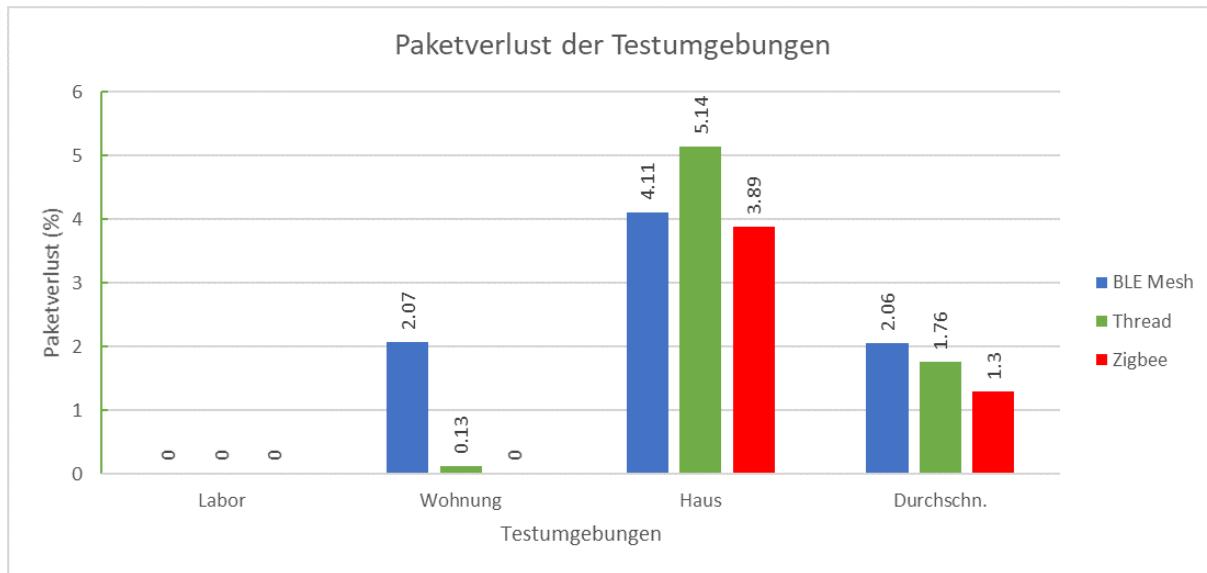


Abbildung 23.7: Durchschnittlicher Paketverlust der einzelnen Testumgebungen im Vergleich

23.3 Fazit

Alle Mesh-Netzwerke mussten diversen Tests standhalten. Die Ergebnisse aus den Messreihen 1 bis 4 aus allen Testumgebungen wurden als Durchschnittswert zusammengefasst, um einen finalen Vergleich zu erzielen. Abbildung 23.8 zeigt alle Ergebnisse auf einen Blick.

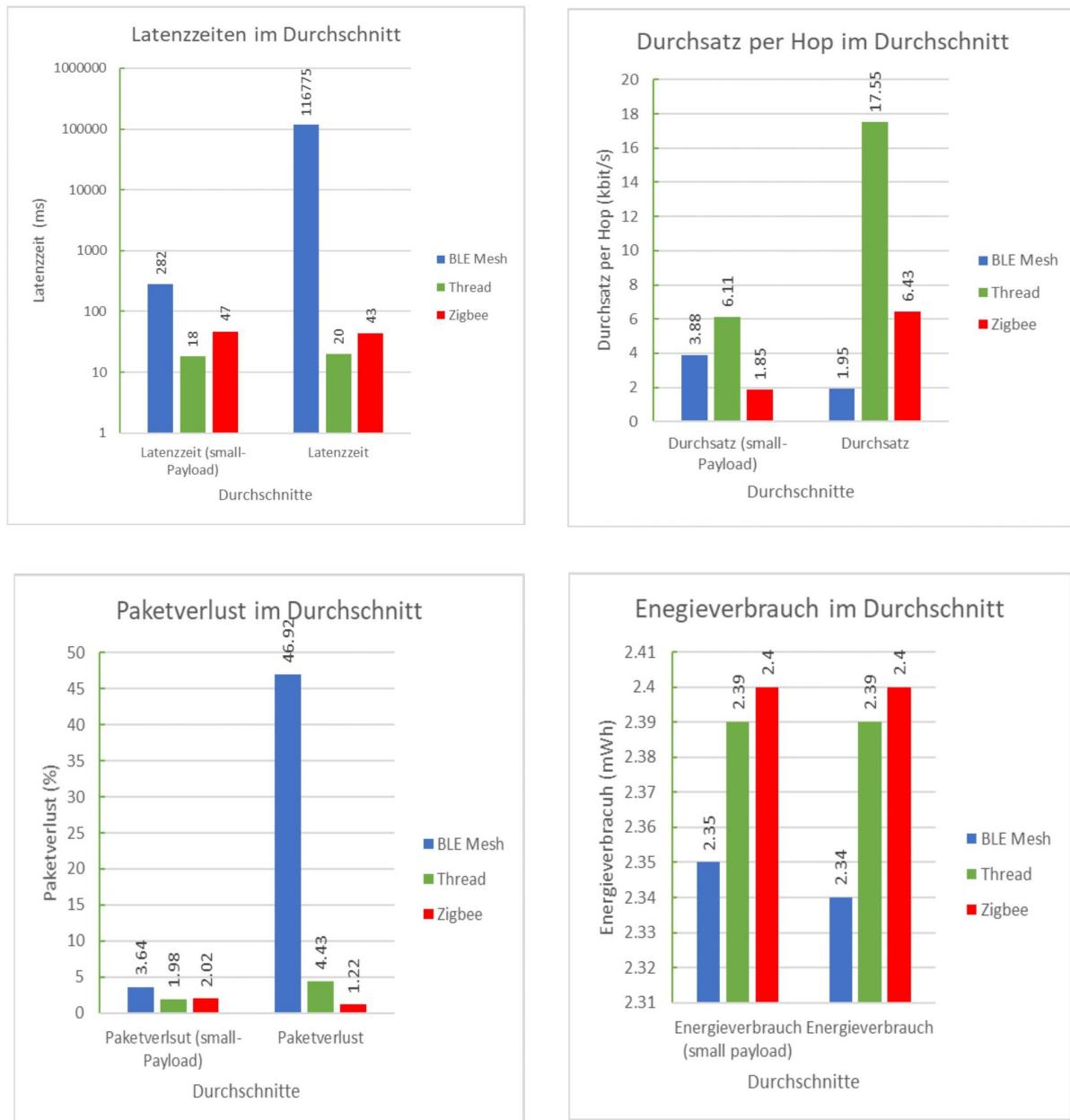


Abbildung 23.8: Durchschnittliche Messgrößen im Vergleich inkl. separater Betrachtung der Messergebnisse, welche mit 8-Byte Paketlänge erzielt wurden

Thread geht shliesslich als klarer Sieger hervor. Dieser Netzwerk-Stack hat die Tests mit den besten Ergebnissen absolviert. Auf dem zweiten Platz steht Zigbee, welches mit seiner konstanten Latenz und als zuverlässigstes Protokoll die Daten verarbeiten konnte. Dabei gilt zu beachten, dass bei Zigbee die Anzahl Hops nicht berücksichtigt wurden (siehe Abschnitt 22.2), was zu einer leichten Verfälschung der Ergebnisse führt. Bluetooth-Mesh kann zwar durch den niedrigen Energiebedarf brillieren, schneidet jedoch in Sachen Performance deutlich schlechter ab als seine Konkurrenten Thread und Zigbee. Im Anschluss wird jedes Protokoll genauer analysiert und auf die Vor- und Nachteile sowie mögliche Verbesserungen eingegangen.

23.3.1 Thread

Wie aus den vorhergehenden Kapiteln ersichtlich, hat der Thread Stack die Messungen mit den besten Ergebnissen abgeschlossen. Das bedeutet aber nicht unbedingt, dass der Stack der Beste ist. Es ist klar zu erkennen, dass der Thread Stack dank seiner automatischen Bestimmung von Routing-Knoten, sich seiner Umgebung gut anpassen kann. Die Latenzzeit der gesendeten Nachrichten ändert sich in verschiedenen Umgebungen nur minimal. Das bedeutet, dass sich der Thread Stack sehr gut für eine Hausautomation eignet. Wenn sich Sensoren oder Aktoren verschieben, z.B. wenn eine Lampe einen neuen Standort erhält, erkennt dies der Stack und kann das Routing der Knoten anpassen. Da sich das Netz sehr gut erweitern lässt und sich die Latenzzeiten eher tief zeigen, kann das Thread Netzwerk ausserdem gut für Industrieanwendungen verwendet werden.

Durch das automatische Routen der Knoten entsteht auf den einzelnen Nodes jedoch ein Overhead, der zum Beispiel BT Mesh nicht aufweist. Aus diesem Grund ist der Energieverbrauch auf den Knoten höher, was sich auch in Abbildung 23.4 zeigt. Dies ist zwar keine repräsentative Messung, da nur die aktiven Radio Zeiten gemessen wurden. Dennoch kann es die vorherige These mit dem Overhead bestätigen.

Anders als bei BT-Mesh können mit dem IEEE 802.15.4 und dem 6LoWPAN Layer grössere Pakete versendet werden. Das bedeutet, dass die Segmentierung im Gegensatz zu BT Mesh später stattfindet. Dies erklärt das Phänomen, dass sich der Durchsatz mit steigender Payload vergrössert. Der Thread Stack erreicht bei manchen Messreihen einen Durchsatz von bis zu 32 kbit pro Sekunde. Dies ist jedoch eher unwahrscheinlich. Da bei den Durchschnittsberechnungen kein Median verwendet wurde, können Extremwerte den Durchschnitt verfälschen. Es gab sehr wenige Latenzmessungen, die fehlerhaft waren und eine Latenz unter 0.5ms aufwiesen. Durch diesen Fehler wurde der Durchsatz in der Auswertung unwahrscheinlich hoch.

23.3.2 Zigbee

Die Resultate und Vergleiche aus den vorhergehenden Abschnitten zeigen, dass das Zigbee Protokoll in dieser Implementation eine solide Performance abliefert. Auch wenn Zigbee nicht ganz an jene Leistung von Thread herankommt, zeigen die Ergebnisse wieso Zigbee zurzeit das am weitest verbreitete WPAN Protokoll ist.

Die durchschnittlichen Latenzzeiten liegen zwischen 30ms und 50ms. Damit schneidet Zigbee besser als Bluetooth ab, jedoch liegen die Ergebnisse hinter jenen Durchschnittswerten von Thread. Dies kommt nicht allzu überraschend. Sehr interessant zu beobachten sind die Verteilungen der Latenzzeiten, wie sie beispielsweise in Abbildung 22.1 zu sehen sind. Anders als Thread weist Zigbee dort deutliche Peaks bei 40ms und 70ms auf. Diese Beobachtung kann bei sämtlichen Messresultaten gemacht werden. Ausreisser nach unten gibt es praktisch keine und solche nach oben nur sehr vereinzelt. Es kann also festgestellt werden, dass bei Zigbee mit einer minimalen Latenzzeit von ca. 30ms gerechnet werden muss, diese Schwelle aber in den seltensten Fällen deutlich überschritten wird. Zigbee wäre also auch für eher zeitkritische Anwendungen geeignet, da die Latenzzeiten deterministisch sind. In diesen Zusammenhang nochmals zu erwähnen ist, dass die Information über die Anzahl Hops die ein Paket passiert hat, leider nicht erfasst werden konnte (siehe Abschnitt 21.2). Es kann davon ausgegangen werden, dass der Peak bei 70ms in den Latenzzeiten, diesem Problem verschuldet ist. Wenn also die Anzahl Hops erfasst werden könnten, dürfte sich die Verteilung der Latenzzeiten wohl nochmals reduzieren und die Ergebnisse würden noch besser ausfallen.

Die Ergebnisse aus Abbildung 23.1 zeigen eine weitere interessante Eigenschaft von Zigbee. In Messreihe 1 beträgt die durchschnittliche Latenzzeit von 86ms mehr als das Doppelte der Durch-

schnittswerte in den übrigen Messreihen. Die Ursache dafür kann nicht abschliessend geklärt werden. Jedoch liegt die Vermutung nahe, dass sich das Mesh Netz während dieser Messreihe noch nicht komplett aufgebaut respektive das *Commissioning* und *Route Discovery* noch nicht abgeschlossen war. Dies verursachte zusätzlichen Traffic und beeinflusste den Benchmark.

Auffallend tief ist auch der Paketverlust. In den meisten Fällen lag dieser sogar bei 0 Prozent. Hier macht sich das CSMA/CA des *IEEE 802.15.4* MAC Layers bemerkbar. Zudem wirkt sich hier wohl die Unicast Adressierung innerhalb von Zigbee positiv aus. Die Gesamtbelastung des Netzes kann dadurch deutlich minimiert werden.

Die Erhöhung der Payload von 8 auf 50 Byte hat die Latenzzeiten bei den Zigbee Messungen nicht merklich beeinflusst. Dies bewirkt, dass der Durchsatz mit grösserer Payload zunimmt. Da die Segmentierung dank *IEEE 802.15.4* erst ab 127 Byte Framelänge startet, ist dies auch nicht weiter verwunderlich. Gegenüber BT-Mesh haben Zigbee wie auch Thread den klaren Vorteil grössere Payloads versenden zu können, ohne segmentieren zu müssen.

Zigbee brilliert in den Anwendungen, in denen es bereits verbreitet eingesetzt wird. Beispielsweise eine komplette Hausautomation oder eine einfachere Lichtsteuerung. Bis zu einer Netzwerkgrösse von ungefähr 200 Nodes mit einigermassen kleinem Verkehrsaufkommen ist Zigbee die ideale Wahl. Ergänzt durch die *Zigbee Cluster Library* welche die Systeme herstellerunabhängig macht wird Zigbee noch interessanter.

23.3.3 Bluetooth Mesh

Die Performance von Bluetooth-Mesh ist stark von der Belastung abhängig, wie aus Abschnitt 23.1 zu erkennen ist. Der grösste Einfluss auf die Performance hatte die Länge der gesendeten Nachrichten. Ursach dafür ist, dass ein einzelnes Bluetooth-Mesh-Paket lediglich 8 bis 10 Byte Payload aufnehmen kann. Der Stack beginnt mit der Segmentierung der Daten in mehrere kleine Pakete. Bei 32-Byte sind dies bereits 4 Frames. Damit ist der Network-Stack überfordert.

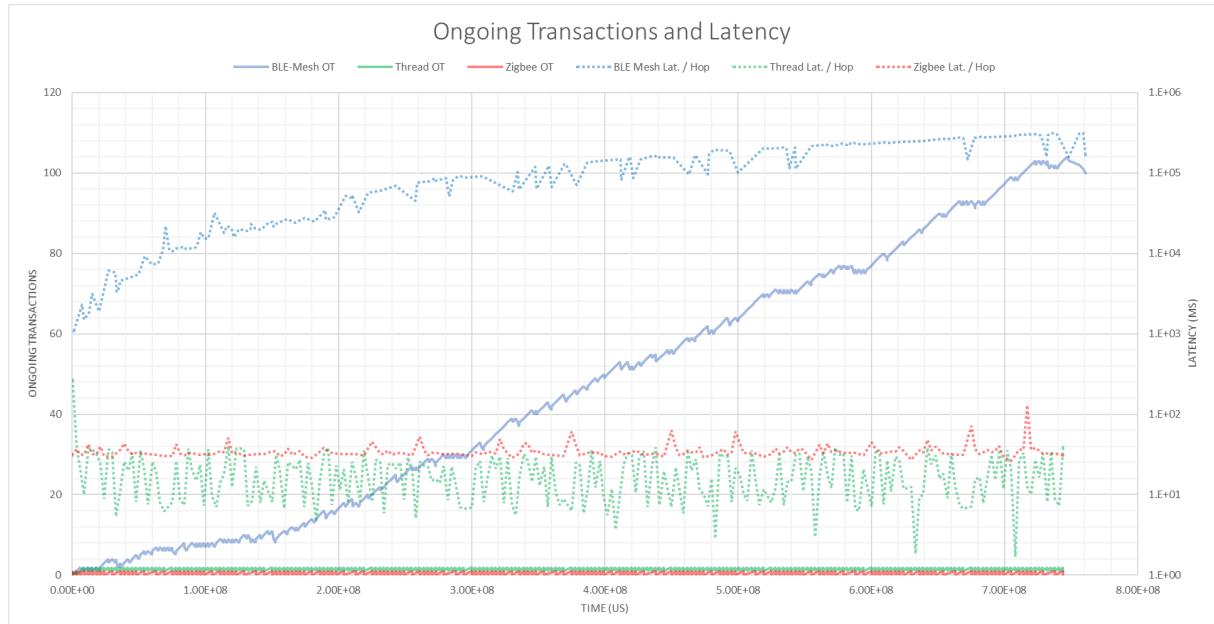


Abbildung 23.9: Ongoing Transactions und Latenzzeiten einer Messung mit 32Byte Paketlänge

Abbildung 23.9 zeigt eine Messung mit 32-Byte Payload bei einer Message-Dichte von 3 Sekunden/Message, welche sequentiell versendet wurden. Dabei ist zu erkennen, dass die Anzahl der

Ongoing Transactions immer weiter ansteigt. Erst in der Nachtbearbeitungszeit sinkt die Kurve wieder. Dies deutet darauf hin, dass zu viel Traffic im Netz vorhanden ist. In Korrelation mit den Ongoing Transactions steigt die Latenz von einer Sekunde auf einige 100 Sekunden an.

Vermutlich entsteht eine zu hohe Message-Dichte, da zu viele Relay-Nodes die empfangenen Messages wiederholen. Durch deaktivieren einiger Relays könnte der Netzaufbau entlastet werden. Jedoch erfordern solche manuellen Eingriffe Fachwissen und verringern die Ausfallsicherheit des Netzes.

24 Schluss

24.1 Zielerreichung

Für die vorliegende Projektarbeit wurden im Pflichtenheft (siehe Anhang B) klare Ziele definiert. In den Tabellen 24.1, 24.2, 24.3 und 24.4 sind diese in kurzer Form nochmals zusammengefasst. Ausserdem ist ersichtlich, welche Ziele erfüllt und welche nicht bzw. nur bedingt erfüllt werden konnten. Falls nötig ist auch noch ein kurzer Kommentar hinterlegt.

Punkt zu Punkt Testinfrastruktur

Projektziele Punkt zu Punkt Testinfrastruktur			
Nr.	Ziel	Erfüllt	Kommentar
P1	Kommunikation mit BMS	Ja	Alle Master Module werden via USB UART angesteuert.
P2	Senden von MAC Frames	Ja	
P3	Rückbestätigung der MAC-Frames	Ja	
P4	Konfiguration der Anzahl BSN	Bedingt	Alle Slave-Nodes werden automatisch erkannt und angezeigt.
P5	Adressierung der BSN	Bedingt	Alle Slave-Nodes werden automatisch erkannt und angezeigt.
P6	Konfiguration der Kanäle	Ja	Kann über den Webserver angepasst werden.
P7	Einstellbare Framelänge	Ja	Die Payload kann über den Webserver eingestellt werden.
P8	Einstellbare Frame und Kanalwechselrate	Nein	
P9	Einstellbare Sendeleistung	Ja	Kann über den Webserver angepasst werden.
P10	Anpassung der Modulationsart	Ja	Kann über den Webserver angepasst werden.
P11	Ein- und Ausschalten der Collision Avoidance (CSMA/CA)	Ja	Kann über den Webserver ein- und ausgeschaltet werden.
P12	Erfassen der Verbindungsqualität	Ja	Der Master sendet und empfängt in einem Zeitintervall Frames von den Slave-Nodes und sendet die Daten an die Serielle-Schnittstelle weiter.
P13	Tool für Feldtests	Ja	Das Tool ist mit dem Webserver und der Firmware einsatzbereit.

Tabelle 24.1: Erreichung der Ziele der Punkt zu Punkt Testinfrastruktur

Performancevergleich Mesh Netzwerke

Projektziele Performancevergleich Mesh Netzwerke			
Nr.	Ziel	Erfüllt	Kommentar
P1	Kommunikation mit BMS	Ja	Alle Master Module werden via USB UART angesteuert.
P2	Konfiguration BSN	Nein	Alle Nodes im Netzwerk fungieren als Routing-Knoten.
P3	Mesh-Netzwerk	Ja	Alle Netzwerke wurden in einem 50 Node Mesh-Netzwerk getestet.
P4	Simulation Sensorwerte	Ja	Über ein Python Programm können Testzeit und Anzahl Nachrichten konfiguriert werden. Es können sogar verschiedene Zufalls-Modi gewählt werden.
P5	Sensordaten	Ja	Die Sensordaten werden im RAM und FLASH des SoCs gespeichert und nach der Messung an den Master gesendet.
P6	Datenauswertung	Ja	Die Daten der Messungen wurden mit Hilfe von Excel Tabellen ausgewertet.
P7	Störimmunität	Ja	Störmessungen mit der P2P-Testinfrastruktur wurden gemacht.
P8	Unterschiedliche Test Bedingungen	Ja	Es wurden die drei Bereiche Haus, Wohnung und Labor getestet.
P9	Test und Validierung	Ja	Die Mesh Netzwerke wurden im Fachbericht vollumfänglich verglichen und ausgewertet.

Tabelle 24.2: Erreichung der Ziele zu den Test Mesh Netzwerken

Steuer- und Auswertesoftware

Projektziele Steuer- und Auswertesoftware			
Nr.	Ziel	Erfüllt	Kommentar
P1	Ansteuerung Funkmodul	Ja	Alle Master Module werden via USB UART angesteuert.
P2	Visualisierung Parameter	Bedingt	Für die P2P-Testinfrastruktur steht ein Webserver zur Verfügung. Die Daten des Mesh-Tests werden in einer Excel-Tabelle ausgewertet und visualisiert.
P3	User Interface (UI)	Ja	Ein Django Webserver stellt ein UI für die Verwaltung der P2P-Testinfrastruktur zur Verfügung.
P4	Konfiguration Mesh-Netzwerk	Ja	Ein Python Programm konfiguriert die Mesh Nodes über den BMN.
P5	Einheitliche Kommunikation von BMS	Ja	Ein Python Programm steuert die Mesh Nodes über den BMN.

Tabelle 24.3: Erreichung der zur Steuer- und Auswertesoftware

Zusatzziele

Projektziele Zusatz			
Nr.	Ziel	Erfüllt	Kommentar
P1	Hardware Testmodul BMN/BSN	Nein	In Absprache mit den Fachcoaches, ist ein Hardwaremodul nicht notwendig.
P2	Vergleich SoC	Bedingt	Die Bluetooth Tests wären mit verschiedenen SoCs möglich, diese Messungen wurden aber nicht ausgeführt.
P3	Drahtlos Konfiguration	Nein	Ein Firmware over the air upgrade wurde nicht umgesetzt.
P4	UI für Mesh Test	Nein	Für die Mesh-Tests wurde aus zeitlichen Gründen kein User-Interface geschrieben

Tabelle 24.4: Erreichung der Zusatzziele

24.2 Fazit

In dieser Arbeit wurden die drei Mesh Protokolle Bluetooth Mesh, Thread und Zigbee unter verschiedenen Testszenarien ausgemessen und anschliessend miteinander verglichen. Die wichtigste Erkenntnis ist, dass es kein bestes Mesh Netzwerk gibt. Alle Protokolle haben ihre Stärken und Schwächen in verschiedenen Umgebungen und mit verschiedenen Messparametern. Nachfolgend wird zusammengefasst, welche Schlüsse aus dem Vergleich der drei Protokolle gezogen worden sind.

- Einer der grössten Vorteile von Bluetooth Mesh ist, die Interoperabilität. Der Mesh-Stack ist mit Milliarden von Geräten Weltweit kompatibel.
- Ein Nachteil von Bleutooth Mesh ist das es über kein Routing verfügt. Das Managed Flooding Prinzip führt zu hohen Netzwerk Belastungen, wenn die Topologie eine sehr hohe Node-Dichte aufweist.
- Der grösste Vorteil von Thread ist, dass sich das Mesh-Netzwerk selber ausmisst. Die routenden Knoten werden automatisch bestimmt. Somit kann sich das Protokoll der Umgebung anpassen und konnte dadurch auch in allen Testumgebungen eine tiefe Latenz aufweisen.
- Ein Nachteil von Thread ist, dass aufgrund des automatischen Routen ein höherer Overhead entsteht. Durch diesen Overhead steigt der Energieverbrauch auf den einzelnen Nodes.
- Zigbee weist eine hohe Zuverlässigkeit auf. Bedingt durch das CSMA/CA des IEEE 802.15.4 MAC und Physical Layers erreichen rund 98 Prozent der Nachrichten ihr Ziel mit konstant tiefer Latenzzeit.
- Der gemessene Durchsatz und die gemessene Latenzzeit von Zigbee, können nicht ganz mit jenen Werten von Thread mithalten. Daher ist die Performance bei hoher Belastung eher schlechter.

Nachfolgend wird auf einige Probleme eingegangen, welche während dem Projekt entstanden sind. Außerdem werden mögliche Lösungen dafür vorgeschlagen. Die Probleme werden in organisatorische und technische Verbesserungsvorschläge aufgeteilt.

Organisatorische Verbesserungen

- Ein frühzeitig definiertes Softwarekonzept hätte sehr viel Redundanz in der verschiedenen Firmware Teilen erspart und somit hätte viel effizienter gearbeitet werden können.
- Der zu Beginn erstellte Zeitplan wurde nicht eingehalten, deswegen wurden die Arbeiten zum Teil unstrukturiert ausgeführt, was zu Zeitmangel führte.

Technische Verbesserungen

- Durchschnittswerte sollten mit dem Median bestimmt werden. Der verwendete Mittelwert wird zu sehr von Ausreissern in der Latenzzeit verfälscht. Daher ist die durchschnittliche Latenz meist nicht repräsentativ.
- Es wurden ungünstige Messparameter verwendet. Die Nachrichtendichte ist zu gross gewählt und muss heruntergesetzt werden, damit sich die Protokolle besser vergleichen lassen.
- Beim Reporting vom BSN zum BMN ist es möglich, dass sich die Daten von zwei Slaves überschneiden. Mit Hilfe der Node ID könnte dies verhindert werden, indem nur der BSN mit der entsprechender Node ID angesprochen wird.
- Beim Zigbee Protokoll ist es leider nicht möglich die Anzahl Hops auszulesen, die eine Nachricht bei der Übertragung passiert. Dies müsste für eine repräsentative Auswertung noch implementiert werden können, da die Latenzzeit pro Hop bestimmt wird.
- Die Message ID der Nodes werden in der Payload der Benchmark-Nachricht mitgegeben. Dadurch ist es nicht möglich eine Payload unter 2 Byte zu versenden. Besser wäre ein Auslesen der Message ID vom entsprechenden Header.

24.3 Ausblick

In diesem Abschnitt wird darauf eingegangen, welche Module zu den bereits bestehenden Modulen noch ergänzt werden könnten, um z.B. die Bedienung noch benutzerfreundlicher zu machen. Diese Informationen dienen als Hinweis, falls die Arbeit von einer anderen Person weitergeführt wird.

- Damit die Bedienung benutzerfreundlicher wird, kann für die gesamte Mesh-Test Umgebung ein GUI erstellt werden. Mit Hilfe einem User Interface können alle Einstellungen und Auswertung direkt an einem Ort visualisiert werden. Durch dies würde eine mühsame Auswertung mit externen Programmen wie Excel entfallen.
- Ein Over-The-Air Firmware Upgrade würde die Bedienung zusätzlich vereinfachen. Mit Hilfe des GUIs kann das entsprechende Protokoll ausgewählt werden und die BSN werden mit der entsprechenden Firmware geladen. Das würde bedeuten, dass ein mühseliges Flashen der einzelnen Nodes entfällt.
- Damit die einzelnen BSN robuster werden und auch weitere Parameter wie z.B. der Stromverbrauch erfasst werden können, ist eine eigene Hardware Entwicklung sinnvoll. Dadurch könnte der Batteriestand von den einzelnen Nodes bestimmt werden und ein Warnung würde dem Benutzer mitteilen, dass die Batterien ausgetauscht werden müssen.

24.4 Schlusswort

Das Ziel dieser Bachelor-Thesis war die drei Mesh-Protokolle BT-Mesh, Thread und Zigbee miteinander zu vergleichen. Dieses Ziel konnte erreicht werden. Der Weg zu diesem Ziel war teilweise steinig und ungewiss. Da das Softwarekonzept und die Arbeitsaufteilung besonders zu Beginn der Arbeit nur teilweise umgesetzt werden konnten, mussten viele Zeilen Code doppelt geschrieben werden. Dank dem sehr grossen Durchhaltewillen aller Teammitglieder und dem starken Entschluss eine sehr gute Bachelor-Thesis zu liefern, konnten trotz diversen Rückschlägen aussagekräftige Resultate erzielt werden. Durch das Einlesen und Einarbeiten in neue Themen wie Mesh-Netzwerke, Firmwareprogrammierung, Webserver, Funksysteme und Zeitsynchronisation konnten zahlreiche neue Erfahrungen gesammelt werden. Dank diesen Erfahrungen wurden die Kenntnisse der Programmiersprachen C, C++, Python, HTML und Java Script verbessert oder gar neu erlernt. Dank der guten Teamarbeit konnte schliesslich ein Resultat erzielt werden, das aussagekräftig ist und ein Grossteil der gesetzten Ziele erfüllt.

Zum Schluss möchten wir uns bei unseren Projektbetreuern Herrn M. Meier und Herrn M. Di Cerbo für die fachliche Unterstützung bedanken. Sie haben uns bei unseren Ideen und unserem Vorgehen stets unterstützt und halfen uns fachlich weiter, wo es nötig war. Wir danken ihnen, dass wir unsere Visionen in dieser Bachelor-Thesis verwirklichen durften.

Literatur

- [1] Github, *P6_Software_P2P*, en, Library Catalog: github.com, Aug. 2020. Adresse: https://github.com/Rouben94/P6_Software/P2P (besucht am 7. Aug. 2020).
- [2] nRF_Connect_SDK, *Radio_Test_Example — nRF Connect SDK 1.3.99 documentation*, Aug. 2020. Adresse: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/samples/peripheral/radio_test/README.html#radio-test (besucht am 8. Aug. 2020).
- [3] Nordic_Semi, *nRF_Infocenter_Radio_States*, Aug. 2020. Adresse: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52840%2Fradio.html&cp=4_0_0_5_19_4&anchor=concept_rnf_nml_4r (besucht am 8. Aug. 2020).
- [4] —, *nRF_Infocenter_Radio_Transmit_Sequence*, Aug. 2020. Adresse: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52840%2Fradio.html&cp=4_0_0_5_19_5&anchor=concept_db5_cnl_4r (besucht am 8. Aug. 2020).
- [5] —, *nRF_Infocenter_Radio_IEEE_Operation*, Aug. 2020. Adresse: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52840%2Fradio.html&cp=4_0_0_5_19_11&anchor=concept_wpg_n3j_4r (besucht am 8. Aug. 2020).
- [6] —, *nRF_Infocenter_Radio_Receive_Sequence*, Aug. 2020. Adresse: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52840%2Fradio.html&cp=4_0_0_5_19_4&anchor=concept_rnf_nml_4r (besucht am 8. Aug. 2020).
- [7] RK, *How_to_deal_with_broadcasting_collision?*, en, Aug. 2020. Adresse: <https://devzone.nordicsemi.com/f/nordic-q-a/13685/how-to-deal-with-broadcasting-collision> (besucht am 9. Aug. 2020).
- [8] Nordic_Semi, *nRF_Infocenter_PPI*, Aug. 2020. Adresse: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52840%2Fppi.html&cp=4_0_0_5_15 (besucht am 8. Aug. 2020).
- [9] Django, *Django documentation*, en, Jan. 2020. Adresse: <https://docs.djangoproject.com/en/3.1/> (besucht am 21. Aug. 2020).
- [10] MDN Web Docs, *Django introduction*, en, Nov. 2019. Adresse: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction> (besucht am 21. Aug. 2020).
- [11] Silicon Laboratories Inc., *AN1142: Mesh Network Performance Comparison*, Aug. 2020. Adresse: <https://www.silabs.com/documents/public/application-notes/an1142-mesh-network-performance-comparison.pdf> (besucht am 24. Aug. 2020).
- [12] N. S. ASA, *nRF52840 - Nordic Semiconductor*, en, Aug. 2020. Adresse: <https://www.nordicsemi.com/en/Products/Low%20power%20short-range%20wireless/nRF52840> (besucht am 9. Aug. 2020).
- [13] Nordic Semiconductor ASA, *nRF52840_PS_v1.1.pdf*. Adresse: https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf (besucht am 22. Aug. 2020).
- [14] —, *nRF52840 DK*, en, Aug. 2020. Adresse: <https://www.nordicsemi.com/en/Software%20and%20tools/Development%20Kits/nRF52840%20DK> (besucht am 9. Aug. 2020).
- [15] —, *nRF52840 Dongle*, en, Aug. 2020. Adresse: <https://www.nordicsemi.com/en/Software%20and%20tools/Development%20Kits/nRF52840%20Dongle> (besucht am 9. Aug. 2020).
- [16] N. S. ASA, *nRF52840_Dongle_User_Guide_v1.1.pdf*, Jan. 2019. Adresse: https://infocenter.nordicsemi.com/pdf/nRF52840_Dongle_User_Guide_v1.1.pdf (besucht am 9. Aug. 2020).
- [17] Distrelec Schweiz AG, *RND 305-00002_eng_tds*, Aug. 2020. Adresse: https://www.distroelec.ch/Web/Downloads/_t/ds/RND%20305-00002_eng_tds.pdf (besucht am 17. Aug. 2020).

