Bauhaus-Universität Weimar
Fakultät Medien
Studiengang Medieninformatik

Bluetooth-based Mesh Networking for Smoke Detectors

Bachelorarbeit

Matti Wiegmann
geb. am: 24.01.1990 in Suhl

Matrikelnummer 112174

1. Gutachter: Junior-Prof. Dr. Florian Echtler
2. Gutachter: Prof. Dr. Unknown Yet

Datum der Abgabe: 31. Februar 2022

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weimar, 31. Februar 2022

.....
Matti Wiegmann

Zusammenfassung

A short summary.

Inhaltsverzeichnis

1 Einleitung **1**

2 Related Work **2**

3 Vorbetrachtung **5**
3.1 Aufbau und Funktionsweise von Rauchmeldern 5
3.2 Wireless Sensor Networks für Sicherheitskritische Anwendungen 8
3.2.1 Technologische Grundlagen 8
3.2.2 Bluetooth Low Energy 11

4 Implementierung **13**
4.1 Konstruktion einer Mesh-Node 13
4.2 Software Microprozessor 16
4.3 Interaktion mit der Außenwelt 23

5 Evaluation **26**
5.1 Anforderungen an die Signalübertragung 26
5.2 Testmethodik 28
5.3 Diskussion der Ergebnisse 29

6 Ausblick **35**
6.1 Sicherheit vor Angriffen und Fälschung 35
6.2 Energieverbrauch 35
6.3 Überwachung der Funktionstauglichkeit 35

Literaturverzeichnis **36**

Kapitel 1

Einleitung

Kurze Beschreibung der Motivation:

- Was es werden soll
- Was es schon gibt (properitär): zu teuer, zu mächtig, warum wifi als basis von IoT schlecht ist (wenn ich dazu quellen finde die das belegen)
- warum man diese schwächen mit BLE lösen kann
- kurz was genau gemacht und untersucht wurde und was dabei rum kam
- das werde ich wohl als letztes schreiben?

Kapitel 2

Related Work

Das Gebiet der automatisierten Erkennung von Feuer ist mit Bezug zur Informatik ausführlich erforscht worden. Dabei stechen vor allem drei Teilbereiche hervor:

- Studien zu verschiedenen Sensortypen und denen Kombinationen. Dazu zählen Versuche mit Fuzzy Logics, neuronalen Netzen und Fusion Algorithmus um vor allem Fehlalarme, spezielle unter widrigen Umständen, zu reduzieren.
- Studien zur Erkennung von Waldbränden, unter anderem Bild- und Videoanalysen von Flugzeug- und Satellitenaufnahmen sowie GSM-Sensornetze.
- Erkennung von Feuern in Gebäuden und bewohnten Gebieten.

Eine gute Übersicht zu diesen Themen findet man in **Automatic Fire Detection: A Survey from Wireless Seonsor Network Perspective** von Majid Bahrepour, Nirvana Meratnia und Paul Havinga[13]. Im Folgenden sollen einige für diese Arbeit besonders relevante Arbeiten kurz vorgestellt werden.

In **Building Fire Emergency Detection and Response Using Wireless Sensor Networks** [19] beschrieben Zeng et al. drei Routing-Protokolle für die Kommunikation zwischen vernetzen Funkrauchmeldern.
Das Fundament dabei bildet das von den Autoren entwickelte Verfahren "Real-time and Robust Routing in Fire (RTRR)". Dieses beschreibt das Routing von fest verbauten Sensoren (Nodes) zu vorher bekannten Empfangsstationen (Sinks). Jede Node kennt dabei einen von den vier Zuständen: safe (kein Feuer), lowsafe (angrenzende Node schlägt Alarm), infire (schlägt Alarm) and unsafe (nicht Funktionstüchtig).
Die anderen beiden Protokolle erweitern dieses Prinzip um mobile Nodes und

Sinks (an den Rettungskräften) für den Fall dass Melder während des Einsatzes ausfallen. Anschließend wird ein Konzept beschrieben wie der bestmögliche Rettungsweg anhand des Routingprotokolls berechnet werden kann.

Ismail, Husny und Abdullah präsentieren in **Smoke Detection Alert System via Mobile Application** [14] ein Konzept für einen Rauchmelder, der den Nutzer durch den Short Messaging System (SMS) informiert, wenn der Melder Alarm schlägt.
Dabei haben die Autoren einen Rauchsensor sowie ein Bluetooth Classic Funkmodul in einen Arduino verbaut. Wenn der Sensor aktiv wird, verbindet sich das Bluetooth-Modul mit einem nahegelegenen "Sender"-Smartphone. Dieses sendet wiederum eine spezielle SMS-Nachricht an das Empfänger-Smartphone. Dort wird die Nachricht von einer App abgefangen, die wiederum den Alarm am Empfänger auslöst und Möglichkeiten bietet, den Alarm abzuschalten.

Ein auf Routing basierendes, speziell für mehrgeschossige Gebäude entworfenes Design, wurde von Lei Zhang und Gaofeng Wang in **Design and Implementation of Automatic Fire Alarm System based on Wireless Sensor Networks** [20] vorgestellt.
Das Netzwerk setzt auf ein kabelgebundenes Backbone-Netz aus zentralen und lokalen Verteilern (Local Center und Surveillance Center). Die Weiterleitung zwischen Detektoren und lokalen Verteilern übernehmen dedizierte Repeater. Die Sensor-Nodes verbinden sich mit einem Repeater in Reichweite, beziehen eine dynamisch generierte Adresse und werden vom Repeater am Netzwerk angemeldet.
Die Kommunikation setzt auf den RF CC1100 Chipset von Texas Instruments. Das vorgestellte Konzept erlaubt eine Weiterleitung von Alarmsignalen in maximal drei Hops. Außerdem kann erkannt werden, welcher Sensor ausgelöst hat und wo der Repeater lokalisiert ist.

In **An Intelligent Fire Detection and Mitigation System Safe from Fire (SFF)** [16] wird ein experimentelles Brandschutzsystem vorgestellt. Es besteht aus einem Arduino als zentraler Verwaltungseinheit an den je zwei Rauch-, Gas-, und Temperatursensoren angeschlossen sind. Die Sensoren werden durch einen Fusion-Algorithmus ausgewertet um Fehlalarme zu reduzieren. Die Position des Feuers im abgedeckten Bereich wird durch eine Fuzzy Logic approximiert und die Sprinkler über einen angeschlossenen Motor zum Brandherd hin ausgerichtet. Wird ein Feuer erkannt, benachrichtigt der Arduino automatisch über ein angeschlossenes GSM-Modul die Feuerwehr sowie den Zuständigen Gebäudeservice.

Die vorgestellten Arbeiten haben sich vor allem damit beschäftigt, wie man (Wireless) Sensor Networks (WSNs) für Rauchmeldesysteme konzipieren und ihre Zuverlässigkeit optimieren kann. Dabei konzentrieren sie sich auf Routing-basierte Architekturen und untersuchen unter anderem Methoden für das Self-Healing des Netzwerkes. Zum Self-Healing zählt man Techniken, die, falls Nodes des Netzwerkes ausfallen, dessen Funktionsweise weiterhin sicherstellen sollen. Das ist vor allem für Routing-basierte Netzwerke ein Problem, weil die ausfallende Node ein Teil einer Route anderer Nodes sein kann. Fällt sie aus, sind entweder Teile des Netzwerkes gar nicht mehr erreichbar, oder der Ausfall muss erkannt und die betroffenen Routen müssen umgeleitet werden. Self-Healing ist eines der wichtigsten Hürden für WSNs in sicherheitskritischen Anwendungen, vor allem für Routing-basierte Netzwerke. Diese Arbeit versucht, die Hürde mithilfe Flooding-basierter Mesh-Netzwerke zu überwinden.

Kapitel 3

Vorbetrachtung

Um eine Implementierung planen zu können, müssen zuerst die technischen Rahmenbedingungen und Vorschriften analysiert werden. In diesem Kapitel sollen die wichtigen Bestandteile von Rauchmeldern sowohl allgemein als auch an einem passenden Testgerät herausgestellt werden. Außerdem wird beschrieben welche Möglichkeiten der Vernetzung bestehen und worauf bei Hard- und Software geachtet werden muss.

3.1 Aufbau und Funktionsweise von Rauchmeldern

Wie in Kapitel 1 erwähnt kann man einen Rauchmelder als System aus Sensoren und Aktuatoren ansehen, die über einen Integrated Circuit (IC) gesteuert werden. Der Aktuator ist der Signalgeber, üblicherweise ein ferroelektrischer Lautsprecher (Piezo-Summer). Die Sensoren können grob in zwei Kategorien unterteilt werden: Rauchsensoren und ergänzende Sensoren.

Bei Rauchsensoren gibt es zwei übliche Typen: Ionisationsrauchsensoren und photoelektrische (Optische) Rauchsensoren [17]. Erstere ionisieren mit Hilfe radioaktiven Materials die Luft zwischen zwei Metallplatten. Eindringender Rauch unterbricht dabei den Fluss der Ionen und löst so den Alarm aus. photoelektrische Rauchsensoren bestehen aus einer Rauchkammer, einer (Infrarot-)LED und einem Lichtsensor. Je nach Bauart des Sensors unterbricht der Rauch dabei entweder den Lichtstrahl oder löst den Lichtsensor durch veränderte Reflexionseigenschaften erst aus. Dieses Prinzip funktioniert besser, je größer die Partikel im Rauch sind. Die Größe ist vor allem abhängig von der Temperatur des Feuers. So sind bei Schwelbränden die Rauchpartikel größer als bei offenem Feuer. Da diese bei Wohnungs- und Gebäudebränden sehr häu-

fig sind und Ionisationsrauchsensoren aufgrund ihrer unsauberen Technologie in Europa nicht vertrieben werden dürfen [3], sind Photoelektrischer Rauchsensoren der de facto Standard.
Photoelektrischer Rauchsensoren sind allerdings anfällig für Falschalarm, ausgelöst durch Wasserdampf, u.a. in Badezimmern, oder Rauch der beim Kochen entsteht. Um diese schwächen auszugleichen werden in hochwertigen Rauchmeldern zusätzliche Sensoren Verbaut. Dazu zählen Temperatursensoren, CO-Sensoren und Feuersensoren [17] (siehe Kapitel 2).

Für Rauchmelder gelten in Deutschland zwei zentrale Normen. Die DIN 14767 [2] regelt Einbau, Betrieb und Test und wird in Kapitel 5 näher beschrieben. Die für Aufbau und Funktionsweise geltende Norm ist die DIN EN 14604 [3]. Die Norm ist für diese Arbeit besonders relevant, da sie das Verhalten der Rauchmelder und das Vorhandensein sowie Verhalten von LED und Testknopf festlegt. Das bedeutet, das sowohl die folgende Analyse eines Testgerätes als auch das in Kapitel 4 vorgestellte Konzept so oder so ähnlich für alle zertifizierten Modelle gilt.

Als Testrauchmelder wurde ein handelsüblicher, nicht vernetzter optischer Rauchmelder Flamingo FA23 von Smartwares ohne zusätzliche Sensoren ausgewählt.

In Abbildung 3.1 ist der Rauchmelder abgebildet. Er besteht aus sechs wichtigen Komponenten.

- Dem optische Rauchsensor (Rauchkammer) mit Infrarot-LED und Lichtsensor, einer 9-Volt Blockbatterie, dem Testschalter, einer roten LED als Warn- und Funktionsanzeige und zusätzlich:
- Dem Signalgeber (Horn). Bei dem Testgerät wurde ein 3-Pin Self-Drive Piezo-Summer verbaut, der mit 9V Wechselspannung versorgt wird. Einfache Piezo-Summer werden üblicherweise über zwei Pins mit einer Wechselspannung zwischen 10 und 100 Volt in Schwingung versetzt und erzeugen so einen hochfrequenten Ton. Der dritte Pin liefert ein Feedback-Signal für den Oszillator. Das hat zur Folge, dass der Schallgeber mit Eigenfrequenz schwingt und so eine höhere Lautstärke erreicht [11].
- Dem IC KD-5810 von Kingstar Technology Ltd. Er befindet sich auf der Unterseite der Platine und ist in Abbildung 3.1 nicht erkennbar. Dieser IC ist ein Nachbau des verbreiteten MC145010 [7] von Freescale Semiconductor mit veränderter Pin-Belegung. Der IC ist von einer Metallplatte verdeckt, so dass die Pins nicht direkt zugänglich sind, ohne den Rauchmelder zu beschädigen.

