

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. Volker Rodehorst

Datum der Abgabe: 1. September 2016

# **Erklärung**

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

Weimar, 1. September 2016

.....  
Matti Wiegmann

## **Zusammenfassung**

Diese Bachelorarbeit befasst sich mit der Konzeption, Implementierung und Evaluation eines auf Bluetooth Low Energy basierenden Mesh-Networks für Rauchmelder.

Diese Arbeit versucht, Anforderungen an funkvernetzte BrandmeldeSysteme im Zeitalter des Internets der Dinge zu definieren und Schwächen der herkömmlich für diesen Zweck benutzen Technologien aufzuzeigen. Dazu werden verschiedene potenzielle Basistechnologien auf ihre Tauglichkeit für Sensornetzwerke im Kontext sicherheitskritischer Anwendungen analysiert. Aus den Funden der Analyse wird ein Konzept für die technische Realisierung eines ausgearbeitet, was den definierten Anforderungen entspricht. Weiterhin wird die Umsetzung dieses Konzepts in ein Mesh-Network für Rauchmelder beschrieben, bei dem jeder Rauchmelder als eine Node agiert. Außerdem wird anhand einer entwickelten App gezeigt, wie das Testsystem in ein bestehendes IoT- oder Smart-Home-System integriert werden kann.

Um die Einsatzfähigkeit dieses Testsystems in einem realistischen Szenario zu zeigen, wird eine Reihe von Mindestanforderungen für den Betrieb erarbeitet. Anhand der erarbeiteten Kriterien wurde das Testnetzwerk evaluiert und die Ergebnisse werden ausführlich beschrieben.

Abschließend werden noch bestehende Schwächen der Implementierung aufgezeigt sowie Möglichkeiten einer potenziellen Weiterentwicklung in verschiedenen Bereichen aufgezeigt.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Related Work</b>	<b>4</b>
<b>3 Vor betrachtung</b>	<b>7</b>
3.1 Aufbau und Funktionsweise von Rauchmeldern . . . . .	7
3.2 Wireless Sensor Networks für Sicherheitskritische Anwendungen . . . . .	10
3.2.1 Technologische Grundlagen . . . . .	10
3.2.2 Bluetooth Low Energy . . . . .	14
<b>4 Implementierung</b>	<b>17</b>
4.1 Konstruktion einer Mesh-Node . . . . .	17
4.2 Software Microprozessor . . . . .	20
4.2.1 Konzeption . . . . .	20
4.2.2 Trickle Implementierung . . . . .	22
4.2.3 Verarbeitung des Alarmfalles . . . . .	28
4.3 Interaktion mit der Außenwelt . . . . .	30
<b>5 Evaluation</b>	<b>34</b>
5.1 Anforderungen an die Signalübertragung . . . . .	34
5.2 Testmethodik . . . . .	36
5.3 Diskussion der Ergebnisse . . . . .	37
<b>6 Ausblick</b>	<b>43</b>
6.1 Sicherheit vor Angriff und Fälschung . . . . .	43
6.2 Energieverbrauch . . . . .	44
6.3 Mesh Supervision . . . . .	45
6.4 Fazit . . . . .	45
<b>Literaturverzeichnis</b>	<b>47</b>

# Kapitel 1

## Einleitung

Jedes Jahr sterben in Deutschland etwa 400 Menschen durch Brände [12]. Der überwiegende Teil davon nicht etwa, wie man vermuten könnte, am Feuer selbst, sondern durch den dadurch entstehenden Rauch. Der Rauch entsteht oft lange bevor ein Brand voll ausbricht und ist schwer zu bemerken, vor allem wenn die Opfer schlafen. In geschlossenen Räumen entsteht bei einem Schwelbrand vor allem Kohlenmonoxid (CO). CO ist ein Gas, welches schon bei wenigen Atemzügen zur Bewusstlosigkeit führen kann. Dementsprechend lebensrettend ist der Einsatz von Rauchmeldern. Gewöhnliche Rauchmelder haben allerdings noch Schwachstellen:

- Sie können überhört werden. Bricht der Brand zum Beispiel in einem anderen Stockwerk aus, kann es für die Bewohner schon zu spät sein. Bis sich der Rauch so weit ausgebreitet hat, dass ein Rauchmelder Alarm schlägt, der von ihnen gehört werden kann, können Fluchtwege bereits unzugänglich sein. Dieses Problem wird durch herkömmliche Funkrauchmelder adressiert. Diese senden in der Regel im 433 MHz ISM-Band über bis zu 100 Meter und lösen jeden Rauchmelder desselben Typs, der das Signal empfängt, ebenfalls mit aus. Die Funkmodule sind aber oft teuer und bieten von sich aus nur diese eine Zusatzfunktion.
- Die Bewohner sind Hilfebedürftig und können alleine nicht richtig auf den Alarm reagieren.
- Der Raum, in dem das Feuer ausbricht, ist aufgrund häufiger Fehlalarme nicht mit einem Rauchmelder ausgestattet. Das sind häufig Küche, Bad und Werkräume. Fehlalarme können beispielsweise durch zusätzliche Sensoren stark reduziert werden, wie in Kapitel 2 “Related Work“ und Abschnitt 3.1 “Aufbau und Funktionsweise von Rauchmeldern“ näher beschrieben ist.

- Das Feuer bricht in einem unbewohnten Gebäude, wie etwa einer Ferienwohnung, aus. Zwar besteht hier keine unmittelbare Gefahr für Menschenleben, aber es können erhebliche Sachschäden entstehen, bis der Brand entdeckt wird. Ein höher entwickeltes Rauchmeldesystem kann die Besitzer, über Internet- oder Mobilfunk, schon früher über das Feuer informieren.

Zur Lösung für die noch bestehenden Probleme bieten sich die Technologien des Internets der Dinge (IoT) an. Die Philosophie des IoT beschreibt die kabellose Vernetzung und Adressierbarkeit aller Sensoren (wie Licht-, Temperatur- und auch Rauchsensoren) und Aktuatoren (wie Lampen, Klimaanlagen und Lautsprecher) über den Internet-Protokollstack. Die am weitesten verbreitete Funktechnologie dafür ist Wireless LAN (Wifi). Wifi implementiert den IPv6-Stack, hat eine hohe Reichweite und Bandbreite. Genau diese Mächtigkeit birgt aber auch Wifis größte Schwächen: Energieverbrauch und Preis. Abgesehen davon sind IP-basierte Technologien auf dedizierte Router angewiesen, die die Kommunikation zwischen den Endgeräten regeln. Bei Wifi muss also immer ein Router in Reichweite jedes Endgerätes sein und die Router müssen miteinander in Verbindung stehen. Fällt ein Router aus, ist die Verbindung zwischen Teilen des Netzwerkes nicht mehr möglich.

Bluetooth Low Energy (BLE) wurde als alternative Technologie für Szenarien entwickelt, in denen vor allem Energieverbrauch wichtiger ist als eindeutige Adressierbarkeit und Bandbreite. Ursprünglich wurde BLE nur für ein Broadcaster-Observer-Schema entworfen. Dazu zählt hauptsächlich die Überwachung von Sensordaten in einem Personal Area Network (PAN), wie etwa einen Blutdruckmesser. BLE wurde wegen seiner Low-Power, Low-Cost Hardware schnell populär und wird mittlerweile von den meisten Verbrauchergeräten wie Smartphones und Tablets unterstützt. Bluetooth Low Energy ist für die Vernetzung von Rauchmeldern deutlich besser geeignet als Wifi. Der geringe Stromverbrauch behindert nicht die typische Laufzeit ohne Batteriewechsel von Rauchmeldern (in der Regel mehrere Jahre), ein regulärer Bluetooth System on a Chip (SoC) ist billiger als ein Wifi-Modul und in den meisten Haushalten sind bereits Geräte verfügbar, die die BLE-Rauchmelder ohne zusätzliche Netzwerk-Geräte überwachen können. BLE wird ständig weiterentwickelt und unterstützt mittlerweile mehr als simple Broadcaster-Observer-Interaktion. Es ist möglich, BLE-Peripherals nicht mehr nur zum Broadcasten von Daten zu verwenden, sondern auch, Broadcasts von anderen zu empfangen. Damit lässt sich ein Netzwerk von Rauchmeldern konstruieren, dass über Broadcasts in einer Mesh-Topologie von Endgerät zu Endgerät weiterreicht und so alle Rauchmelder im Falle eines Feuers auslöst, ohne auf dedizierte Router zurückgreifen zu müssen.

Mit Hilfe der besprochenen Ideen und Technologien können viele schwächen herkömmlicher Rauchmelder ausgeglichen werden. Der geringe Aufpreis für BLE-Integration und die Unabhängigkeit von dedizierten Netzwerkgeräten bieten Potenzial für eine weit höhere Verbreitung funkvernetzter Rauchmelder in allen Bevölkerungsschichten. Diese Arbeit soll zeigen, dass und wie ein Wireless Sensor Network (WSN) basierend auf Bluetooth Low Energy funktioniert, zuverlässig ist und die oben beschriebenen Probleme nicht-vernetzter Rauchmelder löst. Dazu sollen möglichst einfache, BLE-fähige Rauchmelder gebaut und vernetzt werden. Außerdem soll in einer Testreihe untersucht werden, ob die konstruierte Technik in einem realitätsnahen Szenario schnell und zuverlässig genug funktioniert, um dem Einsatz in einem lebensrettendem Bereich gerecht zu werden.

# 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-Sensornetzen.
- Erkennung von Feuern in Gebäuden und bewohnten Gebieten.

Eine gute Übersicht zu diesen Themen findet man in **Automatic Fire Detection: A Survey from Wireless Sensor Network Perspective** von Majid Bahrepour, Nirvana Meratnia und Paul Havinga [16]. Im Folgenden sollen einige für diese Arbeit besonders relevante Arbeiten kurz vorgestellt werden.

In **Building Fire Emergency Detection and Response Using Wireless Sensor Networks** [22] 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) und 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** [17] 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** [23] 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)** [19] 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 “Einleitung“ 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). Piezo-Summer sind Metallplättchen mit einer dünnen Schicht Piezo-Kristalle. Liegt daran Strom an, ziehen sich die Kristalle zusammen und biegen das Plättchen. Bei Wechselstrom biegt sich das Plättchen abwechselnd in beide Richtungen und erzeugt mit dieser Schwingung einen Ton. Die Sensoren können grob in zwei Kategorien unterteilt werden: Rauchsensoren und ergänzende Sensoren.