- [18] N. S. ASA, *PCA10056_Schematic_And_PCB*, Okt. 2019. Adresse: http://infocenter.nordicsemi.com/pdf/nRF52840_DK_User_Guide_v1.4.1.pdf (besucht am 14. Mai 2020).
- [19] Nordic Semiconductor ASA, *PCA10059_Schematic_And_PCB*, Aug. 2020. Adresse: https://www.nordicsemi.com/-/media/Software-and-other-downloads/Dev-Kits/nRF52840-Dongle/nRF52840-USB-Dongle---Hardware-files-1_0_0.zip (besucht am 9. Aug. 2020).
- [20] Nordic_Semi, *Welcome_to_the_nRF_Connect_SDK! — nRF Connect SDK 1.3.99 documentation*, Aug. 2020. Adresse: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/index.html (besucht am 15. Aug. 2020).
- [21] —, *nRF_SDK_for_Thread_and_Zigbee*, Aug. 2020. Adresse: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_sdk%2Fstruct%2Fsdk_thread_zigbee_latest.html (besucht am 15. Aug. 2020).
- [22] —, *nRF_SDK_for_Mesh*, Aug. 2020. Adresse: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_sdk%2Fstruct%2Fsdk_mesh_latest.html (besucht am 15. Aug. 2020).
- [23] Anklin, Bobst, Horath, *Rouben94/P6_Software*, en, Library Catalog: github.com. Adresse: https://github.com/Rouben94/P6_Software (besucht am 7. Aug. 2020).
- [24] Rouben94, *SharedLib_Timesync_Software_Git*, en, Aug. 2020. Adresse: https://github.com/Rouben94/P6_Software/blob/master/SharedLib/bm_timesync.c (besucht am 16. Aug. 2020).
- [25] Nordic_Semi, *nRF5_SDK_Flash_Data_Storage*, Aug. 2020. Adresse: https://infocenter.nordicsemi.com/topic/sdk_nrf5_v16.0.0/group__fds.html (besucht am 21. Aug. 2020).
- [26] Bluetooth_SIG, *Mesh-Technology-Overview.pdf*, Aug. 2020. Adresse: <https://www.bluetooth.com/wp-content/uploads/2019/03/Mesh-Technology-Overview.pdf> (besucht am 12. Aug. 2020).
- [27] —, *Mesh_Netzwerk_Spezifikationen / Bluetooth®-Technologie Website*, de, Aug. 2020. Adresse: <https://www.bluetooth.com/de/specifications/mesh-specifications/> (besucht am 13. Aug. 2020).
- [28] M. Woolley, *How_Bluetooth_Mesh_Puts_the_Large_in_Large-Scale_Wireless_Device_Networks*, en-US, Juni 2018. Adresse: <https://www.bluetooth.com/blog/mesh-in-large-scale-networks/> (besucht am 23. Aug. 2020).
- [29] Thread Group, Inc., Hrsg., *Thread Specification 1.1.1*, en, Feb. 2017. Adresse: <https://www.threadgroup.org/ThreadSpec> (besucht am 5. Aug. 2020).
- [30] G. Erickson, S. Neidig und J. Hui, *Picture protocol layer thread*, en, Dez. 2019. Adresse: <https://www.threadgroup.org/Portals/0/Images/blog/layerprotocols.png> (besucht am 4. Aug. 2020).
- [31] Thread Group, *What_is_Thread*, en, Jan. 2020. Adresse: <https://www.threadgroup.org/What-is-Thread> (besucht am 15. Aug. 2020).
- [32] Z. Shelby, K. Hartke und C. Bormann, *The Constrained Application Protocol (CoAP)*, en, Library Catalog: tools.ietf.org, Juni 2014. Adresse: <https://tools.ietf.org/html/rfc7252> (besucht am 15. Aug. 2020).
- [33] J. Postel, *User Datagram Protocol*, en, Library Catalog: tools.ietf.org, Aug. 1980. Adresse: <https://tools.ietf.org/html/rfc768> (besucht am 15. Aug. 2020).
- [34] P. Levis und T. H. Clausen, *The Trickle Algorithm*, en, Library Catalog: tools.ietf.org, März 2011. Adresse: <https://tools.ietf.org/html/rfc6206> (besucht am 15. Aug. 2020).

- [35] P. Thubert und J. W. Hui, *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*, en, Library Catalog: tools.ietf.org, Sep. 2011. Adresse: <https://tools.ietf.org/html/rfc6282> (besucht am 15. Aug. 2020).
- [36] IEEE Computer Society, *IEEE 802.15.4-2020 - IEEE Standard for Low-Rate Wireless Networks*, en, Juli 2020. Adresse: https://standards.ieee.org/standard/802_15_4-2020.html (besucht am 15. Aug. 2020).
- [37] Openthread, *ot-primer-roles_2x.png (1656×1708)*, en, Juni 2016. Adresse: https://openthread.io/guides/images/ot-primer-roles_2x.png (besucht am 25. Aug. 2020).
- [38] ——, *ot-primer-taxonomy_2x.png (1480×1688)*, en, Juni 2016. Adresse: https://openthread.io/guides/images/ot-primer-taxonomy_2x.png (besucht am 25. Aug. 2020).
- [39] ——, *ot-primer-scopes_2x.png (2708×2676)*, en, Juni 2016. Adresse: https://openthread.io/guides/images/ot-primer-scopes_2x.png (besucht am 25. Aug. 2020).
- [40] Markus Krause, Rainer Konrad, „ZigBee“, Deutsch, in *Drahtlose ZigBee-Netzwerke - Ein Kompendium*, Springer Vieweg, 2014, S. 215–331, ISBN: 978-3-658-05821-0. (besucht am 18. Apr. 2020).
- [41] ——, „IEEE 802.15.4“, Deutsch, in *Drahtlose ZigBee-Netzwerke - Ein Kompendium*, Springer Vieweg, 2014, S. 83–105, ISBN: 978-3-658-05821-0. (besucht am 18. Apr. 2020).
- [42] The ZigBee Alliance, *ZigBee Cluster Library Specification*, English, Jan. 2016. Adresse: <https://zigbeealliance.org/wp-content/uploads/2019/12/07-5123-06-zigbee-cluster-library-specification.pdf> (besucht am 24. März 2020).
- [43] DSR Corporation, *DSR IoT*. Adresse: <https://dsr-iot.com/> (besucht am 26. Aug. 2020).
- [44] Markus Krause, Rainer Konrad, *Drahtlose ZigBee-Netzwerke - Ein Kompendium*, Deutsch. Springer Vieweg, 2014, ISBN: 978-3-658-05821-0. (besucht am 18. Apr. 2020).
- [45] The ZigBee Alliance, *ZigBee Specification*, English, Aug. 2015. Adresse: <https://zigbee-alliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf> (besucht am 22. Juni 2020).
- [46] Cyril Horath, *Zigbee - Read number of hops (radius)*, Support Forum, Juli 2020. Adresse: <https://devzone.nordicsemi.com/f/nordic-q-a/63815/zigbee---read-number-of-hops-radius>.

Abbildungsverzeichnis

3.1	Konzeptschema P2P Testinfrastruktur	6
3.2	Messaufbau P2P Testinfrastruktur	7
4.1	Webserver Nodelist View ausgeklappt	9
4.2	Webserver Connection View	10
4.3	Webserver Nodelist View	11
4.4	Webserver Chart View	11
5.1	Schrittfolge P2P Testinfrastruktur	12
5.2	Radio States [3]	13
5.3	Sende-Ablauf mit Verknüpfungen [4]	14
5.4	Sende-Ablauf mit CCA Idle	15
5.5	Sende-Ablauf mit CCA Busy	15

5.6 Empfangs-Ablauf mit Verknüpfungen [6]	15
5.7 Zeitsynchronisation über Offset mit Fehler	16
5.8 Zeitsynchronisation über Offset mit PPI	17
5.9 Verifizierung Zeitsynchronisation mittels Oszilloskop.	18
5.10 Django Funktionen [10]	19
5.11 Funktionen P2P Webserver	20
5.12 Parameter Form	22
5.13 Port Form	22
5.14 Ablauf Django Serial	23
7.1 Konzeptschema für den Ablauf eines Mesh Benchmarks.	26
7.2 Anordnung Messumgebung 1: Labor	28
7.3 Anordnung Messumgebung 2: Einfamilienhaus	29
7.4 Anordnung Messumgebung 3: Wohnung	30
7.5 MAC Adresse nRF52840-Dongle	35
8.1 nRF52840-DK PCA10056	38
8.2 nRF52840-Dongle PCA10059	38
8.3 Testnode: nRF52840-Dongle inkl. Batteriepack	38
8.4 PCA10056 Copperplane mit IFA-Antenne	39
8.5 PCA10059 Copperplane mit MIFA-Antenne	39
8.6 Abstrahlung in Z-Richtung PCA10056	39
8.7 Abstrahlung in Z-Richtung PCA10059	39
9.1 Vereinfachte Anbindung der <i>Shared Library</i> an die Mesh-Stacks	43
9.2 Flowgraph der State machine	43
9.3 Konzept des Depth-Feld beim Versand einer Control Message	46
9.4 Ablauf des Reportings mit Handshake zwischen Master und Slave.	47
10.1 Parkhaus mit Bluetooth-Mesh	52
10.2 Stadion mit Bluetooth-Mesh	52
10.3 Client - Server Prinzip Bluetooth Mesh [26]	53
10.4 Publish - Subscribe Prinzip Bluetooth Mesh [26]	54
11.1 Beispielhafte Topologie eines Bluetooth-Mesh Netzwerks [27]	55
11.2 Bluetooth-Mesh Stack [26]	56
14.1 Thread Protokoll Layer [30]	63
15.1 Übersicht Thread Protokoll	65
15.2 Thread Device Roles [37]	67

15.3 Thread Device Types [38]	68
15.4 IPv6 Adressräume [39]	69
16.1 Channel Bandbreite zand_fig_nodate	73
16.2 Initialisierung Thread Stack	73
18.1 Zigbee Protokollstack	78
19.1 Zigbee Baumnetzwerk links und Meshnetzwerk rechts [40, S. 221]	79
19.2 Architektur des Zigbee Protokoll Stacks	80
19.3 Zigbee Frame Struktur bei aktivierten Sicherheitsfunktionen [40, S. 286]	82
20.1 nRF5 SDK for Thread and Zigbee Plattform Design Referenz für Zigbee Applikationen [21]	84
20.2 Konkurrenz Zigbee/IEEE und WLAN Funkkanäle [44]	85
20.3 Aufbau APS Datenframe [44]	86
22.1 Messung 2 Wohnung: Verteilung der Latenzzeiten pro Hop	94
22.2 Messung 2 Wohnung: Durchschnittliche Latenzzeit pro Hop	94
22.3 Messung 2 Wohnung: Durchschnittlicher Durchsatz pro Hop	94
22.4 Messung 2 Wohnung: Durchschnittlicher Paketverlust	95
22.5 Messung 2 Wohnung: Durchschnittlicher Energieverbrauch	95
22.6 Messung 2 Wohnung: Ongoing Transactions und Verlauf der Latenzzeiten über die Messdauer.	95
22.7 Durchschnittliche Latenzzeit pro Hop	97
22.8 Durchschnittliche Latenzzeit ohne Berücksichtigung der Hops.	97
23.1 Durchschnittliche Latenzzeit per Hop der einzelnen Messreihen im Vergleich	100
23.2 Durchschnittlicher Durchsatz per Hop der einzelnen Messreihen im Vergleich	101
23.3 Durchschnittlicher Paketverlust der einzelnen Messreihen im Vergleich	102
23.4 Durchschnittlicher Energiebedarf der einzelnen Messreihen im Vergleich	103
23.5 Durchschnittliche Latenzzeit per Hop der einzelnen Testumgebungen im Vergleich	104
23.6 Durchschnittlicher Durchsatz per Hop der einzelnen Testumgebungen im Vergleich	104
23.7 Durchschnittlicher Paketverlust der einzelnen Testumgebungen im Vergleich	105
23.8 Durchschnittliche Messgrößen im Vergleich inkl. separater Betrachtung der Messergebnisse, welche mit 8-Byte Paketlänge erzielt wurden	106
23.9 Ongoing Transactions und Latenzzeiten einer Messung mit 32Byte Paketlänge	108
G.1 Bluetooth-Mesh Stack Upper Layers [27]	196
G.2 Bluetooth-Mesh Stack Lower Layers [27]	197

A Aufgabenstellung



Aufgabenstellung P6 / Thesisarbeit FS20

Wireless Controller for Smart Systems

Test und Vergleich von GHz Low Power Mesh Netzwerken

Studierende	Raffael Anklin Robin Bobst Cyrill Horath
Betreuende Dozenten	Manuel Di Cerbo FHNW Studiengang EIT manuel.dicerbo@fhnw.ch Matthias Meier FHNW Studiengang EIT matthias.meier@fhnw.ch
Ausgangslage	<p>Unter den standardisierten Low Power Mesh Netzwerk Protokollen im freien GHz ISM-Band konkurrieren sich derzeit vorrangig Bluetooth Mesh, Zigbee sowie Thread.</p> <p>Bezüglich MAC und Physical Layer basieren Zigbee und Thread auf IEEE 802.15.4 wogegen Bluetooth Mesh auf Bluetooth Low Energy (BLE) basiert.</p> <p>Jedes dieser Netzwerkprotokolle hat gewisse Vorteile: Bluetooth Mesh, dass BLE mittlerweile von jedem Smartphone und Notebook unterstützt wird, Thread aufgrund seiner IPv6 Basis und damit einfacher Übergang ins Internet sowie Zigbee aufgrund seiner etablierten Verbreitung im Smart-Lampenbereich durch Philips, IKEA und Osram.</p> <p>Hauptproblem aller drei Mesh Netzwerkprotokolle ist nebst physikalisch und distanzbedingter Absorption und Reflexion die Störbeeinflussung durch WLAN (WiFi) und andere Netzwerke im GHz Frequenzbereich.</p>

Ziel der Arbeit

In der vorliegenden Arbeit soll zuerst ein praxistaugliches, einheitliches Testframework für alle drei Mesh Netzwerke erstellt werden, wonach die Tauglichkeit aller drei Mesh Netzwerke unter realitätsnahen Bedingungen ermittelt und verglichen werden soll.

Zwecks besserer Vergleichbarkeit sollen alle drei Testnetze das gleiche Radio-Interface als Grundlage verwenden. Aufgrund der guten Unterstützung aller drei Mesh Protokolle als auch dem im vergangenen P5 gesammelten Wissens, sollen hierfür die nRF52840 SoCs der Firma Nordic eingesetzt werden.

Die zu erstellende Testinfrastruktur soll jeweils aus folgenden Teilen bestehen:

1. Zweier **Punkt-Punkt Testinfrastrukturen auf MAC-Ebene** (für BLE und 802.15.4)
 Diese sollen es ermöglichen, möglichst einfach zwischen zwei beliebigen Standorten kanalweise die Übertragungsbedingungen zu ermitteln. Diese Testinfrastruktur kann somit einerseits in der Planung eines BT Mesh, Zigbee oder Thread Mesh-Netzwerks als Messinstrument dienen, andererseits auch als Stör-Infrastruktur um gezielt Störungen zu generieren, wie sie von konkurrenzierenden resp. interferierenden BLE und/oder 802.15.4 Netzwerken hervorzurufen werden.
2. Dreier **Test Mesh Netzwerke** (für BT Mesh, Zigbee und Thread)
 Mit diesen Test-Netzwerken soll die Robustheit und Mesh-Funktionalität der drei Protokoll-Stacks unter realitätsnahen und nachvollziehbaren Bedingungen ermittelt werden, d.h. bei verschiedenen Netzbelaestungen, Netztopologien und gezielt eingebrachter Störungen mit unterschiedlichen Störmustern.
3. Einem Leitrechner, auf welchem eine Python basierte **Steuer- und Auswertesoftware** läuft. Für kürzere Tests und während der Entwicklung kann somit hierfür ein Notebook eingesetzt werden, für Langzeittests hingegen ein Raspberry Pi.
 Die Steuer- und Auswertesoftware soll möglichst modular aufgebaut und ein einfaches Commandline-Interface haben, mit Ausgabe der verarbeiteten Daten auf Standard Output.

Zwecks einfacher Bedienung und grafischer Anzeige soll darauf aufbauend ein einfaches Python basiertes Web-Interface realisiert werden.

**Eckpunkte der
Punkt-Punkt
Testinfrastrukturen
auf MAC-Ebene
BLE und 802.15.4**

Beide Varianten sollen konzeptionell wie folgt realisiert werden:

- Der Leitrechner steuert via USB oder UART ein Master-Funkmodul an.
- Das Master-Funkmodul sendet gemäss den Vorgaben des Leitrechners regelmässig MAC-Frames an einen oder mehrere Slave-Funkknoten.
- Der oder die batteriebetriebenen Slave-Funkknoten, bestätigen die MAC-Frames zurück.

Folgende Parameter sollen von der übergeordneten Steuer- und Auswertesoftware konfigurierbar sein:

- Anzahl und ID der Slaves
- Welche BLE resp. 802.15.4 Kanäle zyklisch getestet werden
- Einstellbare Framelänge
- Einstellbare Framerate und Kanalwechselrate
- Einstellbare Sendeleistung
- Nur für BLE: Modulationsart resp. Datenrate (2Mbps ... 125kbps d.h. auch Long Range)
- Nur für 802.15.4: mit/ohne Collision Avoidance (CSMA/CA)

Sowohl master- wie auch slaveseitige Erfassung der Verbindungsqualität (RSSI, Package Loss, Collisions, Noise Level, ...). Die Slaves senden hierzu die erfassen Werte im Rückantwortframe dem Master zurück.

Einfaches und für beide Protokolle (BT und 802.15.4) taugliches Protokoll zwischen Master-Knoten und Auswerterechner.

**Eckpunkte der drei
Mesh Testnetzwerke
(BT Mesh, Zigbee,
Thread)**

Erstellen der drei Mesh-Testnetzwerke mit jeweils ca. 10 Netzknoten wovon:

- Ein Master-Node mit wahlweise USB- oder UART Anschluss zum Auswerterechner
- die restlichen Knoten konfigurierbar z.B. 4 Routing und 5 Low Power Sensor Knoten

Die Sensor-Knoten sollen in einem vom Auswerterechner vorgegebenen parametriesierbaren Intervall Sensorwerte simulieren.

Als "Sensordaten" sollen die Netz-Zustandsdaten übermittelt werden (z.B. Paketnummer zwecks erkennen von verlorenen Datenpacketen, Anzahl Retries, Paketverluste, RSSI, Strombedarf resp. aktive CPU- und Radio-Zeiten, ...)

Das Protokoll und Interface zum Leitrechner soll wenn möglich für alle drei Mesh Netze einheitlich sein.

Um die Störimmunität der Netze zu ermitteln sollen auch gezielt "Fremdstörungen" im eingebracht werden, mit definierbarer Tastung und Störframelaenge. Hierfür sollen die "Punkt-Punkt Testinfrastrukturen auf MAC-Ebene" eingesetzt werden.

Umfassende Gegenüberstellung und Validierung aller drei Netzwerke in einem FHNW Gebäude, insbesondere Durchsatz, Antwortzeit, Zuverlässigkeit, Einfachheit der Konfiguration (inkl. Routing), Einfachheit der Ermittlung geeigneter Router-Standorte, Sicherheit und Energieverbrauch, etc.

**Teamaufteilung,
Fachberichte und
Bewertung**

- Jedes Teammitglied realisiert eines der drei Test Mesh Netze.
- Bezuglich Aufteilung der restlichen Arbeiten einigen sich die Teammitglieder untereinander.
- Die Thesisnote setzt sich jeweils hälftig aus einer Individualnote und einer Teamnote zusammen.
- Die Fachberichte resp. Dokumentation setzt sich zusammen aus:
 - Dokumentation der **individuell erstellten Teile**
 - Dokumentation der **gemeinsam erstellten Teile**
 - Einem **Paper** mit Dokumentation und Interpretation der durchgeführten Tests
- Alles Sourcen sollen als Open Source auf GitHub oder GitLab veröffentlicht werden.



Pflichtenheft und Projektvereinbarung

Nach Projektstart soll innert ca. 1 Monat ein technisches Pflichtenheft erarbeitet werden beinhaltend:

- Lösungskonzept und Spezifikation des zu erstellenden Systems,
- Formulierung von Arbeitspaketen (typisch 5-15) und Meilensteinen,
- Zeitplan in Form eines Gantt-Diagrammes.

Projektmanagement, Kommunikation, Abgabetermine, Bewertung:

Das Projekt soll von einem schlanken, ergebnisorientierten Projektmanagement begleitet werden.

Arbeitgeber und betreuender Dozent sollen periodisch (mind. alle 3 Wochen) über den Stand der Arbeiten sowie allfälliger Abweichungen zum Pflichtenheft und Projektplan informiert werden.

Es finden mindestens folgende Meetings statt:

- Kickoffmeeting
- Besprechung Pflichtenheft/Projektvereinbarung
- Schlusspräsentation

Bei Bedarf können mehr Meetings durchgeführt werden.

Betreffend Fachbericht, Ausstellung der Thesisarbeit (inkl. Erstellen eines Factsheets und Posters) sowie Verteidigung und Bewertung gelten die Vorgaben und Richtlinien der FHNW, Hochschule für Technik.

Termine

Es gelten die offiziellen Termine der FHNW, Hochschule für Technik

- Die Thesisarbeit startet KW 8/2020 und umfasst 360 Arbeitsstunden für jeden Probanden (inkl. Erstellung Fachbericht).
- Der Abgabetermin der Arbeit (Fachbericht) ist am Tag der Bachelor-Ausstellung 14.8.2020.
- Die Verteidigung (d.h. Präsentation und Diskussion der Thesis vor den Betreuern und externem Experten) findet gemäss Semesterplanung der FHNW voraussichtlich in der KW36 oder KW37 statt. Der genaue Termin wird noch bekannt gegeben.

B Pflichtenheft

Fachhochschule Nordwestschweiz
Hochschule für Technik

Pflichtenheft

Testumgebung und Performancevergleich von Zigbee,
Thread und Bluetooth Mesh Netzwerken

BACHELOR THESIS - ANKLIN, BOBST, HORATH
15. August 2020

Fachcoach:

Matthias Meier
Manuel Di Cerbo

Team:

Raffael Anklin
Robin Bobst
Cyrill Horath

Studiengang:

Elektro- und Informationstechnik

Semester:

Frühlingssemester 2020

Inhaltsverzeichnis

1 Übersicht	1
1.1 Ausgangslage	1
1.2 Ziel der Arbeit	1
2 Lösungskonzept	2
2.1 Punkt zu Punkt Testinfrastruktur	3
2.2 Test Mesh Netzwerke	3
2.2.1 Bluetooth Mesh	4
2.2.2 Thread	4
2.2.3 Zigbee	4
2.3 Steuer und Auswertesoftware	5
3 Projektziele und Lieferobjekte	6
3.1 Punkt zu Punkt Testinfrastruktur	6
3.2 Test Mesh Netzwerke	7
3.3 Steuer- und Auswertesoftware	8
3.4 Zusatzziele	8
3.5 Lieferobjekte	8
4 Projektmanagement	9
4.1 Projektaufteilung	9
4.2 Projektplan	9
4.3 Risikoanalyse	9
4.4 Projektvereinbarung	10
A Test Kriterien Point to Point auf MAC Ebene	11
B Test Kriterien Mesh Netzwerke	12
C Gesamt Terminplanung	13
D Bluetooth Mesh Terminplanung	14
E Thread Terminplanung	15
F Zigbee Terminplanung	16
G Risikoanalyse	17

1 Übersicht

Das vorliegende Dokument stellt das Pflichtenheft der Bachelorthesen von Raffael Anklin, Robin Bobst und Cyrill Horath an der Fachhochschule Nordwestschweiz Brugg-Windisch im Studiengang Elektro- und Informationstechnik dar. Im kommenden ersten Kapitel soll eine Übersicht über die Ausgangslage sowie das Ziel dieser Arbeit gegeben und somit die Rahmenbedingungen abgesteckt werden. Weiter werden die Lösungskonzepte 2 sowie die Projektziele und Lieferobjekte 3 definiert. Abschliessend soll auch noch das Projektmanagement 4 thematisiert werden.

1.1 Ausgangslage

Unter den standardisierten Low Power Mesh Netzwerk Protokollen im freien GHz ISM-Band konkurrieren sich derzeit vorrangig Bluetooth Mesh, Zigbee sowie Thread. Beziiglich MAC und Physical Layer basieren Zigbee und Thread auf IEEE 802.15.4 wogegen Bluetooth Mesh auf Bluetooth Low Energy (BLE) basiert. Jedes dieser Netzwerkprotokolle hat gewisse Vorzüge: Bluetooth Mesh, dass BLE mittlerweile von jedem Smartphone und Notebook unterstützt wird, Thread aufgrund seiner IPv6 Basis und damit einfacher Übergang ins Internet sowie Zigbee aufgrund seiner etablierten Verbreitung im Smart-Lampenbereich durch Philips, IKEA und Osram. Hauptproblem aller drei Mesh Netzwerkprotokolle ist nebst physikalisch und distanzbedingter Absorption und Reflexion die Störbeeinflussung durch WLAN (WiFi) und andere Netzwerke im GHz Frequenzbereich.

Im Rahmen des P5 mit dem Namen *Bluetooth-Mesh Plattform für IoT Anwendungen*, wurde das Bluetooth-Mesh Protokoll bereits vertieft betrachtet und dessen Vor- und Nachteile aufgezeigt. Basierend auf diesen Erkenntnissen und Erfahrungen und der oben beschriebenen Thematik soll das Bluetooth-Mesh Protokoll mit den Alternativen Thread sowie Zigbee verglichen werden.

1.2 Ziel der Arbeit

In der vorliegenden Arbeit soll zuerst ein praxistaugliches, einheitliches Testframework für alle drei Mesh Netzwerke erstellt werden, wonach die Tauglichkeit aller drei Mesh Netzwerke unter realitätsnahen Bedingungen ermittelt und verglichen werden soll. Zwecks besserer Vergleichbarkeit sollen alle drei Testnetze das gleiche Radio-Interface als Grundlage verwenden. Aufgrund der guten Unterstützung aller drei Mesh Protokolle als auch dem im vergangenen P5 gesammelten Wissens, sollen hierfür die nRF52840 SoCs der Firma Nordic eingesetzt werden. Die zu erstellende Testinfrastruktur soll aus den drei folgenden Teilen bestehen:

- Punkt-Punkt Testinfrastrukturen auf MAC-Ebene
- Test Mesh Netzwerke für BT Mesh, Zigbee und Thread
- Steuer- und Auswertesoftware

Die genauen Anforderungen an die Testumgebung sind einerseits in der Aufgabenstellung im Anhang ?? aufgeführt und andererseits werden sie anhand der Projektziele 3 definiert.

2 Lösungskonzept

Zur Messung und Auswertung der Mesh-Netzwerke dient ein Testframework wie es in Abbildung 2.1 schematisch dargestellt ist.

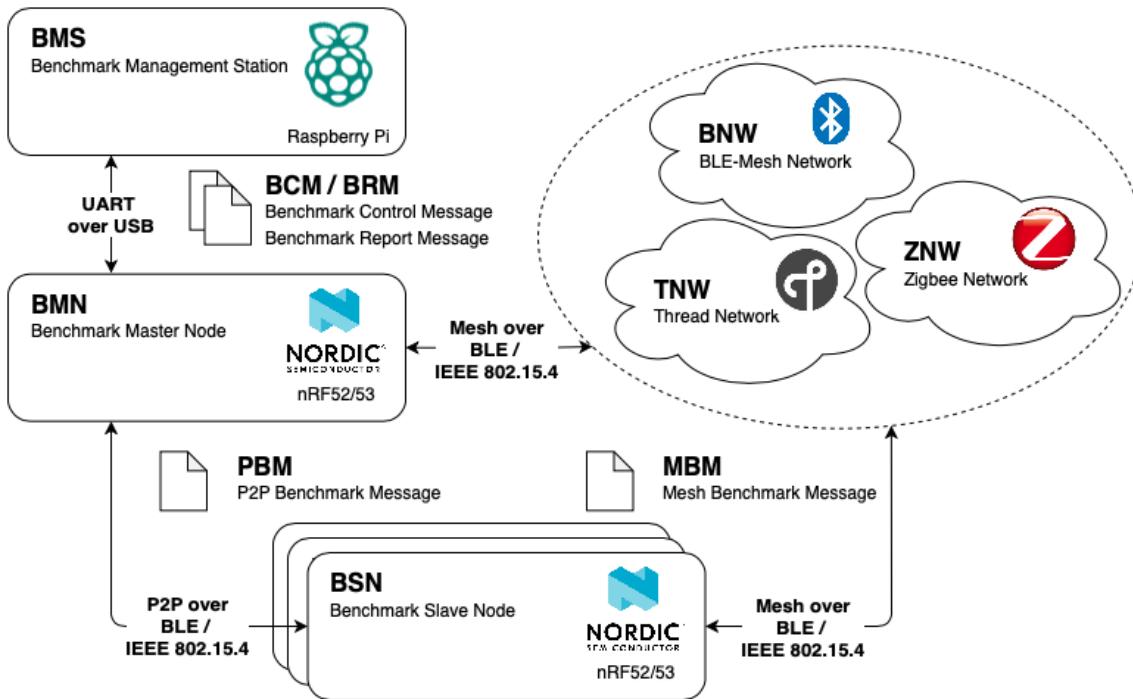


Abbildung 2.1: Konzeptschema Testframework

Das Testframework besteht aus folgenden physikalisch getrennten Teilsystemen:

- **BMS** Benchmark Management Station
Dient zur Verwaltung und Konfiguration des Testframeworks. Beinhaltet einen Webserver um dem Endanwender die Bedienung zu ermöglichen. Realisiert wird die BMS durch einen *Raspberry Pi 4 Model B*. Als Webserver wird das Python-Framework *Django* eingesetzt.
- **BMN** Benchmark Master Node
Dient als Zugangspunkt der BMS für die im Testframework gefahrenen Tests und lässt sich über eine serielle Schnittstelle ansprechen. In der Aufgabenstellung ?? wird der BMN als Master bezeichnet. Realisiert wird der BMN über einen nRF52840 von *Nordic*.
- **BSN** Benchmark Slave Node
Dient als Zugriffspunkt der im Testframework gefahrenen Tests und kann frei in der Testumgebung platziert werden, daher muss die Energieversorgung über einen Akku oder Batterie erfolgen. In der Aufgabenstellung ?? wird von einem Slave gesprochen. Realisiert wird der BSN über einen nRF52840 oder nRF5340 von *Nordic*. In einem Test Netzwerk werden unterschiedliche Arten von BSN eingesetzt wie zum Beispiel Mesh Router oder sogenannte End Devices. Letztere werden häufig auch als Low Power Nodes (LPN) bezeichnet.

Die logischen Komponenten des Testframeworks lassen sich wie folgt aufteilen:

- **BCM** Benchmark Control Message
Beschreibt Nachrichten welche zur Steuerung eines Benchmarks dienen. Dies sind zum Bei-

2.1 Punkt zu Punkt Testinfrastruktur

3

spiel Konfigurations-, Start- oder Stop-Befehle. Werden von der BMS initiiert und gelangen über eine USB-UART Verbindung zum BMN.

- **BRM** Benchmark Report Message

Beschreibt Nachrichten welche den Status oder die Ergebnisse eines Benchmarks zurückmelden. Werden vom BMN initiiert und gelangen über eine USB-UART Verbindung zur BMS.

- **PBM** P2P Benchmark Message

Nachrichten welche während der Durchführung eines Punkt zu Punkt Benchmarks versendet werden. Dies sind zum Beispiel Ping-Anfragen zur Latenzzeitmessung. Sie ermöglichen den Datenaustausch zwischen zwei Teilnehmern auf MAC-Ebene.

- **MBM** Mesh Benchmark Message

Nachrichten welche während der Durchführung eines Mesh Benchmarks versendet werden. Dies sind zum Beispiel Ping-Anfragen zur Latenzzeitmessung. Sie ermöglichen den Daten austausch über ein Mesh-Netzwerk auf Applikations-Ebene.

2.1 Punkt zu Punkt Testinfrastruktur

Die Punkt zu Punkt Testinfrastruktur (P2P) ermöglicht es ein Benchmark auf physikalischer Ebene durchzuführen. Dies soll unabhängig vom Mesh-Protokoll und basierend auf den beiden MAC-Ebenen *BLE* und *IEEE802.15.4* möglich sein. Somit ist ein Vergleich der beiden MAC-Ebenen machbar. Weiter soll diese Infrastruktur einem Endanwender die Möglichkeit bieten die Sende- und Empfangsbedingungen an gegeben Örtlichkeiten auf physikalischer Ebene auszumessen um geeignete Standorte für die Mesh Router zu bestimmen. Die Datenerfassung und grafische Aufbereitung erfolgt dabei auf der BMS. Eine Erfassung der Momentanwerte soll ebenso möglich sein wie die Aufzeichnung der Bedingungen in Form eines Langzeittests über mindestens 24 Stunden. Schliesslich soll also ein Messinstrument zur Bestimmung der Signalqualitäten im Feld entstehen. Im Anhang A werden die Testkriterien für die P2P Infrastruktur aufgeführt und beschrieben. Diese Tests werden mit Hilfe bereits bestehenden Beispiel Firmware (Radio-Example) aus der nRF Connect SDK auf dem nRF52840 durchgeführt. Optional wird dies auch noch auf den nRF5340 portiert welcher leistungsfähiger ist. Weiter soll diese P2P Testinfrastruktur dazu genutzt werden können, gezielte Störungen auf die Test Mesh Netzwerke zu richten die nachfolgend unter 2.2 beschrieben werden.

2.2 Test Mesh Netzwerke

Der Mesh-Benchmark soll die verschiedenen Mesh-Netzwerke möglichst identisch ausmessen um damit deren Protokoll Stacks untereinander zu vergleichen. Dazu dient bei allen Mesh-Netzwerken die Applikations-Schicht. Ein Mesh Netzwerk wird zwischen dem BMN und den BSN aufgebaut. Dazu werden die einzelnen Nodes über die BMS mit der entsprechenden Firmware geladen und anschliessend im Raum verteilt. Das Laden ist über eine Kabelverbindung (UART) vorgesehen. Allenfalls könnte dies zu einem späteren Zeitpunkt drahtlos mithilfe eines Bootloaders möglich gemacht werden. Dabei handelt es sich jedoch um ein Zusatzziel (siehe 3.4). Die Test Mesh Netzwerke sollen anders als die P2P Testinfrastruktur jedoch nur zu Testzwecken innerhalb dieser Arbeit eingerichtet werden und nicht als Messinstrument für Feldmessungen dienen. Im Anhang B sind die Testkriterien für den Mesh-Benchmark aufgeführt. Anhand dieser sollen die Messungen durchgeführt und analysiert werden. Zusätzlich sollen die Messungen auch unter verschiedenen Bedingungen durchgeführt werden. Beispielsweise ist zu erwarten, dass die Messresultate aufgrund von sich verändernder Störbelastung durch andere Geräte in unmittelbarer Nähe unterschiedlich ausfallen, abhängig von Örtlichkeit und Anordnung der Nodes. Alle drei Mesh Netzwerke verfügen über unterschiedliche Nodetypen mit entsprechend differenzierten

2.2 Test Mesh Netzwerke

4

Funktionen wie beispielweise Router oder End Devices. Die Mesh Netzwerke sollen möglichst realitätsnahe aufgebaut werden und somit mindestens diese beiden Typen beinhalten. Realitätsnah bedeutet in diesem Kontext dass eine mögliche Anwendung, als Beispiel in der Heimautomation, nachgebaut wird in welcher die Nodetypen und auch die Kommunikationswege klar definiert sind.