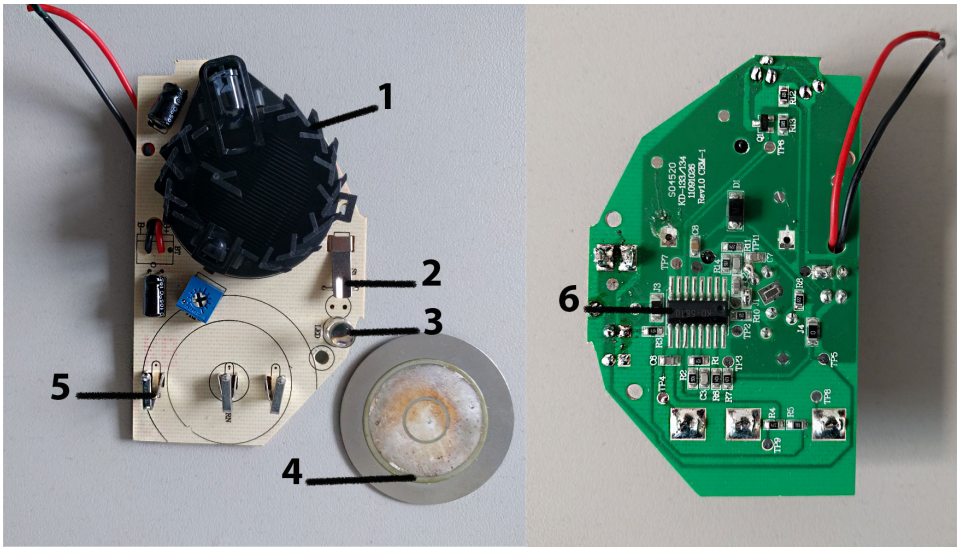

Abbildung 3.1: Innenleben des Testrauchmelders. (links) Oberseite: (1) Rauchsensor (geöffnet), (2) Testschalter, (3) Warnanzeige (LED), (4) Piezo-Element, (5) Signalgeber Pins, (rechts) Unterseite mit Schaltkreis, (6) IC ohne Abdeckung

Der Rauchmelder hat die folgende vier Verhaltensmuster [7]:

- **IDLE:** Der Signalgeber ist aus. Der Rauchmelder führt alle 38.9 bis 47.1 Sekunden eine Prüfung der Elektronik durch. Dabei wird die Ladung der Batterie überprüft (Low-Supply) und ob die Rauchkammer ordnungsgemäß funktioniert (Chamber-Sensitivity). Die LED blinkt alle 38.9 bis 47.1 Sekunden.
- **Test:** Wenn der Testschalter gedrückt ist. Der Signalgeber ist durchgehend an und die LED blinkt alle 0.6 bis 0.74 Sekunden.
- **ON:** Wenn der Testschalter nicht gedrückt ist und Rauch in der Rauchkammer ist. Der Signalgeber ist durchgehend an und die LED blinkt alle 0.6 bis 0.74 Sekunden
- **Fehler:** Wird bei der Prüfung der Elektronik ein Fehler festgestellt, piepst der Rauchmelder alle 38.9 bis 47.1 Sekunden. Die LED blinkt alle 38.9 bis 47.1 Sekunden. Bei Low-Supply piepst der Melder zeitgleich mit dem blinken der LED, bei Chamber-Sensitivity piepst er in der Mitte des Blinkintervalls.

Die tatsächliche Intervalldauer ist abhängig von der Taktung des Oszillators, bestimmt durch die Größe der verbauten Widerstände und Kondensatoren.

Für das Testmodell wurden 44 Sekunden für das lange Intervall und 0.7 Sekunden für das kurze Intervall gemessen. Laut Vorschrift müssen diese kürzer als eine Minute beziehungsweise eine Sekunde sein.
Die Zustände ON und Test sind vom Verhalten her identisch. Sie können dadurch Unterschieden werden dass entweder der Testschalter direkt oder der Pin16 des ICs (TEST) abgetastet wird.

Ohne den IC direkt abzutasten kann der Zustand des Rauchmelders auch anhand des Blinkmusters ausgelesen werden. Dabei kann allerdings nicht festgestellt werden ob eine Störung der Elektronik vorliegt, da das Blinkmuster mit Störung identisch ist zu dem im Idle-Modus.

3.2 Wireless Sensor Networks für Sicherheitskritische Anwendungen

Im folgenden Abschnitt soll erläutert werden, welche Technologien für Wireless Sensor Networks (WSN) in Frage kommen und wieso sich diese Arbeit mit Bluetooth LE (BLE) Broadcast-Mesh-Netzwerken befasst. Außerdem soll die benutzte Hardware sowie für die Implementation wichtige Konzepte des BLE-Stacks beschrieben werden.

3.2.1 Technologische Grundlagen

Die Idee hinter dem Internet der Dinge (IoT) ist jedes Gerät eindeutig Adressierbar zu machen und zentral zu steuern. Die Schlüsseltechnologien dafür sind die Internettechnologien TCP/IP und Wifi. Dabei bekommt jedes Gerät (mit Sensoren/Aktuatoren) eine Adresse aus dem IPv6-Adressraum. Die Kommunikation innerhalb des Netzwerkes und mit der Außenwelt wird durch (dedizierte) Router geregelt, die in Empfangsreichweite der IoT-Geräte sein müssen.

Wifi ist eine mächtige Technologie für das Internet der Dinge, vor allem da sie den vollen Internet-Protokollstack implementiert, einen hohe Reichweite und Datenrate hat. Das bringt jedoch das Problem mit sich, dass der Energieverbraucht relativ hoch ist. Low-Power Geräte wie Sensor-Nodes sollten mit einer gewöhnlichen 3 Volt Knopfzelle (etwa 50-300 mAh) viele Monate bis Jahre betrieben werden können, was mit herkömmlichen Wifi nicht möglich ist.

Um dieses Problem zu lösen wurde eine Vielzahl neuer Technologien und Standards speziell für den Low-Power Bereich entwickelt. Die wichtigsten davon sind [1]:

- **6LoWPAN** (IPv6 Low-Power Wireless Personal Area Network) und Thread sind Netzwerk-Protokolle. Sie basieren auf IPv6 und sind für Low-Power Geräte und Mesh-Netzwerke entwickelt worden. Der Standard für 6LoWPAN ist der RFC6282, für Thread IEEE802.15.4 und 6LowPAN.
- **Zigbee** basiert auf IEEE802.15.4. Es sendet im 2,4 GHz Band und unterstützt Datenraten von bis zu 250 Kbit/s auf einer Reichweite von bis zu 100 Metern.
- **Z-Wave** ermöglicht Datenraten von bis zu 100 Kbit/s auf bis zu 30 Meter Reichweite. Gesendet wird im Gegensatz zu Zigbee und BLE im 900 MHz Band, wodurch es weniger anfällig gegenüber Interferenzen ist.
- **Bluetooth Low Energy** (auch Bluetooth Smart / BLE) ist eine Erweiterung der Bluetooth Core Specification ab Version 4.0. Gesendet wird im 2.4 GHz Band mit Datenraten von bis zu 1 Mbit/s und einer Reichweite von theoretisch über 100 Metern. Seit Core Specification 4.2 implementiert BLE auch 6LoWPAN, so dass Geräte über das Internet Protocol Support Profile mit IP-basierten Netzwerken kommunizieren können.

Zigbee und Z-Wave implementieren Mesh-Funktionalitäten bereits, für BLE ist nativer Support in Entwicklung. Bluetooth hat allerdings den Vorteil, dass es eine wesentlich weitere Verbreitung im privaten Sektor hat und von vielen Smartphones und ähnlichen Geräten unterstützt wird. IPhones werden seit 2011 mit BLE Sensoren ausgeliefert. Android unterstützt BLE ab API-Level 18, das umfasst über 80% alle Geräte [10].
Da in industriellen und öffentlichen Gebäuden meistens verkabelte Brandmeldeanlagen vorgeschrieben werden [3] ist der private Sektor für diese Arbeit am interessantesten. Daher bietet sich Bluetooth Low Energy als technologische Basis an.

Die meisten der erwähnten Technologien für das Internet der Dinge setzen auf IPv6 und zählen somit zu den Routed-Networks. Das heißt, dass jedes Gerät eine Adresse hat. Die Daten, die über das Netzwerk versendet werden, werden mit den Adressen des Absenders und des Empfängers versehen. Die Router wissen, im Gegensatz zu den Endgeräten, wie diese Pakete über das Netzwerk versendet werden müssen und sind deswegen nötig um die Kommunikation zwischen Endgeräten zu ermöglichen.
Dieses Prinzip ist dann essenziell, wenn Absender und Ziel genau identifiziert werden müssen. Ist das nicht der Fall, bleiben einige Nachteile:

- Es sind dedizierte Geräte (Router) notwendig, damit das Netzwerk funktioniert.

- Jedes Gerät im Netzwerk muss eingerichtet werden.
- Fällt ein Gerät aus, müssen im besten Fall die Routen neu bestimmt werden. Im schlechtesten Fall können Teile des Netzwerkes nicht mehr kommunizieren.

Um diese Probleme zu umgehen, hat sich die Wissenschaft mit alternativen Lösungen beschäftigt (siehe Kapitel 2).

Sicherheitskritische Anwendungen, wie zum Beispiel Rauchmeldernetzwerke, sind nicht darauf angewiesen, dass jedes Gerät eindeutig Adressierbar ist. Die Aufgabe des Netzwerkes ist, das jeder Rauchmelder im Netzwerk Alarm schlägt, wenn nur ein einziger Melder Rauch erkennt.
Strasser et al. haben für diese Art von Netzwerken drei "fundamental challenges" [18] herausgestellt: zuverlässige Datenübertragung (reliable data delivery), geringe Latenz (low latency) und geringer Energieverbrauch (low energy consumption).
Um diesen Anforderungen gerecht zu werden schlagen Strasser et al. vor, Flooding anstatt Routing zu benutzen. Beim Flooding wird die Nachricht von der Sender-Node an alle Nodes in Empfangsreichweite gesendet. Diese leiten das Paket wiederum an alle Nodes in Reichweite weiter, bis alle Nodes im Netzwerk die Information erhalten haben.
Beim Broadcast-Flooding, im Gegensatz zum Unicast-Flooding, werden die Daten unselektiert an alle Nodes in Reichweite weiter gesendet. Damit das Netzwerk allerdings Funktionsfähig bleibt, müssen in regelmäßigen Abständen Status-Nachrichten gesendet werden. Passiert das nicht, wissen die Nodes und eventuelle Supervisor nicht ob das Netzwerk noch vollständig besteht. Flooding erfüllt die Challenge der zuverlässigen Datenübertragung im Gegensatz zum Routing immer, da jede Node die Nachrichten immer an alle Anderen in Reichweite sendet. Fällt eine Node aus, stört das die Kommunikation aller anderen Nodes nicht, solange die ausfallende Node das Mesh nicht in zwei Teilnetze trennt, die außerhalb ihrer jeweiligen Sendereichweite liegen .

Um energiesparendes Broadcast-Flooding zu ermöglichen, haben Levis et al. 2011 den Trickle-Algorithmus (RFC6206)[15] entwickelt. Trickle verfügt über ein variables Rebroadcasting-Intervall. Dieses Intervall wird länger, je länger der Abstand zum letzten Update wird. Dadurch kann der Abstand zwischen zwei Broadcasts auf bis zu mehrere Stunden ansteigen. Empfängt eine Node einen inkonsistenten Wert, überprüft sie anhand eines Alters-Flags ob es sich um eine Update handelt. Ist das der Fall, wird das Broadcasting-Intervall wieder auf den Minimalwert zurückgesetzt und so sichergestellt, dass Updates so schnell wie möglich über das Netzwerk propagiert werden.