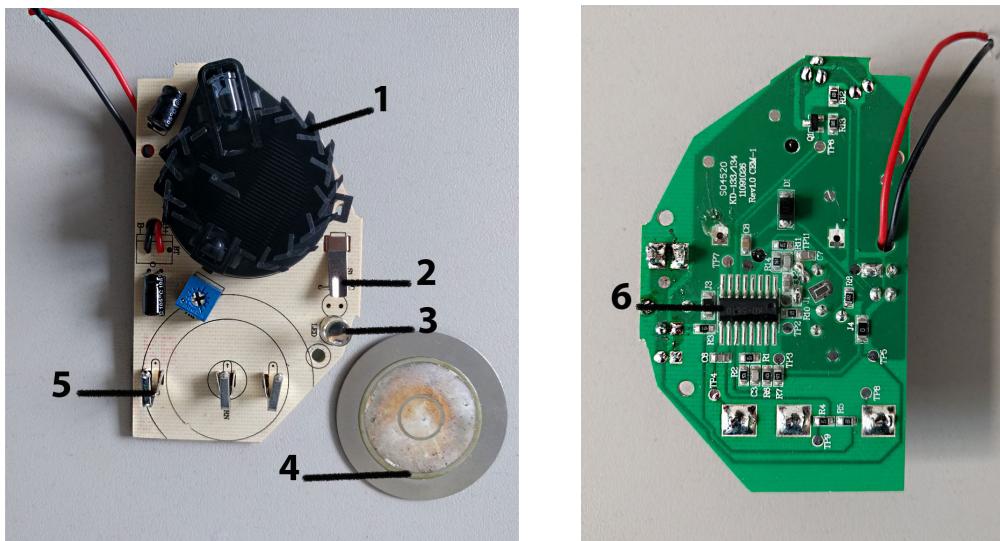
Bei Rauchsensoren gibt es zwei übliche Typen: Ionisationsrauchsensoren und photoelektrische (optische) Rauchsensoren [20]. Erstere ionisieren mit Hilfe radioaktiven Materials die Luft zwischen zwei Metallplatten. Eindringender Rauch unterbricht dabei den Fluss der Ionen und löst so den Alarm aus. Photoelektrische 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 Schmelzbränden die Rauchpartikel größer als bei offenem Feuer. Da diese bei Wohnungs- und Gebäudebränden sehr häufig sind und Ionisationsrauchsensoren aufgrund ihrer Technologie in Europa nicht vertrieben werden dürfen [3], sind photoelektrische Rauchsensoren der De-Facto-Standard. Photoelektrische Rauchsensoren sind allerdings anfällig für Falschalarm, ausgelöst durch Wasserdampf, unter anderem in Badezimmern, oder Rauch der beim Kochen entsteht. Um diese Schwächen auszugleichen werden in hochwertigen Rauchmeldern zusätzliche Sensoren verbaut. Dazu zählen Temperatursensoren, Kohlenmonoxidsensoren und Feuersensoren [20] (siehe Kapitel 2 “Related Work“).

Für Rauchmelder gelten in Deutschland zwei zentrale Normen. Die Norm DIN 14767 [2] regelt Einbau, Betrieb und Test und wird in Kapitel 5 “Evaluation“ näher beschrieben. Für Aufbau und Funktionsweise gilt die Norm 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, dass sowohl die folgende Analyse eines Testgerätes als auch das in Kapitel 4 “Implementierung“ vorgestellte Konzept 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 optischen 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. Bei nur 9 Volt Spannung würde ein 2-Pin-Summer aber nicht die Lautstärke, die die Norm vorschreibt, erreichen. Der Summer des Rauchmelders hat dafür einen dritten Pin. Dieser liefert ein Feedback-Signal, wodurch der Schallgeber mit Eigenfrequenz schwingen kann und so eine höhere Lautstärke erreicht [14].



(a) Oberseite: (1) Rauchsensor (geöffnet),  
 (2) Testschalter, (3) Warnanzeige (LED),  
 (4) Piezo-Element, (5) Signalgeber Pins.  
 (b) Unterseite mit Schaltkreis:  
 (6) IC ohne Abdeckung

**Abbildung 3.1:** Innenleben des Testrauchmelders.

- Dem Integrated Circuit (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 [9] 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.

Der Rauchmelder hat die folgende vier Verhaltensmuster [9]:

- **IDLE:** Der Signalgeber ist aus. Der Rauchmelder führt alle 38.9 bis 47.1 Sekunden eine Prüfung der Funktionsfähigkeit 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 Sicherheits-kritische Anwendungen

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

### 3.2.1 Technologische Grundlagen

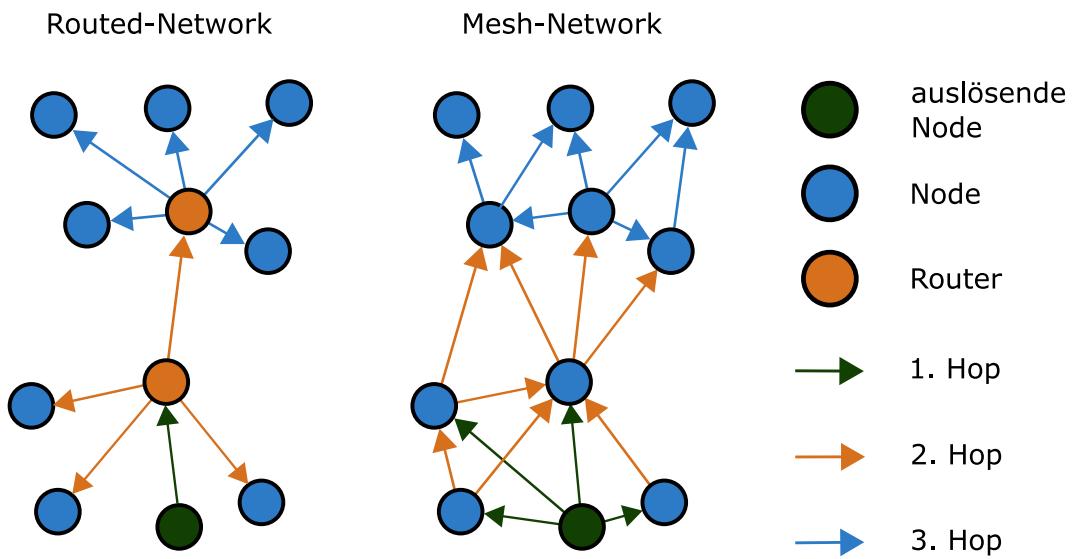
Die Idee hinter dem Internet der Dinge (IoT) ist, jedes Gerät eindeutig zu adressieren und zentral zu steuern. Die Schlüsseltechnologien dafür sind die Internettechnologien TCP/IP und Wireless LAN (Wifi). Dabei bekommt jedes Gerät (mit Sensoren/Aktuatoren) eine Adresse aus dem IPv6-Adressraum. Die Kommunikation 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, eine 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, signifikant länger als mit einem Wifi-Sender [5].

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) ist ein Netzwerk-Protokoll. Es basiert auf IPv6 und ist für Low-Power Geräte und Mesh-Netzwerke entwickelt worden. Der Standard für 6LoWPAN ist der RFC6282.
- **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 Broadcast-Mesh-Netzwerke bereits, für BLE ist nativer Support noch in Entwicklung. Bluetooth Low Energy 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 (“Jelly Bean”, released im November 2012), das umfasst heute über 80% alle Geräte [13]. In industriellen und öffentlichen Gebäuden werden meistens verkabelte Brandmeldeanlagen vorgeschrieben [3], daher ist das wichtigste Einsatzgebiet von Funkrauchmeldern der private Sektor. Da BLE-fähige Endgeräte gerade im privaten Sektor viel weiter verbreitet sind als Zigbee- oder Z-Wave-Gateways, bietet sich Bluetooth Low Energy als technologische Basis für diese Arbeit an.



**Abbildung 3.2:** Datenübertragung zwischen Nodes in Routed-Networks und Broadcast-Mesh-Networks

Die meisten der erwähnten Technologien für das Internet der Dinge setzen auf IPv6 und zählen somit zu den Routed-Networks (siehe Abbildung 3.2). Das heißt, dass jedes Gerät eine eigene Adresse hat. Die Daten, welche 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 geeignet, wenn Absender und Ziel genau identifiziert werden müssen. Ist das nicht der Fall und die Identifizierung als Absender oder Ziel ist nicht notwendig, verbleiben durch dieses System jedoch 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 “Related Work“).

Sicherheitskritische Anwendungen, wie zum Beispiel Rauchmeldernetzwerke,

sind nicht darauf angewiesen, dass jedes Gerät eindeutig adressierbar ist. Die Aufgabe des Netzwerkes ist, dass 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“ [21] 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 (siehe Abbildung 3.2). Damit das Netzwerk allerdings funktionsfähig bleibt, müssen in regelmäßigen Abständen Status-Updates 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) [18] 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 Flags, ob es sich um ein Update oder einen alten Wert handelt. Ist der empfangene Wert ein Update, also eine neuere Version, wird das Broadcasting-Intervall wieder auf den Minimalwert zurückgesetzt und so sichergestellt, dass Updates so schnell wie möglich über das Netzwerk propagiert werden (siehe Unterabschnitt 4.2.2 “Trickle Implementierung“). Für diese Arbeit soll also ein auf Trickle basierendes Broadcast-Flooding-Mesh-Network für Rauchmelder mit Bluetooth Low Energy Kommunikationstechnologie entwickelt und seine Einsatzfähigkeit verifiziert werden.

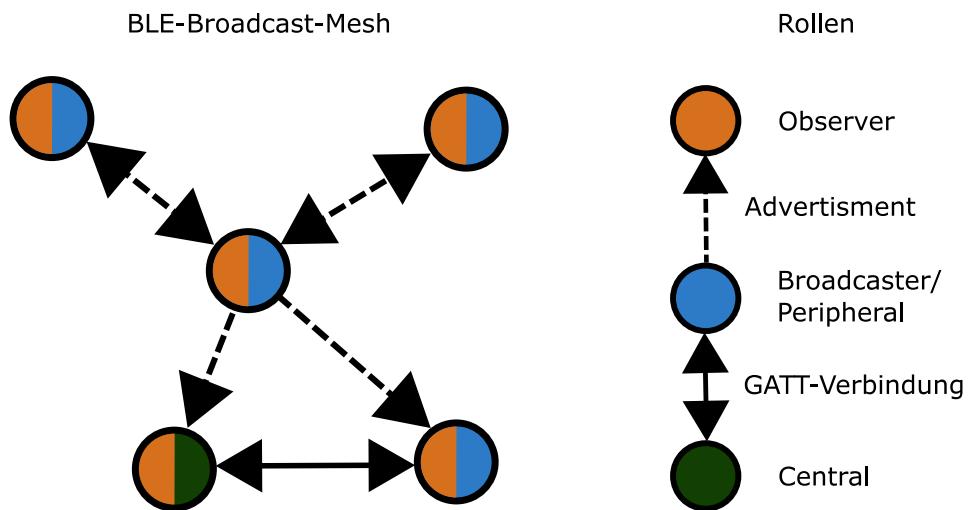
### 3.2.2 Bluetooth Low Energy

Da Bluetooth Low Energy ein offener Standard ist, beschrieben in der Bluetooth Core Specification [15], 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 “Implementierung“).

Da der BLE-Protokoll-Stack [10] sehr komplex ist, sollen die für diese Arbeit wichtigsten Konzepte kurz erläutert werden. Dazu zählen vor allem die Top-Level Protokolle GAP und GATT.