2.2.1 Bluetooth Mesh

Die BLE-Mesh Firmware der Nodes werden aus Beispielen der nRF Connect SDK und Zephyr abgeleitet. Das Mesh-Demo Beispiel erlaubt es die essentiellen Netzwerk Parameter fix vorzugeben. Dadurch müssen die Nodes nicht mehr Provisioniert werden und sind sofort einsatzbereit.

2.2.2 Thread

Die OpenThread Firmware wird mit Hilfe der API und den Tutorials von der offiziellen OpenThread Webseite erstellt. Die offizielle Seite von Google beschreibt das Netzwerk und alle Informationen die benötigt werden, um eine Firmware zu schreiben.

Die Abbildung 2.2 zeigt das Framework des Openthread Netzwerkes auf. Die Kommunikation von der BMS zum BMN findet Seriell mit UART over USB statt. Der Serielle Kanal wird mit Hilfe eines Python-Skripts ausgeführt.

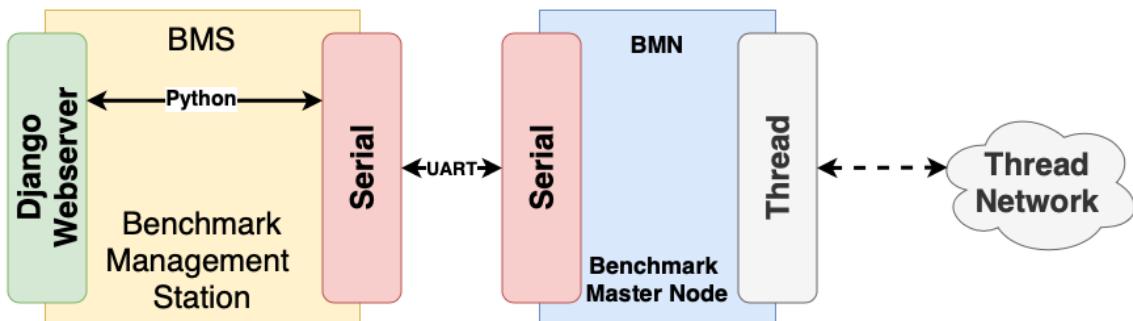


Abbildung 2.2: Konzept OpenThread Framework

2.2.3 Zigbee

Das Zigbee Test Mesh Netzwerk wird mithilfe der *nRF SDK for Thread and Zigbee* eingerichtet und für die Messungen vorbereitet. Der BMN wird dabei als Zigbee Coordinator und gleichzeitig als Zigbee Router eingesetzt. Die BSN können wieder als Router oder aber als Zigbee End Device betrieben werden.

2.3 Steuer und Auswertesoftware

Die Steuerung des Testframeworks erfolgt über eine Weboberfläche. Diese wird von der BMS mittels WLAN auf den Benutzergeräten zur Anzeige gebracht. Als Webserver dient das Python-Framework *Django*. Zur Steuerung der Benchmarks dienen Schleifenketten, welche mit der Firmware auf dem BMN kommunizieren. Als letzter Schritt eines Benchmarks werden die Ergebnisse geloggt, nachbearbeitet und wiederum zur Anzeige gebracht.

Die Abbildung 2.3 stellt ein erstes Konzept dar, wie der Webserver für das P2P Test Framework aussehen kann. Die vom BMN ersichtlichen BSN werden aufgelistet und es können verschiedene Aktionen durchgeführt werden. Mit Reitern soll auf verschiedene Seiten gewechselt werden auf welchen wiederum Resultate oder Logs der Kommunikation ersichtlich sein sollen.

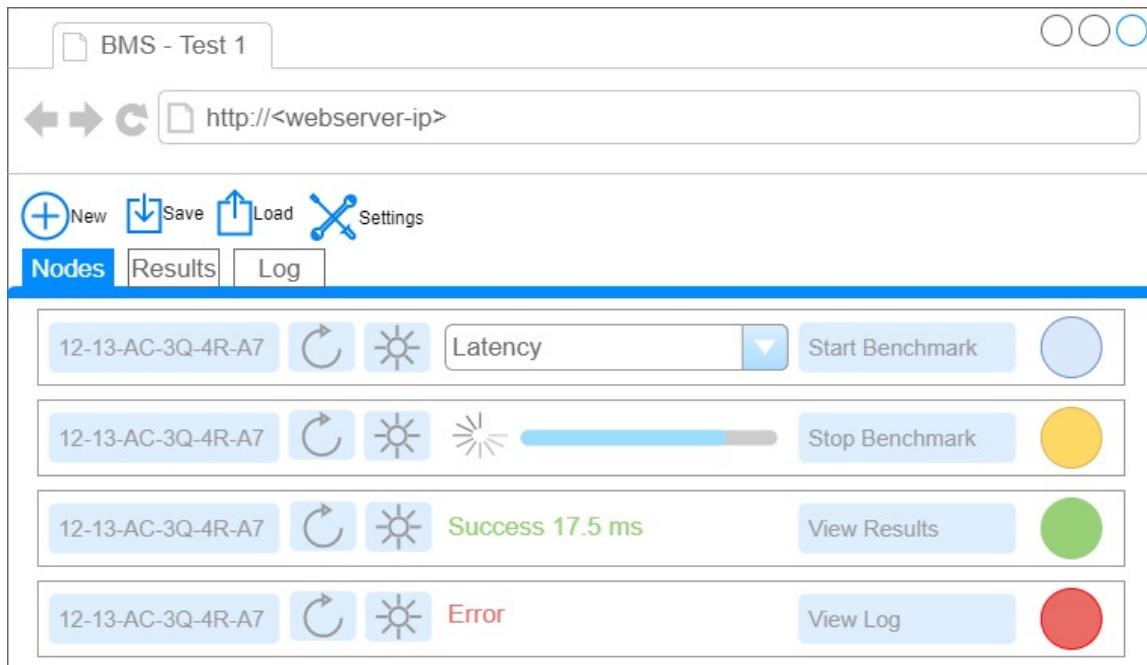


Abbildung 2.3: Entwurf Webserver

3 Projektziele und Lieferobjekte

Nachfolgend sind alle Projektziele aufgelistet. Die Ziele wurden in vier Teile unterteilt. Die ersten drei Teile entsprechen den Hauptzielen die in Kapitel 1.2 bereits erwähnt wurden. Die Tabelle 3.4 zeigt zusätzliche Ziele die als Wunschziele betrachtet werden und somit nicht im Fokus der Arbeit stehen. Wie bereits unter Kapitel 2.1 sowie 2.2 erwähnt, sind ergänzende Beschreibungen von Testkriterien in den Anhängen A und B zu finden.

3.1 Punkt zu Punkt Testinfrastruktur

Projektziele		
Nr.	Ziel	Beschrieb
P1	Kommunikation mit BMS	Der BMN kommuniziert via USB oder UART mit dem BMS.
P2	Senden von MAC-Frames	Das BMN sendet gemäss den Vorgaben der Testkriterien und gesteuert durch das BMS MAC-Frames an einen oder mehrere BSN.
P3	Rückbestätigung der MAC-Frames	Der oder die batteriebetriebenen BSN, bestätigen die MAC-Frames mit entsprechender Payload zurück.
P4	Konfiguration der Anzahl BSN	Die Anzahl der BSN ist über eine Steuer- und Auswertesoftware konfigurierbar.
P5	Adressierung der BSN	Die Adressierung der BSN ist über eine Steuer- und Auswertesoftware konfigurierbar.
P6	Konfiguration der Kanäle	Die BLE resp. 802.15.4 Kanäle können über eine Steuer- und Auswertesoftware ausgewählt werden.
P7	Einstellbare Frame-länge	Die Framelänge der MAC-Frames ist über eine Steuer- und Auswertesoftware konfigurierbar.
P8	Einstellbare Frame- und Kanalwechselrate	Die Frame- und Kanalwechselrate ist über eine Steuer- und Auswertesoftware konfigurierbar.
P9	Einstellbare Sende-leistung	Die Sendeleistung der BSN ist über eine Steuer- und Auswertesoftware konfigurierbar.
P10	Anpassung der Modulationsart	In BLE soll die Modulationsart über eine Steuer- und Auswertesoftware konfigurierbar sein, um die Datenrate von 125kbps auf 2Mbps und die Long Range Funktion einzustellen.
P11	Ein- und Ausschalten der Collision Avoidance (CSMA/CA)	Beim 802.15.4 Protokoll soll die Collision Avoidance über eine Steuer- und Auswertesoftware ein- und ausgeschaltet werden können.
P12	Erfassen der Verbindungsqualität	Sowohl master- wie auch slaveseitige Erfassung der Verbindungsqualität (RSSI, Package Loss, Collisions, Noise Level, ...). Die BSN senden hierzu die erfassten Werte im Rückantwortframe dem BMN zurück. (Siehe auch Anhang A)
P13	Tool für Feldtests	Es soll ein Tool entstehen, welches dem Anwender die Möglichkeit gibt Messungen durchzuführen und somit sein Mesh Netzwerk zu planen.

Tabelle 3.1: Projektziele der Punkt zu Punkt Testinfrastruktur

3.2 Test Mesh Netzwerke

Projektziele		
Nr.	Ziel	Beschrieb
P1	Kommunikation mit BMS	Das BMN kommuniziert via USB oder UART mit dem BMS.
P2	Konfiguration BSN	Die BSN lassen sich frei zu einem Routing-Knoten, End-Knoten oder einem Low-Power Knoten konfigurieren.
P3	Mesh-Netzwerk	Alle drei Technologien Bluetooth, Thread und Zigbee müssen als Mesh-Netzwerk mit mindestens 10 BSN aufgebaut werden.
P4	Simulation Sensorwerte	Die BSN sollen in einem vom BMS vorgegebenen parametrisierbaren Intervall Sensorwerte simulieren
P5	Sensordaten	Als Sensordaten sollen die Netzzustandsdaten übermittelt werden: Paketnummer, Anzahl Retries, Paketverluste, RSSI, Strombedarf und aktive CPU- und Radio-Zeiten.
P6	Datenauswertung	Die Auswertung der gemessenen Daten soll entweder direkt auf dem BMS oder alternativ auf einem Client Rechner erfolgen. Eine Gegenüberstellung der Daten der drei Mesh Protokolle ist dabei ebenfalls gewünscht.
P7	Störrimmunität	Um die Störrimmunität der Netzwerke zu ermitteln sollen gezielt Fremdstörungen mit definierbarer Tastung und Störfraumlänge eingebracht werden. Hierfür soll die Punkt zu Punkt Testinfrastruktur auf MAC-Ebene eingesetzt werden.
P8	Unterschiedliche Test Bedingungen	Die Messungen und Tests an den Mesh Netzwerken sollen unter unterschiedlichen Bedingungen bezüglich Testumgebung durchgeführt werden. Einerseits soll dies in einem Gebäude der FHNW sein und andererseits in einer Umgebung im Heimbereich.
P9	Test und Validierung	Umfassende Gegenüberstellung und Validierung der Messresultate aller drei Netzwerke. Insbesondere Durchsatz, Antwortzeit, Zuverlässigkeit, Einfachheit der Konfiguration (inkl. Routing), Einfachheit der Ermittlung geeigneter Router-Standorte, Sicherheit und Energieverbrauch.

Tabelle 3.2: Projektziele der Test Mesh Netzwerke

3.3 Steuer- und Auswertesoftware

8

3.3 Steuer- und Auswertesoftware

Projektziele		
Nr.	Ziel	Beschrieb
P1	Ansteuerung Funkmodul	Das BMS steuert via USB oder UART ein BMN an.
P2	Visualisierung Parameter	Die Parameter der Ziele von Kapitel 3.2 und 3.1 sollen vom BMS visualisiert und eingestellt werden können.
P3	User Interface (UI)	Die Testinfrastruktur beinhaltet ein benutzerfreundliches UI.
P4	Konfiguration Mesh-Netzwerk	Das BMS verwaltet und konfiguriert über einen BMN das Mesh-Netzwerk.
P5	Einheitliche Kommunikation von BMS	Das Protokoll und Interface zum BMS soll für alle drei Mesh-Netzwerke einheitlich sein.

Tabelle 3.3: Projektziele der Steuer- und Auswertesoftware

3.4 Zusatzziele

Projektziele		
Nr.	Ziel	Beschrieb
W1	Hardware Testmodul BMN/BSN	Entwickelung einer eigenen Hardware für das Testmodul mit unabhängiger Stromversorgung und deziiderter Strommessung um den Stromverbrauch aufzuzeichnen.
W2	Vergleich SOC	Vergleichen zwischen nRF52840, nRF5340 und weiterer kompatibler SOCs.
W3	Drahtlos Konfiguration	Drahtlose Konfiguration der BSN / BMN Firmware. Somit könnte ein Wechsel zwischen BLE Mesh, Thread und Zigbee während der Runtime möglich werden.
W4	UI für Mesh Test	Für die Mesh Tests soll analog zu den P2P Tests ein User Interface implementiert werden.

Tabelle 3.4: Zusatzziele des Gesamtprojektes

3.5 Lieferobjekte

Zusätzlich zu den Projektzielen, folgen in diesem Kapitel die Lieferobjekte mit dem jeweiligen Fälligkeitsdatum. In der Tabelle 3.5 sind diese aufgelistet.

Nr.	Datum	Lieferobjekt
1	02.03.2020	Abgabe Pflichtenheft, 1. Version
2	08.03.2020	Abgabe Pflichtenheft, definitive Version
3	14.08.2020	Abgabe Fachbericht
4	14.08.2020	Abgabe Paper
5	14.08.2020	Abgabe Testaufbau
6	14.08.2020	Abgabe Factsheet
7	14.08.2020	Abgabe Poster
8	01.09.2020	Projektpräsentation

Tabelle 3.5: Lieferobjekte

4 Projektmanagement

Nachfolgend werden die wichtigsten Punkte bezüglich Projektmanagement behandelt. Im Zentrum stehen dabei vor allem die Aufteilung der Zuständigkeiten 4.1 sowie die Projektplanung 4.2.

Um den Projektfortschritt zu überwachen und allfällige Probleme zu besprechen sind zweiwöchentliche Sitzungen mit den Fachcoaches vorgesehen. Ausserordentliche Termine für Besprechungen werden bei Bedarf zusätzlich definiert.

4.1 Projektaufteilung

Wie bereits im Kapitel 1 erwähnt, stellt die vorliegende Projektarbeit die Bachelorthesen von Raffael Anklin, Robin Bobst und Cyrill Horath dar. Die einzelnen Thesen werden grundsätzlich als separate Projekte umgesetzt welche jedoch einen gemeinsamen Teil beinhalten. Innerhalb dieses gemeinsamen Teils welcher als Framework bezeichnet wird, werden die Arbeitspakete dynamisch an die 3 Mitarbeiter verteilt. In der Tabelle 4.1 ist die genaue Zuweisung der Zuständigkeiten ersichtlich.

Bezeichnung	Inhalt	Zuständigkeit	Kennung
Bluetooth Mesh	Aufbau, Messung und Analyse eines Bluetooth Mesh Netzwerks.	Raffael Anklin	EIT-P-20FS-030
Thread	Aufbau, Messung und Analyse eines Thread Mesh Netzwerks.	Robin Bobst	EIT-P-20FS-031
Zigbee	Aufbau, Messung und Analyse eines Zigbee Mesh Netzwerks.	Cyrill Horath	EIT-P-20FS-032
Framework	Bereitstellung der Test- und Messinfrastruktur. Dazu gehört der Punkt zu Punkt Testaufbau sowie die Steuer- und Auswertesoftware.	Alle	-

Tabelle 4.1: Zuweisung der Zuständigkeiten

4.2 Projektplan

Die Terminpläne zum Projekt *Wireless Controller for Smart Systems* sind im Anhang zu finden. Sie sind in die 4 Teile unterteilt die in der Projektaufteilung 4.1 behandelt wurden. Unter Anhang C ist der Gesamt Terminplan ersichtlich, welcher unter anderem das Framework sowie Projektmanagement Aufgaben beinhaltet. Die Anhänge D, E und F sind die Terminpläne für die 3 persönlichen Bachelorthesen.

4.3 Risikoanalyse

Für die Projektarbeit wurde ausserdem eine schlanke Risikoanalyse erstellt. Diese ist in Anhang G ersichtlich und soll dem Team bei der Beseitigung von Problemen hilfreich sein.

4.4 Projektvereinbarung	10
-------------------------	----

4.4 Projektvereinbarung

Projektcoach und Auftraggeber

Di Cerbo Manuel

Ort, Datum:

Unterschrift:

Meier Matthias

Ort, Datum:

Unterschrift:

Projekt: EIT-P-20FS-030

Anklin Raffael

Ort, Datum:

Unterschrift:

Projekt: EIT-P-20FS-031

Bobst Robin

Ort, Datum:

Unterschrift:

Projekt: EIT-P-20FS-032

Horath Cyrill

Ort, Datum:

Unterschrift:

A Test Kriterien Point to Point auf MAC Ebene

Index Messung	MAC-1	MAC-2	MAC-3	MAC-4	MAC-5	MAC-6	MAC-7	MAC-8	MAC-9
Bezeichnung									
	Latency Time	Response Time	Data Transmission Rate	Noise Level Detection	RSSI	Packet-loss	Active radio-time	Active CPU-time	Theoretical power consumption
	Bestimmung der Latenzzeit eines Node	Bestimmung der Antwortzeit eines Node	Bestimmen der Datenübertragungsrate	Bestimmung des Noise Level in dbm in Verschiedenen Kanälen	Bestimmung des Noise Level in dbm	Bestimmung des Received Signal Strength Indicator	Bestimmung der Anzahl verlorenen Pakete	Bestimmung der Aktiven Radio Zeiten	Bestimmung der theoretischen Leistungsaufnahme.
Beschreibung									
Messgröße	Latenzzeit	Antwortzeit	Datenübertragungsrate	Störsignalleistung	Empfangsleistung	Paketverlust	Zeit	Zeit	Leistung
	Millisekunden (ms)	Millisekunden (ms)	kBit/s	dBm	dBm	Verhältnis gesendete Pakete zu verlorene Pakete in %	s	s	mW
Einheit									
	Zu Beginn findet eine Zeit sowie Kanal-Synchronisation zwischen Master und den Slaves statt. Anschließend sendet der Master einzeln Testpakete mit der Sendezeit T1 an die Nodes. Jeder Node vergleicht die Empfangszeit T2 mit der Sendezeit T1 und schickt die Differenz als Latenzzeit dem Master zurück.	Zu Beginn findet eine Zeit sowie Kanal-Synchronisation zwischen Master und den Slaves statt. Anschließend sendet der Master einzeln Testpakete mit der Sendezeit T1 an die Nodes. Jeder Node sendet eine Empfangsbestätigung an den Master zurück. Dieser vergleicht die Empfangszeit T2 mit der Sendezeit T1 und generiert daraus die Antwortzeit.	Zu Beginn findet eine Zeit sowie Kanal-Synchronisation zwischen Master und den Slaves statt. Anschließend sendet der Master einzeln Testpakete mit der Sendezeit T1 und Zufallsdaten in verschiedenen Längen an die Nodes. Jeder Node speichert die Empfangszeit T2 nach vollständig erhaltenem Paket ab und bildet mit der Sendezeit T1 die Differenz. Aus dem Quotienten der Datenumenge und Latenzzeit wird die Datenübertragungsrate gebildet. Diese meldet der Node dem Master zurück.	Zu Beginn messen das Noise Level auf allen Kanälen auf Anfrage des Masters und senden dieses an den Master zurück. Dabei ist jeweils nur ein Node aktiv womit nur Störsignale erfasst werden.	Der RSSI Wert wird von den verschiedenen Nodes erfasst und als Payload den Nachrichten mitgegeben und dem Master zugeschickt.	Die Paketnummer vom empfangenen Signal wird ausgelernt und mit der tatsächlichen Paketnummer, die in der Payload mit geliefert wird verglichen. Das Verhältnis zwischen den Werten stellt den Paketverlust dar.	Beim Einschalten und Ausschalten der Rx / Tx-Schnittstelle wird ein Timer gestartet bzw. gestoppt, so wird die aktive Radio Zeit ermittelt.	Beim Ein- und Ausschalten der CPU soll ein timer gestartet bzw. gestoppt werden, so wird die aktive CPU Zeit gemessen.	Anhand der gemessenen Radio und CPU Zeiten wird die Leistung berechnet.
Vorgehen									
Störfaktoren	Umliegende Kommunikationsgeräte, welche das 2.45GHz ISM Band benutzen. Gezielte Störung des Testsetups sind nicht vorgesehen.								
Anzahl Wiederholungen	1-10	1-10	1-10	1-10	1-10	1-10	1-10	1-10	1-10
Einstellbare Parameter	Modulationsarten (Ble1Mbps, Ble2Mbps, BleLR, IEEE802.15.4), Kanäle, Ziel Node(s), Anzahl Wiederholungen	Modulationsarten (Ble1Mbps, Ble2Mbps, BleLR, IEEE802.15.4), Kanäle, Ziel Node(s), Anzahl Wiederholungen	Modulationsarten (Ble1Mbps, Ble2Mbps, BleLR, IEEE802.15.4), Kanäle, Ziel Node(s), Anzahl Wiederholungen	Kanäle, Ziel Node(s), Anzahl Wiederholungen	Kanäle, Ziel Node(s), Anzahl Wiederholungen	Modulationsarten (Ble1Mbps, Ble2Mbps, BleLR, IEEE802.15.4), Kanäle, Ziel Node(s), Anzahl Wiederholungen	Ziel Node(s), Anzahl Wiederholungen	Ziel Node(s), Anzahl Wiederholungen	Ziel Node(s), Anzahl Wiederholungen

B Test Kriterien Mesh Netzwerke

Index Messung	Mesh-1	Mesh-2	Mesh-3	Mesh-4	Mesh-5	Mesh-6	Mesh-7	Mesh-8	Mesh-9	Mesh-10	
Bezeichnung	Latency Time	Number of hops	Data Transmission Rate Unacknowledged	Data Transmission Rate Acknowledged	RSSI	Packet-loss	Active radio-time	Active CPU-time	Theoretical power consumption	Number of retries	
Beschreibung	Bestimmung der Latenzzeit von Aktor zu Sensor über Anzahl Hops.	Bestimmung der Anzahl Hops, die eine Nachricht nehmen musste.	Bestimmen der Datenübertragungsrate (Unbestätigt)	Bestimmen der Datenübertragungsrate (Bestätigt)	Bestimmung des RSSI von verschiedenen Nodes	Bestimmung der Anzahl verlorenen Pakete	Bestimmung der Aktiven Radio Zeiten	Bestimmung der Aktiven CPU Zei	Bestimmung der theoretischen Leistungsaufnahme.	Anzahl Retries	
Messgröße	Latenzzeit	$n = \text{Anzahl Hops}$	Datenübertragungsrate	Datenübertragungsrate	Empfangsleistung	Paketverlust	Zeit	Zeit	Leistung	$n = \text{Anzahl Retries}$	
Einheit	Millisekunden (ms)	-	bit/s	kbit/s	dBm	Verhältnis gesendete Pakete zu verlorene Pakete in %	Millisekunden (ms)	Sekunden (s)	Milliwatt (mW)	-	
Vorgehen	Die Latenzzeit wird immer von einem Sensor zu einem Aktor gemessen, z.B. von einem Lichtschalter zum Lampen. Wenn die Nachricht vom Sensor zum Sensor geht, wird ein Timestamp als Payload der Nachricht hinzugefügt. Beim Aktor werden weitere Timestamps zum Payload hinzugefügt und dem Sensor als Acknowledge zurückgeschickt. Im Sensor wird danach die Latenzzeit anhand der Timestamps berechnet.	Auf einem Node werden die Next Hop Informationen lokal gespeichert. Diese Information wird der Nachricht als Payload hinzugefügt und am Ziel Node auszuwerten wie viele Hops die Nachricht genommen hat.	Es werden Datenpakete verschiedener Länge [1byte - ca. 1MBYTE] zufällig generiert. Diese Ablauf ist mit T3 identisch, außer dass der Erhalt von jedem Datenpaket (ebenfalls gesendet) bestätigt wird. Die Zeitmessung ist mit der letzten Bestätigung an den Sensor abgeschlossen.	Der Ablauf ist mit T3 identisch, außer dass der Erhalt von jedem Datenpaket (ebenfalls gesendet) bestätigt wird. Die Zeitmessung ist mit der letzten Bestätigung an den Sensor abgeschlossen.	Der RSSI Wert wird von den verschiedenen Nodes erfasst und als Payload den Nachrichten mitgegeben und dem Master zugeschickt.	Die Paketnummer vom empfangenen Signal wird ausgelesen und mit der tatsächlichen Paketnummer, die in der Payload mit geliefert wird verglichen. Das Verhältnis zwischen den Werten stellt den Paketverlust dar.	Beim Einschalten und Ausschalten der Rx- / Tx-Schnittstelle wird ein Timer gestartet bzw. gestoppt, so wird die aktive Radio Zeit ermittelt.	Beim Ein- und Ausschalten der CPU soll ein Timer gestartet bzw. gestoppt werden, so wird die aktive Radio Zeit gemessen.	Anhand der gemessenen Radio und CPU Zeiten wird die Leistung berechnet.	Wird das Acknowledge nicht quittiert, wird die Nachricht erneut gesendet. Diese Anzahl Retries werden ermittelt und der Payload mitgegeben.	
Störfaktoren	Umliegende Kommunikationsgeräte, welche das 2.45GHz ISM Band benutzen. Periodisch										
Anzahl Wiederholungen	-	Anzahl Hops kann begrenzt werden	Packetsize	Packetsize	-	-	-	-	-	-	
Einstellbare Parameter	-										
Voraussetzungen	Node muss bereit und konfiguriert sein. Zeit der Nodes muss synchronisiert.	Node muss bereit und konfiguriert sein.									
Allgemeine Bedingungen	Die Tests werden unter belastetem und unbelastetem Mesh-Netzwerk durchgeführt										

C Gesamt Terminplanung

Personen:

RAN Raffel Anklin
RBO Robin Bobst
CHO Cyrill Horath

D Bluetooth Mesh Terminplanung

Personen:

RAN Raffel Anklin
RBO Robin Bobst
CHO Cyril Horath

E Thread Terminplanung

Personen:

PERSONEN

RAN	Raffel Anklin
RBO	Robin Bobst
CHO	Cyrill Horath

F Zigbee Terminplanung

Personen:

PERSONEN:

RAN	Raffel Anklin
RBO	Robin Bobst
CHO	Cyrill Horath

G Risikoanalyse

Nr.	Ereignis			Risiko ohne Massnahmen	Prävention	Risiko mit Massnahmen			Verantwortlich	Indikator
	Risiko	Ursachen	Konsequenzen			si	pi	si		
	A Teammitglied fällt kurzfristig aus	Unvorhergesehener Termin, leichte Krankheit, leichter Unfall	Weniger Personalressourcen, kleiner Mehraufwand	3 2 6	Reservezeit einplanen, Transparenter Informationsfluss im Team	1	2	2	CHO	Abwesenheit
B Teammitglied fällt längerfristig aus	Militärdienst, schwere Krankheit, Studienabbruch, schwerer Unfall	Größere Umlaufplanung, Neuverteilung der Arbeiten	3 2 6	Strukturierte Datenablage, guter Kommunikationsfluss	1	2	2	CHO	Abwesenheit	
C Datenverlust oder Zugriffsprobleme	Löschnung der Projektdateien, Unzugänglichkeit von Onedrive, kein Internetverbindung	Zugriff auf Daten nicht möglich, Sämtliche Projektdateien nicht mehr vorhanden	2 1 2	Regelmäßige Backups, Dokumente zusätzlich lokal abspeichern	1	1	1	RAN	Arbeiten auf dem Stand des letzten Backups	
D Software kann nicht mehr ausgeführt werden	Datenverlust, Softwareupdate	Schlimmstenfalls Verlust der gesamten Arbeit, vorübergehende Arbeitspause bis Update komplett	2 3 6	Fertige Softwareteile werden zusätzlich im Onedrive gespeichert (Revisionsverwaltung) / Github	2	1	2	RAN	Fehlmeldung	
E Softwarekonzept nicht ausführbar	Mangelnde Vorkenntnisse, schlechte Planung	Überdenken der Arbeit, Verzug der Arbeiten	2 2 4	Mit Software-Fachcoach besprechen	1	1	1	RAN	Nicht funktionierendes Skript	
F Softwareprojekt von Funk Masternode und Leitrechner nicht verknüpfbar	Schnittstelle wurde nicht korrekt eingehalten	Verzögerung der Arbeit, Mehraufwand	2 2 4	Kommunikation zwischen Softwareteam	1	1	1	RAN	Softwareteam können Vorhaben nicht weiterführen	
G Zu komplizierter Sachverhalt	Inhalt kann nicht umgesetzt werden	Stillstand der Arbeit, Projekt nicht durchführbar	1 3 3	Frühzeitige Besprechung mit Fachcoaches	1	1	1	RAN	Kein Weiterkommen	
H Soziale Spannungen im Team	Meinungsverschiedenheiten, schlechte Arbeitsaufteilung, keine Kompromissbereitschaft	Motivation sinkt, Arbeitsmoral sinkt, schlechte Projektarbeit, unzufriedener Arbeitgeber	2 2 4	Gegenseitige Kontrolle, Fehler offen im Team besprechen, Konstruktive Kritik	2	1	2	BOB	Schlechte Arbeitsmoral	
I Mangelnde Kommunikation	zu wenig Sitzungen, Angst vor Demütigung	Schlechtes Zusammenspiel, schlechtere Arbeit	2 1 2	Häufigere Sitzungen, höhere Wertschätzung der einzelnen Teammitglieder	1	1	1	BOB	Zurückhaltung	
J Nicht Termingerechte Abgabe der Arbeiten	Faulheit, mangelnder Einsatz, falsche Prioritäten, schlechte Projektführung	Terminplan kann nicht eingehalten werden	2 2 4	striktere Projektführung, gegenseitige Kontrolle, frühzeitiges Melden	2	1	2	BOB	Schlechte Arbeitsmoral	
K Qualitativ minderwertige Arbeit	Faulheit, mangelnder Einsatz, schlechter Teamgeist	Mehraufwand, Qualitativ ungenügende Arbeit, Zeitliche Probleme	2 2 4	Gegenkontrolle der Arbeiten	2	1	2	CHO	Schlechte Arbeitsmoral	
L Schlechte Terminplanung	Aufwand unterschätzt, keine Reserve eingeplant	Mehraufwand, Überarbeitung des Terminplans, Engpässe	3 2 6	Genug Reservezeit einplanen	2	1	2	CHO	Terminverzug	

si=Eintrittswahrscheinlichkeit

pi=Auswirkung

C Bericht emv Messung Development Kits



Fachhochschule Nordwestschweiz
Hochschule für Technik

Projektbericht emv

Abstrahlcharakteristik von PCB Antennen

EMV FS20 - ANKLIN, HORATH

10. Juni 2020

Team:

Raffael Anklin
Cyrill Horath

Dozent:

Pascal Schleuniger

Studiengang:

Elektro- und Informationstechnik

Semester:

Frühlingssemester 2020

Inhaltsverzeichnis

1 Ausgangslage und Ziele	1
2 Funktionsbeschrieb	2
2.1 Hardware	2
2.2 Firmware	3
3 Messungen	4
3.1 Verfahren	4
3.2 Aufbau	5
3.3 Messerwartung	6
3.4 Ablauf	6
3.5 Resultate	6
4 Auswertung und Fazit	9
Literatur	10
Abbildungsverzeichnis	11

1 Ausgangslage und Ziele

Im Rahmen des Moduls emv (Elektromagnetische Verträglichkeit) soll eine Messaufgabe mit emv-Thematik durchgeführt werden. Die in unserer Bachelorthesis verwendete Hardware soll dabei als Messobjekt dienen. Es handelt sich dabei um zwei Arten von Development Kits (DK) für das System-on-Chip nRF52840 von Nordic Semiconductors. Dieses verwenden wir im Rahmen der Thesis für einen Performance Vergleich der drei gängigsten Low Power Mesh Netzwerken Bluetooth Mesh, Thread und Zigbee. Alle drei Protokolle arbeiten unter anderem auf dem 2.4 GHz ISM Band welches im Fokus unserer Betrachtungen steht. Die beiden DK's beinhalten nebst anderer Peripherie eine PCB-Antenne die jeweils unterschiedlich ausgeführt ist. Auf die Ausführung der Antennen wird im Abschnitt 2.1 genauer eingegangen. Ziel der vorliegenden Messungen ist es, die PCB-Antennen der beiden DK auszumessen und damit deren Abstrahlcharakteristik kennen zu lernen und zu dokumentieren. Daraus können dann wiederum Vor- und Nachteile der verschiedenen Antennenformen ersichtlich gemacht werden. Außerdem ist eine bekannte Abstrahlcharakteristik hilfreich bei der Optimierung von Netzwerken.

2 Funktionsbeschrieb

Der folgende Abschnitt soll einen Überblick über die in der Thesis und in den Messungen verwendete Hardware sowie über die in den Messungen verwendete Firmware geben.