Für diese Arbeit soll also ein auf Trickle basierendes Broadcast-Flooding-Mesh-Network für Rauchmelder mit Bluetooth Kommunikationstechnologie entwickelt und seine Einsatzfähigkeit verifiziert werden.

3.2.2 Bluetooth Low Energy

Da Bluetooth Low Energy ein offener Standard, beschrieben in der Bluetooth Core Specification [12], ist, gibt es eine Vielzahl an Hardwareherstellern. Für dieses Projekt wurde die auf dem ARM Cortex M0 basierende nRF51822 MCU (Micro Controller Unit) von Nordic Semiconductor ausgewählt (siehe Kapitel 4).

Da der Bluetooth Stack sehr komplex ist, sollen die für diese Arbeit wichtigsten Konzepte kurz erläutert werden.
Ein BLE Gerät ist Grundsätzlich entweder ein Central oder ein Peripheral. Peripherals sind verteilte, low-energy Geräte die an den jeweiligen Sensoren oder Aktuatoren angeschlossen sind. Centrals sind die mächtigeren, zentralen Geräte. Sie sollen die Datan aus mehreren Peripherals auslesen. Die Kommunikation zwischen BLE(-fähigen) Geräten wird durch zwei sogenannte Profile geführt.

Das Generic Access Profile (GAP) managt Advertising und Verbindungsfähigkeit. Das Advertising ist ein Datenpaket, das regelmäßig und unaufgefordert vom Peripheral gesendet wird und von allen anderen BLE Geräten empfangen werden kann. GAP muss implementiert und aktiv sein, damit die Node von anderen Geräten gesehen werden kann. Laut Spezifikation kommunizieren BLE Geräte über 40 Channel mit je 2 MHz Abstand im 2,4 GHz ISM Band. Um Interferenzen mit Wifi zu vermeiden werden für GAP-Advertisments nur die Channel 37, 38 und 39 genutzt, auf denen Wifi nicht sendet.
Das Advertisment ist die Broadcast-Methode von BLE. In der Core Specification ist allerdings nicht vorgesehen, dass Peripherals über Advertisments kommunizieren. Außerdem ist es im Standard nicht vorgesehen, dass während einer Verbindung gebroadcastet werden kann. Für diese Funktionen kann aber das Softdevice der Nordic-SDK benutzt werden.

Die BLE-Funktionen sind für Geräte von Nordic Semiconductor in einer Art Betriebssystem-API implementiert, die Softdevice genannt wird. Dieses Softdevice ist vorkompiliert und abhängig von der benutzten Version der Nordic SDK und dem Typ des BLE-Gerätes. Zusätzlich zu den festgelegten Funktionen bietet dieses SDK ab Version 8.0 die sogenannte Timeslot-API [9].

Diese API ermöglicht genau das, dass mehrere Advertising-"Threads"nebeneinander stattfinden und von den anderen Peripherals mit der gleichen Softdevice Version auch empfangen werden können. Das GAP-Advertisment, das von Centrals gescannt und zum Verbindungsaufbau benötigt wird, wird davon nicht beeinflusst.

Über GAP können die Peripheral und Central miteinander verbunden werden, was das Advertising in der Regel für die Dauer der Verbindung stoppt. Das Generic Attribute Profile (GATT) beschreibt die Kommunikation zwischen verbundenen Peripheral und Central. Das Peripheral implementiert dafür Profiles, Services und Characteristics. Das sind hierarchisch verschachtelte Objecte, wobei jedes Profile mehrere Services und jeder Service mehrere Characteristics beinhalten kann.
Das Central kann Lese- und Schreibanfragen an eine Characteristic des verbundenen Peripherals schicken. Das Peripheral kann ein verbundenes Central informieren, wenn sich der Wert einer Characteristic ändert. GATT benutzt für die Datenübertragung die verbliebenen 37 Channel im ISM-Band und ermöglicht so Übertragungsraten von bis zu 1 MHz/s.

GATT ist für das Flooding nicht sinnvoll. Selbst wenn die Nodes sich schnell genug Verbinden und Trennen würden, wäre der Energiebedarf dafür trotzdem zu hoch um lange Laufzeiten beizubehalten. Es kann jedoch als Schnittstelle für ein Central-Device dienen, dass den Status des Netzes überwacht oder zu Testzwecken manipuliert.

Kapitel 4

Implementierung

Aufbauend aus den Ergebnissen der Vorbetrachtung (Kapitel 3) soll in diesem Kapitel, wie ein funktionsfähiges Rauchmelder-Netzwerk auf Basis eines BLE-Meshs gebaut und programmiert werden kann. Dazu sollen zuerst handelsübliche Rauchmelder an BLE-Sender angeschlossen werden. Die Sender so programmiert werden, dass sie die wichtigen Mesh-Funktionen bieten. Außerdem soll eine beispielhafte Schnittstelle zur Außenwelt in Form einer Android-App simuliert werden.

4.1 Konstruktion einer Mesh-Node

Jede Node des Meshs besteht aus einem Rauchmelder-Modul und einem BLE-Sender. Als Rauchmelder wird ein Flamingo FA23 von Smartwares benutzt (siehe Abschnitt 3.1). Ein externer MPU, wie ein nRF51822, kann entweder direkt an die Pins des ICs angeschlossen oder in den Schaltkreis (Testschalter, Piezo-Lautsprecher und Signal-LED) integriert werden.

Der Pin 7 (I/O) des ICs ist dafür vorgesehen, beim Auslösen des Rauchsensors angeschlossene Geräte zu Informieren oder von angeschlossenen Geräten ausgelöst zu werden. Pin 7 hat ein LOW von 1.5 Volt und ein HIGH von 3.2 Volt und kann von einem 3.3 Volt Microprozessor ohne Spannungstransformation angesteuert bzw. ausgelesen werden. Der Pin 16 (Test) des ICs wird vom Testschalter des Rauchmelders angesprochen und löst den Alarm aus, ohne dass Rauch in der Rauchkammer ist. Dieser Pin erwartet eine HIGH-Spannung von mindestens 8.5 Volt.

Für eine eventuelle industrielle Fertigung eines BLE-fähigen Rauchmelders, bei dem der BLE-Chip und Antenne fest in demselben Schaltkreis integriert werden ist diese Methode aufgrund der Zuverlässigkeit der Signale vorzuziehen.

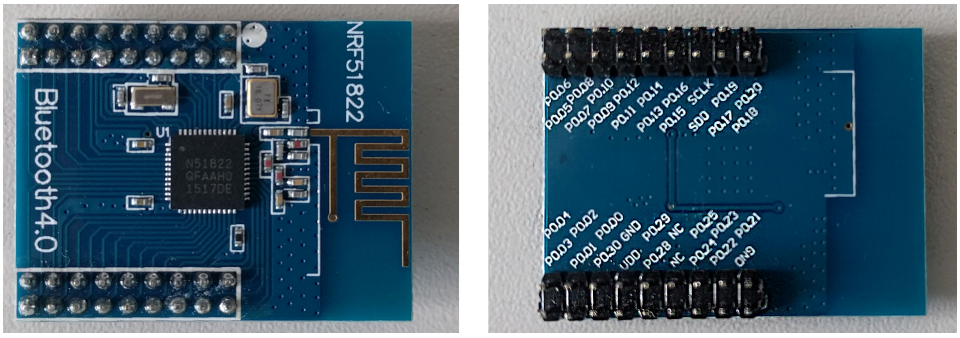

Abbildung 4.1: nRF51822 Breakout-Board. (rechts) Unterseite mit Chip und Antenne. (links) Vorderseite mit Pin-Headers

Beim Flamingo FA23 ist der IC allerdings durch eine fest verbaute Metallplatte abgeschirmt. Diese kann nur mit Beschädigung des Rauchmelders entfernt werden.

Aus der Menge der möglichen BLE-Sender wurde ein Nordic nRF51822 Breakout-Board (siehe Abbildung 4.1 ausgewählt. Es besteht aus dem SoC (System on a Chip) und einer integrierten Antenne, ist 2.5 cm breit und 3 cm lang und kostet etwa 6 Euro. Die Pins des SoC sind als zwei 2x9-Pin-Header ausgeführt. Davon sind 29 frei belegbare Pins (PO00 - PO25, PO28 - PO30), zwei nicht Verbunden (N.C.), zwei Ground, einer für Versorgung (VCC) und zwei sind für das flashen des Codes notwendig (SCLK, SDO).

Es werden mindestens zwei Pins für eine Node benötigt: einer, die das Auslösen des Rauchmelders abtastet und einer, die den Rauchmelder auslöst. Zwei weitere sind optional denkbar: einer, der den Batteriestatus des Rauchmelders überwacht, falls Melder und Sender nicht über dieselbe Spannungsquelle versorgt werden und einer, der den Testschalter überwacht um zwischen Test und Alarm unterscheiden zu können.

Welcher der frei belegbaren Pins des nRF51822 genutzt wird spielt keine Rolle, allerdings gib es durch Nordics Board Support Package (BSP) einen Status Quo. Im BSP werden unter anderem statische Bezeichner für die Pins der offiziellen Nordic Hardware vergeben. Zu den im BSP definierten Bezeichnern gehören unter anderem PO21 als Status-LED für Softdevice-Funktionen (wie etwa Advertising / Verbindungsstatus) und PO00 als Reset-Button.

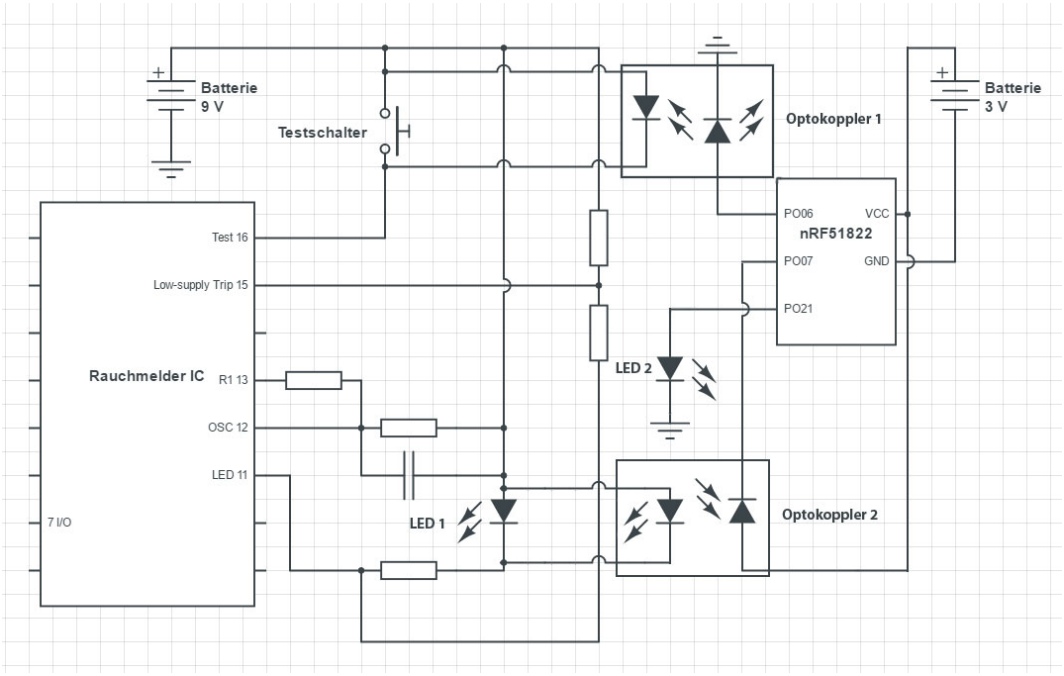

Abbildung 4.2: Schaltplan einer Mesh-Node mit externer 3V Spannungsquelle für den BLE-Sender (Auszug)

Um die Beschädigung des Rauchmelders durch entfernen der Metallplatte zu vermeiden wird das nRF51822-Breakout-Board für das Testsetup an den Testschalter und die LED angeschlossen (Abbildung 4.2). Dabei werden zwei Optokoppler verbaut. Der Erste davon schaltet den Testschalter, wenn an PO06 des nRFs Spannung anliegt. Der zweite Optokoppler schaltet Spannung an PO07 des nRFs wenn die LED 1 des Rauchmelders blinkt. An PO21 des nRFs ist eine LED angeschlossen, welche den Verbindungsstatus anzeigt (siehe Kapitel 4.2).