Die Specification definiert zwei Patterns, um die Rolle eines BLE-Gerätes zu klassifizieren: Broadcaster-Observer-Pattern sowie Central-Peripheral-Pattern (siehe Abbildung 3.3). Ein Broadcaster überträgt seine Daten, wie etwa Werte angeschlossener Sensoren, in Form einer fest definierten Datenstruktur genannt Advertisments (siehe Abbildung 4.7). Der Observer scannt nach den omnidirektional gesendeten Advertisments. In einem Mesh-Netzwerk ist jede Node sowohl Broadcaster als auch Observer. Das andere Pattern ist das Central-Peripheral-Pattern. Darin ist ein BLE Gerät entweder ein “Central“ oder ein “Peripheral“. Peripherals sind dabei verteilte, low-energy Geräte die an den jeweiligen Sensoren oder Aktuatoren angeschlossen sind. Centrals sind die mächtigeren, zentralen Geräte. Sie sollen mit bestimmten Peripherals kommunizieren, indem sie eine Verbindung aufbauen und aus dem verbundenen Peripheral Daten auslesen sowie Daten schreiben. In einem Mesh-Netzwerk sind alle Nodes des Peripherals. Da BLE-Peripherals in der Regel sowohl Peripheral als auch Observer sind, wird im weiteren Verlauf nur noch von Peripherals anstatt von Peripheral und Observer gesprochen.

Die Kommunikation zwischen BLE-Geräten wird über zwei sogenannte Profile geführt. Das Generic Access Profile (GAP) managt Advertising und Verbindungsfähigkeit. Das Advertisement 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 Advertisement 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 wäh-



**Abbildung 3.3:** Rollen von BLE-Geräten und Kommunikationswege in einem BLE-Broadcast-Mesh

rend 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 vorkompliiert 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 [11]. Diese API ermöglicht, dass mehrere Advertising-“Threads“ nebeneinander stattfinden und von den anderen Peripherals mit der gleichen Softdevice Version auch empfangen werden können. Das GAP-Advertising, das von Centrals gescannt und zum Verbindungsauftakt 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 drei hierarchisch verschachtelte Container: Profiles, Services und Characteristics. Dabei kann 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

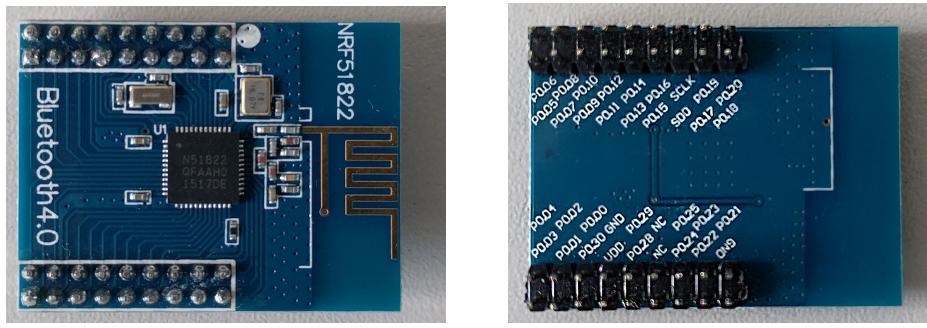
Aufbauend aus den Ergebnissen der Vorbetrachtung (siehe Kapitel 3 “Vorbertragung“) soll in diesem Kapitel aufgezeigt werden, wie ein funktionsfähiges Rauchmelder-Netzwerk auf Basis eines BLE-Meshs gebaut und programmiert werden kann. Dazu sollen zuerst handelsübliche Rauchmelder an BLE-Sender angeschlossen werden. Die Sender wurden so programmiert, 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 “Aufbau und Funktionsweise von Rauchmeldern“).

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 (siehe Abbildung 4.2). 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 Mikroprozessor 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 die Antenne fest in demselben Schaltkreis inte-



(a) Unterseite mit Chip und Antenne.

(b) Oberseite mit Pin-Headers.

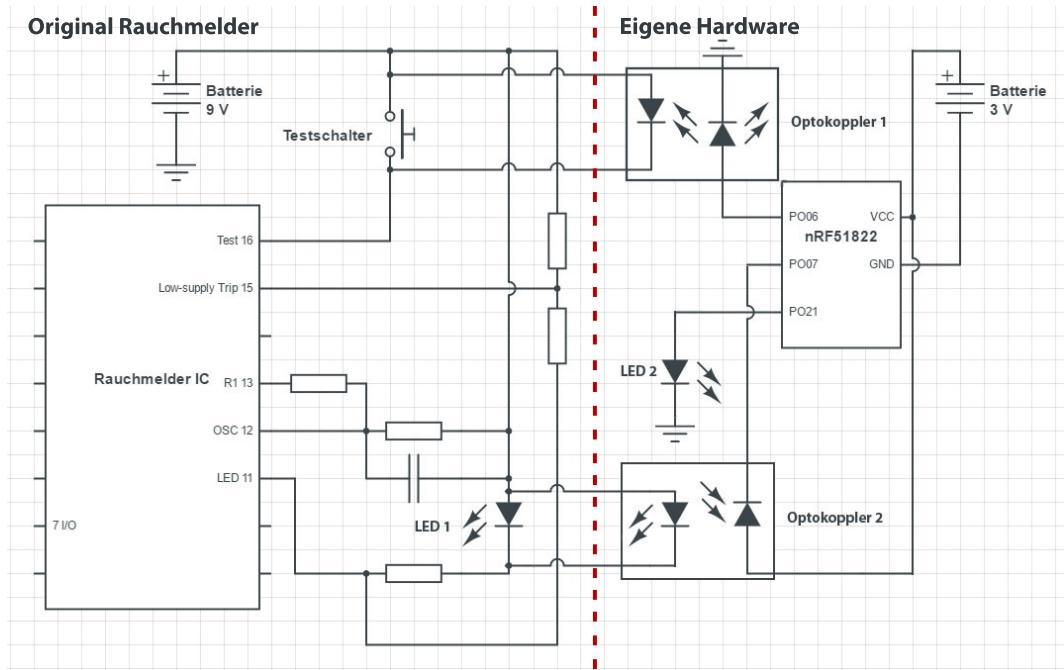
**Abbildung 4.1:** nRF51822 Breakout-Board.

griert werden, ist diese Methode aufgrund der Zuverlässigkeit der Signale vorzuziehen. 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 36 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, der das Auslösen des Rauchmelders abtastet und einer, der 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.

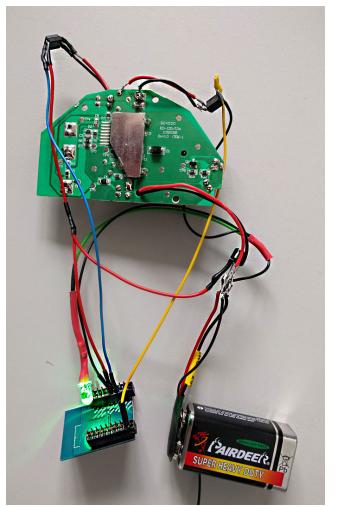
Welcher der frei belegbaren Pins des nRF51822 genutzt wird, spielt keine Rolle, allerdings gibt es durch Nordics Board Support Package (BSP) einen Status quo. Im BSP werden unter anderem statische Bezeichner für die Pins der offiziellen Nordic Hardware vergeben. Zu den im BSP definierten Bezeichnern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



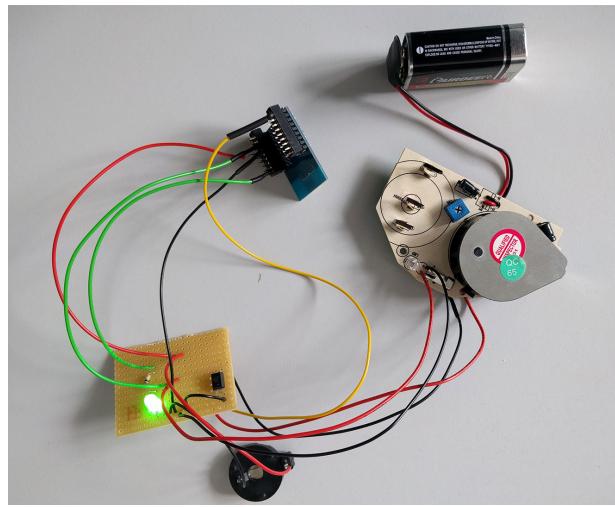
**Abbildung 4.2:** Schaltplan einer Mesh-Node mit externer 3V Spannungsquelle für den BLE-Sender (Auszug)

den 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. 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.



(a) Lösung mit einer 9 Volt Blockzelle.



(b) Lösung mit einer 9 Volt Blockzelle für Rauchmelder und einer 3 Volt Knopfzelle für nRF.

**Abbildung 4.3:** Konstrukt aus Rauchmelder mit angebautem nRF51822.

## 4.2 Software Microprozessor

Im folgenden Absatz wird beschrieben, wie die Software des nRF51822-MPUs funktioniert. Dazu wird zuerst grundlegend beschrieben, welche Anforderungen bestehen und welche Module dafür verwendet werden. Danach wird im Detail erklärt, wie der Trickle-Algorithmus für Rauchmelder-Netzwerke implementiert und adaptiert ist. Abschließend wird beschrieben, wie der Rauchmelder gesteuert und abgetastet wird.

### 4.2.1 Konzeption

Der Prozessor des nRF51822-SoC ist ein ARM Cortex M0 und wird in C99 programmiert und mit GCC kompiliert. Um den Chip zu flashen wurde für dieses Projekt 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.

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 können so alle Rauchmelder, die nicht direkt auslösen, stumm geschaltet werden (siehe Abschnitt 4.3 “Interaktion mit der Außenwelt“).

Die Software besteht aus drei übergeordneten Modulen, in Auszügen dargestellt in Abbildung 4.4. Die Nordic nRF5 SDK [8] enthält die Treiber für die Hardware, wie Radio, Pins und den Cortex M0 (Taktgeber etc.). Außerdem enthält die SDK die Bibliotheken, um die Hardware-Module zu steuern. Für diese Arbeit wurde die SDK Version 8.1.0 benutzt.

Um die Bluetooth Low Energy Funktionen nutzen zu können, wird für Peripherals das Softdevice S110 [10] benötigt (siehe Unterabschnitt 3.2.2 “Bluetooth Low Energy“), welches den BLE-Stack implementiert. Das Softdevice ist vorkompiliert in der SDK enthalten und muss separat an den Anfang des Flashspeichers 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 Multi-protocol Timeslot API“. Über diese API kann das Programm von  $100 \mu\text{s}$  bis 100 ms den Zugriff auf das Radio-Modul erhalten, um Nachrichten zu versenden.