2.1 Hardware

nRF52840

Der nRF52840 ist ein System-on-Chip (SoC) vom Hersteller Nordic Semiconductor der um eine 32-Bit Cortex M4 CPU aufgebaut ist und mit 1MB Flash sowie 256KB RAM ausgestattet wurde. Der SoC unterstützt die drahtlosen Übertragungsprotokolle Bluetooth 5, Bluetooth Mesh, Thread, Zigbee sowie weitere proprietäre Protokolle. Er ist also bereits ausgestattet mit einem 2.4 GHz Transceiver. Weitere Peripherien via UART, SPI, I2S wie auch NFC werden von diesem SoC ebenso unterstützt. [1]

nRF52840-DK PCA10056

Das nRF52840-DK PCA10056 (siehe Abbildung 2.1) ist das offizielle Hardware Development Kit für den nRF52840 SoC welches von Nordic Semiconductor speziell für die Entwicklung von Bluetooth, Thread und Zigbee Produkten angeboten wird. Es beinhaltet nebst dem SoC diverse Peripheriebausteine wie Buttons, LEDs und Pins sowie einen Segger J-Link Programmer und Debugger. Ausserdem ist auf dem DK eine PCB Antenne umgesetzt in Form einer Inverted-F Antenna (IFA) die ganz rechts in Abbildung 2.2 ersichtlich ist. [2]

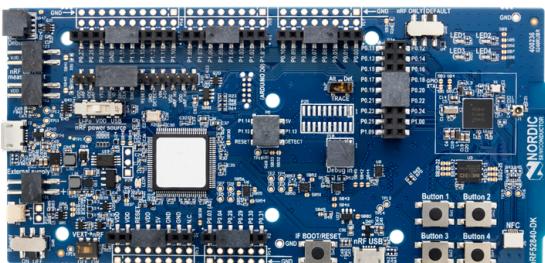


Abbildung 2.1: PCA10056 [2]

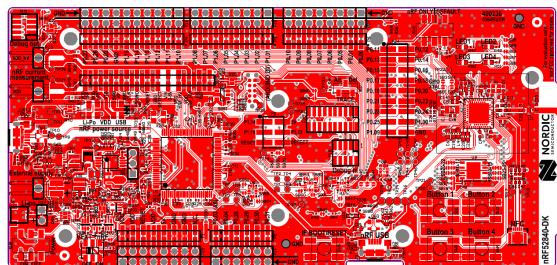


Abbildung 2.2: PCA10056 Copperplane [3]

nRF52840-Dongle PCA10059

Der nRF52840-Dongle PCA10059 (siehe Abbildung 2.3) ist die reduzierte Variante des Development Kit von Nordic Semiconductor. Nebst dem SoC ist nur die absolut nötigste Peripherie verbaut wozu beispielsweise ein Button sowie ein RGB-LED gehören. Der Dongle ist als USB Gerät Typ A entworfen wodurch die Handhabung sehr komfortabel wird. Aufgrund der engen Platzverhältnisse ist für die 2.4 GHz Kommunikation auf dem Board eine Meandered Inverted-F Antenna (MIFA) umgesetzt welche in der Abbildung 2.4 unter dem Nordic Semiconductor Schriftzug sichtbar ist. [4]

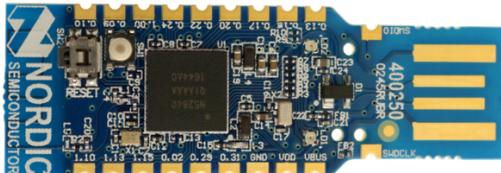


Abbildung 2.3: PCA10059 [4]

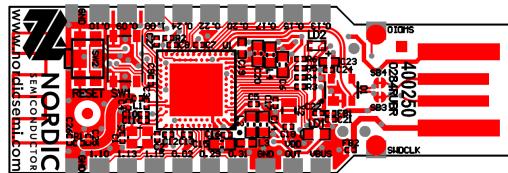


Abbildung 2.4: PCA10059 Copperplane [5]

Speisung

Die Speisung der beiden Messobjekte ist mit einem Batteriepaket umgesetzt damit die Geräte

möglich autonom funktionieren können. Das Batteriepaket besteht aus zwei in Serie geschaltete Mignonzellen vom Typ AA. Zusammen stellen Sie die Speisespannung von 3V zur Verfügung.

2.2 Firmware

Basierend auf Beispielen aus dem nRF5 SDK for Mesh wird für die Messungen eine Firmware verwendet die ununterbrochen Bluetooth Beacons aussendet. Dadurch kann eine kontinuierliche Messung durchgeführt werden. Da die Abstrahlcharakteristik der Antennen lediglich von der Frequenz, unabhängig jedoch vom verwendeten Protokoll ist, wird nur diese eine Firmware getestet. Auch bei der Sendeleistung wird der Standardwert der SDK verwendet da es primär um den relativen Vergleich der Antennen geht und nicht um die Bestimmung der effektiven Sendeleistung. Der Vergleich der Abstrahlcharakteristiken sollte unabhängig von der Sendeleistung sein. Die Beacons werden in einem Intervall von rund $500\mu\text{s}$ versendet und sind jeweils 36 Bytes gross. Als Modulationsart wird GMSK mit 1MBit/s Übertragungsrate eingesetzt. Die Bandbreite des Signals beträgt dabei 2MHz.

3 Messungen

Im folgenden Abschnitt wird auf die Messung der im vorhergehenden Kapitel beschrieben Hardware eingegangen. Zuerst wird das Messverfahren erläutert. Anschliessend der Aufbau der Messungen beschrieben und schliesslich die Messerwartungen und Resultate aufgeführt.

3.1 Verfahren

Um die Abstrahlcharakteristik der Antennen zu bestimmen wird die abgestrahlte Emission in Abhängigkeit des Winkels gemessen. Dazu muss der zu prüfenden Gegenstand um 180° gedreht und um 90° vor und zurück gekippt werden können. Dieses Verfahren wird in der EMV Prüftechnik normalerweise dazu eingesetzt, um zu überprüfen ob die Emissions-Grenzwerte eingehalten werden. Die Norm IEC 61000-6-3/4 regelt die Vorschriften für neu entwickelte Geräte im Wohn- und Gewerbebereich respektive im Industriebereich. Als Beispiel dafür zeigt Abbildung 3.1 eine Emissionskurve einer LED. Der Grenzwert stellt die Norm EN55015 dar und ist als rote Linie eingezeichnet. [6]

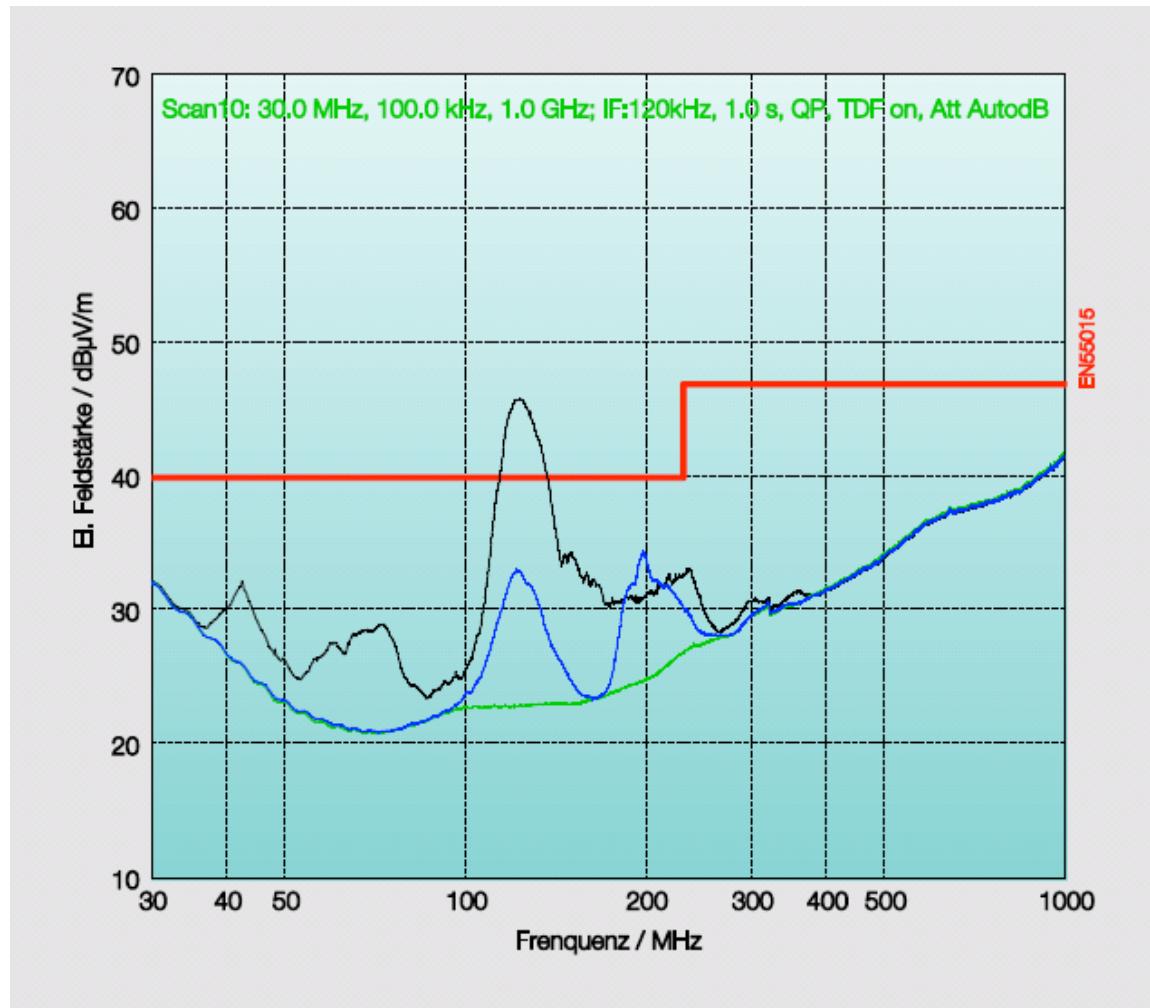


Abbildung 3.1: Beispiel Emissionskurve LED im Test mit Grenzwert (rote Linie) gemäss EN55015 [7]

Das Ziel dieser Arbeit besteht jedoch nicht im Nachweisen der Einhaltung bestimmter Normen, vielmehr soll die Messung dem Erfassen der Abstrahlcharakteristik im dreidimensionalen Raum dienen. Der Messaufbau um diese Daten zu erhalten wird im folgendem Abschnitt genauer erläutert.

3.2 Aufbau

Der Messungen wurden in einem geerdetem und abgeschirmten Prüfcontainer (EMV-Messhalle) durchgeführt um die Störeinflüsse aus der Umgebung so klein wie möglich zu halten. Die wichtigsten Bestandteile des Messaufbaus sind:

- **Messvorrichtung** dient dazu das Messobjekt zu justieren und somit die unterschiedlichen Messwinkel anzufahren.
- **Empfangsantenne** welche gegenüber des Messobjekts montiert ist, detektiert die Emission.
- **Spektrum Analyzer** an welchem die Empfangsantenne angeschlossen ist, erfasst den Pegel der empfangenen Signale.
- **Messcomputer** steuert die Messungen sowie die Positionierung der Messvorrichtung und speichert die Messdaten ab.

Die Messvorrichtung wird in Abbildung 3.2 gezeigt. Das Messobjekt, in diesem Fall das nRF52840-DK PCA10056, wird auf der Messplatte eingespannt. Diese ist über zwei Achsen gelagert und verstellbar. Um störende Reflexionen durch die Messvorrichtung zu verhindern, wurde ein Pappbecher als Abstandshalter zwischen Messplatte und Messobjekt verwendet. Der Messcomputer steuert die Servomotoren an, welche für die Ausrichtung des Messobjekts zuständig sind.

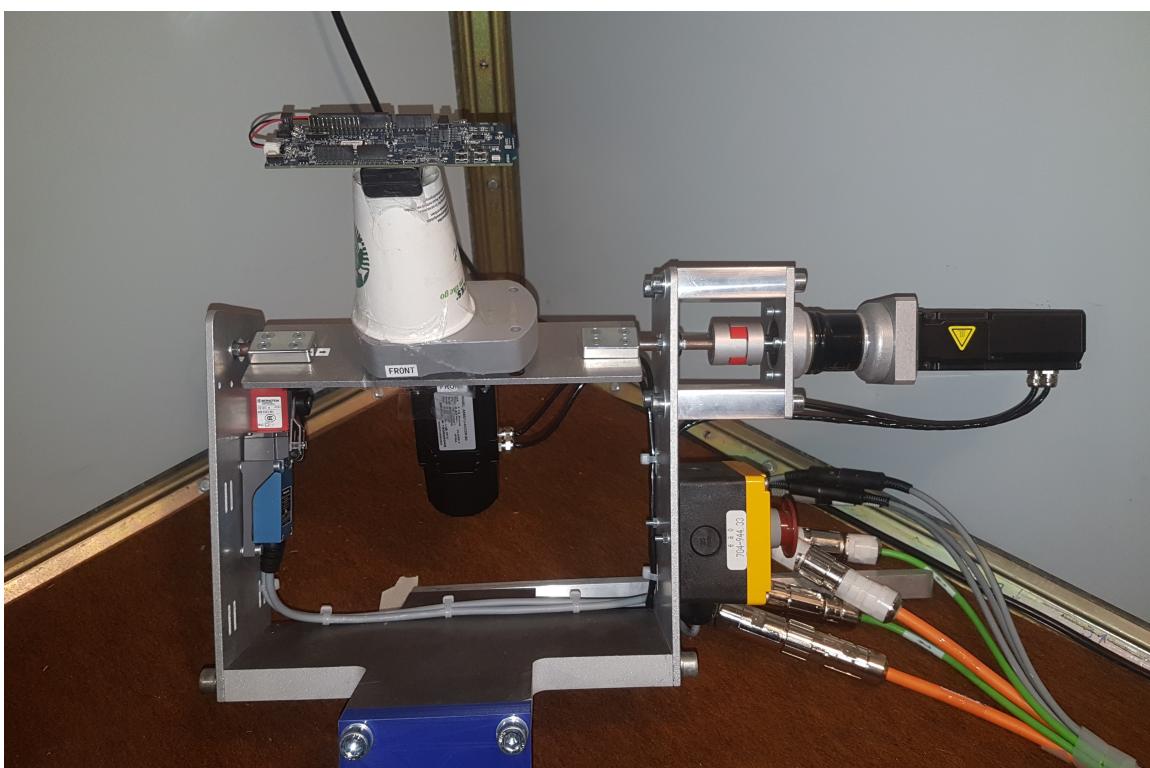


Abbildung 3.2: Eingespannetes Messobjekt in Messvorrichtung

Die Empfangsantenne ist eine Vivaldi Antenne vom Typ PowerLOG 70180 (siehe Abbildung 3.4). Diese weist ein sehr breitbandiges Empfangsspektrum auf und ist daher ideal um alle möglichen Spektren auszumessen. [8] [9]

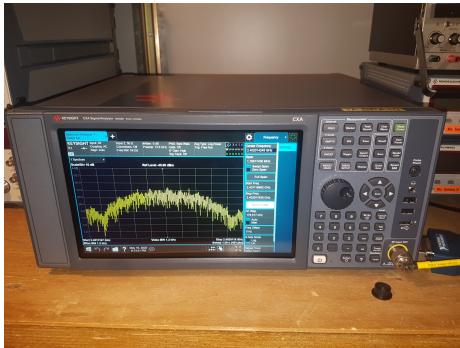


Abbildung 3.3: Oszilloskop an Antenne angeschlossen



Abbildung 3.4: Vivaldi Antenne PowerLOG 70180 zur Messung [9]

Die Empfangsantenne ist an ein Oszilloskop (siehe Abbildung 3.3) angeschlossen. Es dient als Empfänger welcher die Daten aufbereitet und an den Messcomputer weiterleitet. Der Computer koordiniert die Messung und Speichert die Daten ab. Zur Auswertung wird der maximal empfangene Pegel und der durchschnittliche Pegel in ein Excelfile geschrieben. Mithilfe eines Matlab-Skripts werden die Daten zur Visualisierung gebracht.

3.3 Messerwartung

In vorhergehenden Feldversuchen hat sich gezeigt, dass das Board PCA10056 eine grössere Reichweite gegenüber dem PCB PCA10059 aufweist. Daher wird auf dem PCA10056 eine stärkere Abstrahlung gegenüber dem kleineren Board erwartet. Bei beiden Messobjekten wird eine gewisse Abhängigkeit bezüglich der Antennenausrichtung erwartet.

3.4 Ablauf

Die Messungen wurden dreimal durchgeführt. Zuerst wurde nur das Grundrauschen gemessen um als Referenzpegel zu dienen. Anschliessend wurden die Sender eingeschaltet und nacheinander in der Messvorrichtung eingespannt. Die Messung erfolgte in 6° Grad Schritten und dauerte damit ungefähr 45 Minuten.

3.5 Resultate

In Abbildung 3.5 und Abbildung 3.7 ist die Abstrahlcharakteristik über eine Halbkugel verteilt dargestellt. Die Bilder zeigen die Abweichung gegenüber dem Grundrauschen in Leistungs-dB. Dazu wurden die Spitzenwerte des Senders gegenüber dem durchschnittlichen Grundrauschpegel ausgewertet. Die Leiterplatten wurden ungefähr gemäss Abbildung 3.6 und Abbildung 3.8 orientiert dazu eingespannt.

3.5 Resultate

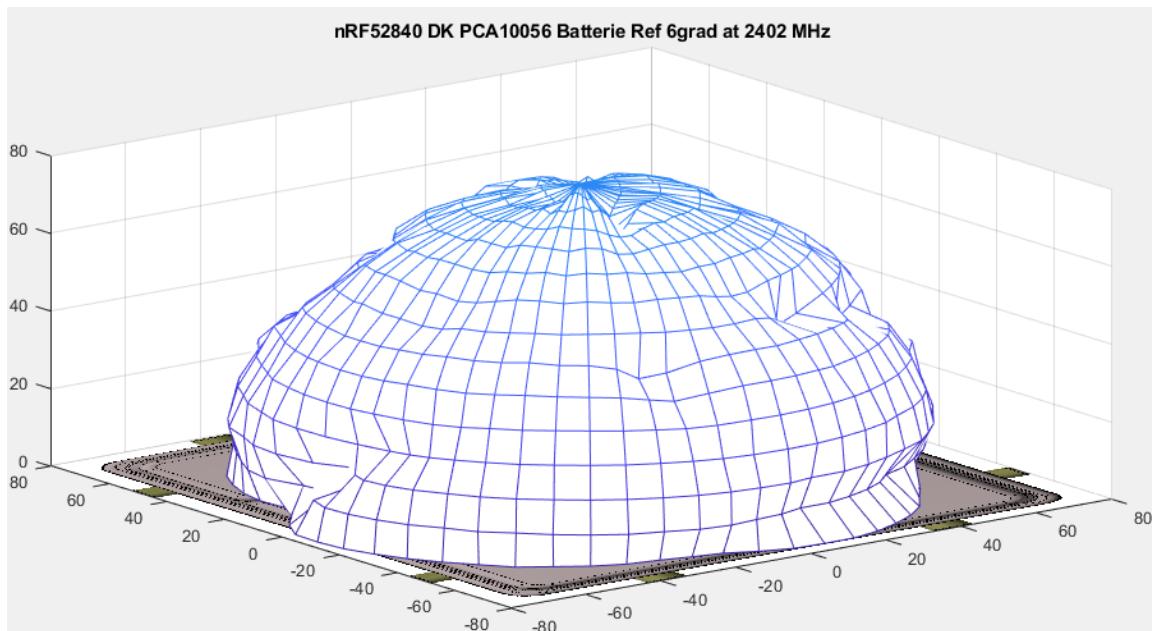


Abbildung 3.5: Abstrahlungscharakteristik der IFA Antenne auf dem Board PCA10056

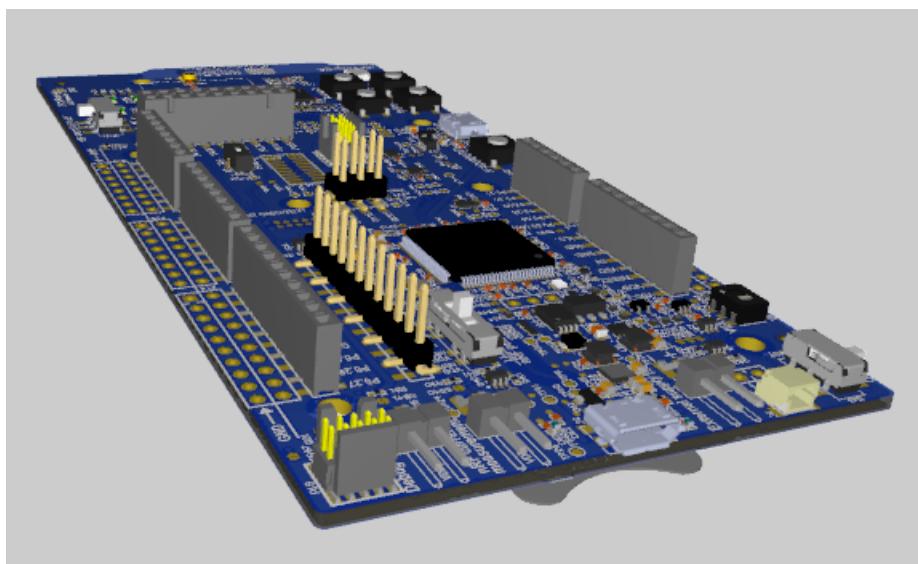


Abbildung 3.6: Ausrichtung PCA10056

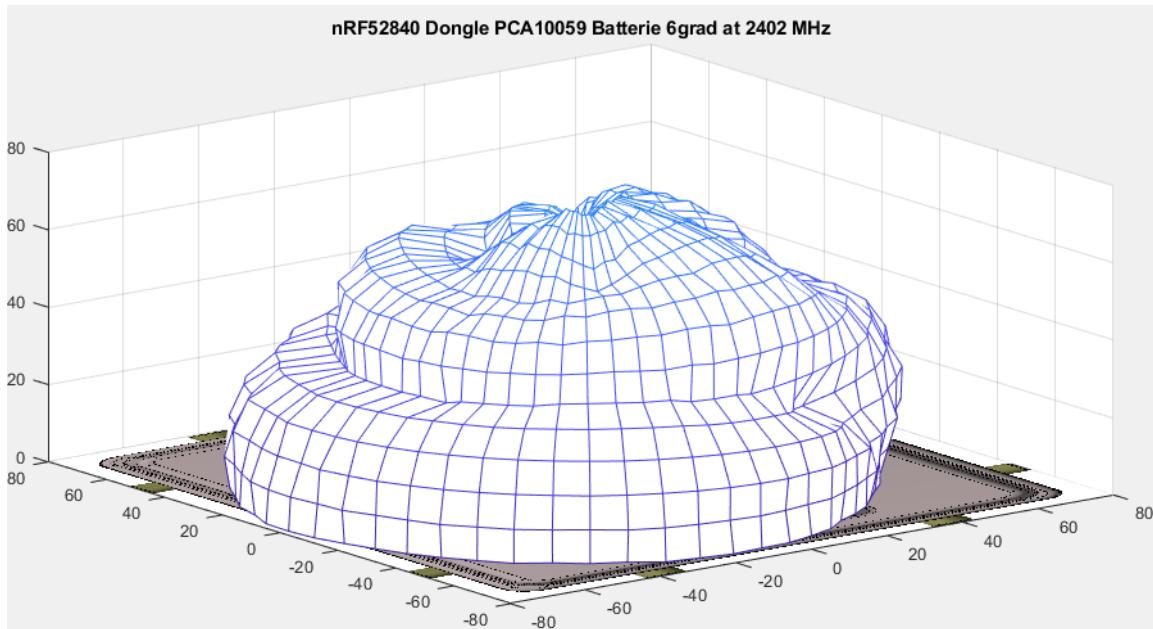


Abbildung 3.7: Abstrahlungscharakteristik der MIFA Antenne auf dem Board PCA10059

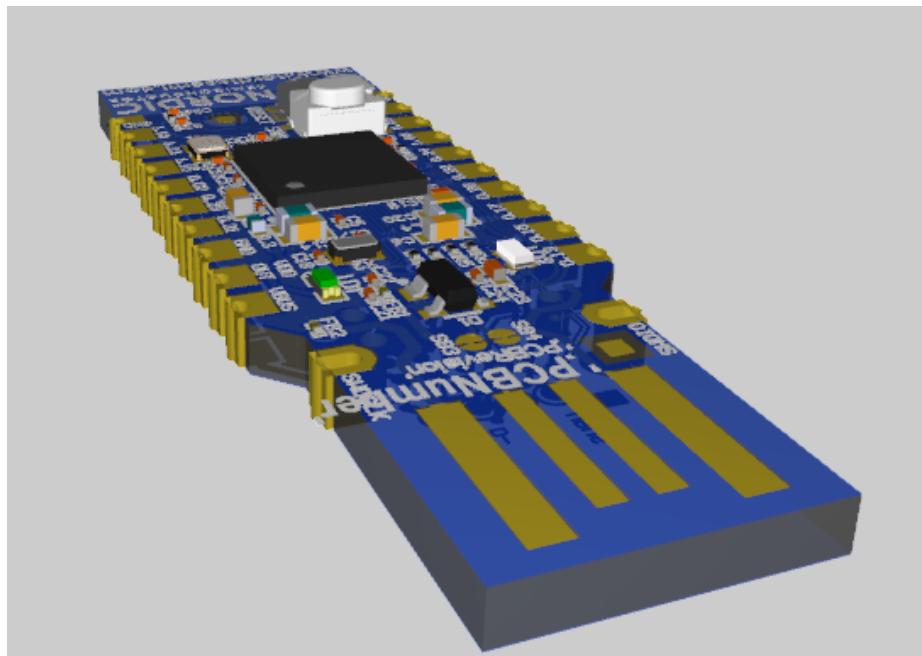


Abbildung 3.8: Ausrichtung PCA10059

4 Auswertung und Fazit

Die in den Abbildungen im Abschnitt 3.5 dargestellten Abstrahlcharakteristiken der Antennen weisen eine erstaunlich gleichmässige Verteilung auf. Besonders die IFA-Antenne des PCA10056 erzeugt eine nahezu ideale Abstrahlung (3.5) über die gesamte Halbkugelfläche. An einzelnen Punkten weisst das 3D-Modell kleinere Einbuchtungen auf welche jedoch auf die Sende- und Empfangsqualität keinen oder nur einen sehr geringen Einfluss haben sollten. Im Gegensatz dazu weisst die MIFA-Antenne des PCA10059 wesentlich grössere Schwachstellen auf. In der Abbildung 3.7 sind diese klar ersichtlich in Form von Dellen in der halbkugelförmigen Abstrahlung. Diese Schwachstellen wirken sich in der Art und Weise auf die Sende- und Empfangsqualität aus, dass die Ausrichtung des Boards und somit der Antenne eine nicht zu vernachlässigende Rolle spielt. Im Idealfall ist demzufolge die Antenne Frontal auszurichten. Das heisst dass der Empfänger im rechten Winkel zur Oberfläche des PCA10059 steht. Allerdings sind auch in dieser Richtung Schwachstellen in der Charakteristik auszumachen welche beim PCA10056 so nicht vorhanden sind.

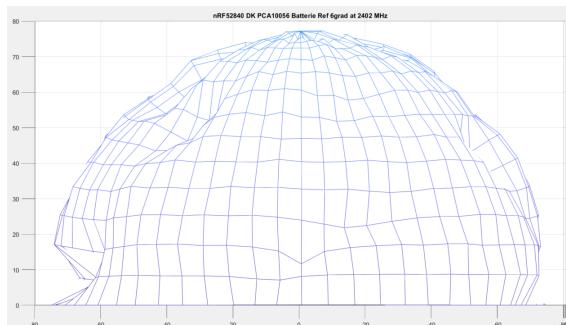


Abbildung 4.1: Abstrahlung in Z-Richtung
PCA10056

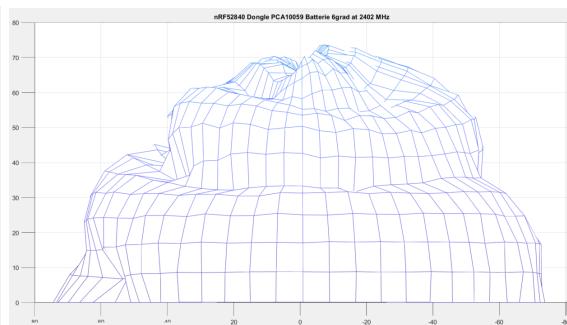


Abbildung 4.2: Abstrahlung in Z-Richtung
PCA10059

Wie bereits im Abschnitt 3.5 beschrieben, sind die Achsen der Grafiken in dBm dargestellt. Ausserdem handelt es sich bei der Darstellung um die Differenz der Spitzenleistungen und des Grundrauschen und nicht um Absolutwerte. Mit diesem Wissen kann beim Vergleich der Grafiken 4.1 und 4.2 zusätzlich noch ein Unterschied im maximal Pegel festgestellt werden. Die Spitze der Halbkugel in Z-Richtung liegt beim PCA10059 um ca. drei bis sechs dB tiefer als beim PCA10056. Dies bedeutet eine 2 bis 4 fach höhere Dämpfung an diesem Punkt was wiederum ein beträchtlicher Faktor ist. Wenn die deutlichen Einschnürungen an der Spitze noch mit betrachtet werden, ist die Dämpfung an diesem Punkt sogar noch höher.

Die gemessenen Abstrahleigenschaften der beiden Antennen bestätigen schliesslich unsere Erfahrungen die wir mit den Boards bereits gemacht haben. Die MIFA Antenne des PCA10059 weisst einige klare Nachteile gegenüber der IFA Antenne des PCA10056 auf, weshalb wenn möglich die Verwendung einer IFA Antenne empfohlen ist. Bei beschränkten Platz Verhältnissen wie dies in vielen Anwendungen der Fall ist, bietet jedoch auch die MIFA Antenne ein ordentliches Spektrum. Die Messerwartungen 3.3 wurden also erfüllt.

Literatur

- [1] Nordic Semiconductor, *nRF52840 DK*, en, Mai 2020. Adresse: <https://www.nordicsemi.com/en/Software%20and%20tools/Development%20Kits/nRF52840%20DK> (besucht am 14. Mai 2020).
- [2] ——, *nRF52840 - Nordic Semiconductor*, en, Mai 2020. Adresse: <https://www.nordicsemi.com/en/Products/Low%20power%20short-range%20wireless/nRF52840> (besucht am 14. Mai 2020).
- [3] N. S. ASA, *PCA10056_Schematic_And_PCB*, Okt. 2019. Adresse: http://infocenter.nordicsemi.com/pdf/nRF52840_DK_User_Guide_v1.4.1.pdf (besucht am 14. Mai 2020).
- [4] Nordic Semiconductor, *nRF52840 Dongle*, en, Mai 2020. Adresse: <https://www.nordicsemi.com/en/Software%20and%20tools/Development%20Kits/nRF52840%20Dongle> (besucht am 14. Mai 2020).
- [5] Nordic Semiconductor ASA, *PCA10059_Schematic_And_PCB*, Juni 2018. Adresse: http://infocenter.nordicsemi.com/pdf/nRF52840_Dongle_User_Guide_v1.1.pdf (besucht am 14. Mai 2020).
- [6] Schurter, „EMV-Emissionen in der Maschinenindustrie“, de, S. 27, Jan. 2014. Adresse: http://archiv.swisstmeeting.ch/tl_files/images/EMV%202014/2_3_Maschinenindustrie_Herbert%20Blum.pdf.
- [7] LED_Know_How, *EMV_Emission_LED*, Mai 2020. Adresse: <http://www.led-know-how.ch/de/ansteuerung/emv> (besucht am 23. Mai 2020).
- [8] RadarBasics, *Vivaldi- Antenne - Radar Basics*, de, Library Catalog: www.radartutorial.eu Publisher: Dipl.-Ing. (FH) Christian Wolff, Mai 2020. Adresse: <https://www.radartutorial.eu/06.antennas/Vivaldi-Antenne.de.html> (besucht am 23. Mai 2020).
- [9] Aaronia, *Aaronia_PowerLOG_Horn_Antennas.pdf*, Mai 2020. Adresse: https://downloads.aaronia.com/datasheets/antennas/PowerLOG/Aaronia_PowerLOG_Horn_Antennas.pdf (besucht am 26. Mai 2020).