Als Stromversorgung dient dem Rauchmelder eine 9 Volt Blockbatterie. Der BLE-Sender braucht eine Versorgung mit 3 - 3.3 Volt, am häufigsten werden dafür Knopfzellen wegen ihrer geringen Größe verwendet. Für die Mesh-Node können entweder beide Batterien verwendet werden, wobei jedes Modul seine eigene hat. Alternativ kann der nRF auf über einen 3.3 Volt Step-Down-Wandler mit an die Blockzelle angeschlossen werden. Die Lösung mit getrennten Batterien bietet eine höhere Lebensdauer der Node. Bei einer gemeinsamen Versorgung muss der nRF gegebenenfalls nur eine Batterie überwachen, was dessen Verbrauch reduzieren würde.

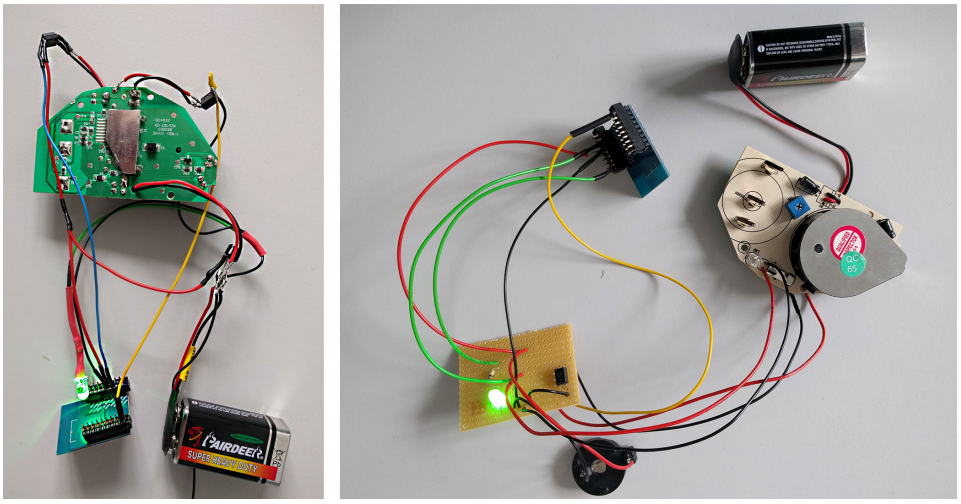

Abbildung 4.3: Konstrukt aus Rauchmelder mit angebautem nRF51822. (links) Lösung mit einer 9 Volt Blockzelle. (rechts) Lösung mit einer 9 Volt Blockzelle für Rauchmelder und einer 3 Volt Knopfzelle für nRF

Für dieses Projekt wurden zwei Nodes konstruiert und auf Funktionsfähigkeit getestet (Abbildung 4.3), eine davon mit zwei Batterien und die andere mit einer Batterie.

4.2 Software Microprozessor

Der Prozessor des nRF51822-SoC ist ein ARM Cortex M0 und wird in C99 programmiert und mit GCC kompiliert. Um den Chip zu flashen kann ein Buspirate benutzt werden um das Programm direkt in den Flash-Speicher des Chips zu kopieren. Dieses Vorgehen ist für der Entwicklung sehr aufwendig und bietet kaum Möglichkeiten für effektives Debugging. Für dieses Projekt wurde deswegen ein J-Link Lite CortexM SWD Emulator von Segger genutzt. Er bietet eine Schnittstelle zwischen USB und dem Serial Wire Debug (SWD) Interface, über den das Breakout-Board angeschlossen wird. Als Entwicklungsumgebung wurde ARMs Keil IDE verwendet. Die IDE enthält den Compiler und integriert das flashen über J-Link Hardware. Der Code für die essenziellen Hardware-Funktionen des nRFs ist Teil der Nordic nRF5 SDK [6].

Um die Bluetooth Low Energy Funktionen nutzen zu können, wird für Peripherals das Softdevice S110[8] benötigt (siehe Unterabschnitt 3.2.2, welches den BLE-Stack implementiert. Das Softdevice ist in der SDK enthalten, allerdings vorkompiliert, und muss separat an den Anfang des Flashes geschrieben

werden. Das Softdevice übernimmt den Großteil der Module des SoCs, wie Taktgeber, Radio und I/O-Pins. Über die Makros und Methoden des Softdevices kann das Programm dann unter anderem Timer und I/O-Interrupts von Softdevice anfordern und Pins setzen sowie die BLE-Funktionen steuern. Das Softdevice enthält auch die Concurrent Multiprotocol Timeslot API. Über diese API kann das Programm von 100 μs bis 100 ms den Zugriff auf das Radio-Modul erhalten, um Nachrichten zu versenden.

Die Software jedes BLE-Senders muss die folgenden Funktionen erfüllen:

- Erkennen, in welchem Zustand sich der Rauchmelder befindet. Löst der Rauchmelder aus, muss der Sender den Wert mit dem Alarmzustand ändern und in das Mesh flooden.
- Den Rauchmelder auslösen, wenn eine entsprechende Nachricht empfangen wird.
- Empfangene Nachrichten weiterleiten.
- Den Zustand des Meshs nach außen hin propagieren. Dadurch soll das Mesh im einfachsten Fall durch Geräte in Reichweite überwacht werden oder in bestehende IoT-Systeme integriert werden können.
- Nachrichten von außen empfangen und weiterleiten. Das soll vor allem für Tests dienen um die Integrität des Netzes sicherzustellen. Außerdem soll so alle Rauchmelder, die nicht direkt auslösen, stummschalten zu können (siehe Kapitel 4.3).

Es gibt mit dem nRF OpenMesh [5] bereits eine Open-Source Implementierung des Trickle-Algorithmus für nRF51 SoCs und das nRF SDK 8.1, welche auf der Timeslot API aufbaut. Dieses Framework kann mit einer Reihe von Modifikationen für die Implementierung der Mesh-Nodes verwendet werden.
formeln aufstellen
Des nRF OpenMesh implementiert den Trickle-Algorithmus wie in der RFC spezifiziert: Bei der Initialisierung des Netzwerkes werden minimales und maximales Intervall I_{min} und I_{max} sowie eine Redundanzkonstante k definiert. Der Algorithmus setzt, für jeden Wert, das Intervall auf I_{min} und sucht, um Kollisionen zu reduzieren, einen zufälligen Zeitpunkt aus der zweiten Hälfte des Intervalls als Übertragungszeitpunkt aus. Alle im Intervall vor diesem Zeitpunkt empfangenen Pakete des jeweiligen Wertes werden gezählt. Ist die Anzahl der (mit dem lokalen Wert konsistenten) Pakete kleiner als die Redundanzkonstante wird der Wert gebroadcastet. Am Ende des Intervalls wird dessen Zeitspanne verdoppelt, bis zu einer maximalen Länge von I_{max}. Wird

ein inkonsistenter Wert (Update) empfangen, wird das Intervall auf I_{min} zurückgesetzt.

Visualisierung trickle? Oder reicht die Beschreibung?

Die Implementierung des Trickels im nRF OpenMesh sieht nur eine freie Auswahl von I_{min} von und definiert k als 3 und I_{max} als 2048 * I_{min}. Je kleiner k ist, desto weniger werden konsistente Werte rebroadcasted, da die Wahrscheinlichkeit bei langen Intervallen hoch ist vor dem gewählten Zeitpunkt Pakete zu empfangen. Ein hohes k erhöht allerdings die Integrität des Netzwerkes, was vor allem für sicherheitskritische Anwendungen relevant ist. Der Standardwert von 3 ist ein guter Kompromiss zwischen Zuverlässigkeit und Energiesparsamkeit.

Die Verzögerung vor dem ersten Senden eines neuen Wertes ist nach oben beschränkt durch I_{min}, die Länge des Timeslots und der Zeit, bis das Softdevice den Timeslot bereitstellt. Das Framework fordert Timeslots mit einer Länge von 10 ms an. Dieses Intervall wird vom Softdevice bereitgestellt, sobald keine Module mit höherer Priorität verarbeitet werden. Als I_{min} für den Alarmzustand wurden 128ms ausgewählt. Das ergibt als theoretisches Infimum um einen neuen Wert über das ganze Mesh zu propagieren ist (100ms I_{min} + Bereitstellungszeitraum) * maximale Anzahl der Hops im Mesh. Werden mehrere Werte über das Mesh verteilt, erhöht sich der Bereitstellungszeitraum um die Dauer der Timeslots für jeden Wert in der Queue und den Zeitraum zwischen den Timeslots, in denen das Softdevice die volle Kontrolle über die Hardware braucht. Das Rauchmelder-Mesh propagiert allerdings nur einen Wert (Alarmstatus). Deswegen ist der Bereitstellungszeitraum maximal so lang, wie GAP und GATT Interaktion dauern. Die Autoren des Frameworks geben an, dass das Softdevice währen des Advertisings etwa 5-25% der Zeit die Kontrolle über das Radio-Modul übernimmt, während einer GATT-Verbindung allerdings bis zu 80%.

Die maximale Intervalllänge im Mesh sind für I_{min} = 128ms und dem Standard-Multiplikator von 2048 im OpenMesh gleich 262144ms. Für die Berechnung der Intervalllängen ab dem ersten werden in der Implementierung immer die nächstliegenden Potenzen von 2 verwendet. Das sind rund 4,37 Sekunden für I_{max}, was viel zu kurz ist hinsichtlich Batterie-Lebensdauer. Der Alarmzustand ändert sich im Mesh nur dann, wenn ein Rauchmelder auslöst. Je weniger Pakete der Rauchmelder sendet, desto geringer ist der Stromverbrauch. Es ist also notwendig das rebroadcasting konsistenter Werte noch über die Redundanzkonstante hinaus zu reduzieren, in dem I_{max} erhöht wird. Ein sinnvoller Kompromiss zwischen Mesh-Integrität und weniger Rebroadcasts ist ein I_{max} von etwa einer Stunde. Die nächste 2er-Potenz für den Multiplikator ist 2^{15} mit resultierendem I_{max} von rund 1 Stunde und 10 Minuten.