Es gibt mit dem nRF OpenMesh [6] bereits eine auf Timeslots basierende Open-Source-Implementierung des Trickle-Algorithmus für die nRF51 SoCs und das nRF SDK 8.1. Dieses Framework kann mit einer Reihe von Modifikationen für die Implementierung der Software für die Mesh-Nodes verwendet werden.

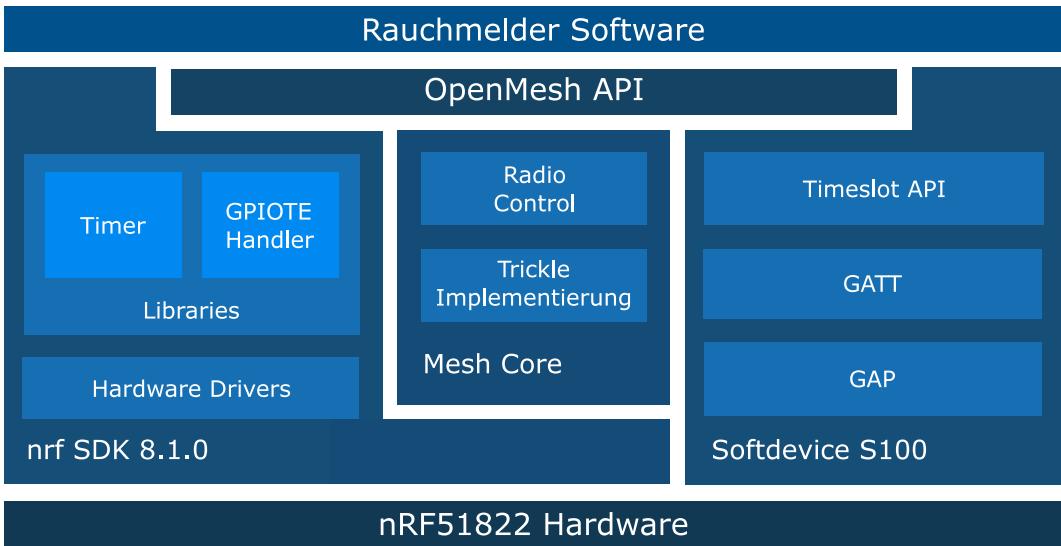


Abbildung 4.4: Übersicht ausgewählter Module der Software des nRFs

#### 4.2.2 Trickle Implementierung

Das nRF OpenMesh implementiert den Trickle-Algorithmus wie in der RFC spezifiziert (siehe Abbildung 4.5).

Während der Initialisierung werden minimale und maximale Intervalllänge  $I_{min}$  und  $I_{max}$ , sowie eine Redundanzkonstante  $k$  definiert. Der Algorithmus startet mit dem Intervall  $I_{min}$  und sucht, um Kollisionen zu reduzieren, einen zufälligen Zeitpunkt aus der zweiten Hälfte des Intervalls als Übertragungszeitpunkt  $t$  aus. Alle im Intervall vor diesem Zeitpunkt empfangenen Pakete mit konsistentem Wert werden gezählt. Ist die Anzahl der empfangenen Pakete, die mit dem lokalen Wert konsistent sind, kleiner als die Redundanzkonstante, wird der Wert gebroadcastet. Wurden mehr als  $k$  konsistente Updates empfangen, wird nicht rebroadcastet. Am Ende des Intervalls wird die Intervalllänge verdoppelt, bis zu einer maximalen Länge von  $I_{max}$ . Wird ein inkonsistenter Wert (Update) empfangen, wird das Intervall sofort auf  $I_{min}$  zurückgesetzt.

Die Implementierung des Trickles im nRF OpenMesh sieht nur eine freie Auswahl von  $I_{min}$  vor, definiert  $k := 3$  und  $I_{max} := 2048 \cdot I_{min}$ . Je kleiner  $k$  ist, desto höher ist die Wahrscheinlichkeit, vor  $t$  mehr als  $k$  Pakete zu empfangen und kein konsistentes Update zu rebroadcasten. Ein kleines  $k$  reduziert also unnötige Updates und verringert so den Energieverbrauch. Ein hohes  $k$  erhöht allerdings die Integrität des Netzwerkes, was vor allem für sicherheitskritische

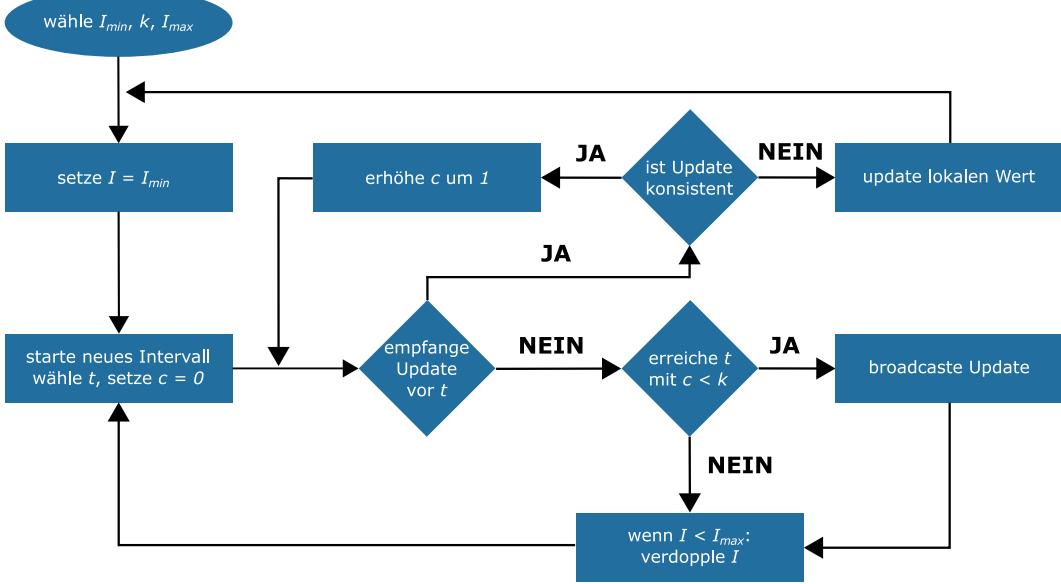


Abbildung 4.5: Flowchart des Trickle-Algorithmus

Anwendungen relevant ist. Die vom OpenMesh vorgeschlagene Redundanzkonstante von 3 ist ein guter Kompromiss zwischen Zuverlässigkeit und Energie sparsamkeit.

Als minimale Intervalllänge für den Alarmzustand wurde  $I_{min} := 100$  ms aus gewählt. Für die Berechnung der Intervallobergrenzen weicht die Implementierung des Trickle-Algorithmus im OpenMesh von der Definition in der RFC ab. Im OpenMesh wird die Obergrenze von  $I_{min}$  auf die nächstliegende 2er-Potenz gerundet, in diesem Fall 128 ms. Die untere Intervallgrenze für  $I_{min}$  wird jedoch nicht geändert und ist 50 ms. Die maximale Intervalllänge ist

$$I_{max} = I_{min} \cdot \text{Multiplikator}$$

Mit dem vorgegebenen Standard-Multiplikator von 2048 ergäbe sich ein Maximalintervall von 262144 ms = 4.37 Sekunden. Dieses Maximalintervall ist angemessen, wenn sich die Sensorwerte häufig ändern. Bei Rauchmeldern geschieht das allerdings nur selten, weswegen eine höhere, maximale Intervalllänge in Betracht des Energieverbrauchs durchaus angemessen ist. Ein Kompromiss zwischen Mesh-Integrität und weniger Rebroadcasts ist ein  $I_{max}$  von etwa einer Stunde. Der Multiplikator für  $I_{min} = 128ms$  um möglichst nahe an diese 1 Stunde heranzukommen ist  $2^{15}$ , mit resultierendem  $I_{max}$  von rund 1 Stunde und 10 Minuten. Das heißt, dass das Intervall 15 mal in Folge verdoppelt wird, bis es seine maximale Länge erreicht.

Sind  $I_{min}$  und  $I_{max}$  bekannt, können die Intervallgrenzen bestimmt werden (siehe Abbildung 4.6). Aus den Intervallgrenzen folgen die Zeiträume, in denen die Zeitpunkte  $t_r$  der jeweiligen Rebroadcasts liegen.

Sei  $r$  definiert als der Wiederholungsindex, dann ist

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

Daraus ergeben sich für  $I_{min} = 128$  ms die akkumulierten Intervallgrenzen seit Auslösen als

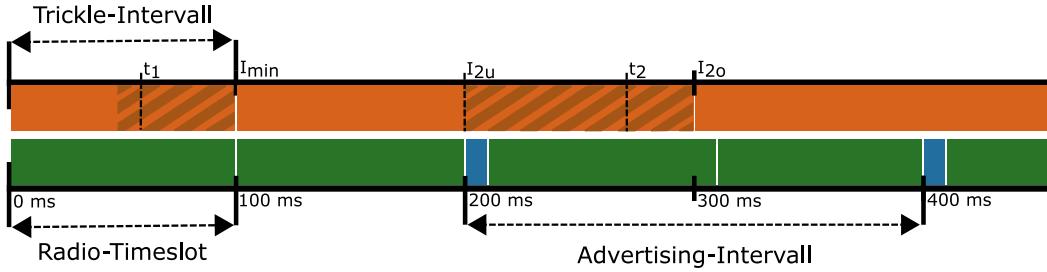
$$\begin{aligned} [I_{ru}, I_{ro}] &= [I_{r-1} + \frac{2^r * I_{min}}{2}, I_{r-1} + 2^r * I_{min}] \\ &= \{[50, 128], [228, 356], [612, 868], \dots\} \end{aligned}$$

Dabei bezeichnet  $I_{ru}$  die untere und  $I_{ro}$  die obere Grenze des Sendezeitpunktes im Trickle-Intervall in Abhängigkeit von  $r$ .

Der tatsächliche Sendezeitpunkt ist allerdings später als der theoretisch Berechnete. Der Grund dafür ist, dass das OpenMesh Framework nicht immer die Kontrolle über das Radio-Modul des nRFs hat (siehe Abbildung 4.6). Läuft auf dem nRF ein Softdevice, übernimmt es Kontrolle über bestimmte Hardware-Module, unter anderem das Radio-Modul. Um das Radio-Modul benutzen zu können, kann eine Software mit der Timeslot-API den Zugriff zeitweise anfordern. Das Softdevice stellt dann einen “Timeslot“ über einen angeforderten Zeitraum bereit. Dieser Zeitraum beträgt maximal 10 Millisekunden, kann aber auf bis zu 1 Sekunde verlängert werden. Während das OpenMesh einen Timeslot hat, scannt es durchgehend auf Updates aus dem Mesh. Wird ein Sendezeitpunkt  $t_r$  erreicht, wird ein Update sofort versendet. Das Softdevice verzögert oder entzieht dem Framework aber den Timeslot, sobald es das Radio-Modul selbst braucht.