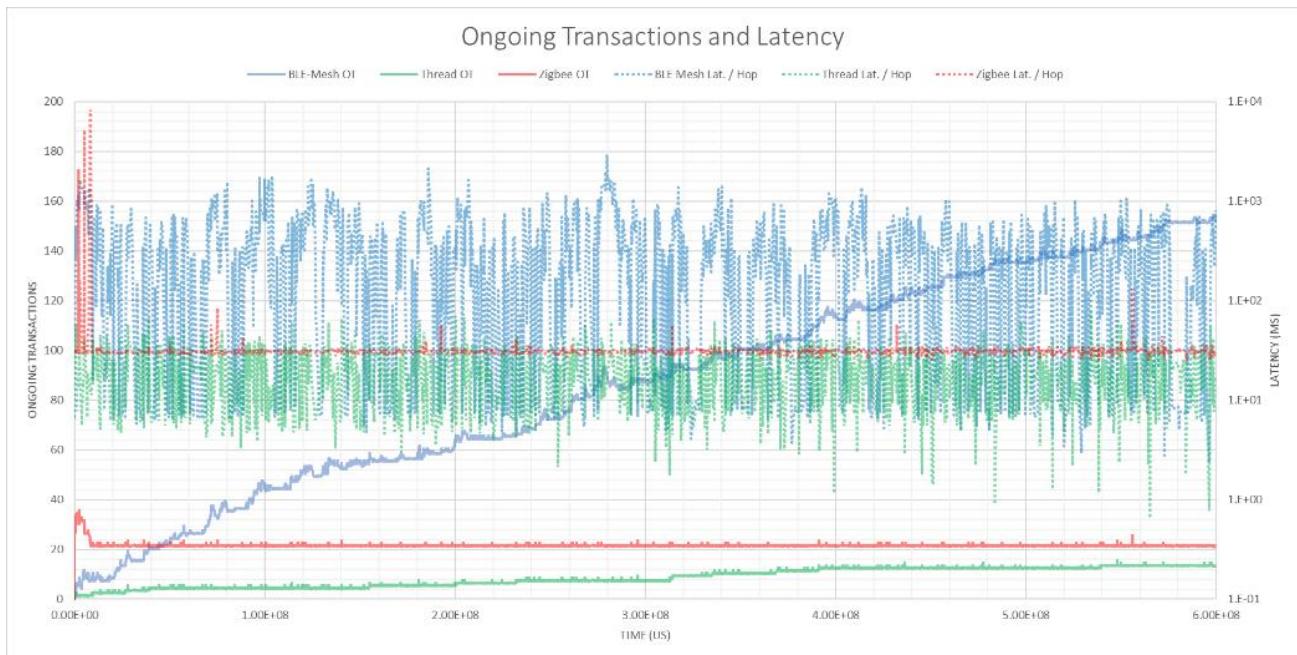
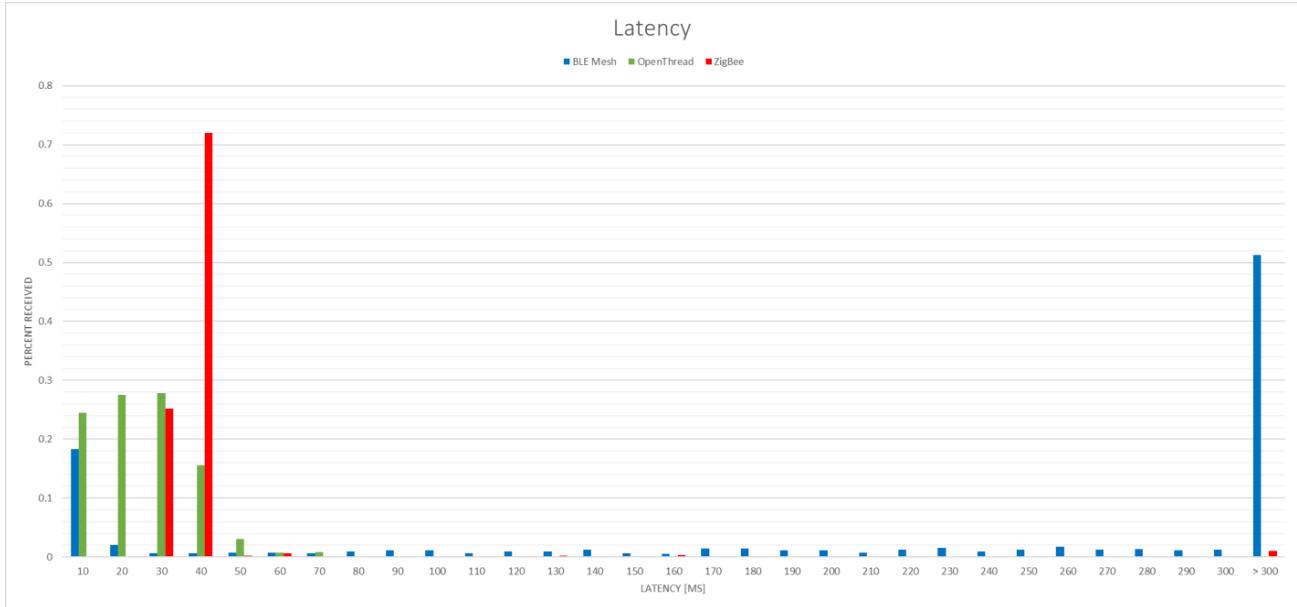
Abbildungsverzeichnis

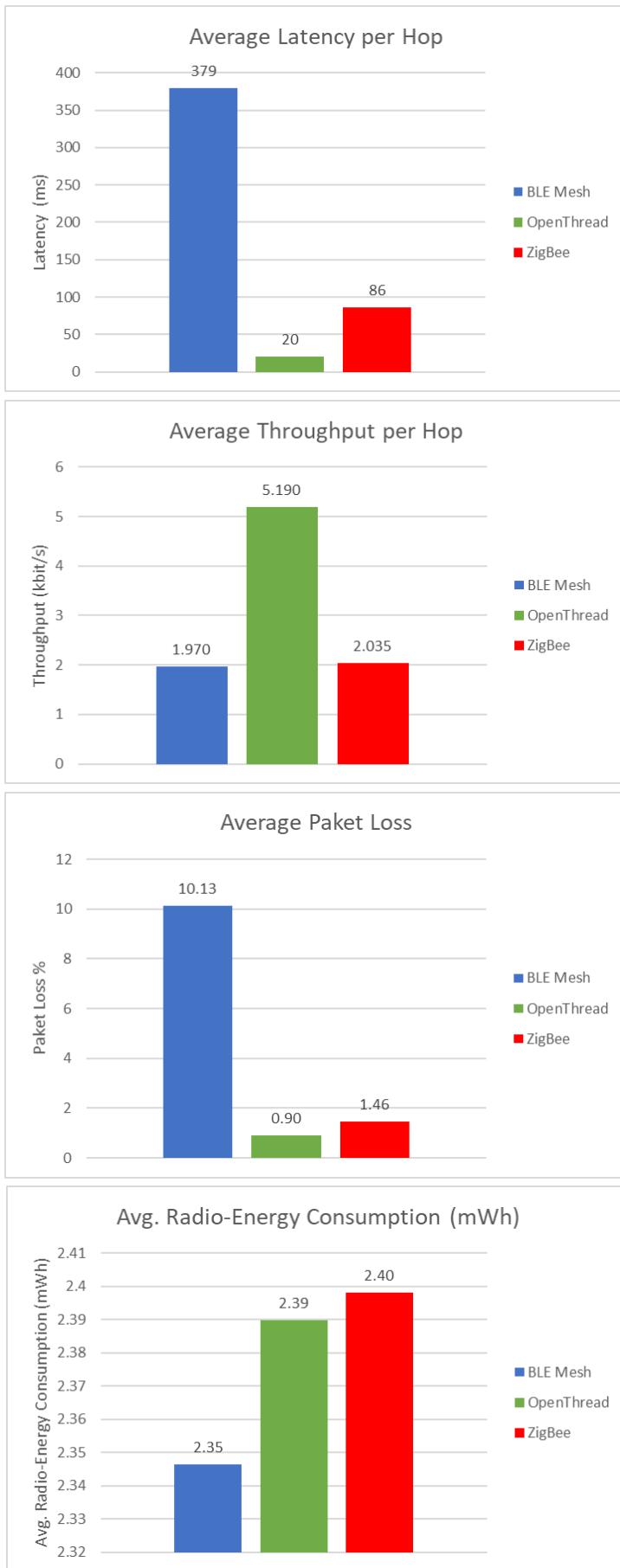
2.1	nRF52840-DK PCA10056	2
2.2	PCA10056 Copperplane mit IFA-Antenne	2
2.3	nRF52840-Dongle PCA10059	2
2.4	PCA10059 Copperplane mit MIFA-Antenne	2
3.1	Beispiel Emissionskurve LED im Test mit Grenzwert	4
3.2	Eingespannetes Messobjekt in Messvorrichtung	5
3.3	Oszilloskop an Antenne angeschlossen	6
3.4	Vivaldi Antenne PowerLOG 70180 zur Messung [9]	6
3.5	Abstrahlungscharakteristik der IFA Antenne auf dem Board PCA10056	7
3.6	Ausrichtung PCA10056	7
3.7	Abstrahlungscharakteristik der MIFA Antenne auf dem Board PCA10059	8
3.8	Ausrichtung PCA10059	8
4.1	Abstrahlung in Z-Richtung PCA10056	9
4.2	Abstrahlung in Z-Richtung PCA10059	9

D Messprotokolle Mesh Benchmark

Auswertung Mesh Benchmark

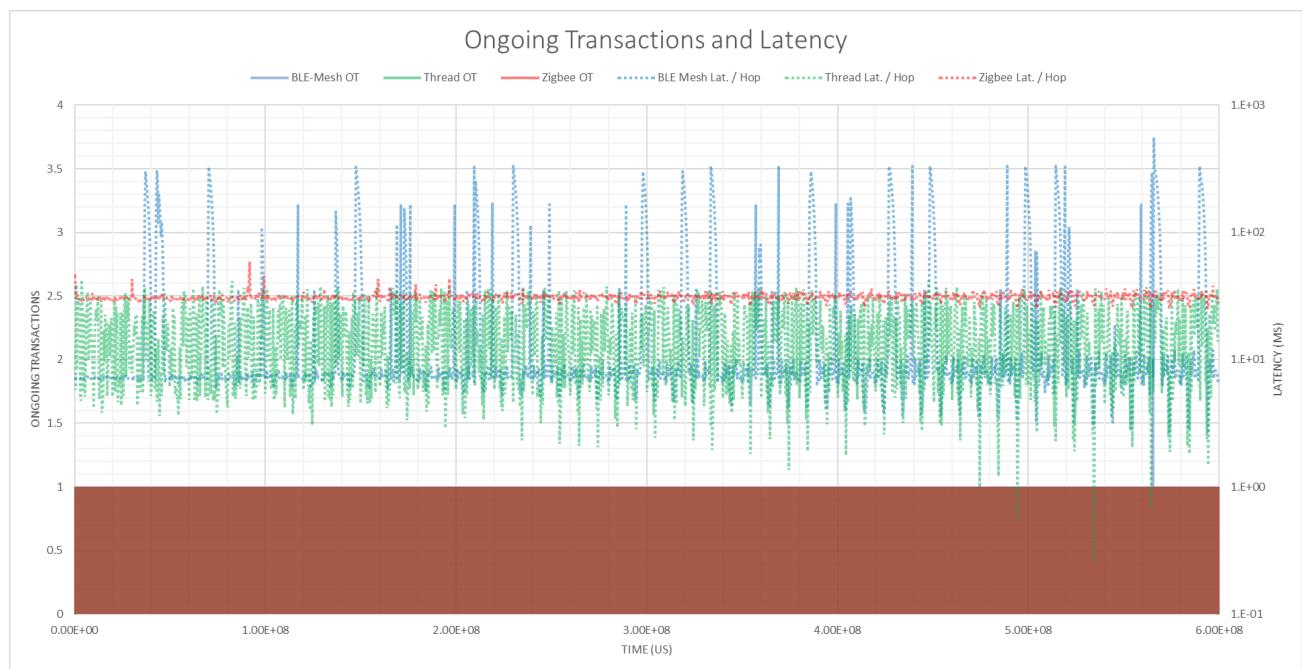
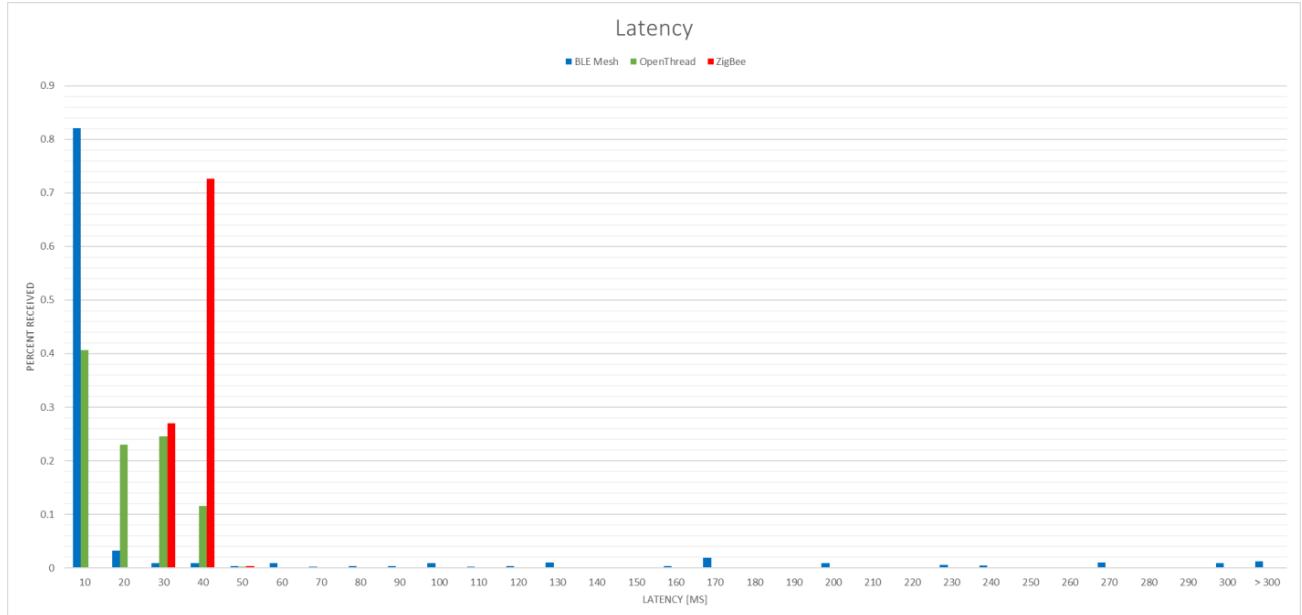
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
1	Rand	600s	60	Small	No	Labor

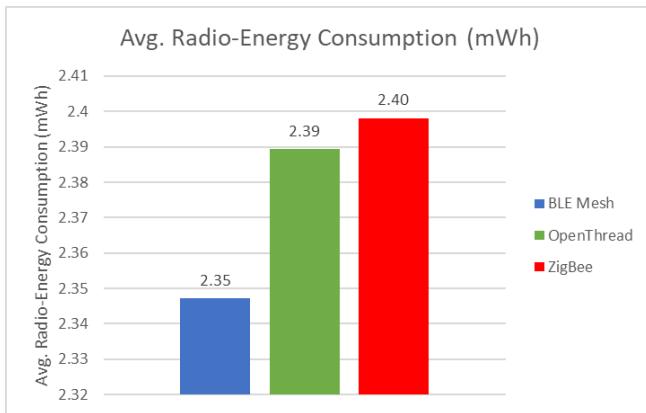
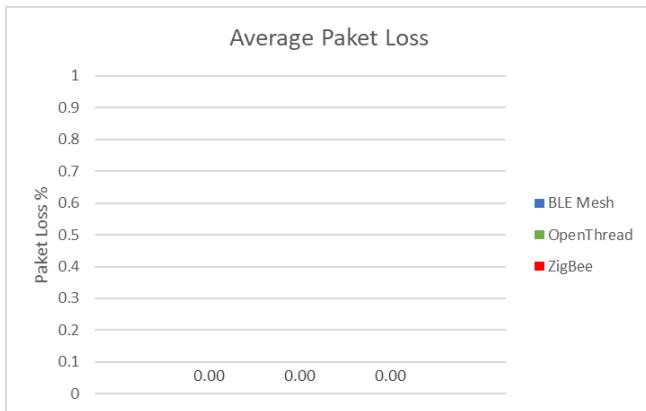
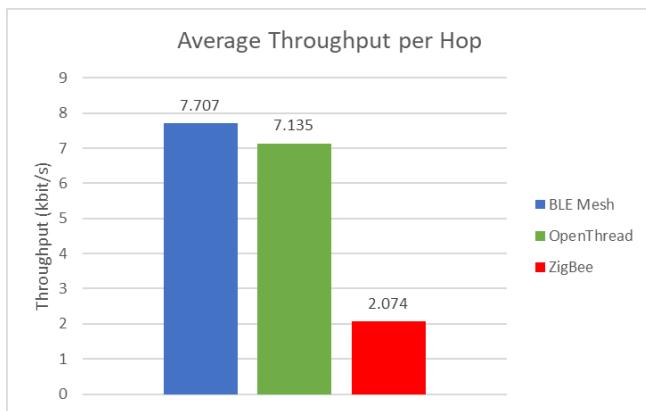
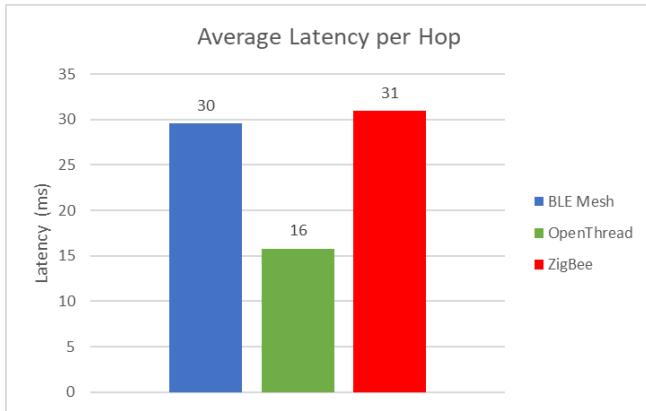




Auswertung Mesh Benchmark

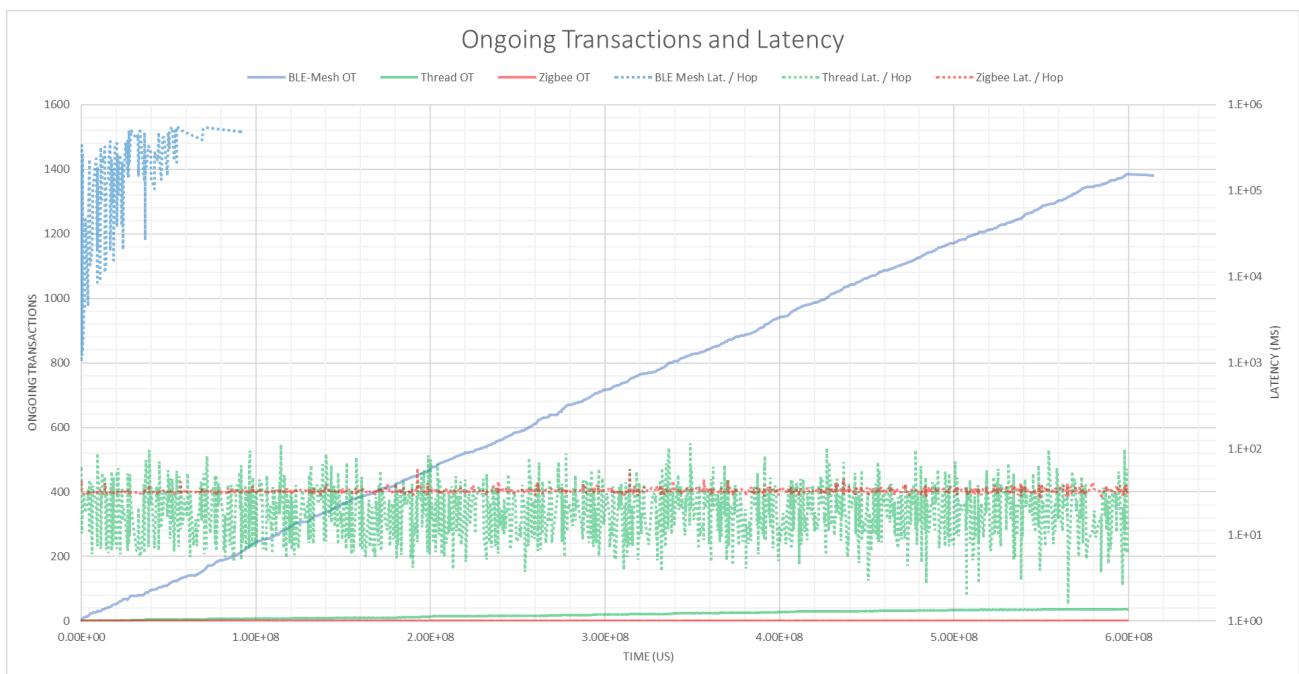
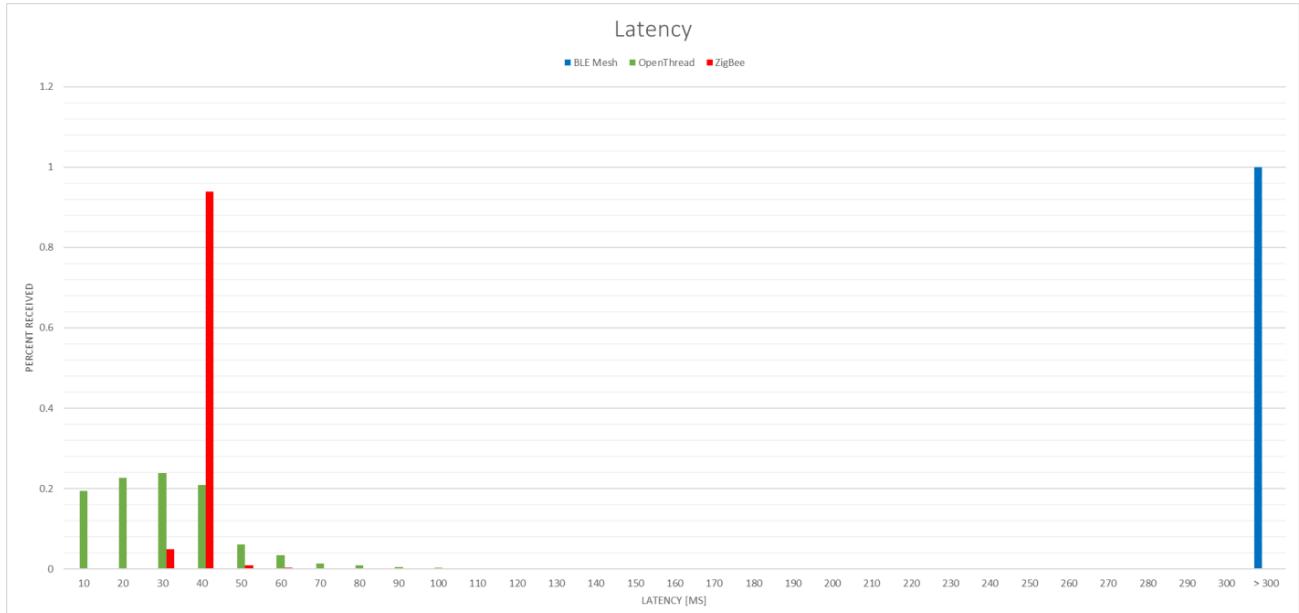
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
2	Seq	600s	60	Small	No	Labor

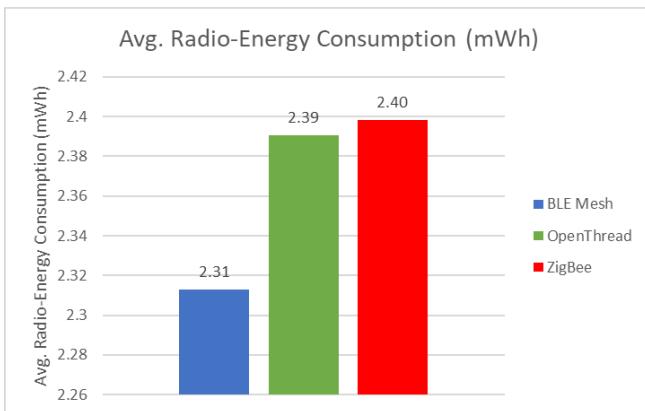
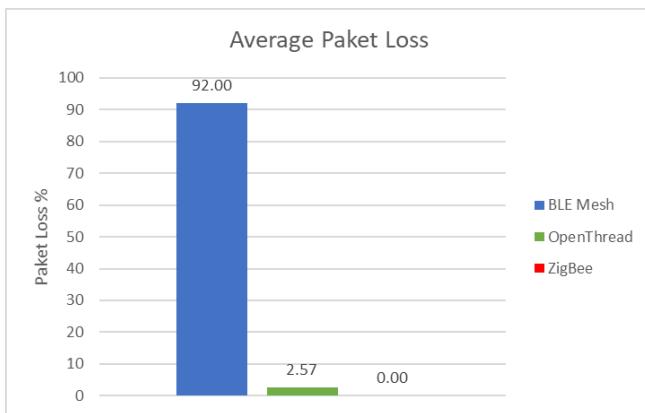
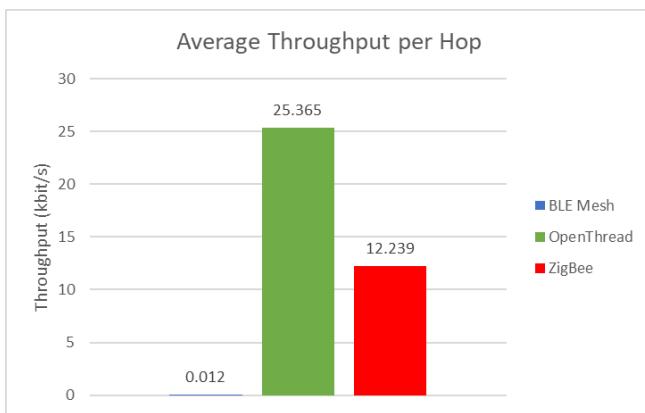
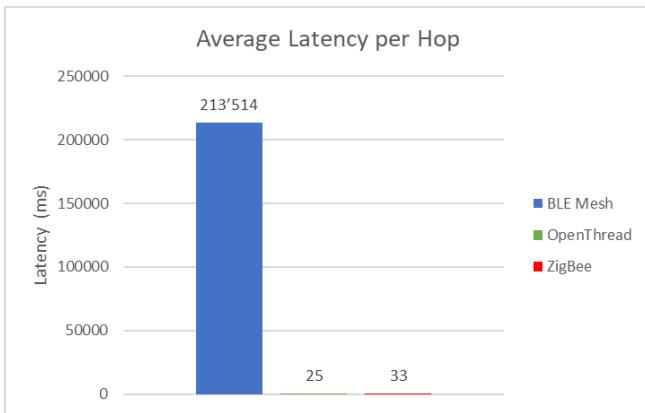




Auswertung Mesh Benchmark

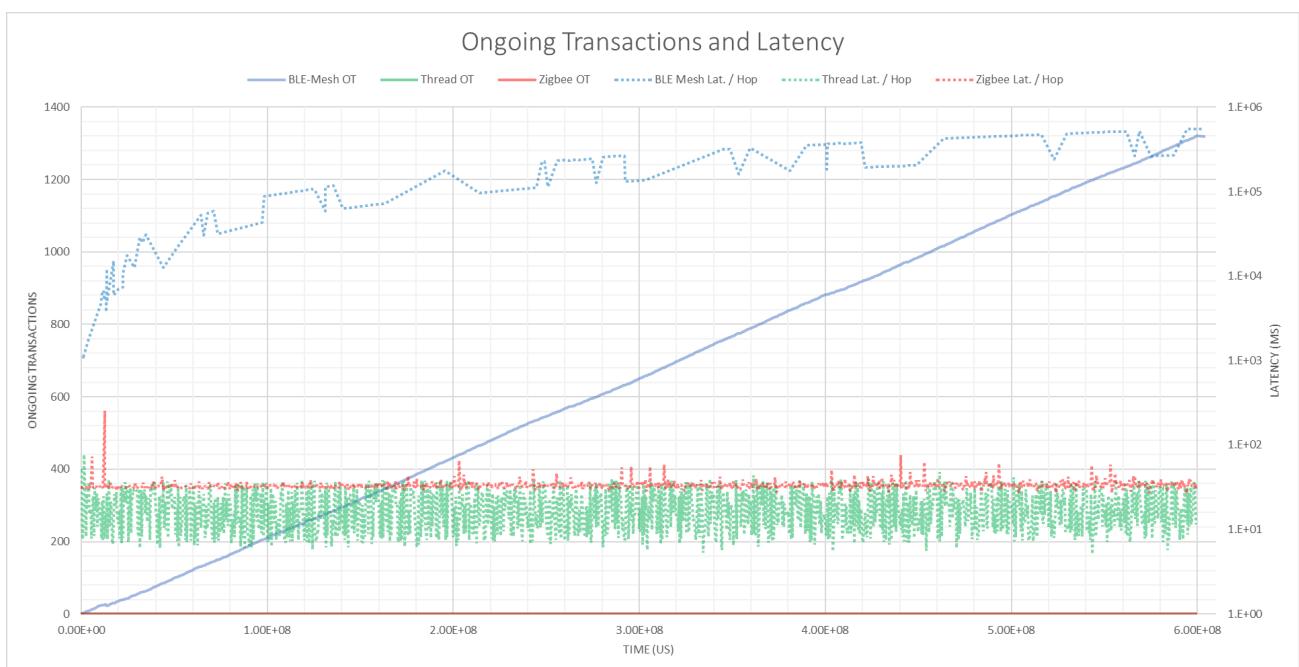
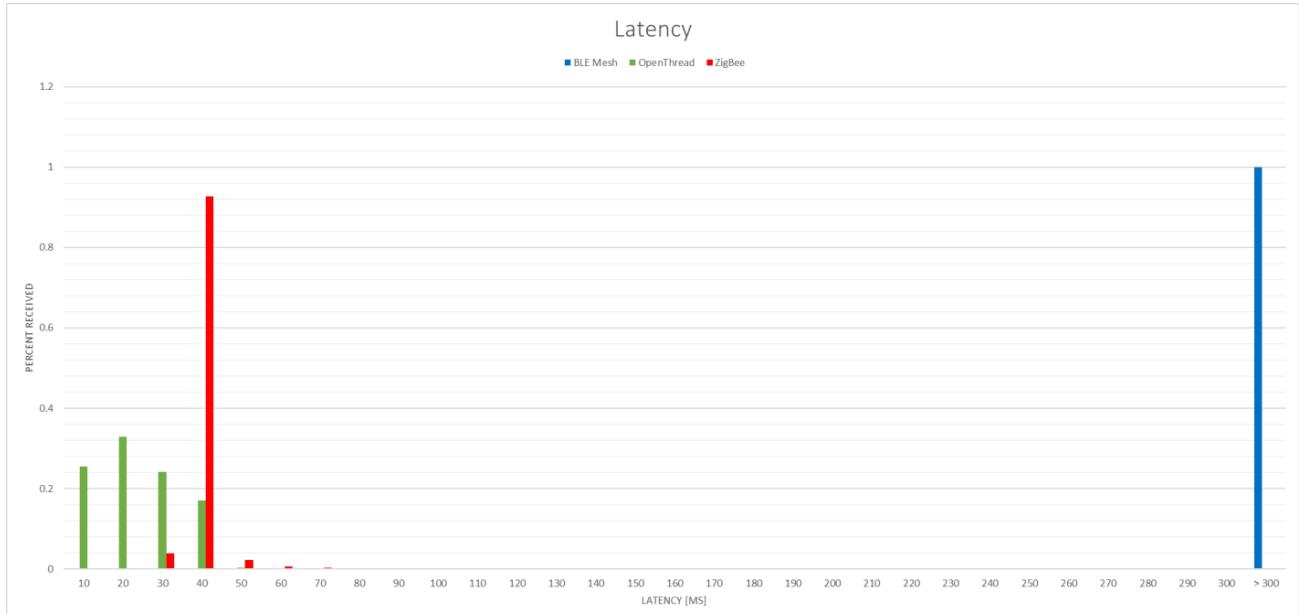
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
3	Rand	600s	60	Large	No	Labor

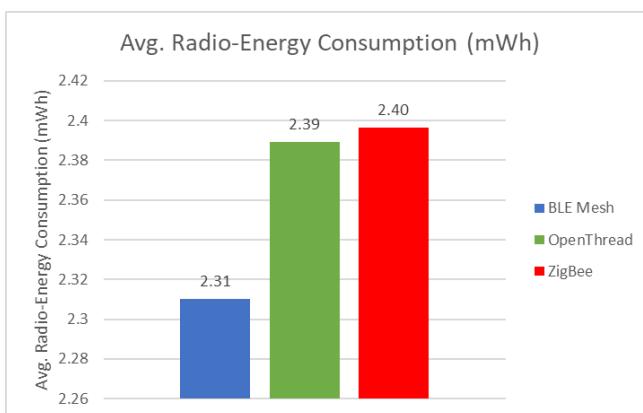
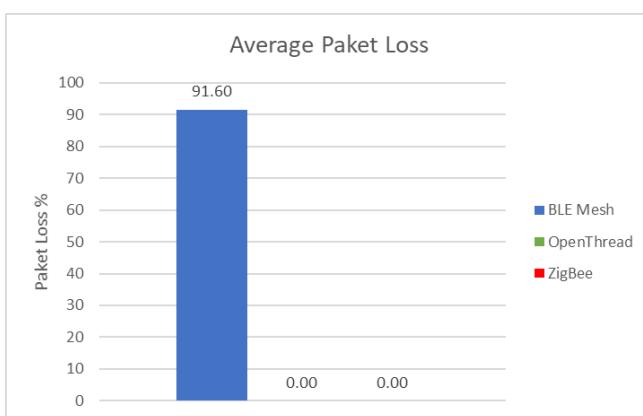
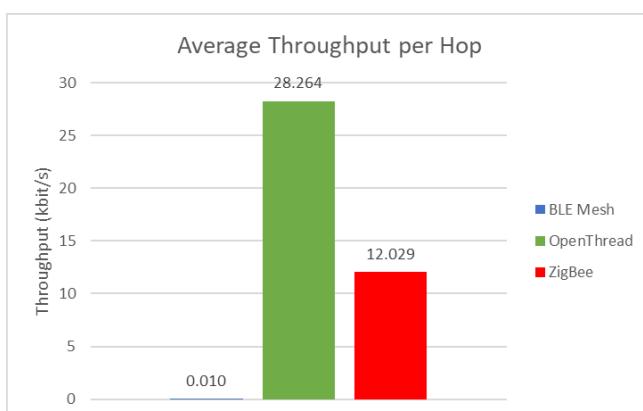
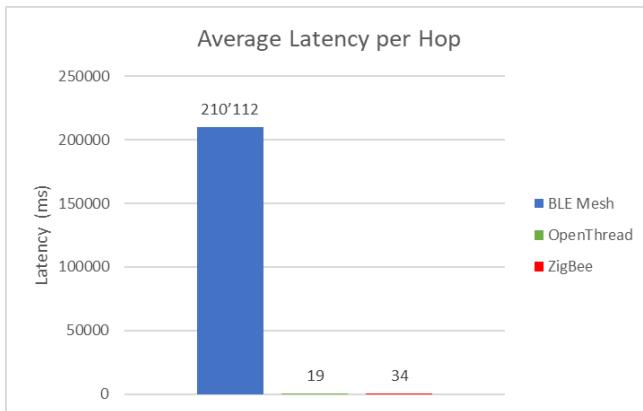




Auswertung Mesh Benchmark

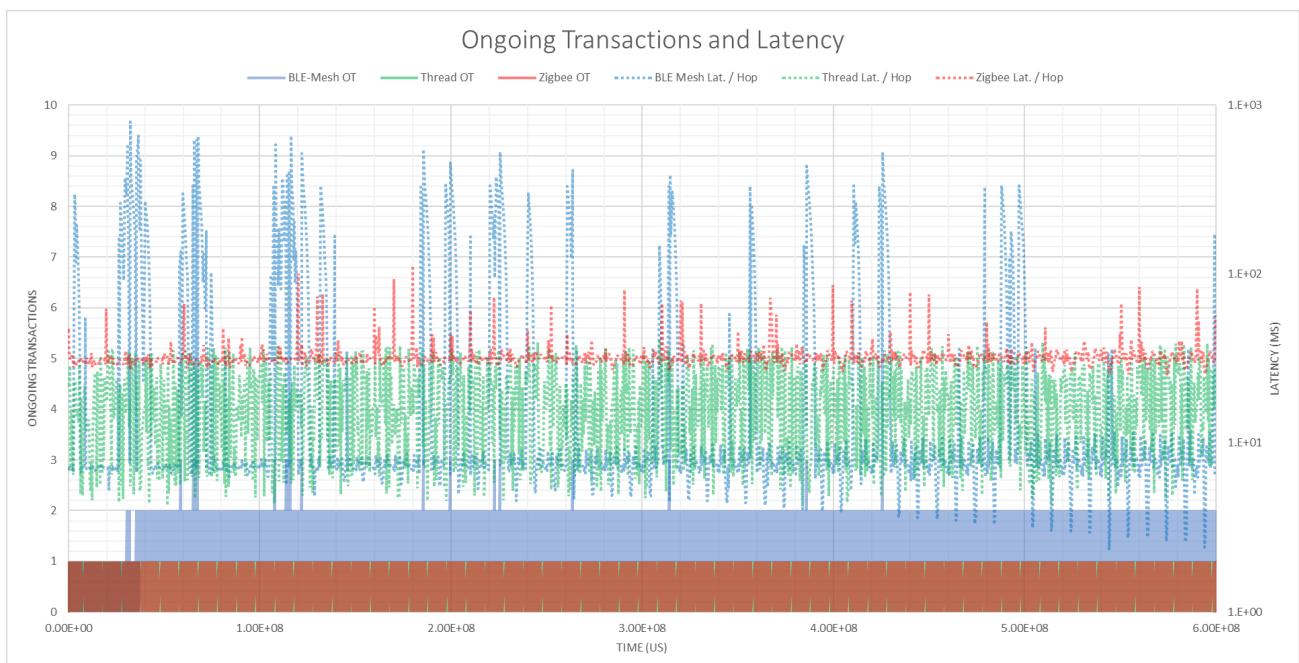
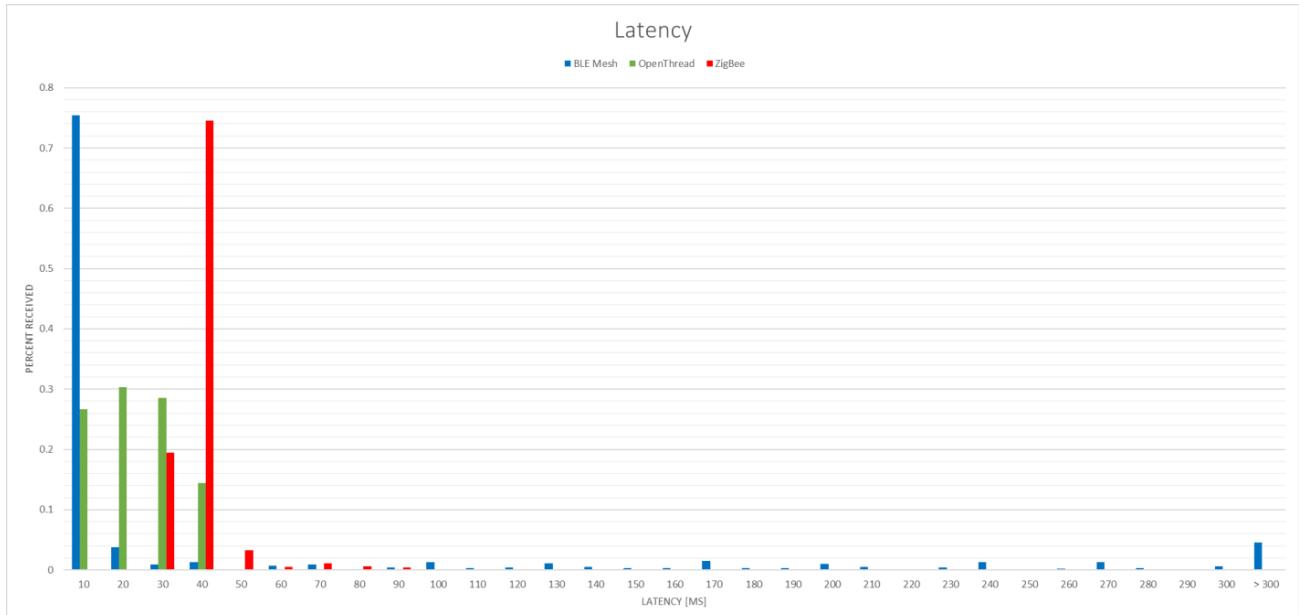
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
4	Seq	600s	60	Large	No	Labor