Visualisierung wann welches Modul Zugriff auf Radio hat???
Eine längere Distanz zwischen Rebroadcasts hat außerdem Vorteile für das Empfangen von Broadcasts. Dafür fordert das Framework Timeslots von 10 Sekunden Länge an und verlängert diese auf bis zu 1 Sekunde. Danach wird ein neuer Timeslot angefordert. Während dieser TImeslot aktiv ist, werden alle ankommenden Sende-Events sofort ausgeführt. Sind alle Sende-Events abgearbeitet, scannt das Framework auf BLE-Advertisment Packages in Reichweite. Die Mesh-Broadcasts sind dafür als reguläre BLE-Advertisments implementiert und werden von allen anderen durch die AD-Type und Service-UUID Flags unterschieden. Reguläre BLE-Advertisments haben eine Länge von 47 Bytes (0-46), die Advertisment-Data sind in den Bytes 13-43. AD-Type ist das Byte 14 des Advertisment-Packets (Byte 1 der Advertisment-Data) und die Service-UUID ist Byte 2-3 (15-16). AD-Type von Mesh-Updates ist 0x16 und Service-UUID ist 0xFEE4. Alle gescannten Pakete die nicht diesen Werten entsprechen werden verworfen. Danach werden die nicht verworfenen Pakete auf Handle (Byte 4-5), Version(Byte 6-7) und Content(Byte 8 - 30) geparst. Das Handle ist die einzigartige ID des jeweiligen Mesh-Wertes. Version ist die Angabe des Ältersdes übertragenen Wertes. Anhand der Version kann unterschieden werden, ob es sich bei einer Inkonsistenz um ein Update oder einen älteren Wert handelt. Content enthält den eigentlichen Wert mit 1 - 23 Bytes Länge. Wird nur ein Byte des Contents gesetzt, reduziert sich das gesamte Advertising Packet auf 25 Bytes Länge, was wiederum die Sendezeit und damit Energieverbrauch reduziert.
Note: Das Framework holt sich nen timeslot für 10ms wann immer es kann, arbeitet sending ab, und nutzt die Restzeit + Verlängerung zum Scannen. Repeat. Kommst da so rüber oder soll ich die Absätze zum Senden und scannen kombinieren und anhand der timeslots erklären? Ist das vielleicht zu viel Allgemeinwissen zum Framework und zu wenig im Verhältnis zur konkreten Implementierung?
Visualisierung byte-struktur?
Das Rauchmelder-Netzwerk hat nur einen Mesh-Wert: den Alarmstatus mit Handle 01. Er beträgt 0x00 für kein Alarm und 0x01 für Alarm. Wird im Framework ein passendes Advertising-Paket erkannt, wird ein Callback aufgerufen. Dort kann Anhand des Eventtyps eine entsprechende Aktion ausgeführt werden:

- **Update:** Der empfangene Wert ist inkonsistent und hat eine höhere Version. Ist der neue Wert 0x01 wird der PO06 des nRFs gesetzt. Das aktiviert den angeschlossenen Optokoppler, welcher den Testschalter des Rauchmelders kurzschließt. Das versetzt den Rauchmelder in den Zustand TEST, welcher vom Verhalten her identisch ist mit ON. Ist der neue Wert 0x00 wird der PO06 gecleart und ein Timer gestartet, der für

die nächsten 10 Sekunden ein erneutes setzen des Pins verhindert. Dieser Timer soll ein kurzzeitiges manuelles muten aller nicht direkt auslösenden Rauchmelder ermöglichen. Der Rauchmelder wechselt in den Zustand IDLE. Alle anderen Werte außer 0x01 und 0x00 werden verworfen.

- **New:** Alle New-Events, die nicht das Handle 01 haben, werden verworfen und das neue Handle wird aus dem Speicher gelöscht. Dieses Verhalten wurde als Schutzmechanismus implementiert, damit kein Cache-Overflow entstehen kann. Das Famework führt zwei Caches: einen persistenten und einen nicht persistenten. Ist der persistente Cache voll, können ältere Werte gelöscht werden. Die Größe des Caches wurde bewusst auf 6 reduziert um den reservierten Speicher des nRFs klein zu halten. Überflüssige Werte können dadurch entstehen, dass entweder ein anderes Mesh in Reichweite betrieben wird oder wenn das Mesh angegriffen wird (siehe Kapitel 6.1).

- **Conflicting:** Ein empfangenes Packet ist dann Conflicting, wenn die Version identisch mit der Lokalen ist, aber der Wert unterschiedlich. Dieses Szenario ist im normalen Betrieb sehr unwahrscheinlich, etwa wenn mehrere Melder schneller zwischen ON und IDLE wechseln als Updates über das Mesh Propagiert werden können. Dabei piepsen die betroffenen Melder und soll manuell vom Benutzer repariert werden. Der wahrscheinlichere Fall ist eine Manipulation von außen, weswegen Conflicting-Values immer verworfen werden.

Die Timeslot API erlaubt es, das Mesh-Packets auf nur einem Channel gesendet werden. Dadurch müssen weniger Channel gescannt werden. Der Channel wird beim initialisieren des Meshs festgelegt. Für das Rauchmelder-Netzwerk wurde der Channel 38 ausgewählt um Kollisionen mit Wifi zu vermeiden.

Der nRF tastet über den PO07 den Zustand des Rauchmelders ab (siehe Kapitel 3. Bei integrierten Lösungen können I/O, TEST und die Signalgeber Pins (BRASS und SILVER) abgetastet werden um zwischen allen vier Zuständen unterscheiden zu können. Bei der Methode nur die LED abzutasten kann nur zwischen IDLE/Fehler und TEST/ON unterschieden werden, da die Blinkmuster in den jeweiligen Fällen gleich sind. Diese Unterscheidung ist allerdings unwichtig für die Funktionsfähigkeit, da die Rauchmelder trotzdem lokal piepsen wenn ein Fehler festgestellt wird. Beim Abtasten bekommt der nRF kein konstantes Signal sondern ein Muster mit Abständen von 700ms für das kurze Intervall I_{kurz} im ON/TEST-Zustand und 44000ms für das lange Intervall I_{lang} im IDLE/Fehler Zustand.
Um den Pin abzutasten wurde in der Software der GPIOTE-Handler der nRF

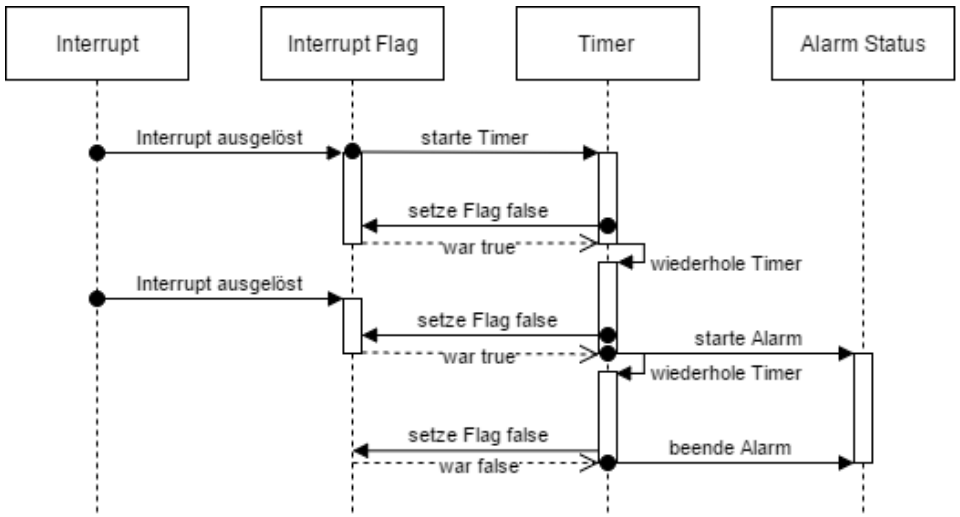

Abbildung 4.4: Algorithmus zum Abtasten des Blinkintervalls (Abstrahiert)

SDK implementiert. Dabei wird bei Systemstart der PO07 als Input-Pin initialisiert. Wechselt die Versorgung am Pin von LOW zu HIGH, wird ein I/O-Interrupt ausgelöst und der zugehörige Callback aufgerufen. In diesem Callback muss unterschieden werden, ob es sich bei dem Interrupt um das kurze oder das lange Blinkintervall handelt. Der Algorithmus dazu ist in Abbildung 4.4 dargestellt. Der Algorithmus startet nach einem Delay von 10ms einen Timer mit Länge $I_{kurz}/2$ und setzt ein Flag, dass ein Interrupt erfolgt ist. Der Timeout-Handler setzt dieses Flag auf false und der Timer wird wiederholt. Ist das Flag nach der Wiederholung true, ist ein zweiter Interrupt ausgelöst worden und das Programm weiß das die LED im kurzen Intervall blinkt und der Rauchmelder folglich auslöst. Der Timer wird nach diesem Schema solange Wiederholt, bis das Flag nach einer Wiederholung immer noch false ist und der Rauchmelder folglich nicht mehr auslöst. Wird so ein Zustandswechsel festgestellt, wird der Mesh-Value für den Alarmzustand geändert, das Trickle-Intervall resettet und der neue Wert über das Mesh propagiert. Das Länge des Timers ist für $I_{kurz}/2 = 350ms$ gesetzt. Das Auslösen des Rauchmelders kann anhand des Blinkintervalls ab der zweiten Wiederholung des Timers mit Delay nach $I_{kurz} = 710ms$ erkannt werden. Der Delay soll verhindern dass durch Verzögerungen durch die Ressourcenverteilung des Softdevices der zweite Timeout-Handler ausgeführt wird, bevor der Interrupt-Handler aufgerufen wurde.

Ändert sich der Alarmzustand entweder lokal oder über das Mesh werden auch die Manufacturer-Data im GAP-Advertisment geändert. Die Standard-

Implementierung des OpenMesh beinhaltet auch GAP und GATT. Das GAP-Advertisment wurde dahingehend angepasst, zusätzlich zum Namen der Node noch Manufacturer-Data zu enthalten. Dieser optionale Teil kann selbst gewählte Bytes an das Advertisment-Packet anhängen. Das Packet darf dabei 31 Bytes nicht überschreiten. Die ersten 4 Bytes der Manufacturer-Date enthalten Länge, Type-Code und die Manufacturer-ID, hier 0xFFFF für üoch in Entwicklung. Danach wird 1 Byte gesendet, das den Wert des Handles 01 enthält, den Alarmzustand. Das Intervall, in dem das GAP-Advertisment gesendet wird wurde auf 200ms eingestellt. Dieses Intervall wurde in Hinsicht auf den Entwicklungsprozess so schnell eingstellt, kann und sollte für ein Produktivsystem aber auf bis zu 10.24 Sekunden erhöht werden. Das GAP-Advertisment bietet sich dafür an, von externen Geräten, wie etwa der Schnittstelle nach Außen, gelesen zu werden. Diese Schnittstelle soll vor allem Benutzer informieren, die nicht in Hörreichweite der Rauchmelder und somit nicht unmittelbar gefährdet sind.

Das GAP-Advertisment wird besonders für GATT-Services benötigt (siehe Unterabschnitt 3.2.2).Das Framework implementiert einen Service mit zwei Characteristics. Die Mesh-Metatdata-Characteristic enthält die Konfigurationsparameter des Meshs:

- Die Access Address (Byte 1-4 im Advertising Packet). Sie ist für GAP-Advertisments immer 0x8E89BED6. Für die Mesh-Advertisments kann sie aber abweichen.
- Die Länge des Intervalls I_{min}.
- Die Anzahl der maximal möglichen Values, die die Node speichern kann.
- Der BLE-Channel auf dem das Mesh agiert.

Die Mesh-Values-Characteristic erlaubt den Zugriff auf Werte (Einfügen und Updaten) und das Abfragen der Persistenz und des Übertragungsstatus. Der Übertragungsstatus ist rein informativ. Die Persistenz spielt nur eine Rolle wenn es mehr Mesh-Values gibt als der Cache umfasst, was im Rauchmelder-Mesh nicht der Fall ist. Das Setzen eines bestimmten Wertes von Außerhalb des Meshs eröffnet jedoch die Möglichkeit, das Mesh bis auf die tatsächlich auslösende Node (für eine Weile, siehe oben) stummzuschalten und einen Testalarm auszulösen ohne direkt mit einer Node interagieren zu müssen. Um einen Wert zu setzen, muss sich ein Central-Device mit der dem GATT-Server der Node verbinden. Dann muss ein Byte-Array von mindestens 5 Byte Länge in die Characteristic geschrieben werden. Dieses Byte-Array enthält den Code für die Operation (1. Byte, 0x00 für Value Set), das Handle (2. und 3. Byte),

die Länge des Wertes der geschrieben werden soll (4. Byte) und den Wert selbst (5. bis maximal 28. Byte).
Diese Charakteristic ist ein Sicherheitsrisiko, weil die Implementierung keine Authentifizierung enthält. Zwar werden alle neuen und alle nicht validen Werte wie oben beschrieben ausgefiltert, ein beliebiger Adversary kann trotzdem einen Falschalarm auslösen oder das Netz im Alarmfall stummschalten. Die Characteristic soll mit Verweis auf diese Schwäche trotzdem in der Implementierung enthalten bleiben. Dadurch können auch nicht spezialisierte BLE-Centrals mit dem Mesh interagieren. Das erleichtert speziell die Entwicklung und wissenschaftliche Arbeit mit dem System. Für ein Produktivsystem sollten besonders der GATT-Server entfernt werde. Zum einen ist eine GATT-Verbindung nachteilig hinsichtlich der der Timeslot-Allokierung, zum anderen wäre Authentifizierung für die Sicherheit relevant, was wiederum Mehraufwand für den nRF und damit erhöhten Verbrauch bedeutet. Eine Alternative dazu wäre eine Mesh-Node als integriertes Gateway zu benutzen. Dieses Gateway tritt dann an Stelle des in Abschnitt 4.3 erwähnten BLE-Interfaces und erlaubt dem Kommunikationsmodul Zugriff über Pin-I/O.