Eine Mesh-Node versendet zwei unterschiedliche Advertisements. Das Mesh-Advertisement wird vom OpenMesh Framework versendet, um Updates zwischen Nodes auszutauschen. Es kann von regulären Observern nicht gelesen werden. Das GAP-Advertisement ist Teil des BLE-Protokoll-Stacks, wird vom Softdevice versendet und broadcastet um den Alarmzustand an reguläre Observer. Das Softdevice entzieht dem Framework den Timeslot, wenn es ein geplantes GAP-Advertisement versenden will oder wenn über eine GATT-Verbindung Daten ausgetauscht werden sollen.

Die Entwickler des Frameworks geben an, dass das Softdevice während des



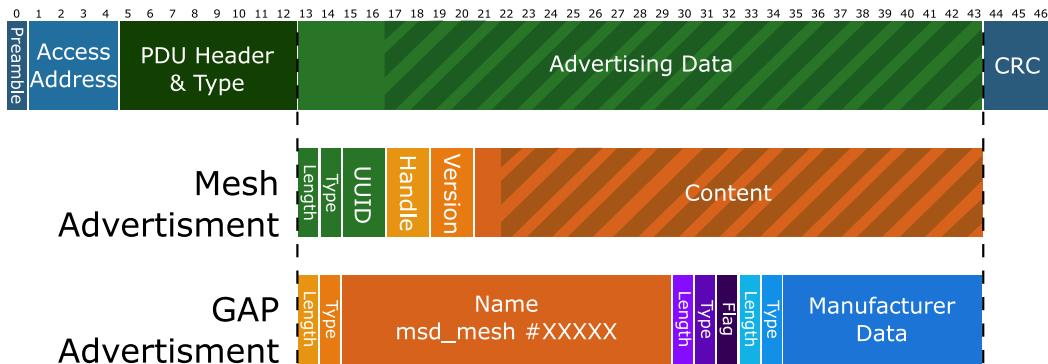
**Abbildung 4.6:** Übersicht der zeitlichen Einordnung von (oben) Trickle-Intervallen und (unten) geteilter Zugriff auf das Radio-Modul zwischen (grün) OpenMesh und (orange) Softdevice

GAP-Advertisings etwa 5-25% der Zeit die Kontrolle über das Radio-Modul übernimmt, während einer GATT-Verbindung sogar bis zu 80%.

Das Versenden eines GAP-Advertisments dauert laut Softdevice-Specification [10] zwischen 7 und 12 Millisekunden. Dabei wird konsekutiv für jeden der drei Advertising-Channel (37, 38, 39) das Advertisement gesendet. Die Timeslot API erlaubt es, das Mesh-Advertisments auf nur einem der drei für Advertisements reservierten Channel (37, 38, 39) gesendet werden. Dadurch muss auch nur ein Channel gescannt werden. Der Channel wird beim initialisieren des Meshs festgelegt. Für das Rauchmelder-Netzwerk wurde der Channel 38 ausgewählt. Während dieser Periode kann das Central die Mesh-Node scannen um eventuell eine GATT-Verbindung aufzunehmen. Dieser Scan verlängert die Dauer des Advertisings und damit die Verzögerung des Sendezeitpunktes im Trickle-Intervall zusätzlich.

Das GAP-Advertisement einer Mesh-Node wird im Intervall von 200 ms gesendet. Dieses Intervall wurde in Hinsicht auf den Entwicklungsprozess so schnell eingestellt, kann für ein Produktivsystem aber auf bis zu 10.24 Sekunden erhöht werden. Ein hohes Intervall reduziert den Energieverbrauch deutlich. Das Advertisement-Intervall kann zur Laufzeit verändert werden. Ändert sich der Alarmzustand auf ON, kann das Intervall wieder auf 200 ms reduziert werden. Zu niedrige Intervalle sind aber nachträglich für die Mesh-Funktion, weil das OpenMesh Framework keinen Zugriff auf das Radio-Modul hat, während das Softdevice das GAP-Advertisement sendet.

Die Mesh-Broadcasts sind wie reguläre BLE-Advertisments strukturiert, aber inhaltlich modifiziert (siehe Abbildung 4.7). Jedes BLE-Advertisement startet mit einer Preamble und der Access-Address und endet mit einem CRC zu



**Abbildung 4.7:** Byte-Struktur von (oben) BLE-Advertisement generell und (unten) Advertisement-Data des Mesh-Advertisments

Verifizierung. Die Access-Address ist immer 0x8E89BED6 für BLE-Advertisements. Anhand dieser Zeichenfolge kann ein Observer erkennen, dass gerade ein BLE-Paket empfangen wird. Im OpenMesh wird diese Access-Address allerdings geändert, was mehrere Vorteile mit sich bringt: Die Mesh-Updates können von anderen Advertisements einfacher unterschieden werden, werden von anderen Central-Devices nicht beachtet und es können mehrere Netzwerke mit dieser Technologie im selben Bereich eingesetzt werden, ohne sich gegenseitig zu behindern. Die 6-37 Bytes zwischen Access-Address und CRC ist die "Packet Data Unit" (PDU). Im PDU-Header steht die Länge und Typ des Advertisements. Der Typ legt fest, ob ein Empfänger Verbindungsanfragen stellen kann und ob bei einem Scan zusätzliche Informationen verfügbar sind. Das Mesh-Advertisement ist vom Typ "Nonconnectable", erlaubt also weder Scan noch Verbindung. Die verbliebenen 31 Bytes einer regulären PDU können beliebig viele AD-Structures beinhalten. Jede AD-Structure besteht aus je einem Byte Länge und AD-Type und den AD-Data. Das Mesh-Advertisement definiert eine AD-Structure mit AD-Type 0x16, gefolgt von einer Service-UUID mit 0xFFE4, dem Handle, Version und Content. Das Handle ist die einzigartige ID des jeweiligen Mesh-Wertes. Version ist die Angabe des "Alters" des 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 Advertisement auf 25 Bytes Länge, was wiederum die Sendezeit und damit Energieverbrauch reduziert.

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 Advertisement erkannt, wird ein Callback aufgerufen. Dort kann anhand des Eventtyps eine entsprechende Aktion ausgeführt werden:

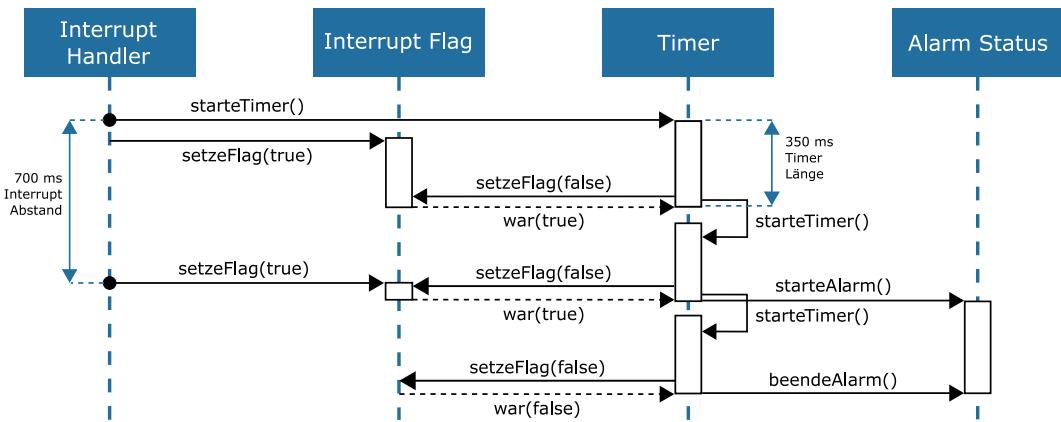
- **Update:** Der empfangene Wert ist inkonsistent und hat eine höhere Version. Ist der neue Wert 0x01 wird der PO06 des nRFs gesetzt. Das aktiviert den angeschlossenen Optokoppler, welcher den Testschalter des Rauchmelders kurzschließt. Das versetzt den Rauchmelder in den Zustand TEST, welcher vom Verhalten her identisch ist mit ON. Ist der neue Wert 0x00 wird der PO06 gecleared 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 Framework speichert jedes Handle-Version-Value Tripel in einem von zwei unterschiedlichen Caches: einen persistenten und einen nicht persistenten. Standardmäßig werden alle Tripel im persistenten Cache gespeichert. Ist der persistente Cache aber voll, werden die ältesten Werte, deren letztes Update am längsten her ist, in den nicht-persistenten Cache verschoben. Ist im RAM des nRFs nicht genug Speicher vorhanden, um alle Tripel zu behalten, werden die ältesten aus dem nicht-persistenten Cache gelöscht. Die Größe des Caches wurde bewusst auf 6 Tripel reduziert um den reservierten Speicher des nRFs klein zu halten. Überflüssige Werte können dadurch entstehen, dass entweder ein anderes Mesh mit derselben Access-Address in Reichweite betrieben wird oder wenn das Mesh angegriffen wird (siehe Abschnitt 6.1 “Sicherheit vor Angriff und Fälschung“). Würden neue Werte mit einem anderen Handle als 0x01 (dem des Alarmzustandes) angenommen, könnte ein Angreifer das Tripel des Alarmzustandes zeitweise aus dem Cache herausdrängen.
- **Conflicting:** Ein empfangenes Paket 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. Der wahrscheinlichere Fall ist eine Manipulation von außen, weswegen Conflicting-Values immer verworfen werden.

### 4.2.3 Verarbeitung des Alarmfalles

Der nRF tastet über den PO07 den Zustand des Rauchmelders ab (siehe Kapitel 3 “Vorbetrachtung“). 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, bei der nur die LED abgetastet wird, 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 700 ms für das kurze Intervall  $I_{kurz}$  im ON/TEST-Zustand und 44000 ms für das lange Intervall  $I_{lang}$  im IDLE/Fehler Zustand.

Um den Pin abzutasten, wurde in der Software der GPIOE-Handler der nRF 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 dazu gehörende 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 4.8 dargestellt. Der Algorithmus startet nach einem Delay von 10 ms 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 erkennt, dass 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. Die Länge des Timers ist für  $I_{kurz}/2 = 350$  ms gesetzt. Das Auslösen des Rauchmelders kann anhand des Blinkintervalls ab der zweiten Wiederholung des Timers mit Delay nach  $I_{kurz} = 710$  ms erkannt werden. Der Delay von 10 ms soll verhindern, dass durch Verzögerungen durch die Ressourcenverteilung des Softdevices der zweite Timeout-Handler ausgeführt wird, bevor der Interrupt-Handler aufgerufen wurde.