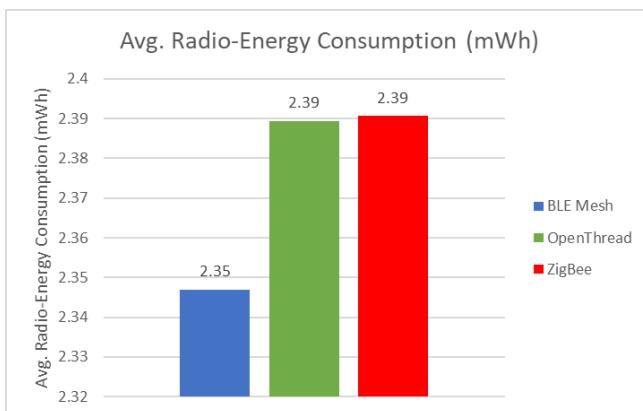
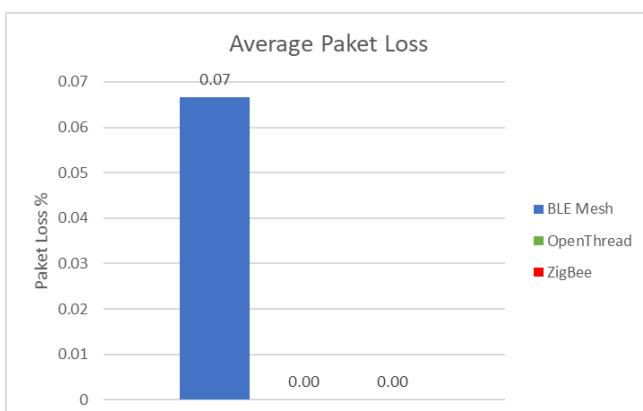
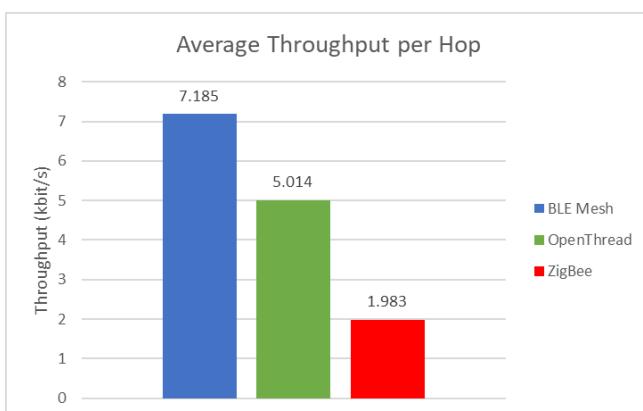
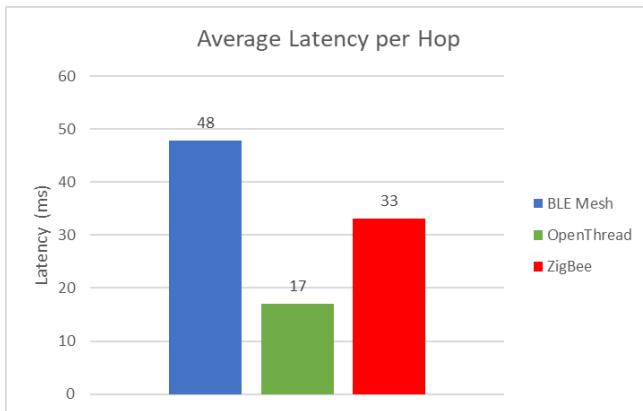




Auswertung Mesh Benchmark

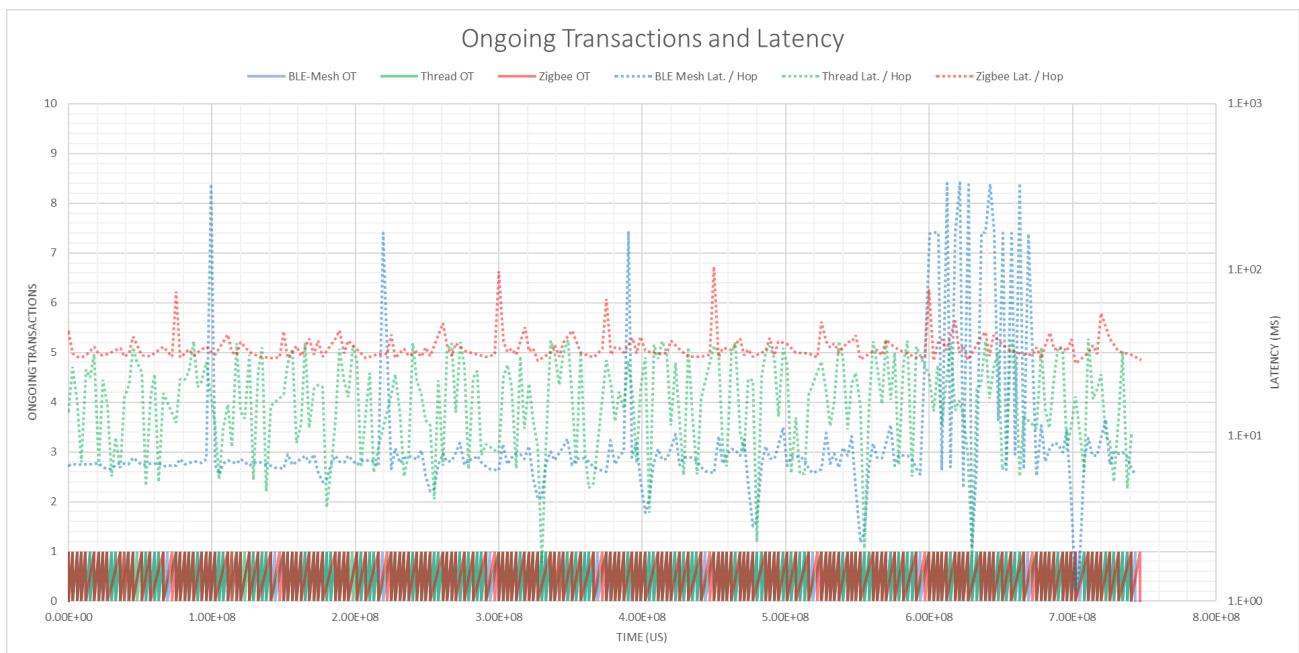
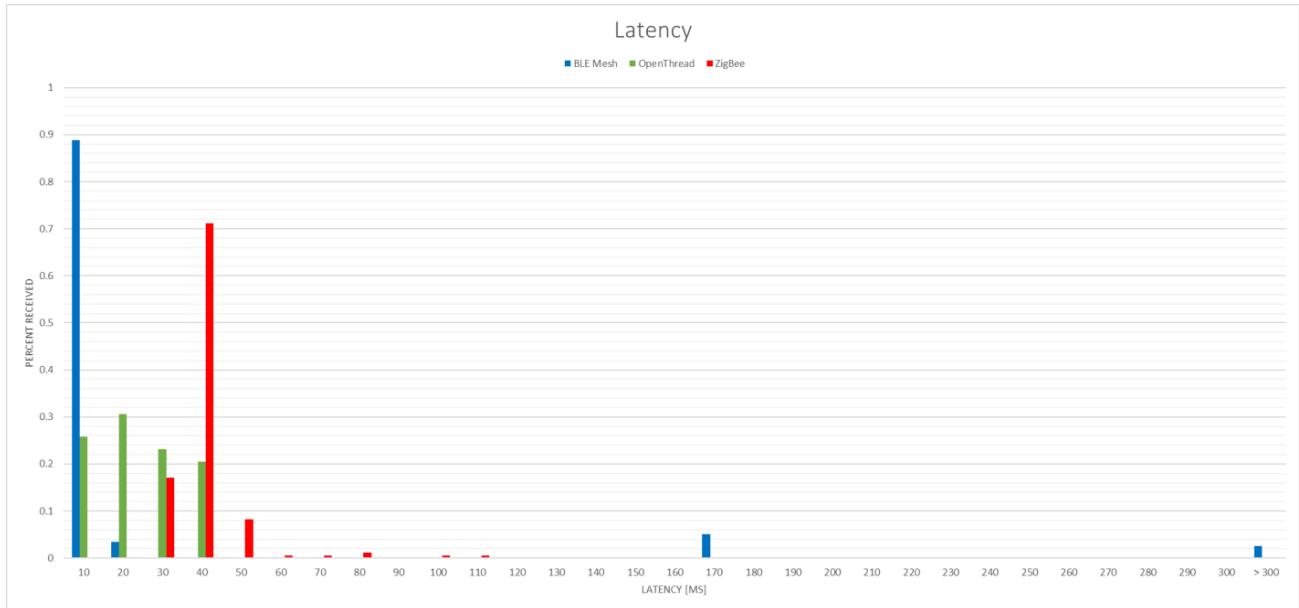
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
6	Seq	600s	60	Small	Yes	Labor

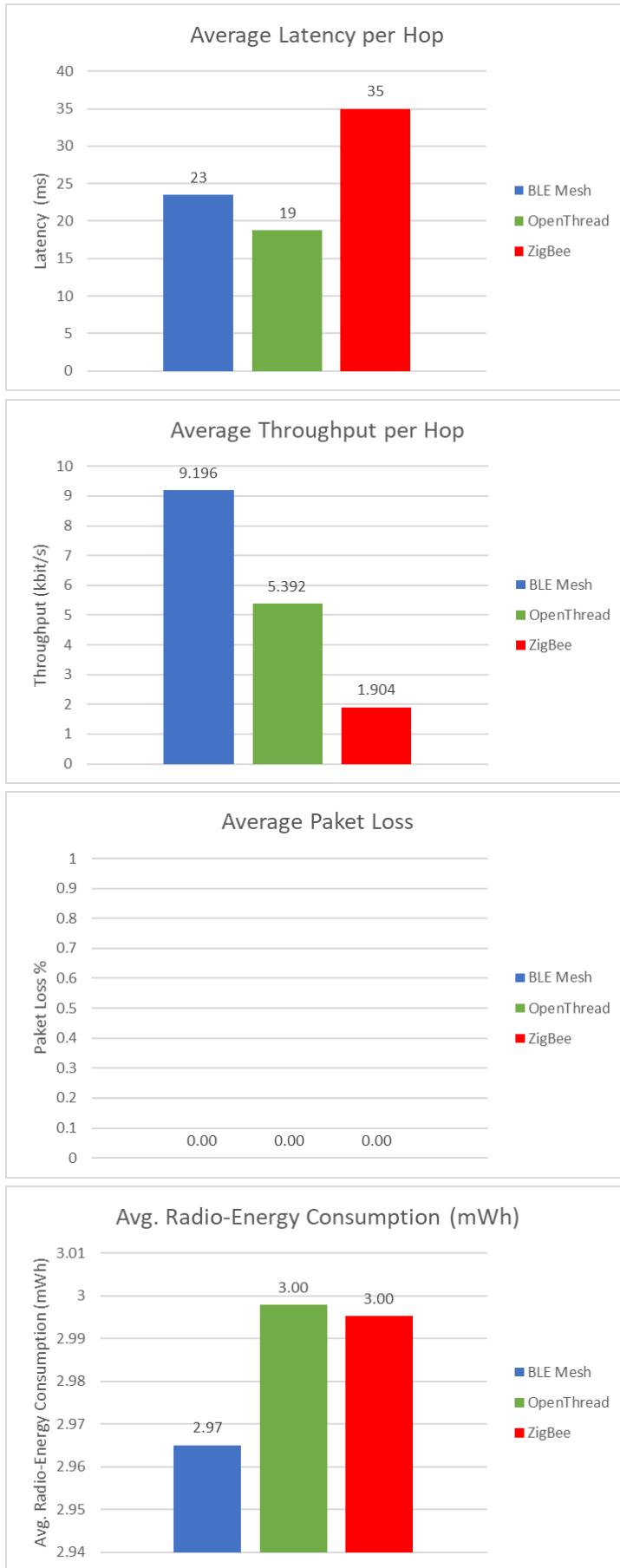




Auswertung Mesh Benchmark

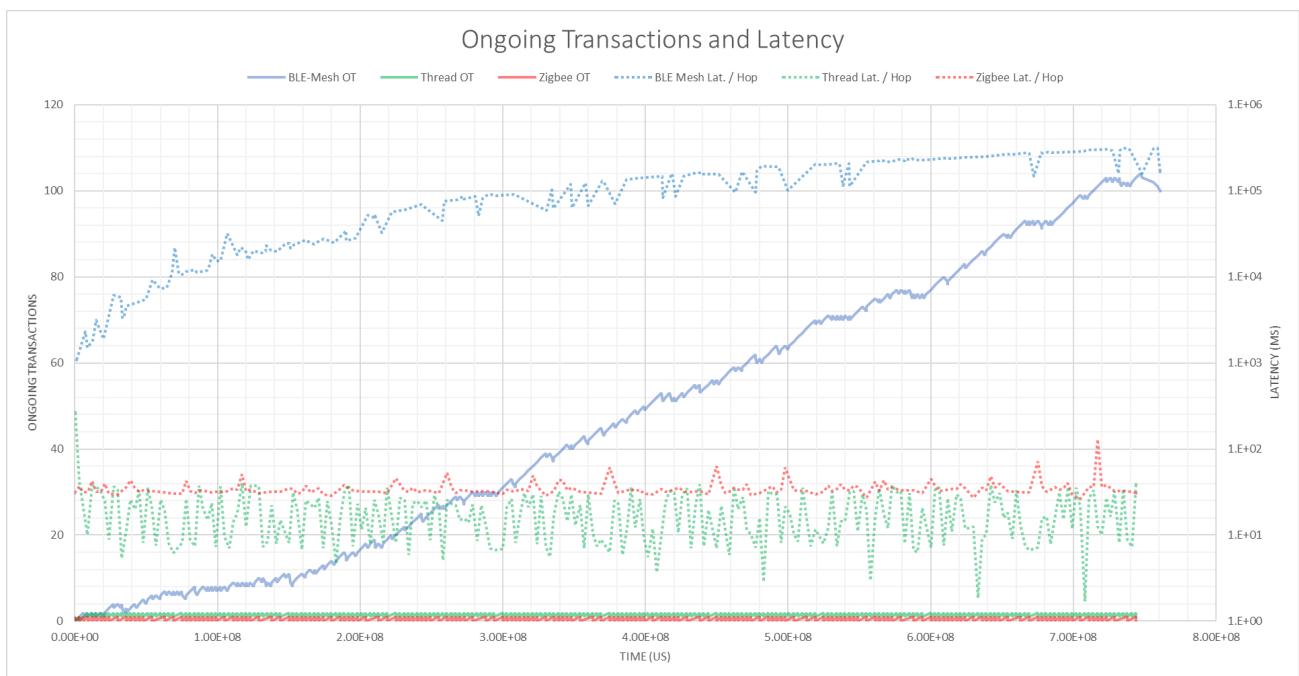
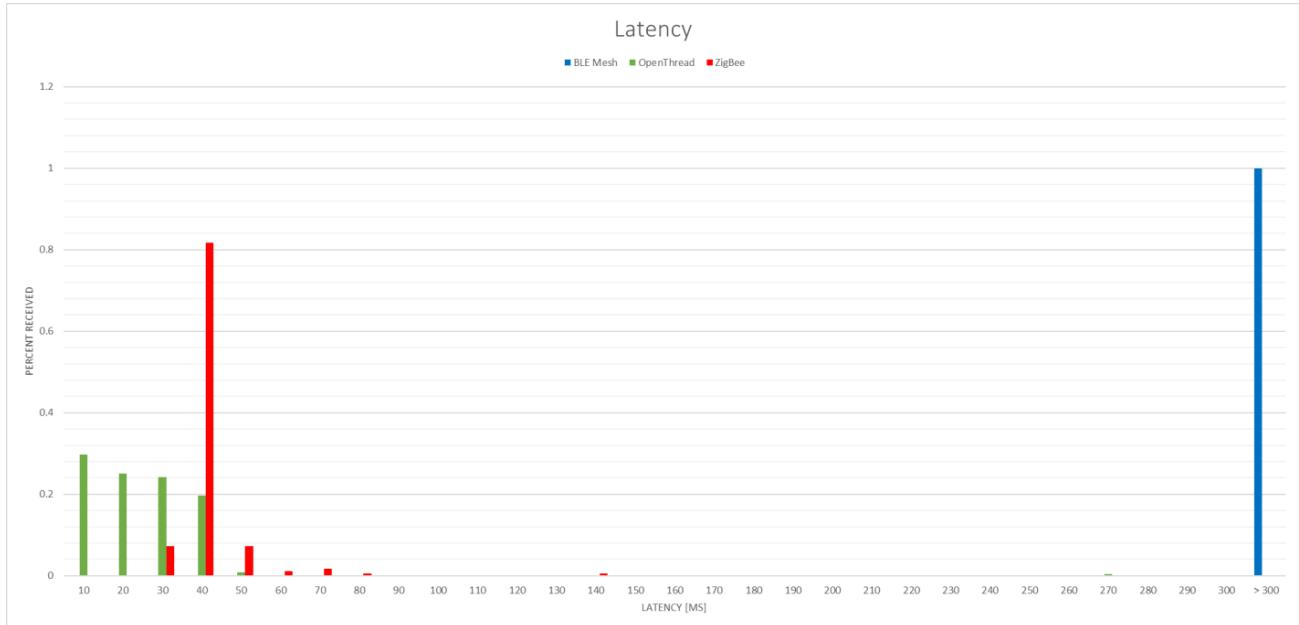
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
7	Seq	750s	10	Small	No	Labor

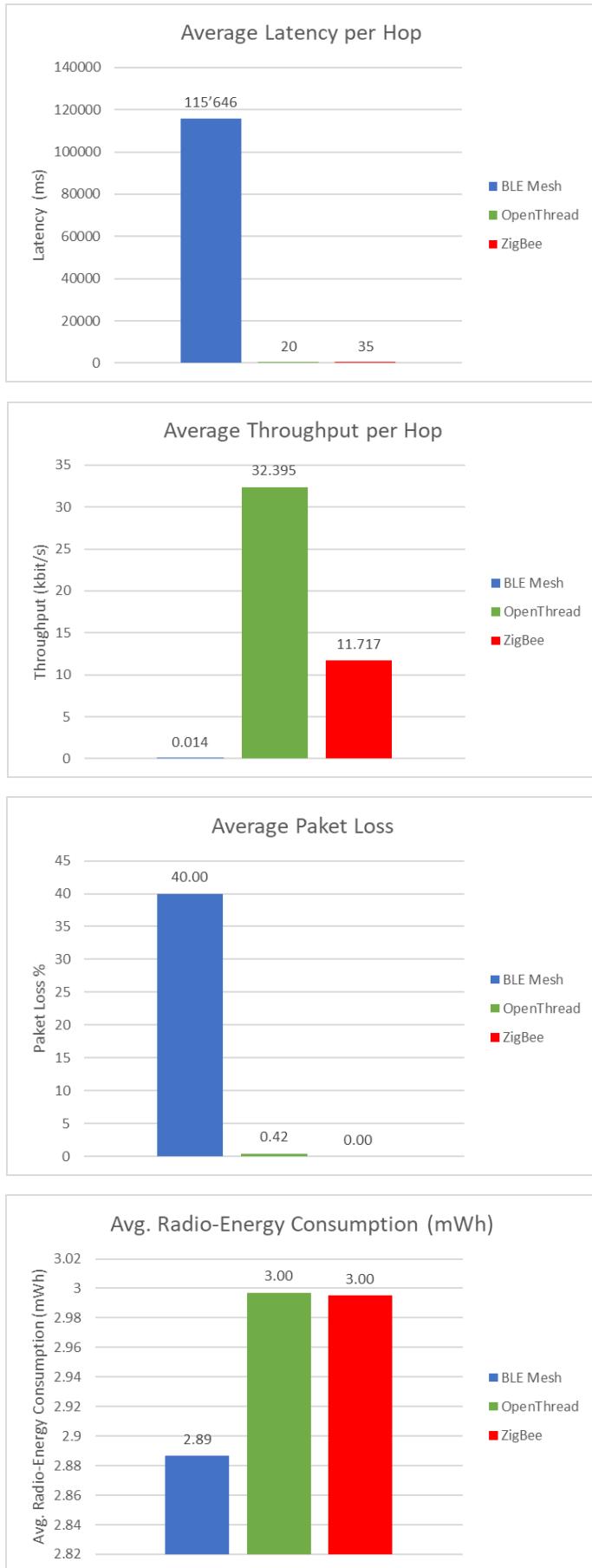




Auswertung Mesh Benchmark

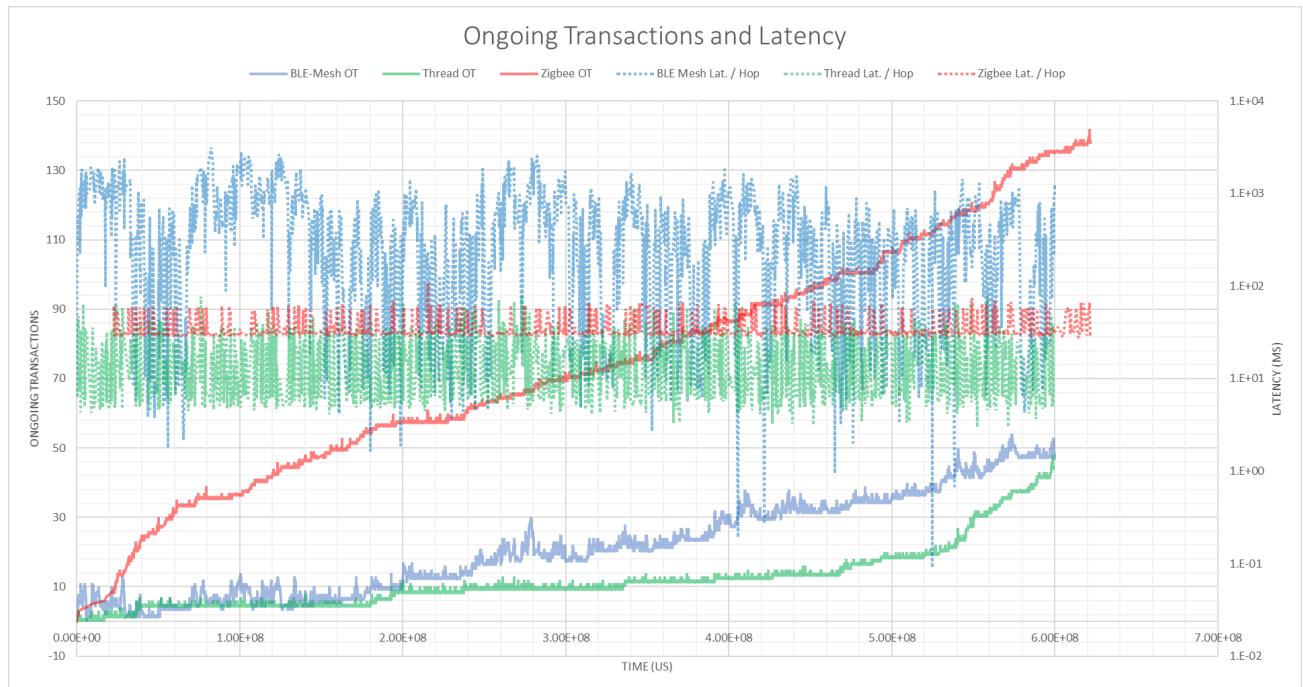
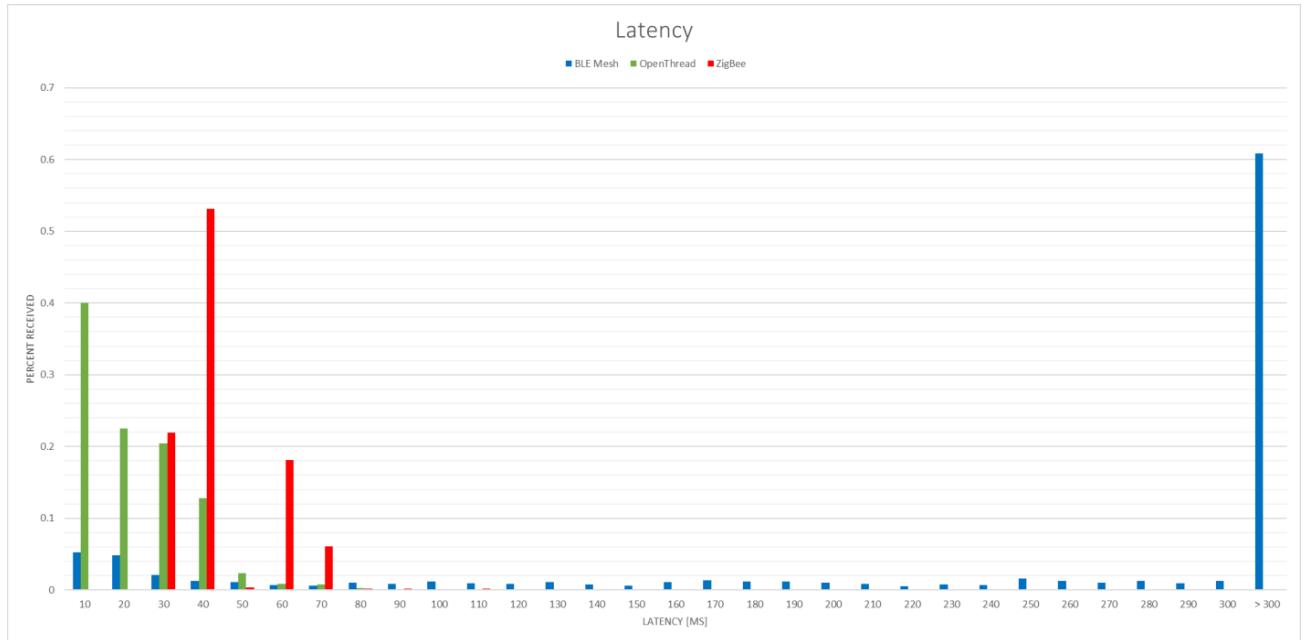
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
8	Seq	750s	10	Large	No	Labor

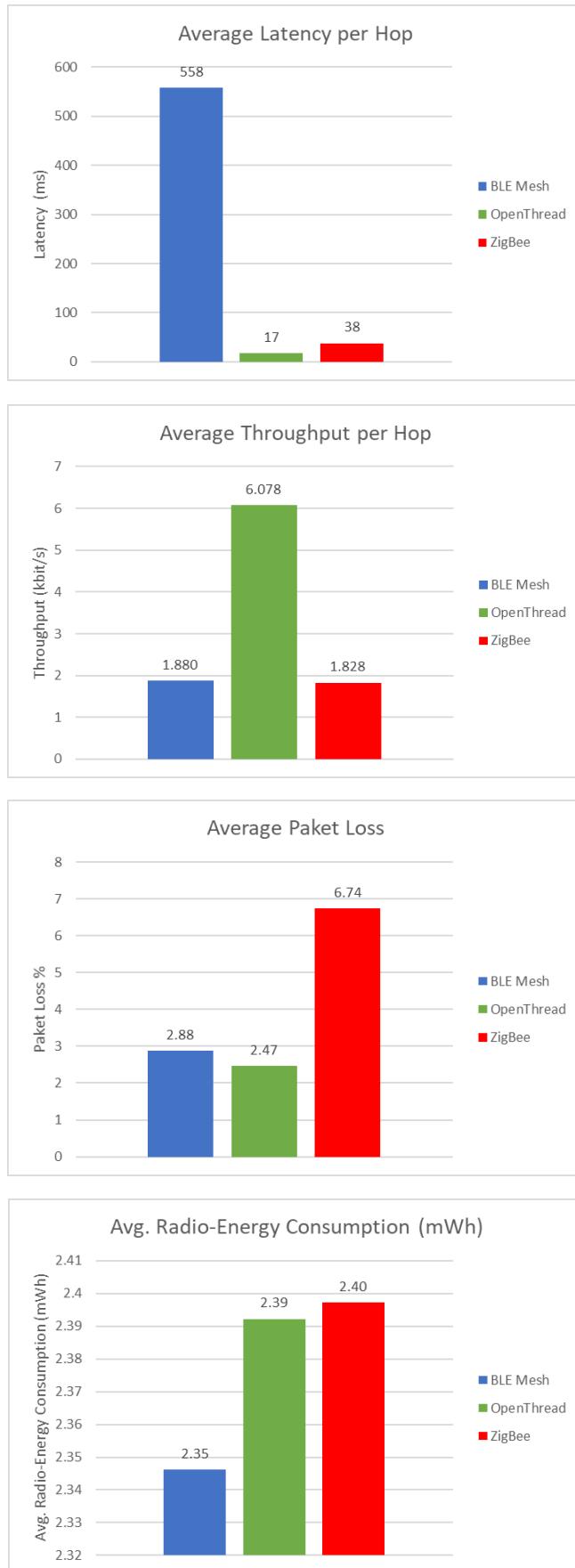




Auswertung Mesh Benchmark

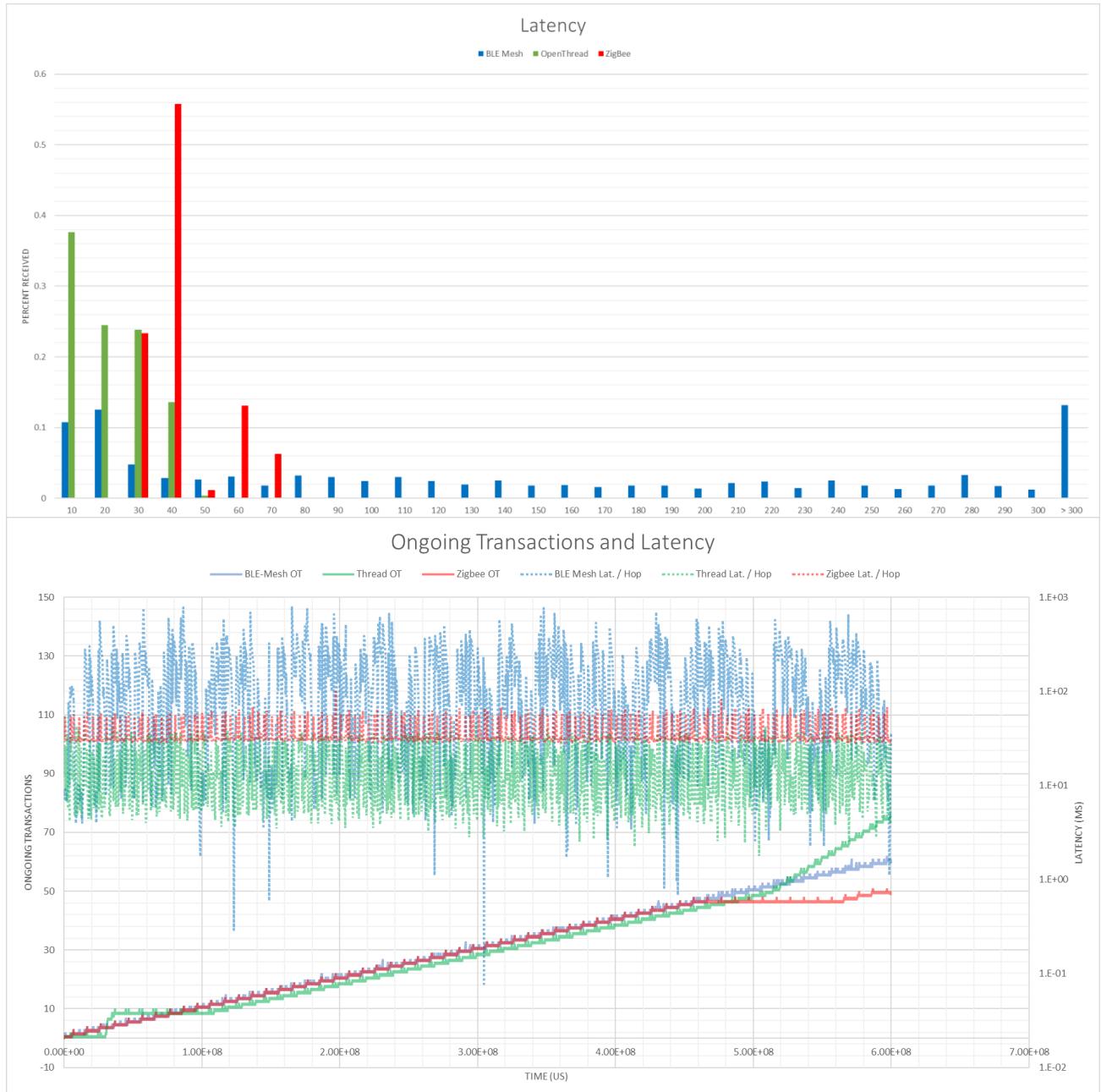
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
1	Rand	600s	60	Small	No	Haus