4.3 Interaktion mit der Außenwelt

Die meisten etablierten Funk-Module für Rauchmelder bauen auf Technologien im 433 MHz ISM-Band auf. Der Grund, warum für dieses Projekt Bluetooth Low Energy gewählt wurde ist unter anderem, dass auf das Netz einfach zugegriffen werden kann. Jedes Gerät, das den BLE-Standard implementiert, kann auf die GAP- und GATT-Server der Mesh-Nodes zugreifen. Dazu zählen Smartphones und Tablets als auch dedizierte BLE-Central-Devices wie BLE-Sniffer oder integrierte Gateway-Lösungen als Interface für andere IoT-Netzwerke beziehungsweise das S̈mart Home.̈ Für die Verwaltung des Netzwerkes von außen wurde ein Konzept für ein Gateway entworfen und beispielhaft in Form einer Android-App realisiert.

Das Konzept hat drei Komponenten:

- Ein BLE-Interface, dass permanent alle GAP-Advertisments scannt, die Advertisments des Meshes herausfiltert und die Verbindungsinformation speichert. Es ist wichtig dass das Interface immer Scannt und eine Liste der aktiven Nodes führt. Mobile Central-Devices scannen oft nur in Intervallen weil BLE-Emfänger sonst unnötig viel Strom verbraucht. Das GAP-Advertisment der Mesh-Nodes wird aber nur einmal pro Mi-

nute gesendet. Das kann zu unnötigen Verzögerungen führen, wenn das Scan-Intervall nicht mit dem Sendezeitpunkt zusammenfällt. Mit den zwischengespeicherten Verbindungsdaten kann eine GATT-Verbindung aufgebaut werden, um Mesh-Values in die entsprechende Characteristic schreibt. Dazu muss, trifft eine Verbindungsaufforderung am BLE-Interface ein, nicht auf den nächsten Sendezyklus der Mesh-Node gewartet werden.

- Ein UI-Modul, etwa die S̈mart Home¨-Software oder eine App durch die die Verwaltung des Netzwerkes geführt wird.
- Ein Kommunikationsmodul, das Befehle zwischen den anderen Modulen austauscht.

Die Android-App soll all der Einfachheit halber alle drei Module implementieren.

Als UI-Modul dient ein Fragment mit vier Views. Eine einfache Text-View zeigt den Alarmstatus in drei Stufen an: Uninitialized, Quiet und Alarm, zwei Button-Views bieten Schaltflächen um die Rauchmelder stummzuschalten und einen Testalarm auszulösen und eine List-View zeigt alle empfangenen Advertisments mit Namen und Manufacturer-Data an (siehe Abbildung 4.5).

Als BLE-Interface wurden zwei Services implementiert. Der ScanService läuft immer und liest alle gesendeten Advertisments. Empfängt der Service ein Advertisment, wird ein ScanResult Objekt an einen Callback übergeben, der Zugriff auf das Kommunikationsmodul hat. Der BleGattConnectService wird vom Kommunikationsmodul gestartet und bekommt von ihm die Liste der gespeicherten Verbindungsdaten und den auszuführenden Befehl. Der Service probiert dann, mit einem Gerät der Liste eine GATT-Verbindung aufzubauen. Ist dies erfolgreich, sucht er Anhand der UUIDs den Service (0000fee4-0000-1000-8000-00805f9b34fb), und daraus die Value-Characteristic (2a1e0005-fd51-d882-8ba8-b98c0000cd1e) heraus. Hat er die Characteristic gefunden, kann er das Bytearray mit dem entsprechenden Befehl schreiben. Die möglichen Befehle sind (0x00, 0x01, 0x00, 0x01, 0x01) für T̈estalarm auslösenünd (0x00, 0x01, 0x00, 0x01, 0x00) für M̈esh temporär stummschalten.̈ Danach trennt der Service die GATT-Verbindung wieder. Der Service wirft einen Fehler wenn Service oder Chatracteristic nicht gefunden wurden, das Schreiben des Befehls fehlschlägt oder keine Verbindungsdaten im Speicher sind.

Das Kommunikationsmodul besteht aus einem Singleton und einem LocalBroadcastManager. Das Singleton führt eine Liste mit den aktuellsten gefun-

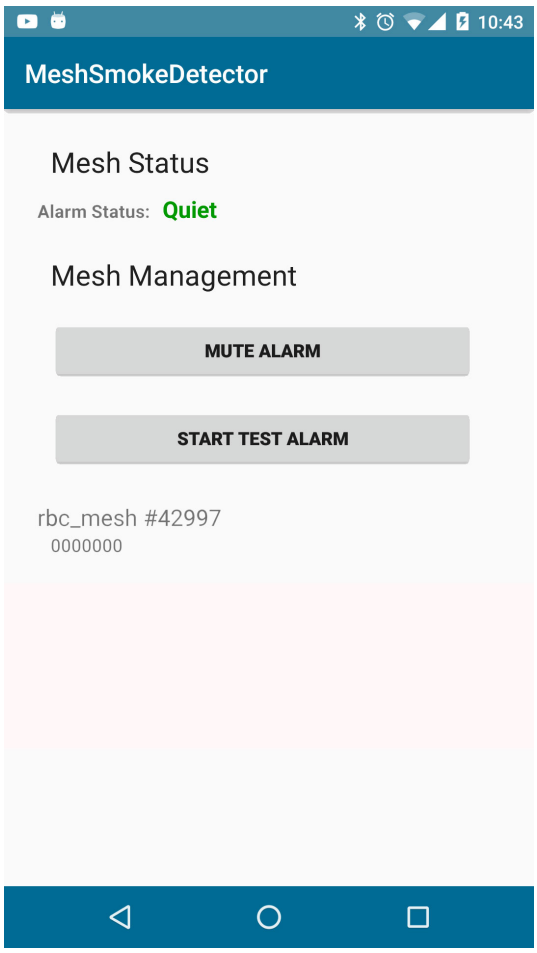

Abbildung 4.5: Das UI-Fragment der Android-App

denen ScanResults für jede Mesh-Node und sowohl ScanService als auch UI-Fragment greifen darauf zu. Ein ScanResult-Objekt beinhaltet sowohl die Daten des Advertisment wie Name der Node und die Manufacturer-Data, als auch die Verbindungsdaten notwendig zum Verbindungsaufbau. Wird der ScanCallback aufgerufen, speichert dieser das erhaltene ScanResult im Singleton. Dort wird aus der Liste der gespeicherten ScanResults das passende Result mit dem neueren ersetzt oder gegebenenfalls hinzugefügt. Danach wird der Alarmstatus aus den Manufacturer-Data geparst und über den BroadcastManager ein Intent an das Fragment geschickt. Das Fragment kann daraus die UI anpassen und im Alarmfalle den Benutzer informieren. Wird von Benutzer einer der Buttons gedrückt, sendet der BroadcastManager einen Intent mit dem zum Button gehörenden Befehl an den BleGattConnectService.

Kapitel 5

Evaluation

Die zweite von Strasser et al. definierte fundamental challenge für sicherheitskritische Anwendungen für WSNs ist geringe Latenz bei der Übertagung von Updates. Die Performance des entwickelten Systems soll in diesem Kapitel gegen diese Anforderung evaluiert werden. Dazu wurde an Hand der Normen und Gesetze ein Verteilungsschema ermittelt, das als Grenzwert für die zu erwartende Leistung dient. Darauf aufbauend werden konkrete Anforderungen über die Performance des Meshs aufgestellt, durch automatisierte Tests geprüft und anschließend ausgewertet werden.

5.1 Anforderungen an die Signalübertragung

Die Landesbauverordnungen schreiben in allen Bundesländern in Deutschland vor, dass Rauchmelder in allen Schlaf- und Kinderzimmern sowie Fluren, die als Rettungswege dienen, verbaut werden. Diese Räume müssen auch mit den Fluren oder zumindest miteinander verbunden sein. Man kann also annehmen, dass für jeden verbauten Rauchmelder im daneben liegenden Raum auch ein Rauchmelder ist.
Die DIN 14676 schreibt vor, wo im Raum der Rauchmelder verbaut werden muss. Anhand dieser Vorschriften kann der maximale Abstand zwischen zwei Meldern bestimmt werden. Ohne Betrachtung von Ausnahmeregelungen bestimmt die Norm zwei allgemeine Regeln:

- In Räumen muss 1 Rauchmelder pro $60m^2$ Fläche mittig an der Decke montiert werden.
- In Fluren bis maximal 3m Breite dürfen maximal 15m Abstand zwischen zwei Rauchmeldern sein. Der Abstand zur Stirnseite eines Flures darf maximal 7,5m betragen. Flure die Breiter als 3m sind werden als Räume betrachtet.

Zu den Ausnahmeregelungen zählen unter anderem Wandmontage, schräge Decken und L-förmige Räume. Alle Ausnahmeregelungen schränken die maximale Distanz weiter ein, weswegen sie hier nicht weiter beachtet werden.

Aus den allgemeinen Regeln ergibt sich eine theoretische maximal erlaubte Distanz zwischen zwei Rauchmeldern von 17,5m. Das ist dann der Fall, wenn ein Raum mit $60m^2$ Fläche und 3m Breite an die Stirnseite eines Flures angrenzt. Für eine allgemeine Hypothese eignet sich allerdings die maximale Distanz zwischen Rauchmeldern in Fluren mit 15m besser.
Daraus ergibt sich die erste Anforderung: Eine Mesh-Node kann ein Update innerhalb der ersten zwei Sendeintervalle empfangen, wenn die nächstliegende Node bis zu 15m entfernt ist.

Die Reichweite der Signale ist abhängig von der Sendeleistung der Antenne und nimmt über Distanz ab. Zusätzlich haben bestimmte Materialien einen Einfluss auf die Signalstärke, genannt Dämpfung [4]. Besonders deutlich ist diese Dämpfung bei Baumaterialien wie Stahl, Mauerwerk und Benton, aber auch bei vorkommen von Wasser wie bei hoher Luftfeuchtigkeit oder bei Menschen. Das Netzwerk soll aber auch bei bestehender Dämpfung funktionieren, besonders wichtig dabei sind Decken und Wände. Ein weiterer Faktor dabei sind Reflexionen. Wird ein Signal von der Wand oder Decke reflektiert, erreicht es den Empfänger nicht.
Es soll also gezeigt werden, dass ein Update trotz Dämpfung und Reflexion von Wänden und Decken bis 3 Meter Höhe in den ersten zwei Sendeintervallen empfangen werden kann.

Die zentrale Funktion des Meshs ist das weiterleiten von Nachrichten an alle Nodes im Netzwerk. Dabei erhöht sich die Gesamtverzögerung für jeden Hop, also für jede Weiterleitung von einer Node an die Nächste. Bei Rauchmelder-Netzwerken ist es fundamental wichtig, dass die Verzögerung selbst bei großen Netzwerken niedrig ist. Die Mesh-Architektur ist dafür gut geeignet, da selbst große Netzwerke mit wenigen Hops vollständig geflooded werden. Verstärkt wird dieser Effekt dadurch, dass Gebäude meist mehrere Stockwerke haben.
Daraus ergibt sich folgende dritte Anforderung: Die Verzögerung des Meshs beträgt bei 3 Hops mit je 10m Abstand nicht mehr als 1000ms.