Wird das Auslösen des Rauchmelders am PO07 oder durch ein Update-Event erkannt, wird dies nicht nur durch die Mesh-Advertisments an alle anderen Nodes propagiert. Über die Manufacturer-Data (siehe Abbildung 4.7) des GAP-Advertisments werden alle Observer, die nicht zum eigentlichen Mesh gehö-



**Abbildung 4.8:** Algorithmus zum Abtasten des Blinkintervalls (vereinfacht)

ren, über den Alarm informiert. Das Softdevice bietet eine Reihe von vordefinierten AD-Structures, die dem Advertisement mitgegeben werden können. Das GAP-Advertisement enthält drei Structures: eine für Name, eine für die Manufacturer-Data und eine für ein Flag, mit dem der Absender deklariert, dass er Verbindungsanfragen akzeptiert. Der Name besteht aus dem Kürzel “msd-mesh\_#“, gefolgt von einer fünfstelligen, beim Systemstart zufällig als ID generierten Zahlenfolge. Die Manufacturer-Data beinhalten zusätzlich zu Länge und AD-Type noch 4 Byte “Manufacturer-ID“, in diesem Fall 0xFFFF für “noch in Entwicklung“ und 1 Byte mit dem Wert des Alarmzustandes. Ändert sich der Alarmzustand, werden auch die Manufacturer-Data im “GAP-Advertisement“ geändert, um den neuen Alarmzustand zu entsprechen.

Das GAP-Advertisement wird auch benötigt, um eine GATT-Verbindung zwischen Peripheral und Central aufzubauen (siehe Unterabschnitt 3.2.2 “Bluetooth Low Energy“). Das Framework implementiert einen Service mit zwei Characteristics. Die Mesh-MetadatCharacteristic enthält die Konfigurationsparameter des Meshs:

- Die Access Address.
- Die Länge des Intervalls  $I_{min}$ .
- Die Anzahl der maximal möglichen Handle-Version-Value Tripel, die die Node speichern kann.
- Den BLE-Channel auf dem das Mesh agiert.

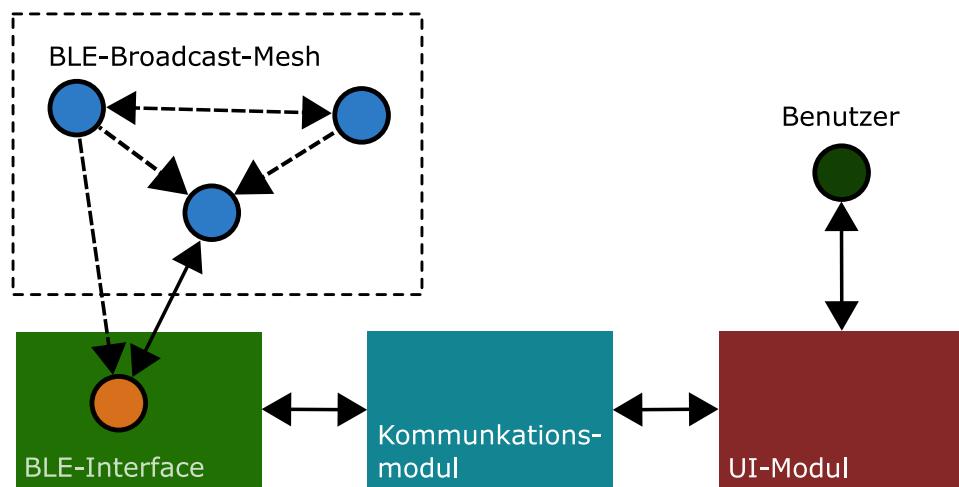
Die Mesh-Values-Characteristic erlaubt den Zugriff auf Werte (Einfügen und Update) und das Abfragen der Persistenz und des Übertragungsstatus. Der

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 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 (siehe Abschnitt 6.1 “Sicherheit vor Angriff und Fälschung“). 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 werden. Zum einen ist eine GATT-Verbindung nachteilig hinsichtlich 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 “Interaktion mit der Außenwelt“ 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 “Smart Home“. Für die Verwaltung des Netz-



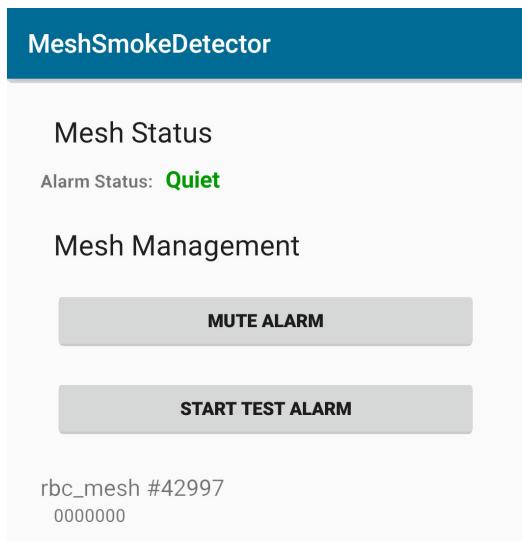
**Abbildung 4.9:** Übersicht über das Verwaltungs-Konzept des Networks

werkes von außen wurde ein Konzept für ein Gateway entworfen und beispielhaft in Form einer Android-App realisiert.

Das Konzept hat drei Komponenten (siehe Abbildung 4.9):

- Ein BLE-Interface, dass permanent alle GAP-Advertisments scannt, die Advertisments des Meshs 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-Empfänger sonst unnötig viel Strom verbrauchen. Das GAP-Advertisement der Mesh-Nodes wird aber nur einmal pro Minute 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 entsprechenden Characteristics zu schreiben. Dazu muss, trifft eine Verbindungsaufruforderung am BLE-Interface ein, nicht auf den nächsten Sendezyklus der Mesh-Node gewartet werden.
- Ein UI-Modul, etwa die “Smart Home“-Software oder eine App, durch die die Verwaltung des Netzwerkes geführt wird.
- Ein Kommunikationsmodul, das Befehle zwischen den anderen Modulen austauscht.

Alle drei Module wurden der Einfachheit halber in einer Android-App implementiert (siehe Abbildung 4.11).

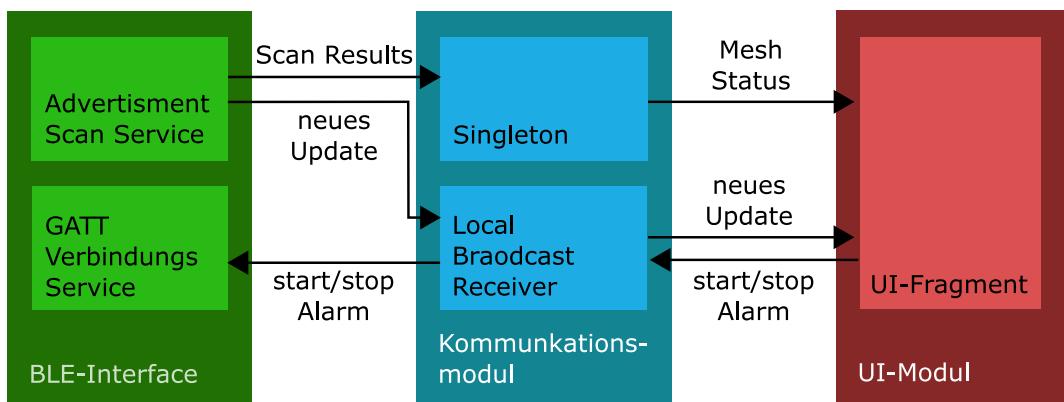


**Abbildung 4.10:** Das UI-Fragment der Android-App

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 Advertisements mit Namen und Manufacturer-Data an (siehe Abbildung 4.10).

Als BLE-Interface wurden zwei Services implementiert. Der ScanService läuft immer und liest alle gesendeten Advertisements. Empfängt der Service ein Advertisement, 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 nach dem Mesh-Service, der Anhand der UUID (0000fee4-0000-1000-8000-00805f9b34fb) erkannt wird. Diese UUID ist im OpenMesh für diesen Service voreingestellt und kann bei Bedarf frei gewählt werden. Wurde der Service gefunden, kann auf die darin enthaltene Value-Characteristic mit der UUID (2a1e0005-fd51-d882-8ba8-b98c0000cd1e) zugegriffen werden. Der entsprechende Befehl wird als Bytearray in diese Characteristic geschrieben.

Die möglichen Befehle sind (0x00, 0x01, 0x00, 0x01, 0x01) für “Testalarm auslösen” und (0x00, 0x01, 0x00, 0x01, 0x00) für “Mesh temporär stumm-



**Abbildung 4.11:** Android-Module nach Zugehörigkeit im Verwaltungs-Konzept

schalten“. 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 gefundenen 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 Verbindungsauflauf.

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. [21] 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 anhand 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 ein Rauchmelder pro  $60\text{ m}^2$  Fläche mittig an der Decke montiert werden.
- In Fluren bis maximal 3 Meter Breite darf maximal 15 Meter Abstand zwischen zwei Rauchmeldern sein. Der Abstand zur Stirnseite eines Flures darf maximal 7,5 Meter betragen. Flure die Breiter als 3 Meter 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 17,5 Meter. Das ist dann der Fall, wenn ein Raum mit  $60 m^2$  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 15 Meter 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 15 m entfernt ist. (A1)*

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 Beton, 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:

*Ein Update kann, trotz Dämpfung und Reflexion von Wänden und Decken bis 3 Meter Höhe, in den ersten zwei Sendeintervallen empfangen werden. (A2)*

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 Rauchmeldernetzwerken 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 10 m Abstand nicht mehr als 1000 ms. (A3)*

## 5.2 Testmethodik

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

- Für die Anforderung (A1) an die Latenz in einem Single-Hop-Szenario, wurden die nRF-Sender und Empfänger ohne den angebauten Rauchmelder in Abständen von 10 cm, 3 m, 7.5 m, 15 m und 20 m Entfernung voneinander platziert.
- Für die Anforderung (A2) wurden Sender und Empfänger mit 3 m Abstand voneinander platziert. Der Ort wurde so gewählt, dass sich dabei einmal eine Wand und einmal eine Raumdecke zwischen den Nodes befand.
- Für die Anforderung (A3) wurde ein Multi-Hop-Szenario getestet. Dabei wurden in vier Tests je 3 Mesh-Nodes mit 7.5 m und 10 m Abstand jeweils sowie 4 Nodes mit 7.5 m und 10 m Abstand jeweils verteilt. Die Distanzen wurden kleiner gewählt als die erwartete Sendestärke um mögliche Verbesserungen der Latenz durch “Überspringen“ einer Node zu untersuchen. Das heißt, dass eine weiter entfernte Node die Updates nach einem anstatt zwei Hops empfängt.

Die Tests wurden in den Räumen der Fakultät für Medien der Bauhausuniversität Weimar durchgeführt. Für die Anforderung (A1) und (A3) wurde ein Flur von etwa 35 Meter Länge und etwa 3 Meter Breite ausgewählt. Die Ergebnisse können durch Menschen, andere 2.4 GHz Geräte (BLE und Wifi) sowie durch die Reflexionseigenschaften der Dachschräge des Flures beeinflusst sein. Anforderung (A2) wurde mit der Wand zwischen dem Flur und einem der Labore sowie der Decke zwischen der Lobby der zweiten und dritten Etage getestet.

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 1 ms genau, wurden mit ihrem Timestamp

in einem Logfile gespeichert und mit Python analysiert.