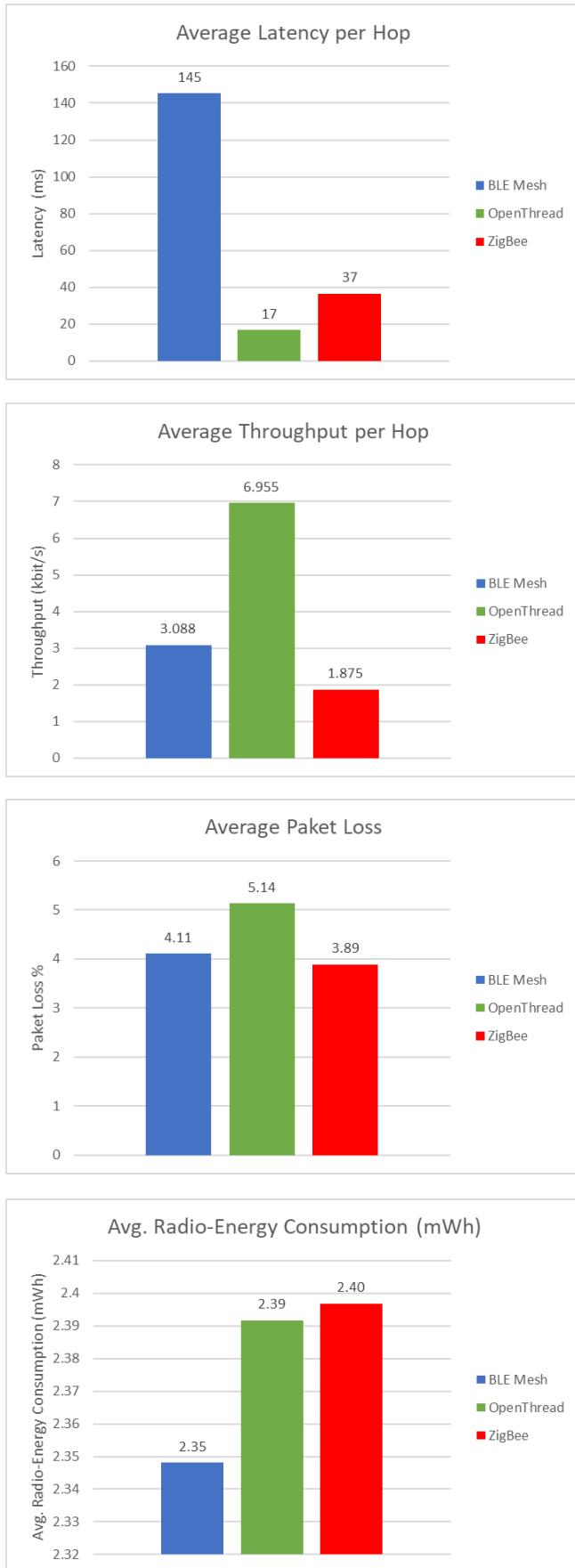




Auswertung Mesh Benchmark

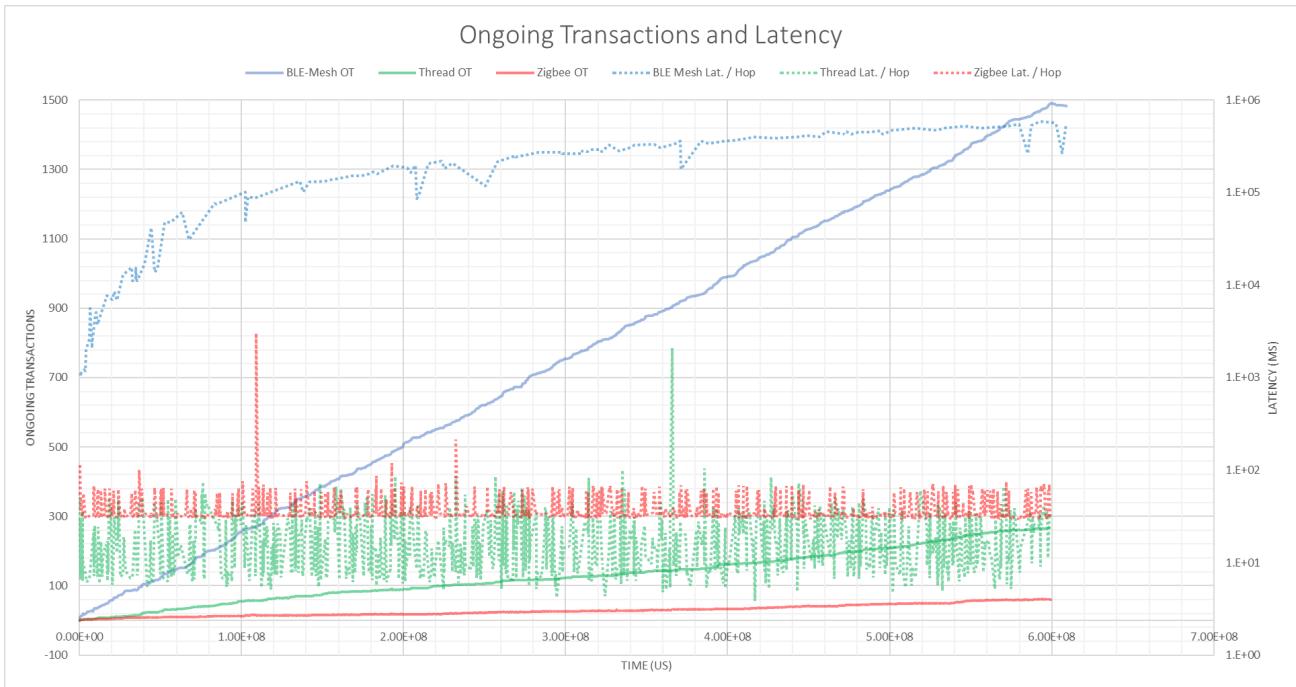
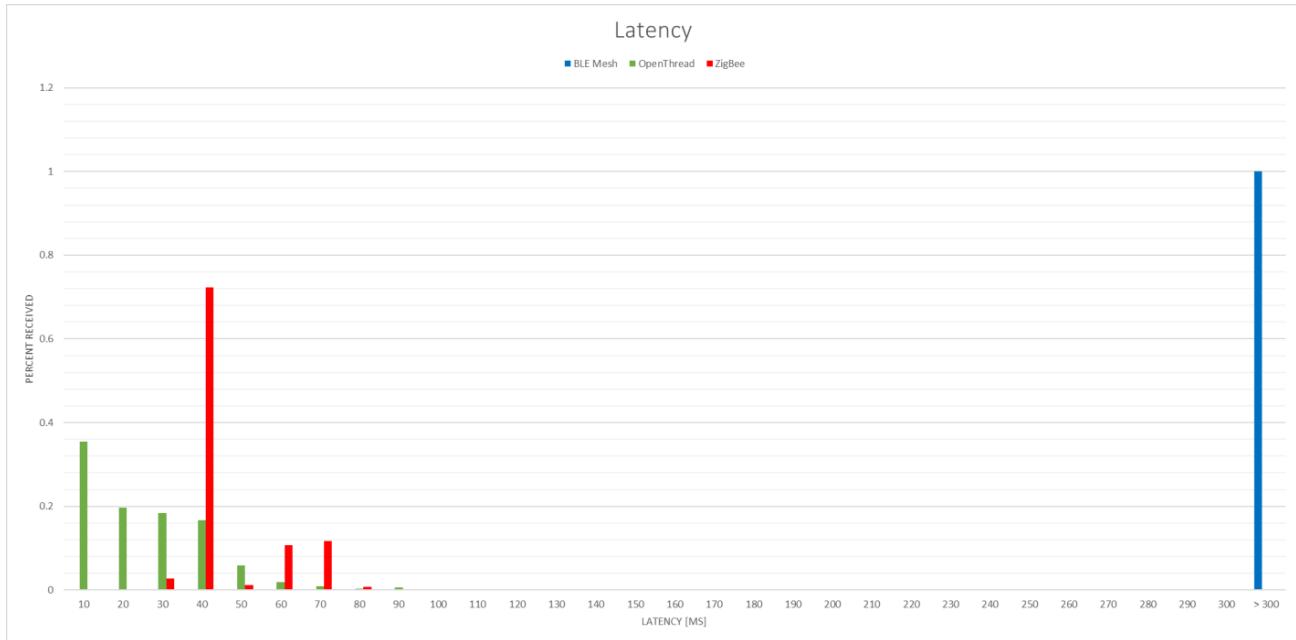
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
2	Sequentiell	600s	60	Small	No	Haus

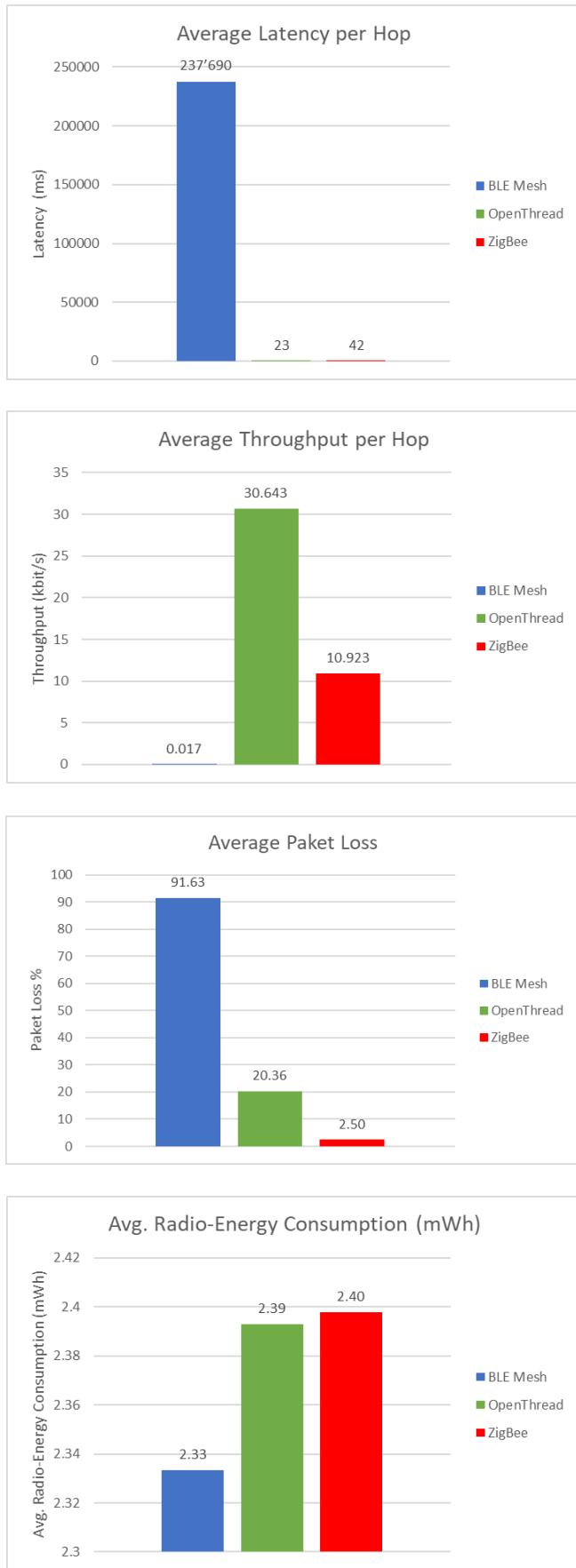




Auswertung Mesh Benchmark

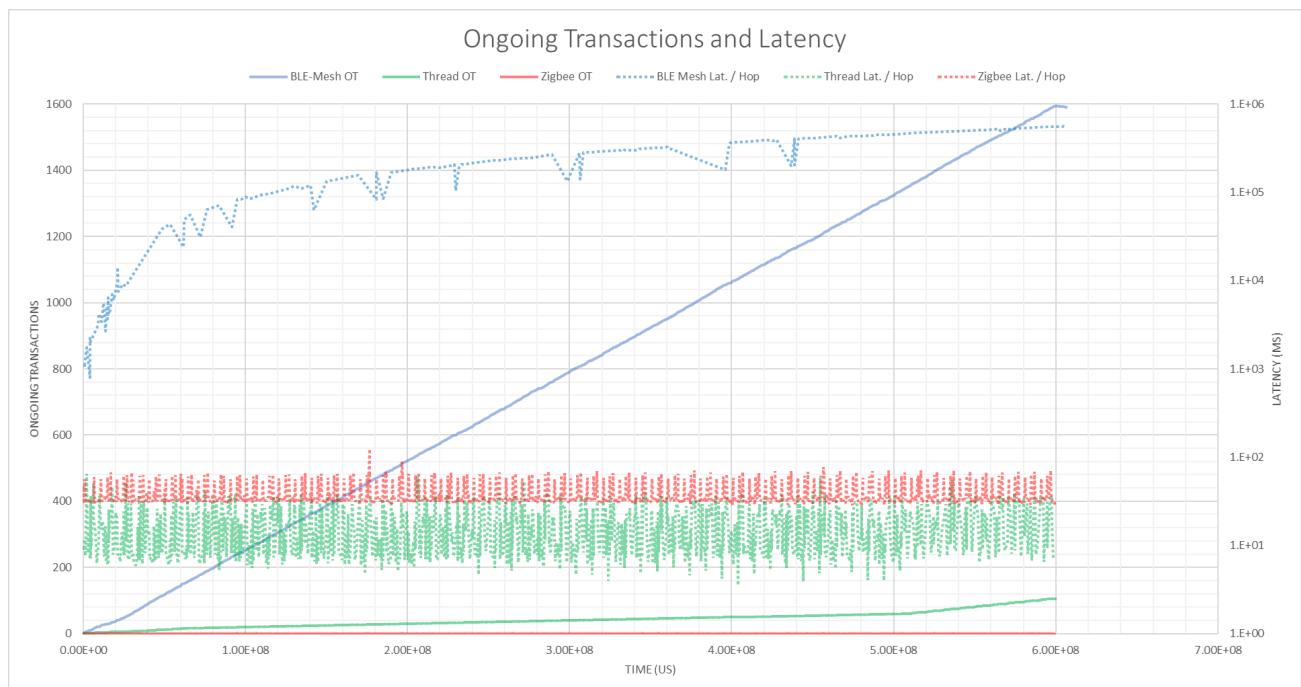
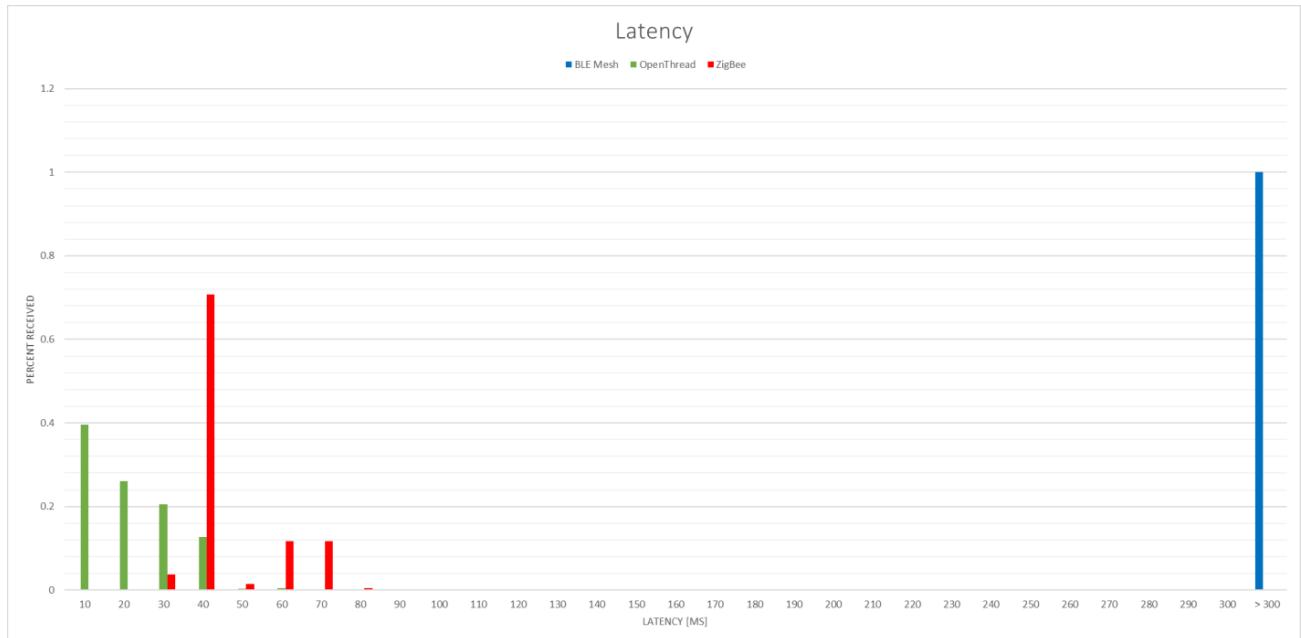
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
3	Rand	600s	60	Large	No	Haus





Auswertung Mesh Benchmark

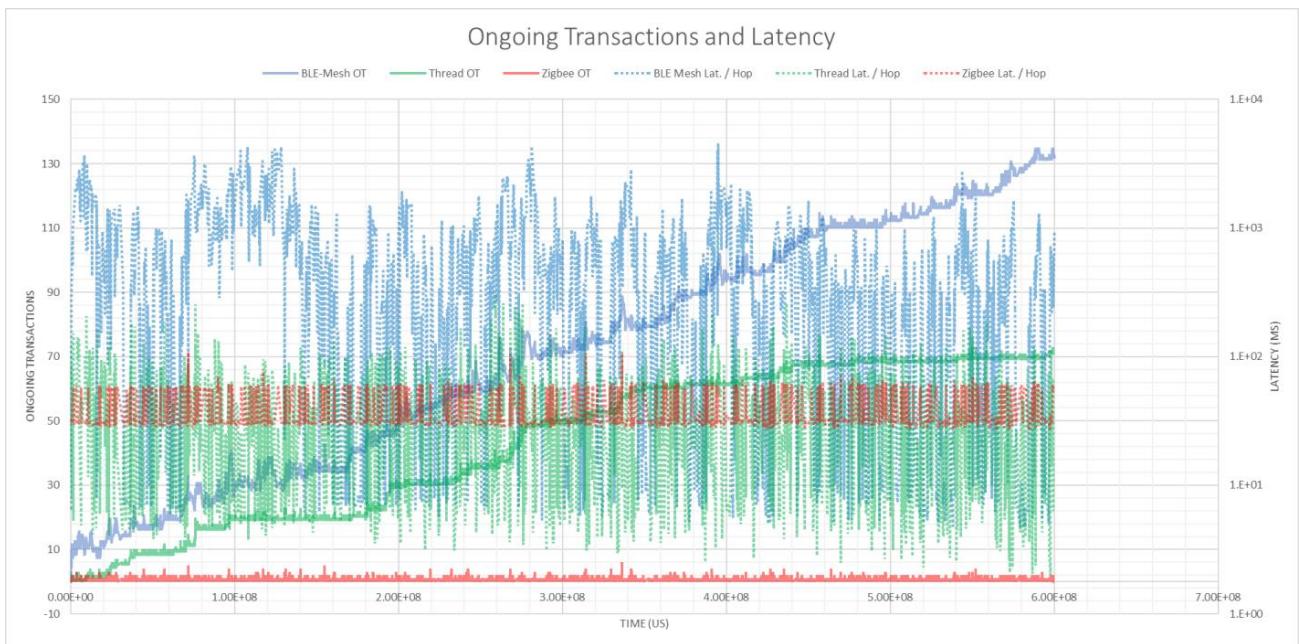
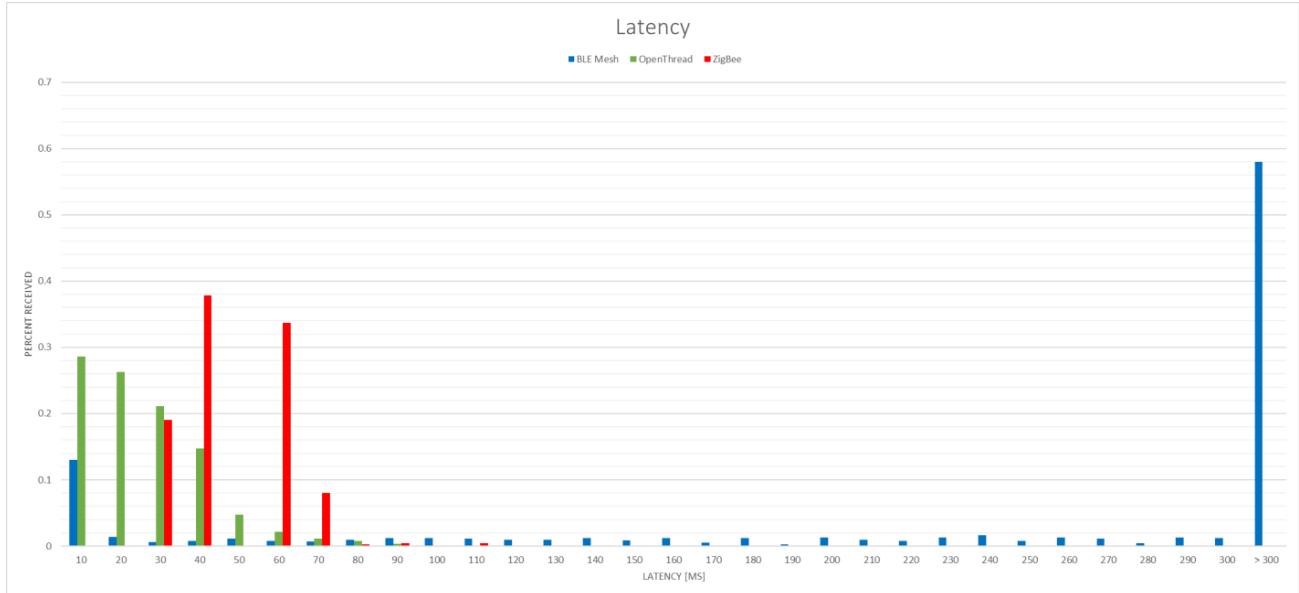
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
4	Seq	600s	60	Large	No	Haus

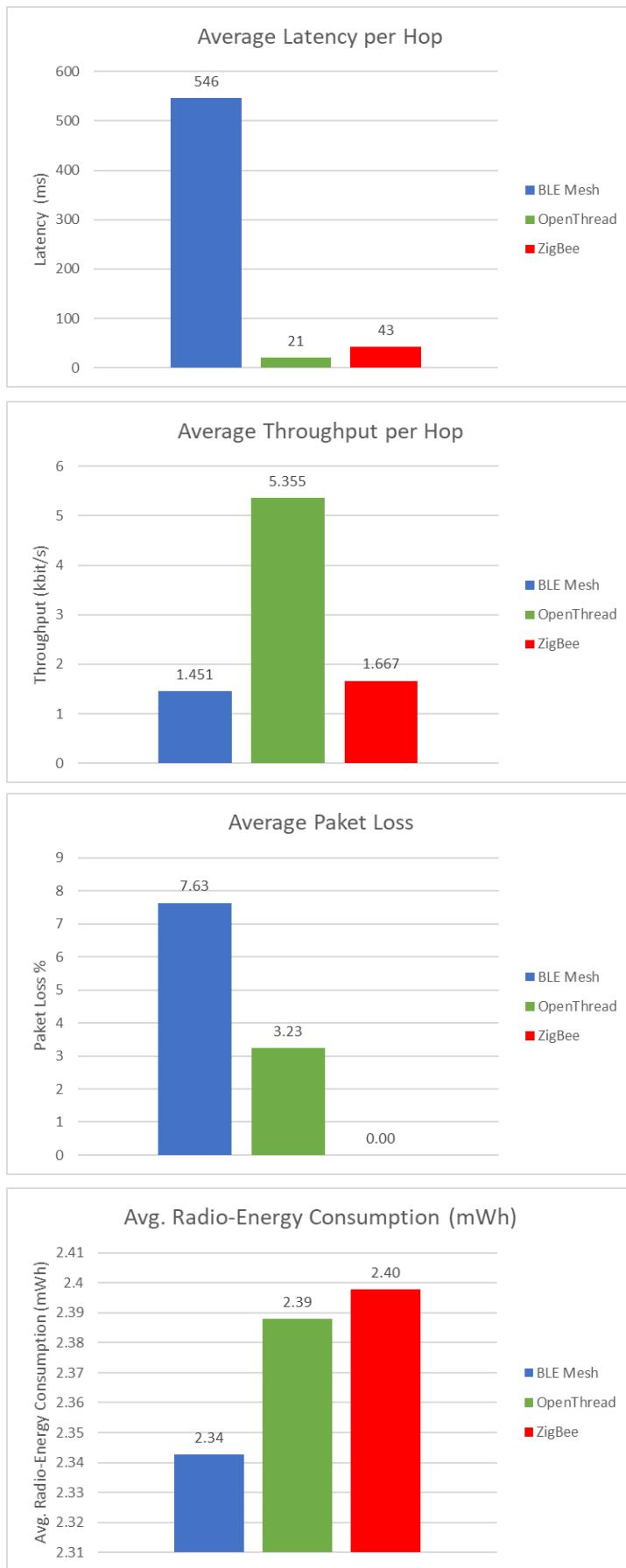




Auswertung Mesh Benchmark

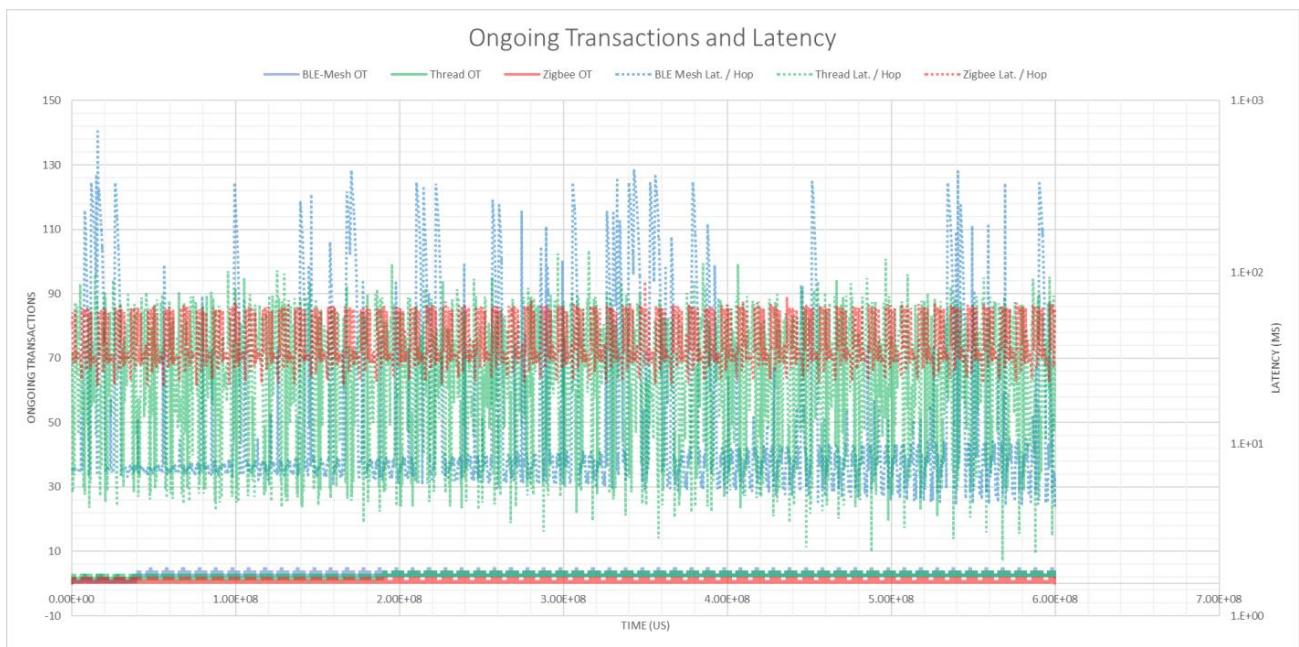
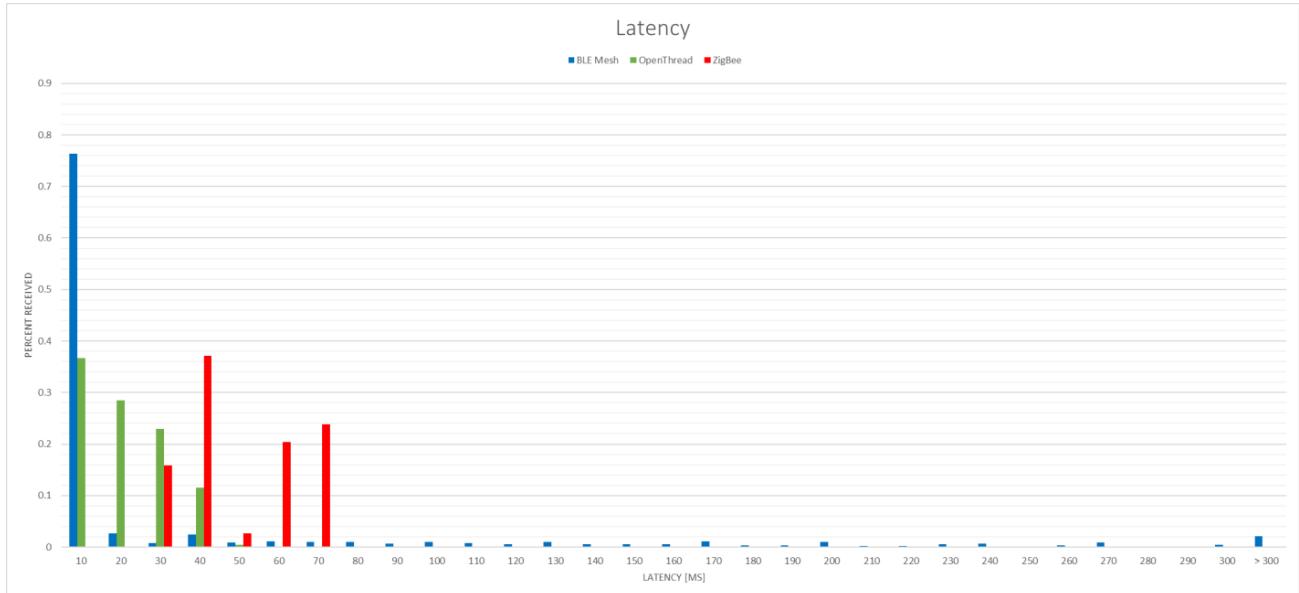
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
1	Rand	600s	60	Small	No	Wohnung

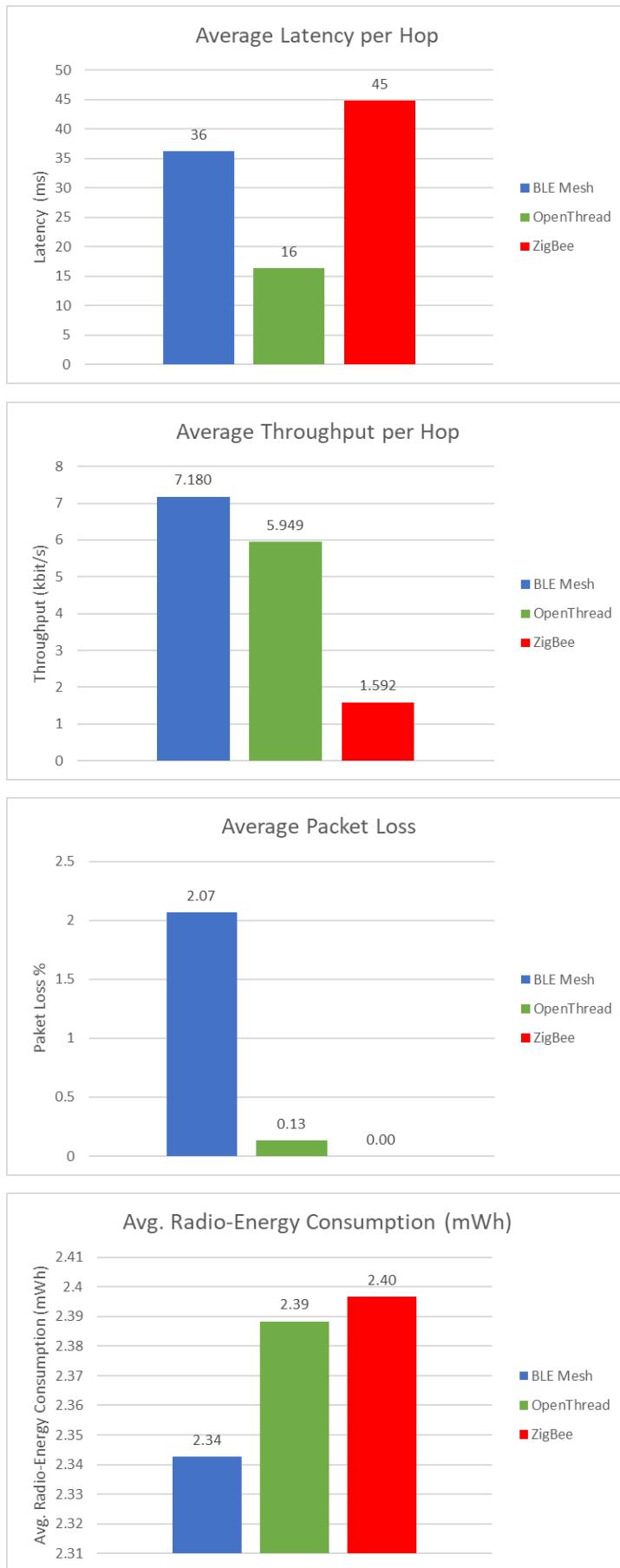




Auswertung Mesh Benchmark

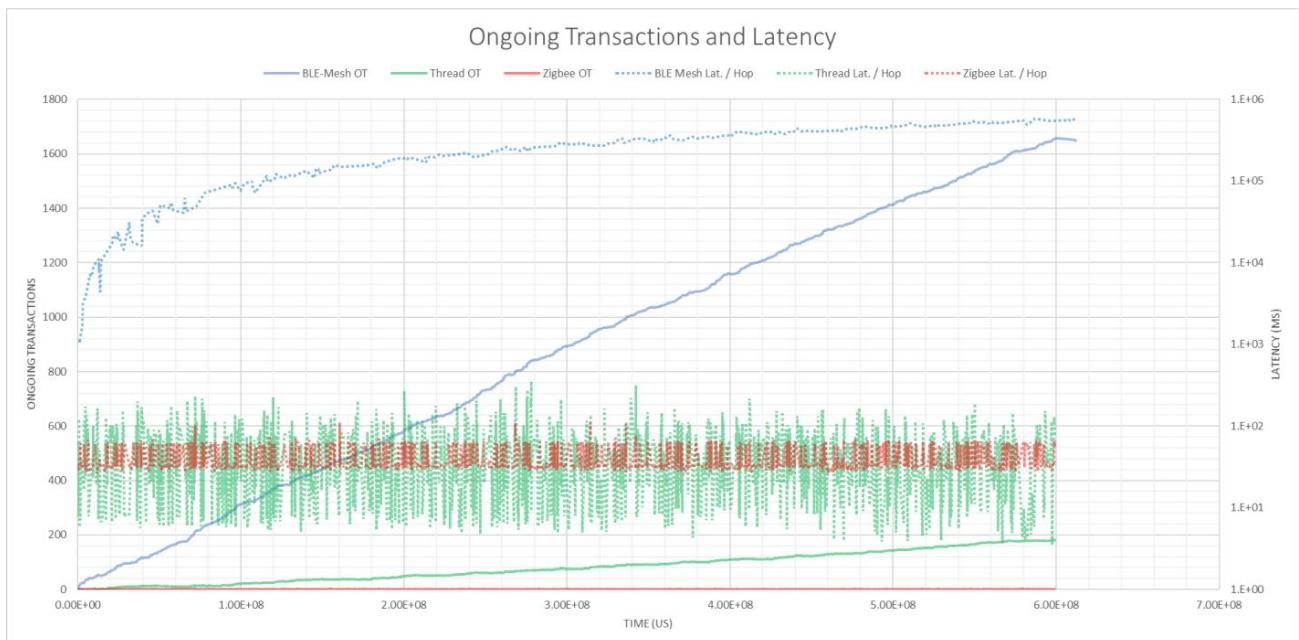
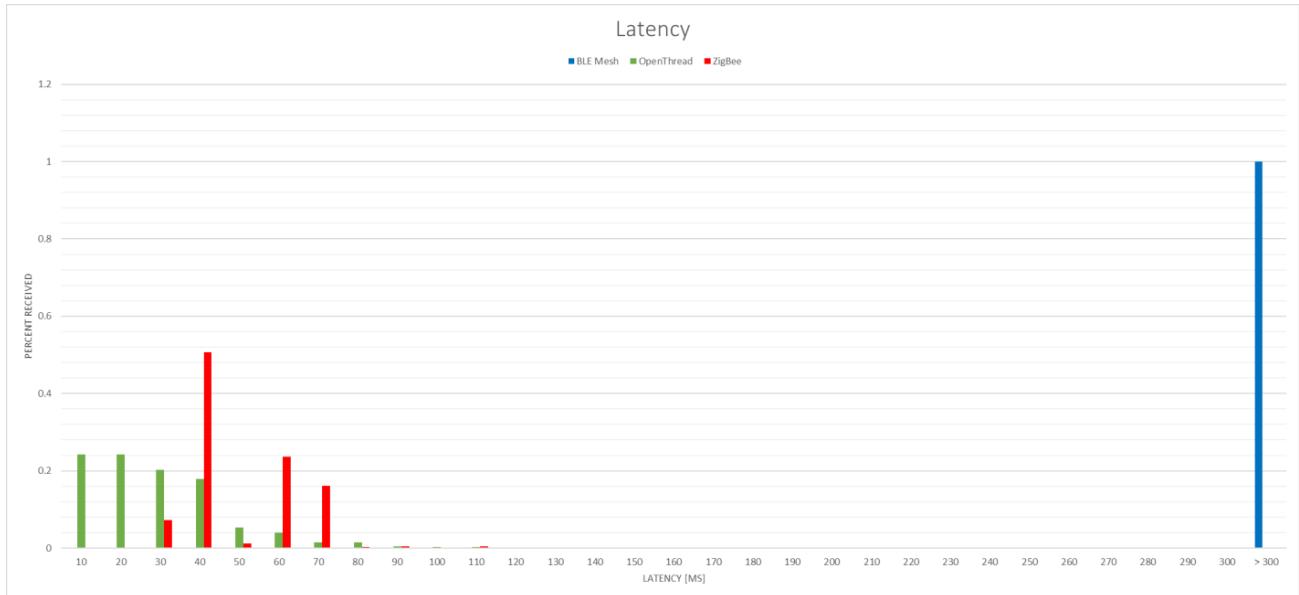
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
2	Seq	600s	60	Small	No	Wohnung