5.2 Testmethodik

Um die Erfüllung drei definierten Anforderungen zu zeigen wurde eine Reihe von Tests durchgeführt. Dabei wurden folgende Szenarien geprüft:

- Für die erste Anforderung an die Latenz in einem Single-Hop-Szenario wurden die nRF-Sender und Empfänger ohne den angebauten Rauchmelder in Abständen von 10cm, 3m, 7.5m, 15m und 20m Entfernung voneinander platziert.
- Für die zweite Anforderung wurden Sender und Empfänger mit 3m Abstand voneinander platziert. Der Ort wurde so gewählt, dass sich dabei einmal eine Wand und einmal eine Raumdecke zwischen den Nodes befand.
- Für die dritte Anforderung zur Latenz ein Multi-Hop-Szenario getestet. Dabei wurden in vier Tests je 3 Mesh-Nodes mit 7.5m und 10m Abstand jeweils sowie 4 Nodes mit 7.5m und 10m Abstand jeweils verteilt. Die Distanzen wurden kleiner gewählt als die erwartete Sendestärke um mögliche Verbesserungen der Latenz durch "Überspringenëiner Node zu untersuchen. Das heißt, dass eine weiter entfernte Node die Updates nach einem anstatt zwei Hops empfängt.

Bei allen Tests wurden die Antennen in die gleiche Richtung ausgerichtet. Die Tests zu Anforderung 1 und 3 wurden in einem Flur mit rund 3m Breite durchgeführt, für Anforderung 2 in offenen Räumen. Um den Einfluss äußerer Faktoren zu minimieren wurden für jedes Szenario mindestens 50 Tests im Abstand von 20 Sekunden durchgeführt. Als Auslöser wurde eine Mesh-Node so modifiziert, dass sie über einen Timer alle 10 Sekunden den Alarmstatus ändert. Die Latenz wurde gemessen, indem ein Arduino Micro den PO07 des Empfängers abtastete. Die Timer des Senders und des Arduino wurden gleichzeitig über einen Button gestartet und beide MCUs nach jedem Testlauf resettet. Die Latenz ergibt sich dann aus dem gemessenen Zeitpunkt, an dem der PO07 des Empfängers auf HIGH wechselt modulo 20000. Um Differenzen zwischen den Timern zu eliminieren wurde der Sender direkt an den Arduino angeschlossen. Die gemessenen Differenzen für jedes Intervall wurden anschließend von den Testergebnissen vor der Auswertung abgezogen. Die gemessenen Ergebnisse sind auf 1ms genau, werden mit ihrem Timestamp in einem Logfile gespeichert und mit Python analysiert.

5.3 Diskussion der Ergebnisse

In Abschnitt 4.2 wurde das Sende- und Empfangsverhalten der Mesh-Nodes beschrieben. Dabei wurden die Sendezeitpunkte nach dem Auslösen des Rauchmelders in Abhängigkeit des Wiederholungsindexes r bestimmt als:

$$Bereitstellungszeitraum + t_r, t_r \in [\frac{2^r * I_{min}}{2}, 2^r * I_{min}], r \in \{0, 1, 2...\}$$

Beachtet man das Erhöhen von I_{min} nach dem ersten Intervall r=0 auf die nächstliegende 2er-Potenz (hier 128), ergeben sich für $I_{min} = 100ms$ die akkumulierten Intervallgrenzen seit Auslösen von

$$[I_{ru}, I_{ro}] = [I_{r-1} + \frac{2^r * I_{min}}{2}, I_{r-1} + 2^r * I_{min}]$$

$$= \{[50, 100], [228, 356], [612, 868], [1380, 1892], [2916, 3940], [5988, 8036], ...\}$$

Anhand dieser Intervalle kann jede gemessene Latenz einem dieser Intervalle zugeordnet werden. Daran wird mit hoher Wahrscheinlichkeit erkannt, in welchem Intervall das erste empfangene Packet gesendet wurde und wie hoch etwa die Bereitstellungszeit ist. Wurde ein Paket aus dem ersten Intervall nicht empfangen gilt es als verloren. Ein Verlust kann dann auftreten, wenn der Empfänger entweder während des Zündzeitpunktes keinen Timeslot reserviert hatte (etwa wenn der Empfänger selber ein GAP-Advertiment sendet) oder wenn das Packet durch Dämpfung oder Interferenz nicht empfangen wurden konnte. Ein Packet gilt als empfangen im Intervall I_r wenn

$$t <= I_r + \frac{2^{r+1} * I_{min}}{2},$$

also bevor das Sendeintervall I_{r+1} beginnt.

An den Ergebnisse der Single-Hop-Tests (siehe Tabelle 5.1) kann man mehrere Beobachtungen machen:
Auf bis zu 15 Meter Abstand werden mindestens 91% der Pakete aus dem ersten Sendeintervall empfangen. In Hinsicht auf Anforderung 1 wurden 96% der empfangenen Pakete im ersten oder zweiten Sendeintervall gesendet. Das Update wurde in jedem 10-Sekunden-Zyklus empfangen. Wird die Distanz weiter erhöht sind die Ergebnisse deutlich schlechte: In 54% der Sendezyklen wurde das Update nicht empfangen. Die Anforderung wird hier nur 61% der Fälle erfüllt. Bei Distanzen von bis zu 7.5 Meter werden bis zu 98% der Pakete aus dem ersten Intervall empfangen. Die verlorenen Pakete können also eher dem Fakt zugeordnet werden, dass der Empfänger nicht immer Zugriff auf das Radio-Modul hat als das Pakete aufgrund von Dämpfung oder Distanz nicht empfangen wurden. Der scheinbare Anstieg der Durchschnittslatenz ist mit hoher

Tabelle 5.1: Single-Hop Latenz der empfangenen Pakete über verschiedene Distanzen

	Mean	Median	t<I_{2u}	I_{2u}<t<I_{3u}	I_{3u}<t	Verlust
10 cm	88ms	94ms	100%	0%	0%	0%
3 m	92ms	91ms	98.82%	1.18%	0%	0%
7.5 m	103ms	99ms	98.82%	1.18%	0%	0%
15 m	504ms	83ms	93.24%	5.41%	4.05%	0%
20 m	1152ms	400ms	46.43%	17.86%	39.29%	54%

Wahrscheinlichkeit auch dadurch Begründet und damit zufällig unterschiedlich. Bei einem Advertising-Intervall von 200ms findet also in 89% der Fälle mindestens ein Advertising während des ersten Sende-Intervalls (50ms-228ms) statt, in dem das Programm den Timeslot zum Empfangen an das Softdevice verliert. Das GAP-Advertising dauert nur wenige Millisekunden insgesamt und überschneidet sich daher nicht oft mit dem Sendezeitpunkt. Verliert das Programm den Timeslot für 2 Sekunden, fällt dies mit dem Sendezeitpunkt in 0.88% der Fälle zusammen. Diese Zufälligkeit lässt sich durch die Software nicht beeinflussen, da sowohl der Sendezeitpunk innerhalb des Sendeintervalls laut Trickle, als auch der genaue Zeitpunkt des GAP-Advertisments laut BLE-Core-Specification im Bereich von 10ms randomisiert wird .
rechnung einfügen? 114 intervalle mit 2ms, davon 25 außerhalb intervall, bleiben 89 intervalle von 2ms innerhalb von Imin = 89/114*1/89 = 0.88%
Die Wahrscheinlichkeit dafür, dass ein Paket im ersten und zweiten Intervall zufällig verloren wird ist aber verschwindend gering, so dass hier davon ausgegangen werden muss, dass Pakete aufgrund anderer Ursachen nicht empfangen wurden. Bei 15 Metern Abstand ist die Verlustrate des ersten und zweiten Intervalls aber so hoch, dass sie nichtmehr nur Zufällig sein kann. Sie muss sich hier auch aus Verlust durch Interferenzen, Distanz und Dämpfung zusammensetzen. Dadurch dass die Intervalllänge exponentiell anwächst haben selbst bei nur 9% Paketverlust im ersten Intervall die Outlier einen so großen Einfluss auf den Durchschnitt, dass dieser kaum geeignet dafür ist, Rückschlüsse auf die generelle Qualität zuzulassen. Dazu kommt die Randomisierung der genauen Sendezeitpunkte innerhalb von 50ms Intervalllänge. Ein genaueres Maß dafür ist der Median, also der Punkt der die untere und obere Hälfte der Messwerte trennt. Visualisiert werden können diese mit Boxplots (siehe Abbildung 5.1). Dabei ist der rote Strich der Median, die Box umfasst die mittleren 50% der

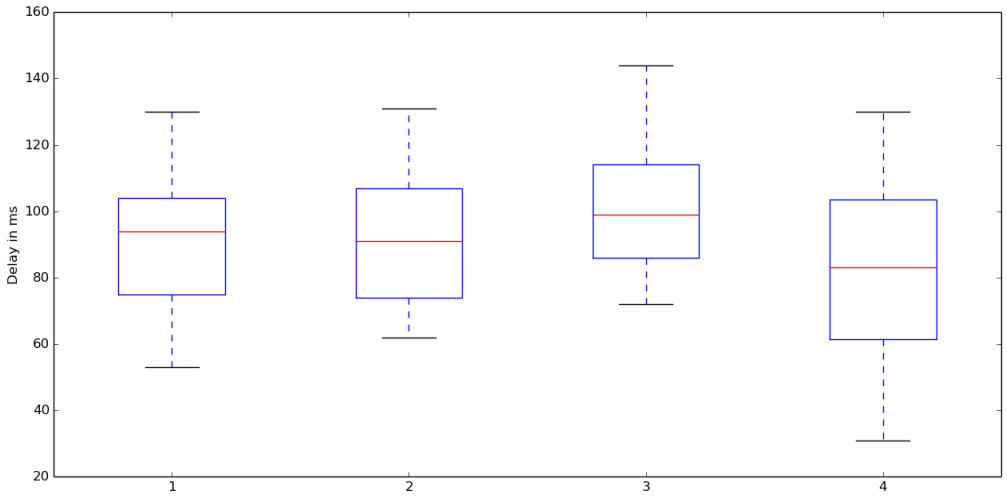

Abbildung 5.1: Single-Hop Latenz über (1) 10 Zentimeter, (2) 3 Meter, (3) 7.5 Meter, (4) 15 Meter

Tabelle 5.2: Latenzen mit Dämpfung über 3 Meter Distanzen

	Mean	Median	t<I_{2u}	I_{2u}<t<I_{3u}	I_{3u}<t	Verlust
ohne Dämpfung	92ms	91ms	98.82%	1.18%	0%	0%
durch Wand	83ms	75ms	96.43%	3.57%	0%	0%
durch Decke	111ms	102ms%	98.82%	1.18%	0%	0%

Daten, der Whisker umfasst die obere und untere Grenze (hier ohne Outlier). An den Plots kann man auch sehen, dass sich der Interquartilsabstand weitestgehend im ersten Intervall befindet.

Die oben beschriebenen Beobachtungen zu Durchschnitt und Verlustrate im ersten Intervall bezüglich Zufälligkeit lassen sich auch bei den Tests zur Dämpfung beobachten (siehe Tabelle 5.2 sowie Abbildung 5.2). Die tatsächliche Verlustrate des ersten Intervalls liegt um 2 Pakete bei dem Wand-Test höher als bei der Referenzgruppe ohne Obstruktion und dem Decken-Test. Das kann auf einen leicht erhöhten Einfluss von Dämpfungserscheinungen schließen lassen, bedarf für eine zuverlässige Schlussfolgerung aber umfangreicherer Tests. Was jedoch geschlussfolgert werden kann, ist dass der Einfluss der Dämpfung gering genug ist um der zweiten Anforderung gerecht zu werden.