Bei der Durchführung wurde zuerst der Sender platziert und mit dem Arduino an einen Laptop angeschlossen. Danach wurden beide Timer gleichzeitig mit einem Schalter gestartet. Anschließend wurde der Sender von der Schaltung entfernt. Der Arduino wurde anschließend mit dem Empfänger an dessen Position verbunden und der Test wurde begonnen. Bei allen Tests wurden die Antennen in die gleiche Richtung ausgerichtet.

### 5.3 Diskussion der Ergebnisse

In Abschnitt 4.2 ‘‘Software Microprozessor’’ 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:

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

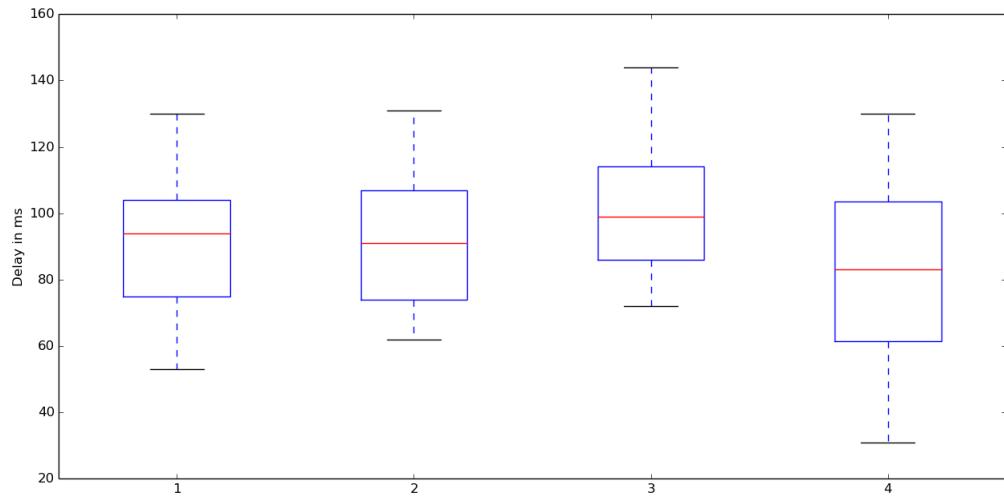
Beachtet man das Erhöhen von  $I_{min}$  nach dem ersten Intervall  $r=0$  auf die nächstliegende 2er-Potenz (hier 128), ergeben sich für  $I_{min} = 100$  ms die akkumulierten Intervallgrenzen seit Auslösen von

$$\begin{aligned} [I_{ru}, I_{ro}] &= [I_{r-1} + \frac{2^r * I_{min}}{2}, I_{r-1} + 2^r * I_{min}] \\ &= \{[50, 128], [228, 356], [612, 868], [1380, 1892], [2916, 3940], [5988, 8036], \dots\} \end{aligned}$$

Anhand dieser Intervalle kann jede gemessene Latenz einem dieser Intervalle zugeordnet werden. Daran wird mit hoher Wahrscheinlichkeit erkannt, in welchem Intervall das erste empfangene Paket 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-Advertisement sendet) oder wenn das Paket durch Dämpfung oder Interferenz nicht empfangen wurden konnte. Ein Paket gilt als empfangen im Intervall  $I_r$  wenn

$$t \leq I_r + \frac{2^{r+1} * I_{min}}{2}$$

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



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

An den Ergebnissen 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 (A1) 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 schlechter: in 54% der Sendezyklen wurde das Update nicht empfangen. Die Anforderung wird hier nur in 61% der Fälle erfüllt. Bei Distanzen von bis zu 7.5 Meter werden bis zu 98% der Pakete

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

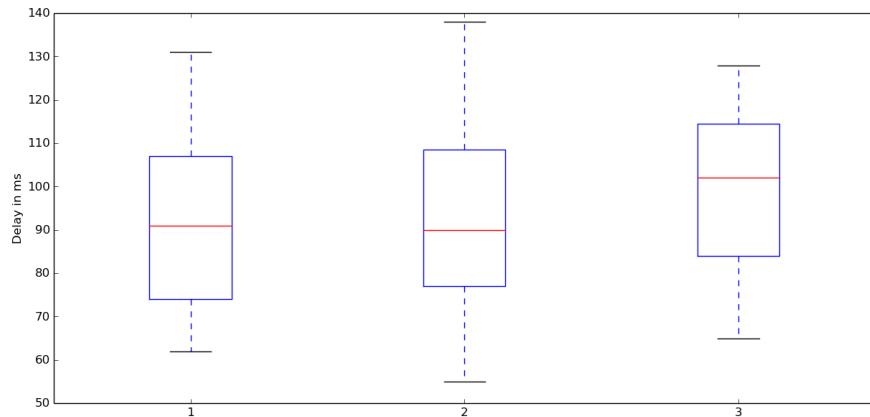
	Mean	Median	$t < I_{2u}$	$I_{2u} < t < I_{3u}$	$I_{3u} < t$	Verlust
10 cm	88 ms	94 ms	100%	0%	0%	0%
3 m	92 ms	91 ms	98.82%	1.18%	0%	0%
7.5 m	103 ms	99 ms	98.82%	1.18%	0%	0%
15 m	504 ms	83 ms	93.24%	5.41%	4.05%	0%
20 m	1152 ms	400 ms	46.43%	17.86%	39.29%	54%

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 dass Pakete aufgrund von Dämpfung oder Distanz nicht empfangen wurden.

Der scheinbare Anstieg der Durchschnittslatenz ist mit hoher Wahrscheinlichkeit ebenfalls dadurch begründet und damit zufällig unterschiedlich. Bei einem Advertising-Intervall von 200 ms findet also in 89% der Fälle mindestens ein Advertising während des ersten Sende-Intervalls (50 ms-228 ms) 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 zwischen 7 und 12 Sekunden, fällt dies mit dem Sendezeitpunkt in 3.5 -6% der Fälle zusammen. Diese Zufälligkeit lässt sich durch die Software nicht beeinflussen, da sowohl der Sendezeitpunkt innerhalb des Sendeintervalls laut Trickle randomisiert, als auch der genaue Zeitpunkt des GAP-Advertisements laut BLE-Core-Specification zufällig um bis zu 10 ms verzögert wird.

Die Wahrscheinlichkeit dafür, dass ein Paket im ersten und zweiten Intervall zufällig verloren wird ist aber verschwindend gering, sodass 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 nicht mehr 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 Ausreißer 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 50 ms 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 Daten, der Whisker umfasst die obere und untere Grenze (hier ohne Ausreißer). 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



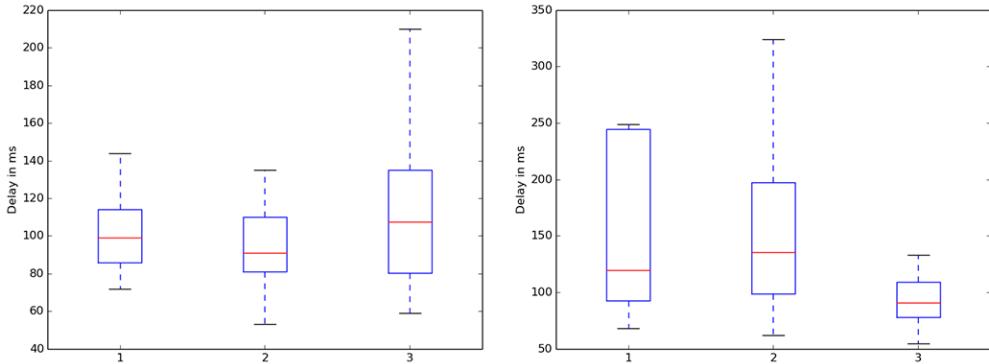
**Abbildung 5.2:** Latenz mit Dämpfung über 3 Meter (1) ohne Dämpfung, (2) durch Wand, (3) durch Decke

einen leicht erhöhten Einfluss von Dämpfungserscheinungen schließen lassen, ausgelöst durch zum Beispiel andere Beton- und Stahlanteile sowie Reflexionen. Für eine zuverlässige Schlussfolgerung bedarf es aber umfangreicherer Tests. Was jedoch geschlussfolgert werden kann, ist, dass der Einfluss der Dämpfung gering genug ist, um der zweiten Anforderung (A2) gerecht zu werden.

Aus den Multi-Hop-Tests lassen sich noch weitere wichtige Eigenschaften beobachten. Die Tests mit 7.5 Meter Distanz sollten vor allem den Einfluss zeigen, den das Überspringen von Updates auf die Latenz hat (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

**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	92 ms	91 ms	98.82%	1.18%	0%	0%
durch Wand	83 ms	75 ms	96.43%	3.57%	0%	0%
durch Decke	111 ms	102 ms	98.82%	1.18%	0%	0%



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

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 100 ms sein. Die Testergebnisse für 3 Hops über je 10 Meter Distanz zeigen also, dass der Trickle 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 geht. 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.

**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, 7.5 m	92 ms	91 ms	100%	0%	0%	0%
3 Hops, 7.5 m	124 ms	108 ms	95.83%	4.17%	0%	0%
2 Hops, 10 m	168 ms	139 ms	82.81%	15.63%	1.56%	0%
3 Hops, 10 m	91 ms	91 ms	100%	0%	0%	0%

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. Dabei sollte beachtet werden, dass 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 (A3) 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 ausreichend genau. Jedoch werden die anderen beschriebenen Beobachtungen und Annahmen durch die Ergebnisse nicht genau nachgewiesen. Um den genauen 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, sodass 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

Es gibt für dieses Projekt viele Erweiterungsmöglichkeiten. Die wichtigsten davon sind Sicherheit, Energieverbrauch und Supervision und sollen in folgendem Kapitel umrissen werden. Abgesehen davon ist auch Forschung in anderen Bereichen denkbar. So können zum Beispiel andere BLE- oder WSN-Module konzipiert und verglichen werden, die auch auf Broadcast-Flooding basieren. Auch kann beispielsweise der Piezosummer direkt angesprochen werden, um etwa verschiedene Tonhöhen, Tonfolgen oder ähnliches zu erzeugen. Das ermöglicht die Untersuchung neuer Interaktionskonzepte, so dass ein Benutzer beispielsweise erkennen kann, wie weit er vom Ursprung des Feuers entfernt ist, oder in welcher Richtung der Brand liegt.