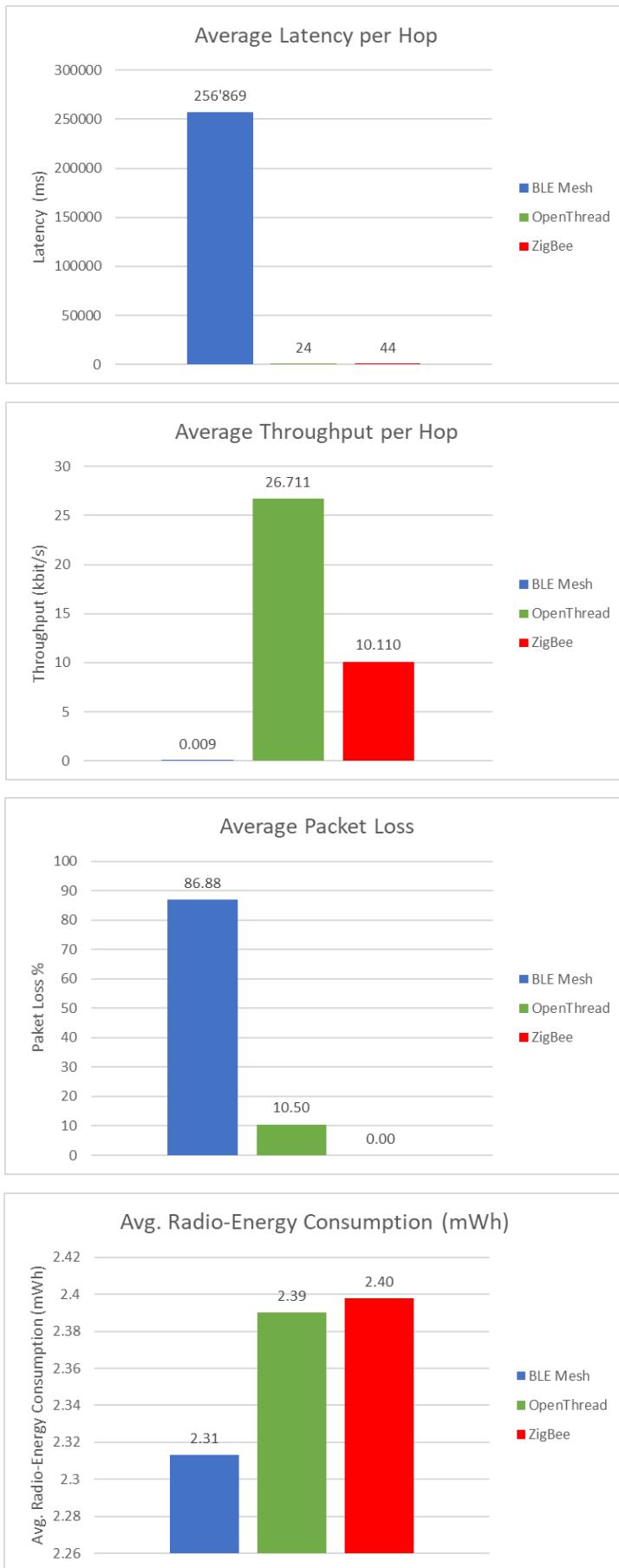




Auswertung Mesh Benchmark

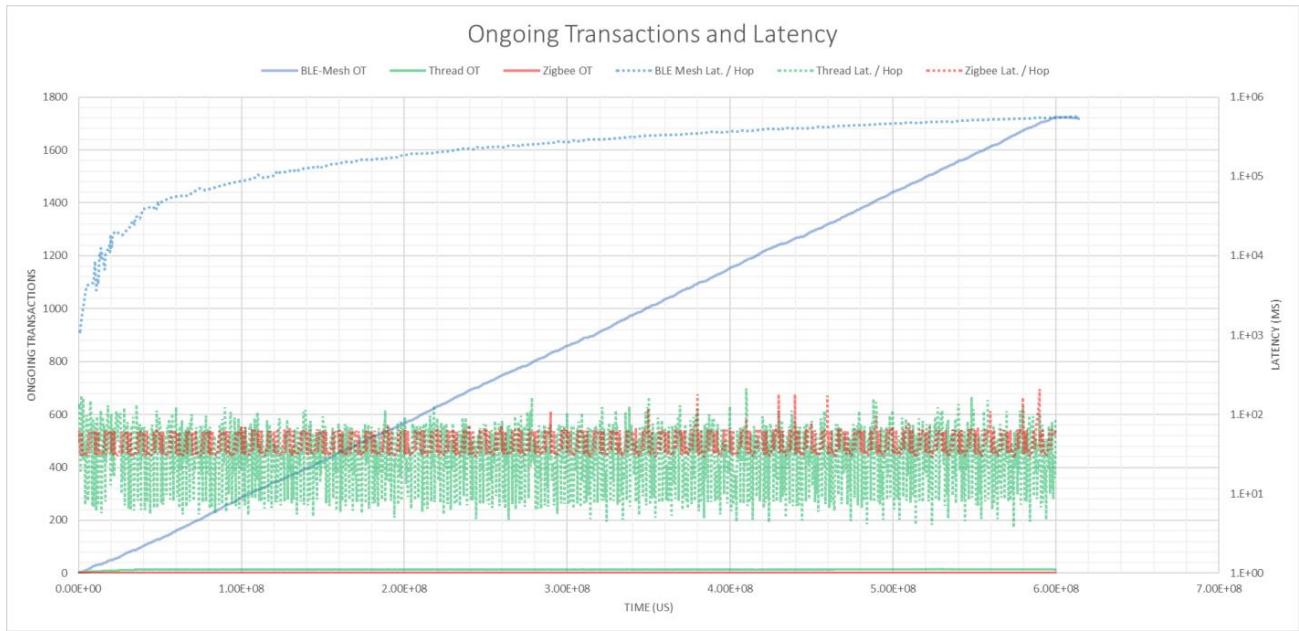
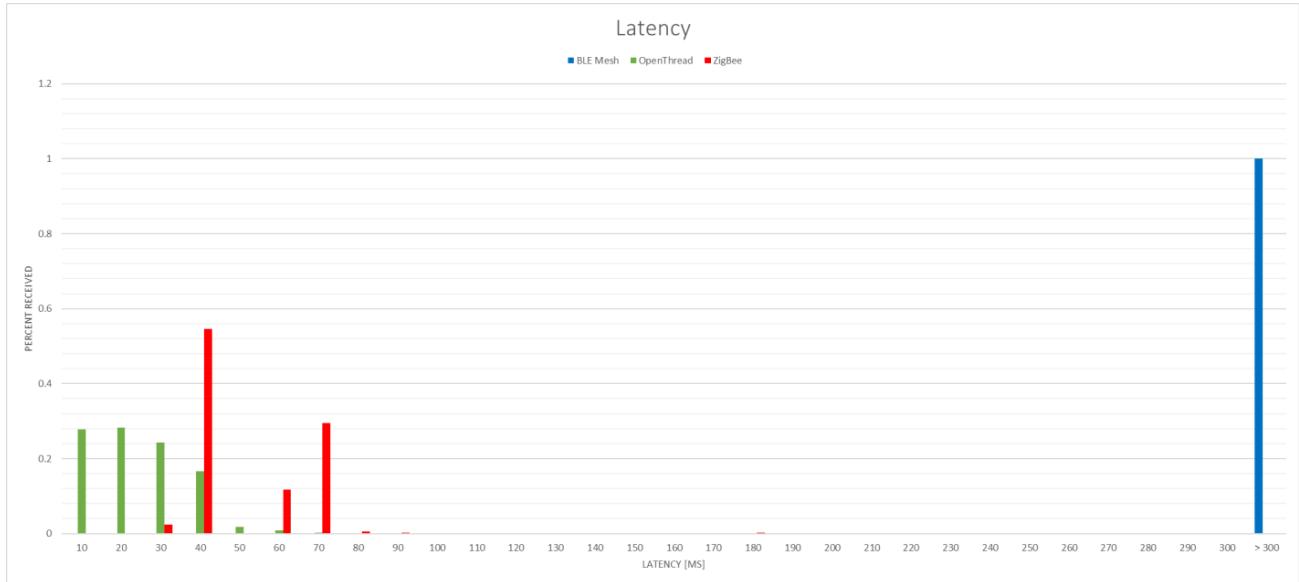
Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
3	Rand	600s	60	Large	No	Wohnung

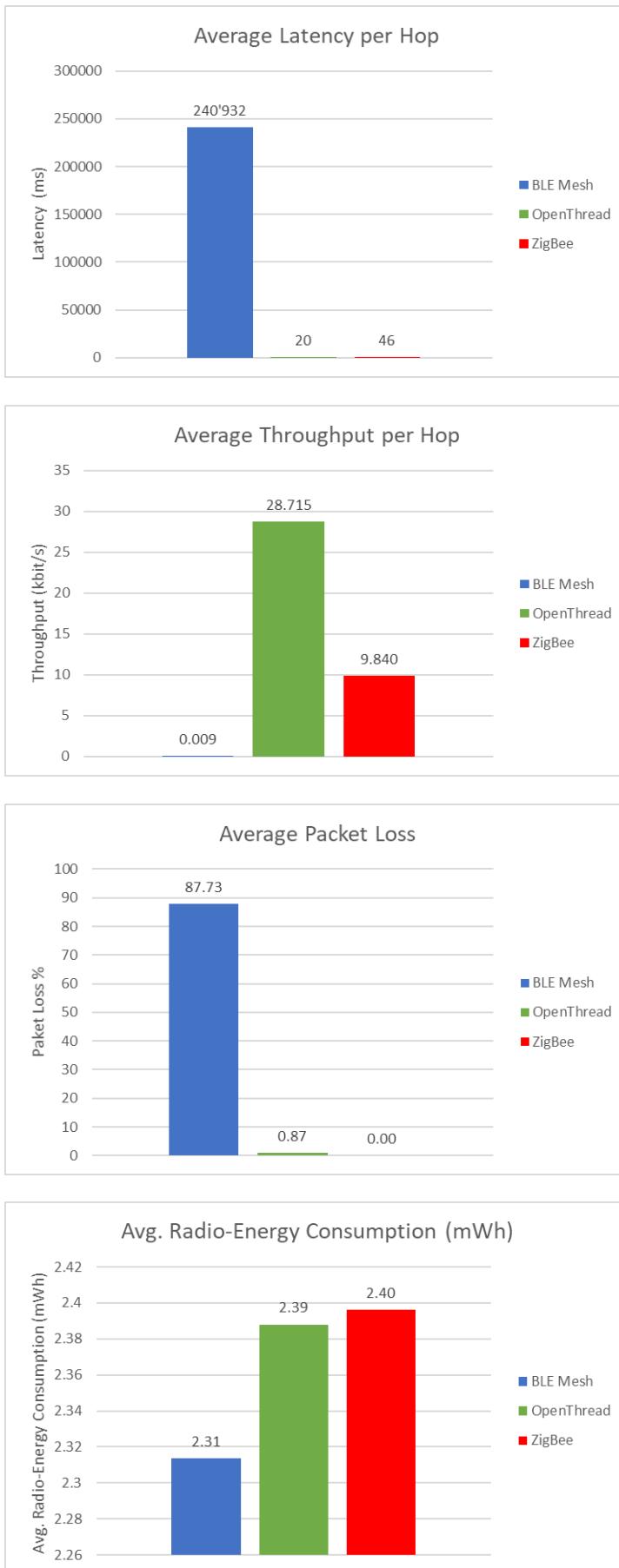




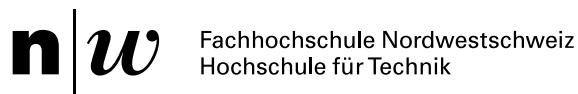
Auswertung Mesh Benchmark

Index Messung	Msg. Gen	Duration	Msg. Cnt	Payload	Disturbance	Messaufbau
4	Seq	600s	60	Large	No	Wohnung





E Paper



Fachhochschule Nordwestschweiz
Hochschule für Technik

FHNW PAPER

Perfomancevergleich von Zigbee, Thread und Bluetooth Mesh Netzwerken

Cyrill Horath¹ | Raffael Anklin¹ | Robin Bobst¹

¹Institut für ??, Fachhochschule
Nordwestschweiz, Windisch, Aargau, 5210,
Schweiz

[Abstract hinzufügen](#)

KEY WORDS

keyword 1, keyword 2, keyword 3, keyword 4, keyword 5, keyword 6,
keyword 7

Correspondence

Team Blau, Institut für ??, Fachhochschule
Nordwestschweiz, Windisch, Aargau, 5210,
Schweiz

Email: TeamBlau@email.com

Funding information

Abbreviations: ABC, a black cat; DEF, doesn't ever fret; GHI, goes home immediately.

1 | EINLEITUNG

In der Einleitung sollen die drei verschiedenen Stacks kurz und knapp erläutert werden und welche Vor- und Nachteile diese haben.

Im 2.4GHz ISM-Band konkurrieren sich derzeit die drei weit verbreiteten Low Power Mesh Netzwerk Protokolle Bluetooth Mesh, Thread und Zigbee. Alle drei wurden konzipiert für die kabellose Übertragung in sogenannten WSN (Wireless Sensor Networks) oder in Netzen für die Heim Automatisierung. Während Thread und Zigbee den IEEE 802.15.4 Standard als Physical Layer benutzen basiert der BT Mesh Stack auf dem BLE (Bluetooth Low Energy) Standard. Aufgrund der hohen Dichte an Netzwerkprotokollen die das 2.4GHz ISM-Band ebenso nutzen (z.B. Wifi) sind die Störeinflüsse auf die Mesh Protokolle eines der grössten Probleme. Die Protokollstacks begegnen diesem und weiteren Problemen auf unterschiedliche Weise. Diese Unterschiede und schliesslich die Performance der Mesh Netzwerke sollen unter unterschiedlichen Testbedingungen aufgezeigt werden wodurch ein objektiver Vergleich der drei Mesh Protokolle möglich wird.

Mesh Netzwerke im Vergleich			
	Bluetooth Mesh	Thread	ZigBee
Markt	Beleuchtung und Smart Home	Industrie und Smart Home	Beleuchtung, Haus Automation und Messtechnik
Veröffentlicht	2017	2015	2003
Applikations Layer	Mesh Model System	Verknüpfbar mit allen IPv6 basierten Protokollen	Cluster Bibliothek
IPv6	Nein	Ja	Nein
Netzwerk Zugriff	Smartphone oder Gateway	Border Router	Gateway
Ökosysteme	Ledvance	Google Nest	Ikea, Phillips Hue, Amazon und weitere
Routing	Managed Flooding	Geroutet	Geroutet
Weiteres	Ist direkt mit Smartphone erreichbar	Automatisiertes Verwalten des Netzwerks	Am meisten verbreitet

T A B L E 1 Vergleich Mesh Netzwerke

2 | METHODE

Um die Performance der drei Mesh Stacks zu vergleichen wurde ein einheitliches Benchmark Konzept erarbeitet. Dieses definiert die Mesh Parameter, Testumgebungen, den Ablauf sowie sämtliche Messgrößen und Messreihen.

2.1 | Messablauf

Für den Vergleich der 3 Mesh Netzwerkstacks Bluetooth Mesh (BT Mesh), Thread und Zigbee wird ein vom Mesh Protokoll unabhängiges Testkonzept umgesetzt welches in der Abbildung 1 als Konzeptschema dargestellt ist. Die Benchmark Slave Nodes (BSN) in der Abbildung als Sensoren und Aktoren mit unterschiedlichen Funktionalitäten dargestellt, bilden zusammen mit dem Benchmark Master Node (BMN) das zu testende Mesh Netzwerk. Innerhalb des Netzwerks wird dessen Organisation vom jeweiligen Protokoll sichergestellt. Das Testnetzwerk soll ein realitätsnahe Netzwerk nachbilden. Beispielsweise wird eine Hausautomation in einem Einfamilienhaus als Referenz angenommen in welchem jeweils nur gewisse Nodes untereinander Applikationsdaten austauschen. Ein Lichtschalter kommuniziert nur mit einer Lichtquelle und umgekehrt. Der selbe Lichtschalter tauscht jedoch keine Applikationsdaten mit dem Temperatursensor aus. Trotzdem bilden die Nodes zusammen ein Mesh Netzwerk.

Die Benchmark Management Station (BMS) welche mit dem BMN via USB/UART kommuniziert, ist zuständig für die Verwaltung und Verarbeitung der Benchmarks. Während eines Benchmark Prozesses sollen sämtliche Messungen jedoch unabhängig von der BMS durchgeführt werden damit allfällige Latenzzeiten der USB/UART Verbindung die Resultate nicht verfälschen.

2.2 | Messaufbau

Unterschiedliche Testumgebungen sollen die Benchmarks und schlussendlich den Vergleich der 3 Mesh Protokolle aussagekräftiger machen. Nachfolgende Umgebungen mit den entsprechenden Eigenschaften sollen getestet werden. Die Abbildungen zu den Testumgebungen zeigen jeweils die Platzierung der Nodes sowie deren Funktion und Gruppen Zugehörigkeit. Die Farbe Grün identifiziert den Node als Client Node während Blau für einen Server Nodes steht. Die Nummerierung zeigt welcher Node zu welcher Adressgruppe gehört. Ein Client Node in Gruppe 1 sendet jeweils Nachrichten zu allen Server Nodes in der selben Gruppe.

Labor

Der Laboraufbau ist ein Extremtest welcher die Leistungsgrenzen der Protokollstacks ausloten soll. Dabei werden die Nodes auf einem Raster gemäss Abbildung ?? angeordnet. Die genauen Abmessungen sind der Abbildung zu entnehmen.

- Testaufbau unter Laborbedingungen auf engstem Raum.
- Ausgeglichene Anzahl Sensoren und Aktoren.
- Sehr Hohe Node-Dichte.
- Geringe bis keine Störbeeinflussung durch die Umgebung zu erwarten.
- Die Mesh-Beziehungen werden künstlich bestimmt sodass einfache P2P Verbindungen mit oder ohne Hop entstehen.

Einfamilienhaus

Die Testgeräte werden in einem Einfamilienhaus installiert und repräsentieren damit eine flächendeckende Heim-Automatisierung. Folgende Eigenschaften soll diese Messung abdecken:

- Einfamilienhaus über mehrere Etagen.
- Anzahl Sensoren und Aktoren vergleichbar gross.
- Node-Dichte relativ gering.
- Kleine Beeinflussung durch Nachbarsysteme sind zu erwarten.

Die Abbildung ?? zeigt den Schnitt des Einfamilienhauses in welchem der Benchmark durchgeführt wurde.

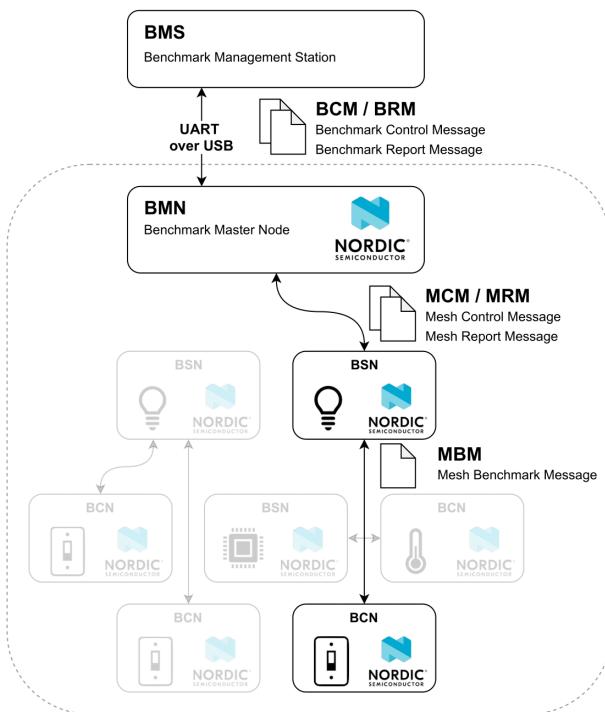


FIGURE 1 Konzeptschema Testablauf

Wohnung

Ebenfalls als Heim-Automatisierung gedacht werden die Messungen in einer Wohnung durchgeführt.

- Wohnung über eine Etage in einem Mehrfamilienhaus
- Anzahl Sensoren und Aktoren vergleichbar gross.
- Node-Dichte höher als im Haus.
- Mögliche Störeinflüsse durch andere Systeme von Nachbarn sind zu erwarten.

Bei der Wohnung handelt es sich um eine 3.5 Zimmer Wohnung mit einer Wohnfläche von 122 Quadratmetern. Die genauen Abmessungen sowie die Platzierung der Nodes ist in Abbildung 4 zu sehen.

2.3 | Messerwartung

Welche Erwartungen haben wir von den verschiedenen Stacks. (Bluetooth routet nicht daher evtl. langsamer)



FIGURE 4 Testaufbau Wohnung

3 | ERGEBNISSE

Die Ergebnisse sollen hier nach verschiedenen Kriterien dargestellt werden (Anzahl Nodes, Anzahl Hops, usw.)

4 | INTERPRETATION

Interpretation der Ergebnisse (Was fällt besonders auf, wo sind die stärken und schwächen der einzelnen Netzwerke, usw.)

5 | VALIDIERUNG

Fehlerabschätzung und Vergleich mit Benchmarks von anderen Organisationen

ERGÄNZENDE INFORMATIONEN

Infos die evtl. wichtig sind aber nicht unbedingt in den Kontext gehören

F Random Traffic Generation

30.7.2020

RANDOM.ORG - Integer Set Generator

[Home](#) [Games](#) [Numbers](#) [Lists & More](#) [Drawings](#) [Web Tools](#) [Statistics](#) [Testimonials](#) [Learn More](#) [Login](#)



Random Integer Set Generator

This form allows you to generate random sets of integers. The randomness comes from atmospheric noise, which for many purposes is better than the pseudo-random number algorithms typically used in computer programs.

Step 1: The Sets

Generate set(s) with unique random integer(s) in each.

Each integer should have a value between and (both inclusive; limits $\pm 1,000,000,000$).

The total number of integers must be no greater than 10,000.

Step 2: Display Options

Each set will be printed on a separate line. You can choose from the following extra options:

- Number the sets sequentially
- Use commas to separate the set members
- Sort the members of each set in ascending order

You can select the order in which the sets are printed:

- Print the sets in the order they were generated
- Order the sets by the values that occur in them (in this case, you should also consider sorting the members of each set)
- Print the sets in random order

Step 3: Choose Output Format

How do you want the sets to be shown?

- On a nicely formatted web page (type text/html)
- As a bare-bones text document (type text/plain)

Step 4: Choose Randomization

Do you want a new randomization or one that was prepared earlier? [[explain this](#)]

- Generate your own personal randomization right now
- Use pregenerated randomization from
- Use pregenerated randomization based on persistent identifier (max 64 alphanumeric characters)

Step 5: Go!

Be patient! It may take a little while to generate your sets...

[Get Sets](#) [Reset Form](#) [Switch to Simple Mode](#)

Note: This generator guarantees the numbers in each set will be unique within each set, but not that the sets themselves are unique amongst each other.

G Bluetooth Mesh Stack Layers

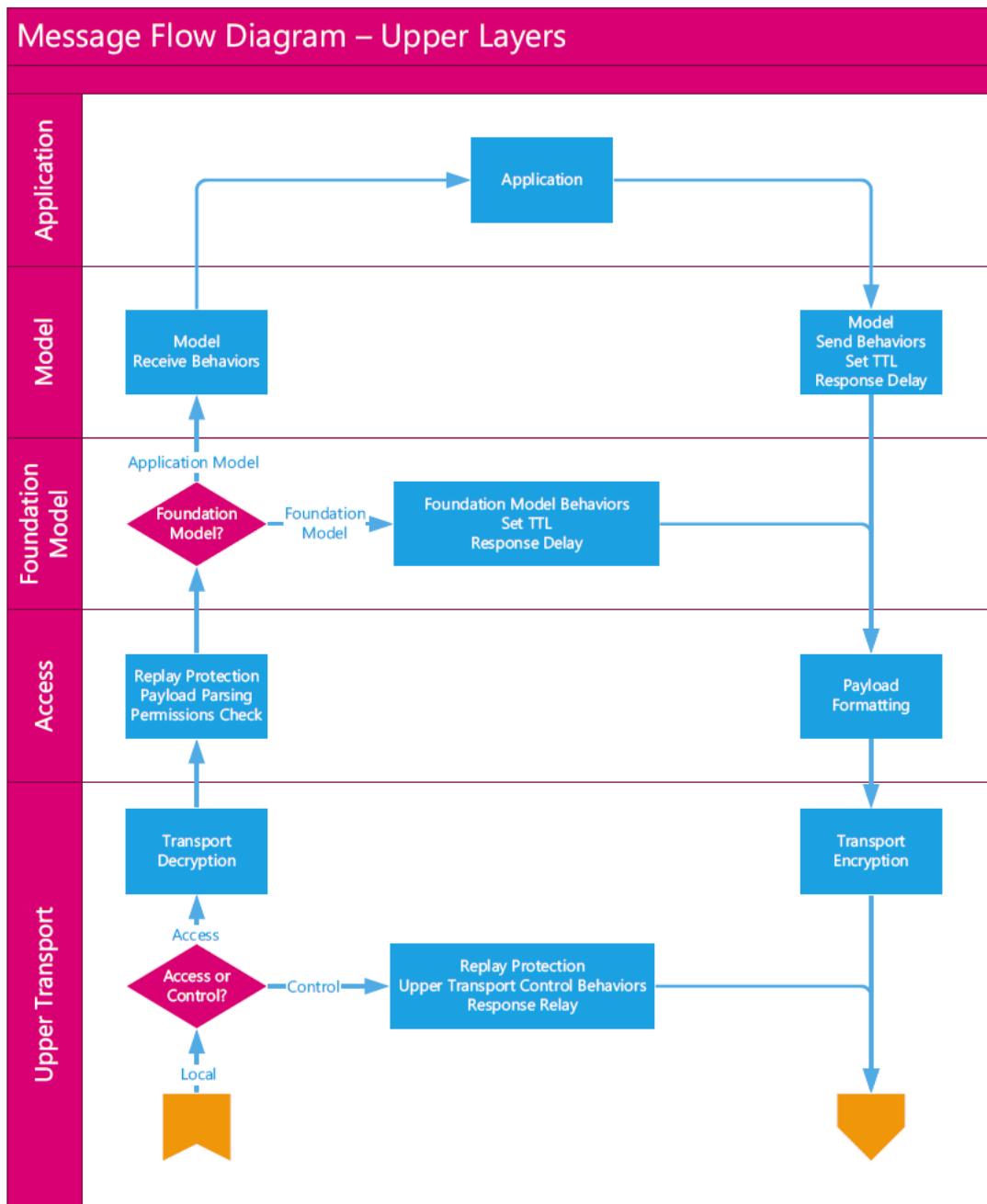


Abbildung G.1: Bluetooth-Mesh Stack Upper Layers [27]

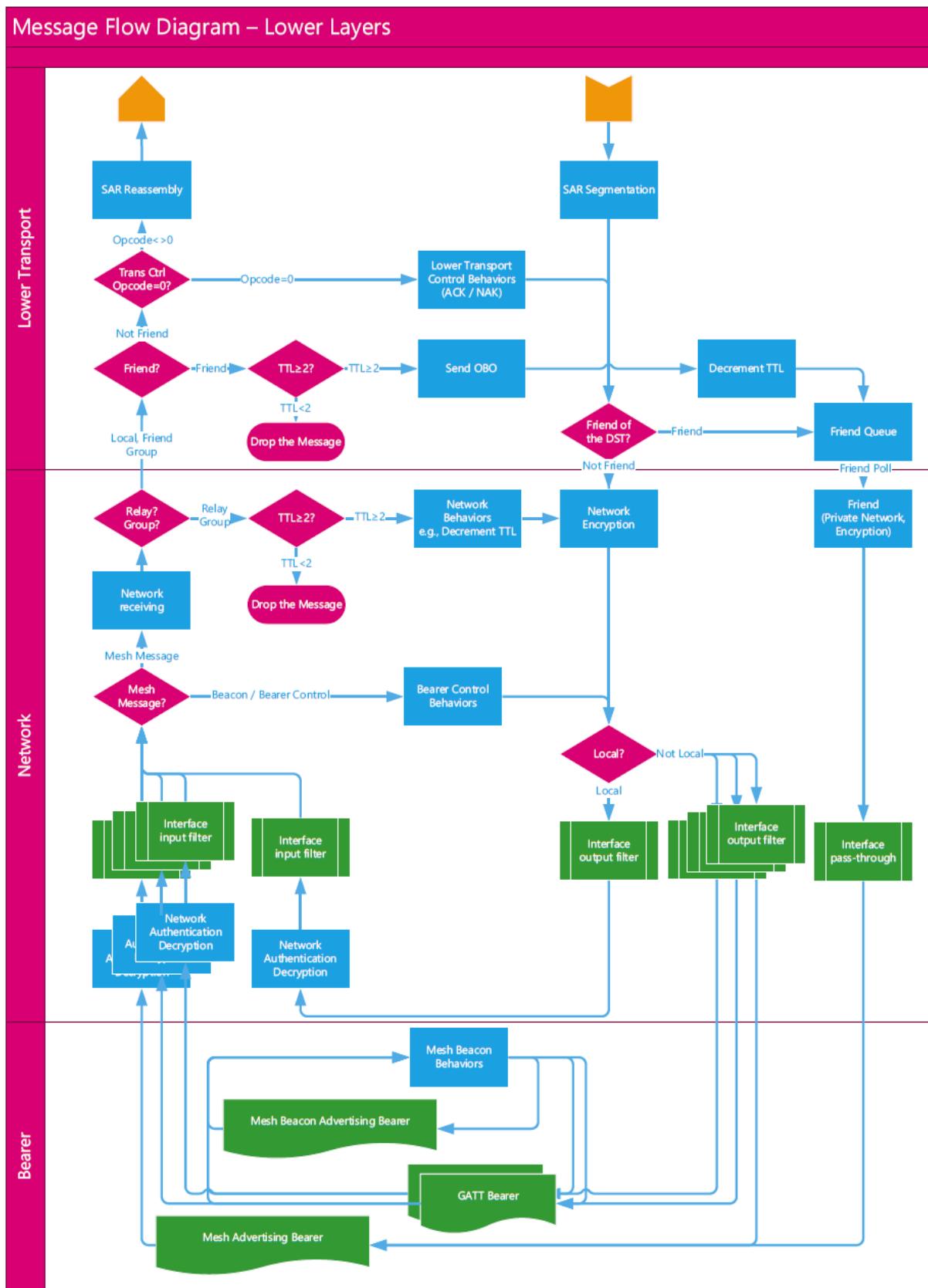


Abbildung G.2: Bluetooth-Mesh Stack Lower Layers [27]