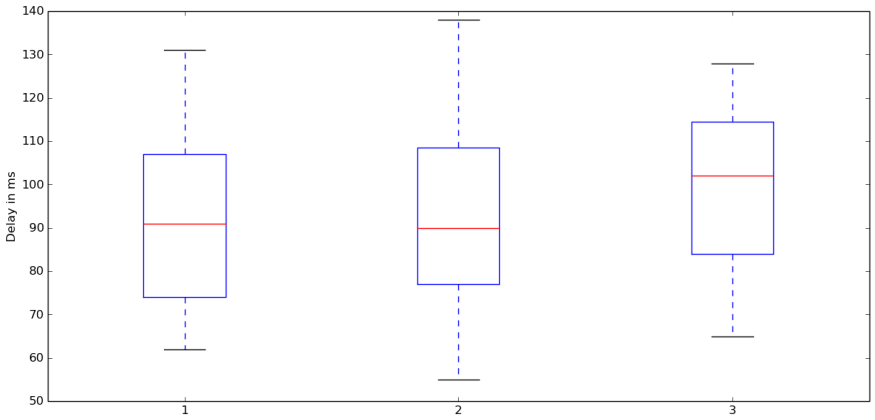

Abbildung 5.2: Latenz mit Dämpfung über 3 Meter (1) ohne Dämpfung, (2) durch Wand, (3) durch Decke

Tabelle 5.3: Multi-Hop Latenzen über verschiedene Distanzen

	Mean	Median	t<I_{2u}	I_{2u}<t<I_{3u}	I_{3u}<t	Verlust
2 Hops, 7.5m	92ms	91ms	100%	0%	0%	0%
3 Hops, 7.5m	124ms	108ms	95.83%	4.17%	0%	0%
2 Hops, 10m	168ms	139ms	82.81%	15.63%	1.56%	0%
3 Hops, 10m	91ms	91ms	100%	0%	0%	0%

Aus den Multi-Hop-Tests lassen sich noch weitere wichtige Eigenschaften beobachten. Die Tests mit 7.5m Distanz sollten vor allem den Einfluss zeigen, den das Überspringen von Updates auf die Latenz haben (siehe Tabelle 5.3). Bei zwei 7.5 Meter Hops ist die maximale Distanz zwischen den Endpunkten 15 Meter. Die Erwartung ist, dass die Latenz vergleichbar ist mit denen des Single-Hop-Tests auf 15 Meter Distanz und die Ergebnisse unterstützen das vollkommen. Bei 3 Hops über 7.5 Meter ist die Latenz aber auch vergleichbar mit denen des Single-Hop 15 Meter Tests, bei 3 Hops über 10 Meter sogar deutlich niedriger. Wie die Single-Hop Tests gezeigt haben ist es sehr unwahrscheinlich, dass auf über 15 Meter Distanz die Verlustrate des ersten Intervalls kleiner als 10% ist. Wird der Sendezeitpunkt nach Trickle RFC berechnet, dürfte die Latenz nach 2 Hops ohne Überspringen nicht niedriger als 100ms sein.

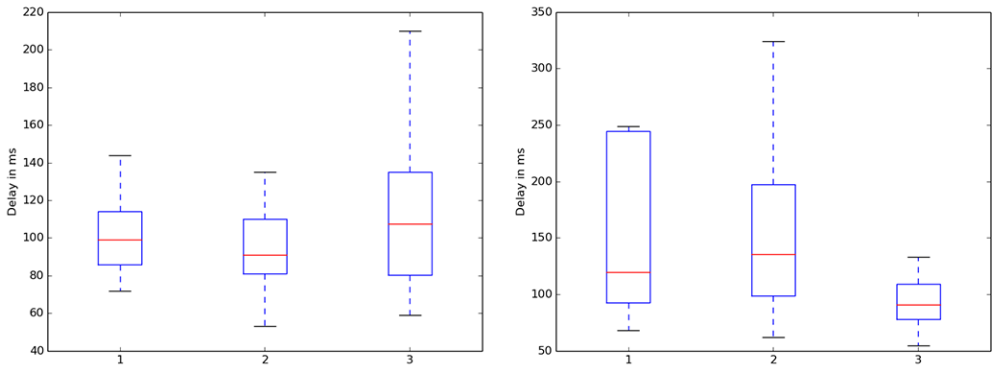

Abbildung 5.3: Multi-Hop Latenz über (links) 7.5 Meter und (rechts) 10 Meter mit Anzahl der Hops

Die Testergebnisse für 3 Hops über je 10 Meter Distanz zeigen also, dass der Trickel Algorithmus nicht genau nach Spezifikation implementiert sein kann, geht man davon aus dass über 30 Meter Distanz zumindest ein erkennbarer Teil der Pakete aus dem ersten Intervall verloren werden. Viel Wahrscheinlicher ist es aber, dass der Trickle für OpenMesh nicht getreu RFC adaptiert wurde und das erste Update nur erst im Anfangsintervall I_min sendet, wenn es sich um ein lokales Update handelt. Wird beim Weiterleiten das Update schon sofort gesendet, bevor das erste Intervall anfängt, kann die Latenz theoretisch nicht von denen der Single-Hop Tests unterschieden werden.

Eine zweite bemerkenswerte Beobachtung ist, dass die Anzahl der Hops zwischen den Endnodes weniger einfluss auf die Latenz hat als andere Störfaktoren. Da die Tests nicht in einer komplett kontrollierten Umgebung stattfanden, können diese Störfaktoren, wie zum Beispiel Interferenzen oder Menschen, die Ergebnisse teilweise deutlich behindern. Besonders deutlich wird das bei der Multi-Hop Latenz über 10 Meter Distanz (siehe Abbildung 5.3), wo der Signalverlust des ersten Intervalls über 2 Hops 17% beträgt. Beachtet werden dabei sollte das die Werte für verschiedene Hops in unterschiedlichen Durchläufen gemessen wurden und keine Rückschlüsse auf die Latenz verschiedener Hops innerhalb eines 10-Sekunden-Zyklus zulassen. In Hinsicht auf Anforderung 3 zeigen die Resultate aber, dass in einem Multi-Hop Szenario mindestens 98.44% der Pakete aus mindestens dem zweiten Sendeintervall empfangen wurden und keins der Updates innerhalb des Zyklus von 10 Sekunden verloren wurde.

Die durchgeführte Testreihe bestätigen also alle definierten Anforderungen an

die Funktionsfähigkeit des BLE-Meshs für Rauchmelder hinreichend genau. Jedoch werden die anderen beschriebenen Beobachtungen und Annahmen durch die Ergebnisse nicht genau nachgewiesen. Um den genau Einfluss des zufälligen Verlierens von Paketen zu zeigen müssen mindestens zwei parallele Empfänger pro Distanz verteilt werden. Dadurch kann gezeigt werden, dass der Verlust eines Paketes in einem Intervall abhängig vom empfangenden Gerät ist. Außerdem sollten die Zeitpunkte mitgeschrieben werden, wann und wie lange Timeslots vom Softdevice unterbrochen werden, so dass Rückschlüsse auf die genaue Wahrscheinlichkeit des zufälligen Verlierens gezogen werden können.

Um den Einfluss äußerer Störfaktoren genau zu untersuchen müssen die Tests in einer besser kontrollierten Umgebung stattfinden. Diese Umgebung muss vor allem von Fremdsignalen abgeschirmt werden, so dass andere Signale wie von BLE- oder Wifi-Geräten nur kontrolliert auftreten können. Im Rahmen dieser Arbeit stand eine solche Umgebung nicht zur Verfügung.

Kapitel 6

Ausblick

Piezo direkt ansprechen -> Verschiedene Tonhöhen, Tonfolgen etc. um zu zeigen wie weit der Ursprung weg ist / was das problem ist

Einbinden in bestehende IoT Systeme / Zugriff über Interwebz ??

6.1 Sicherheit vor Angriffen und Fälschung

neuer Sensor änmeldenän gateway –> sensor-ID muss bestätigt werden - mesh nimmt nur updates von

6.2 Energieverbrauch

Erweitern der Software um Batterie-Messungen. Dann Testen verschiedener Software/Hardware: nRF mit eigener Batterie und Software -> Nordic Beacon Example, Mesh Template (quasi ohne alles), mein SD-Mesh. Danach eventuell wenn die Zeit reicht nRF an 9V des SD anschließen und einfach mal schauen was passiert. Vergleich von openMesh mit implementierung vom neuen Standard + properitäre lösungen

6.3 Überwachung der Funktionstauglichkeit

Literaturverzeichnis

[1] 11 Internet of Things (IoT) Protocols You Need to Know About. http://www.rs-online.com/designspark/electronics/knowledge-item/eleven-internet-of-things-iot-protocols-you-need-to-know-about. letzter Zugriff 02.08.2016. 3.2.1

[2] DIN 14676:2012-09, Rauchwarnmelder für Wohnhäuser, Wohnungen und Räume mit wohnungsähnlicher Nutzung - Einbau, Betrieb und Instandhaltung. 3.1

[3] DIN EN 14604:2009-02, Rauchwarnmelder; Deutsche Fassung (EN 14604:2005). 3.1, 3.2.1

[4] Elektronik Kompendium, Funktechnik. http://www.elektronik-kompendium.de/sites/kom/0810301.htm. letzter Zugriff 07.08.2016. 5.1

[5] nRF OpenMesh. https://github.com/NordicSemiconductor/nRF51-ble-bcast-mesh. letzter Zugriff 05.08.2016. 4.2

[6] nRF51 Software Development Kit 8.1.0. http://developer.nordicsemi.com/nRF51_SDK/nRF51_SDK_v8.x.x/doc/8.1.0/index.html. letzter Zugriff 04.08.2016. 4.2

[7] Photoelectric Smoke Detector IC with I/O - MC145010. http://www.nxp.com/files/analog/doc/data_sheet/MC145010.pdf. letzter Zugriff 23.07.2016. 3.1

[8] S110 nRF51 SoftDevice Specification v2.0. http://infocenter.nordicsemi.com/pdf/S110_SDS_v2.0.pdf. letzter Zugriff 04.08.2016. 4.2

[9] Setting up the Timeslot API. https://devzone.nordicsemi.com/tutorials/16/. letzter Zugriff 02.08.2016. 3.2.2

[10] Verteilung von Android-Geräten nach API-Level. https://developer.android.com/about/dashboards/index.html. letzter Zugriff 02.08.2016. 3.2.1

[11] What's the third wire on a piezo buzzer? http://electronics.stackexchange.com/questions/18212/whats-the-third-wire-on-a-piezo-buzzer. letzter Zugriff 23.07.2016. 3.1

[12] Bluetooth Core Specification 4.2. https://www.bluetooth.com/specifications/adopted-specifications, Dezember 2014. 3.2.2

[13] Majid Bahrepour, Nirvana Meratnia, and Paul Havinga. Automatic Fire Detection: A Survey from Wireless Sensor Network Perspective. http://doc.utwente.nl/65223/1/Automatic_Fire_Detection.pdf, 2008. letzter Zugriff 13.07.2016. 2

[14] Wan Hazimah Wan Ismail, Herny Ramadhani Mohd Husny, and Norhaiza Ya Abdullah. Smoke Detection Alert System via Mobile Application. In *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication*, Danang, Vietnam, Januar 2016. 2

[15] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. RFC 6206, Trickle-Algorithm. https://tools.ietf.org/html/rfc6206, März 2011. 3.2.1

[16] Iftekharul Mobin, Abid Ar-Rafi, Neamul Islam, and Rifat Hasan. An intelligent fire detection and mitigation system safe from fire (sff). *International Journal of Computer Applications*, 133(6), Januar 2016. 2

[17] Tyco Fire Safety. Consultant's Guide for Designing Fire Detection Alarm Systems. http://tfppemea.com/en/_layouts/fsassets/Docs/Detection_FireClass/UKFireClassConsultantsGuide(LR).pdf. letzter Zugriff 23.07.2016. 3.1

[18] Mario Strasser, Andreas Meier, Koen Langendoen, and Philipp Blum. Dwarf: Delay-Aware Robust Forwarding for Energy-Constrained Wireless Sensor Networks. In *Proceedings of the 3rd IEEE international conference on Distributed computing in sensor systems*, Heidelberg, Deutschland, Juni 2007. 3.2.1

[19] Y. Zeng, S. Murphy, L. Sitanayah, T. Tabirca, T. Truong, K. Brown, and C. Sreenan. Building Fire Emergency Detection and Response using Wireless Sensor Networks. In *Ninth IT and T Conference, Dublin Institute of Technology*, Dublin, Irland, Oktober 2009. 2

[20] Lei Zhang and Gaofeng Wang. Design and Implementation of Automatic Fire Alarm System based on Wireless Sensor Networks. In *Proceedings of the 2009 International Symposium on Information Processing*, Huangshan, China, August 2009. 2