### 6.1 Sicherheit vor Angriff und Fälschung

Das in dieser Arbeit vorgestellte System wurde als experimentelle Plattform konzipiert und ignoriert deshalb die Sicherheit vor Angriffen und Fälschung weitestgehend, was zu einer Reihe von Sicherheitslücken führt. So wird zum Beispiel nicht validiert, ob eine neue Node tatsächlich zum Netzwerk gehört. Findet ein Angreifer die Access-Address der Mesh-Advertisments heraus, kann er mit jedem BLE-Broadcaster gefälschte Pakete senden und den Feueralarm auslösen. Diese Schwäche kann nicht dadurch behoben werden, dass der Wert des Mesh-Advertisments um die ID der Node erweitert wird, weil diese ID auch ausgelesen und gefälscht werden kann. Ein möglicher Lösungsansatz wäre die Verschlüsselung der Pakete. Der zugehörige Schlüssel müsste dann in einer streng kontrollierten Initialisierungsphase an die Node übergeben werden. Das könnte beispielsweise über eine GATT-Verbindung geschehen. Die Bluetooth Core Specification beinhaltet bereits ein Konzept für Verschlüsselung und Authentifizierung von GATT-Verbindungen. Die Verschlüsselung und Authentifizierung des Advertisements ist dort nicht vorgesehen, aber die beste-

henden Konzepte können dafür adaptiert oder weiter benutzt werden. Damit die Verschlüsselung tatsächlich sicher ist, bedarf es entweder mehrerer Schlüssel oder ein “randomisiertes Byte“ im Content des Mesh-Advertisments (siehe Abbildung 4.7). Ohne eines dieser Verfahren kann ein Adversary einfach die bereits verschlüsselten Nachrichten abfangen und rebroadcasten.

Die zweite kritische Sicherheitslücke ist der in Abschnitt 4.2 “Software Microprozessor“ beschriebene GATT-Service zum Stummschalten der Rauchmelder und Auslösen des Testalarms. In der aktuellen Implementation ist das Verbinden zu den Nodes noch offen. Das Sicherheitskonzept für Bluetooth Low Energy erlaubt aber einen Key-Exchange für GATT-Verbindungen. Dieser Schlüsselaustausch muss für jedes Central durchgeführt werden, dass Schreibzugriff auf die Characteristics haben soll. Alternativ dazu kann der GATT-Service zum Auslösen des Alarms auch entfernt werden. Ersetzt werden kann er dadurch, dass das in Abschnitt 4.3 “Interaktion mit der Außenwelt“ beschriebene BLE-Interface nicht als GATT-Characteristic implementiert wird, sondern auch durch eine Mesh-Node. Diese Interface-Node kann vom Kommunikationsmodul direkt über die Pins angesprochen werden. Dadurch verliert das Mesh zwar an Flexibilität, bietet aber auch weniger Angriffsfläche.

## 6.2 Energieverbrauch

Eine der “Fundamental Challenges“ für WSNs in sicherheitskritischen Anwendungen ist niedriger Energieverbrauch. Für diese Arbeit war die Challenge hinreichend erfüllt durch BLE als Technologie und Trickle als Flooding-Algorithmus. Bei detaillierter Betrachtung gibt es trotzdem noch Bedarf für Optimierung und Evaluierung. Dazu gehört unter anderem ein Vergleich zwischen der ein- und zwei-Batterie Lösung hinsichtlich der Lebensdauer einer Mesh-Node. Nach Angaben des Herstellers [7] zum allgemeinen Verbrauch des nRF51822, könnte eine auf Energieeffizienz optimierte Node mit einer 150 mAh Knopfzelle theoretisch bis zu 300 Tage Laufzeit erreichen. Realistisch betrachtet dürfte die Lebensdauer aber deutlich geringer ausfallen. Außerdem gilt es herauszufinden, inwieweit die Intervallgrenzen, Advertisingintervalle und die Redundanzkonstante zwischen Energieverbrauch und Latenz optimal balanciert werden können. Die BLE Special-Interest-Group (SIG) hat bereits angekündigt, Mesh-Funktionalität in die Core Specification zu integrieren. Sollten dazu Implementierungen der Hersteller vorliegen, kann diese vor allem hinsichtlich des Energieverbrauchs mit der hier vorgestellten Lösung verglichen werden.

## 6.3 Mesh Supervision

Im Rahmen der Zuverlässigkeit der Übertragung wurde das Self-Healing von Broadcast-based Mesh Networking besprochen. Eine Erweiterung dieses Konzeptes stellt die Überwachung des Zustandes der Mesh-Nodes, oder Supervision, dar. Grundsätzlich ist es ausreichend, dass das Mesh weiterhin funktioniert, wenn einzelne Nodes ausfallen. Es ist jedoch genauso wichtig, dass erkannt wird, wenn und welche Node ausgefallen ist. Durch die Prinzipien des Trickle-Algorithmus, speziell die Redundanzkonstanten, ist es wahrscheinlich, dass einzelne Nodes bei längerem Sendeintervall nie ihren Zustand rebroadcasten. Es ist daher nicht ausreichend, Listen mit den Node-IDs zu führen und zu überprüfen, wann eine Node keine Updates mehr sendet, ohne eine zweite Trickle-Instanz zu führen. Stattdessen kann ein Supervisor-Konzept realisiert werden. Dabei kann über die Schnittstelle nach außen (siehe Abschnitt 4.3 “Interaktion mit der Außenwelt“) in regelmäßigen Abständen beispielsweise ein “Durchzählen“ aller Nodes im Netzwerk initiiert werden. Dabei kann jede Node ein Bit im freien Bereich des Contents (siehe Abbildung 4.7) setzen. Bei 21 freien Bytes, minus 1 Byte für die ID der Zählung, ergibt das 160 freie Bits für die Zählung. Ist die Anzahl der gesetzten Bytes geringer als bei der letzten Zählung, lässt das auf den Ausfall einer Node schließen.

Implementieren die BLE-Sender den Battery-Service, kann das Mesh auch dazu benutzt werden, Low-Power Warnungen zu broadcasten. Dabei kann ein gesetztes Byte des Contents aussagen, dass eine Node wenig Strom übrig hat. Über das UI-Modul kann diese Node dann ausgelöst werden, um sie zu lokalisieren. So kann der Benutzer auch dann die Batterie-Warnung mitbekommen, wenn er über einen längeren Zeitraum nicht in Hörreichweite des Batterie-Alarms des Rauchmelders ist.

## 6.4 Fazit

Diese Arbeit hat gezeigt, wie (Funk-)Rauchmelder, mit Hilfe der modernen Technologien des Internets der Dinge, weitreichend verbessert werden können. Ein besonderer Schwerpunkt dabei war, wie zuverlässige Datenübertragung, geringe Latenz und geringer Energieverbrauch geboten werden können. Es wurde erörtert, warum ein Mesh-Netzwerk aus Bluetooth Low Energy Peripherals dahingehend besser operiert als existierende Technologie.

Um die Funktionsfähigkeit zu zeigen, wurden mehrere Rauchmelder mit BLE-Peripherals nachgerüstet. Es wurde eine Software entworfen, die ein Mesh-Netzwerk zwischen den Rauchmeldern aufspannt und die Rauchmelder koor-

diniert auslösen und stummschalten kann. Es wurde ein Verwaltungs-Konzept entworfen, dass es erlaubt, dass Mesh-Network sowohl in bestehende IoT-Systeme zu integrieren. Dieses Konzept wurde als Smartphone-APP realisiert, die dem Benutzer die direkte Verbindung erlaubt.

Die erwartete zuverlässige Datenübertragung und geringe Latenz wurde in einer Reihe von Tests evaluiert. Diese Tests haben gezeigt, dass Nachrichten in 95% der Fälle in unter 228 Millisekunden über das Netzwerk propagiert werden können. Die Tests werfen aber auch weitere Fragen auf, etwa inwieweit genau Reflexionen der Signale oder Dämpfung durch Menschen und andere Signalquellen das Mesh beeinflussen. Abgesehen davon hat das hier vorgestellte Konzept noch ungelöste Schwachstellen hinsichtlich Energieverbrauch und Sicherheit, welche sich als Themen für weitere wissenschaftliche oder kommerzielle Arbeit eignen.

Abschließend ist zu sagen, dass das vorgestellte System großes Potenzial hat, eine Lösung für die eingangs aufgezeigten Schwächen klassischer (Funk-) Rauchmelder zu sein, dabei preiswert zu bleiben und so mehr Sicherheit in die Haushalte zu bringen.

# 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 wohnungsgleichlicher 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] Energy consumption analysis for bluetooth, wifi and cellular networks. 3.2.1
- [6] nRF OpenMesh. <https://github.com/NordicSemiconductor/nRF51-ble-bcast-mesh>. letzter Zugriff 05.08.2016. 4.2.1
- [7] nRF51 current consumption for common scenarios. <https://devzone.nordicsemi.com/blogs/679/nrf51-current-consumption-for-common-scenarios/>. letzter Zugriff 27.08.2016. 6.2
- [8] nRF51 Software Development Kit 8.1.0. [http://developer.nordicsemi.com/nRF51\\_SDK/nRF51\\_SDK\\_v8.x.x/doc/8.1.0/index.html](http://developer.nordicsemi.com/nRF51_SDK/nRF51_SDK_v8.x.x/doc/8.1.0/index.html). letzter Zugriff 04.08.2016. 4.2.1
- [9] Photoelectric Smoke Detector IC with I/O - MC145010. [http://www.nxp.com/files/analog/doc/data\\_sheet/MC145010.pdf](http://www.nxp.com/files/analog/doc/data_sheet/MC145010.pdf). letzter Zugriff 23.07.2016. 3.1

- [10] S110 nRF51 SoftDevice Specification v2.0. [http://infocenter.nordicsemi.com/pdf/S110\\_SDS\\_v2.0.pdf](http://infocenter.nordicsemi.com/pdf/S110_SDS_v2.0.pdf). letzter Zugriff 04.08.2016. 3.2.2, 4.2.1, 4.2.2
- [11] Setting up the Timeslot API. <https://devzone.nordicsemi.com/tutorials/16/>. letzter Zugriff 02.08.2016. 3.2.2
- [12] Todesursachen in Deutschland, Statistisches Bundesamt. [https://www.destatis.de/DE/Publikationen/Thematisch/Gesundheit/Todesursachen/Todesursachen2120400147004.pdf?\\_\\_blob=publicationFile](https://www.destatis.de/DE/Publikationen/Thematisch/Gesundheit/Todesursachen/Todesursachen2120400147004.pdf?__blob=publicationFile). letzter Zugriff 15.08.2016. 1
- [13] Verteilung von Android-Geräten nach API-Level. <https://developer.android.com/about/dashboards/index.html>. letzter Zugriff 02.08.2016. 3.2.1
- [14] 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
- [15] Bluetooth Core Specification 4.2. <https://www.bluetooth.com/specifications/adopted-specifications>, Dezember 2014. 3.2.2
- [16] 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](http://doc.utwente.nl/65223/1/Automatic_Fire_Detection.pdf), 2008. letzter Zugriff 13.07.2016. 2
- [17] Wan Hazimah Wan Ismail, Herny Ramadhani Mohd Husny, and Norhaiiza 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
- [18] 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
- [19] 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
- [20] Tyco Fire Safety. Consultant's Guide for Designing Fire Detection Alarm Systems. [http://tfppemea.com/en/\\_layouts/fsassets/Docs/](http://tfppemea.com/en/_layouts/fsassets/Docs/)

- Detection\_FireClass/UKFireClassConsultantsGuide(LR).pdf. letzter Zugriff 23.07.2016. 3.1
- [21] 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, 5
  - [22] 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
  - [23] 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