Berliner Hochschule für Technik
Fachbereich VI – Informatik und Medien
Studiengang Medieninformatik

Berliner Hochschule für Technik

Abschlussarbeit zur Erlangung des Grades
Bachelor of Science (B.Sc.)

Welche Routingverfahren eignen sich für ein iPhone-ad-hoc-Netzwerk auf Großveranstaltungen?

Autor
Simon Núñez Aschenbrenner
Matrikelnummer 908606
simon@bleep.chat

Tag der Abgabe
2. August 2024

Gutachter

Prof. Dr. Ing. Zbigniew Jerzak	Berliner Hochschule für Technik
Prof. Dr. rer. nat. Rüdiger Weis	Berliner Hochschule für Technik

Kurzfassung

Diese Bachelorarbeit beschäftigt sich mit der Evaluierung effizienter Peer-to-Peer-Routingverfahren für mobile Ad-hoc-Netzwerke (MANET) unter Verwendung von *Bluetooth Low Energy* (BLE) auf iOS Geräten. Im Fokus steht die Frage, wie trotz Ausfällen oder Überlastungen der Mobilfunkinfrastruktur z.B. auf Großveranstaltungen Telekommunikation ermöglicht werden kann.

Es wurden bestehende Anwendungen analysiert, darunter *Bluetooth Mesh, Serval, FireChat, Berty, Briar* und *Bridgefy*. Dabei zeigte sich, dass sie entweder eingestellt wurden oder zentrale Anforderungen eines solchen Telekommunikationsdienstes nicht erfüllen. Außerdem wurden mehrere Routingverfahren untersucht, kategorisiert und schließlich zwei für einen Vergleich ausgewählt: Der weiterleitungsbasierte *Disconnected Transitive Communication* (DTC) sowie der vervielfältigungsbasierte *Binary Spray and Wait* (BSaW) Algorithmus.

Im praktischen Teil der Arbeit wurde die App **bleep** von Grund auf entwickelt. Sie demonstriert eine verbindungslose Ende-zu-Ende-Nachrichtenvermittlung an Unicast-Adressen in einem vermaschten, verzögerungstoleranten MANET und implementiert beide Algorithmen, um sie bzw. ihre Energieeffizienz unter realen Bedingungen testen zu können. Es stellte sich allerdings heraus, dass die Anwendung vor allem wegen Einschränkungen im iOS Framework *Core Bluetooth* nicht die gewünschte Funktionalität erreichen konnte.

Ein theoretischer Vergleich ergab, dass entgegen der Erwartung DTC kein effizienteres Routing als BSaW ermöglicht. Dies liegt an einer hohen Anzahl erforderlicher Übertragungen auf der Netzzugangsschicht. Die Ergebnisse deuten zudem darauf hin, dass *Bluetooth Mesh* eine bessere Grundlage für den beschriebenen Telekommunikationsdienst darstellt.

Der Quellcode meiner Softwareanwendung ist unter folgendem Link einsehbar:
github.com/simon-nunez-aschenbrenner/bleep-eval

Inhaltsverzeichnis

Einleitung

Motivation

Telekommunikation auf Großveranstaltungen

Telekommunikation spielt eine fundamentale Rolle für die soziale Interaktion. Die Möglichkeit über weite Strecken zu kommunizieren und Informationen auszutauschen hat die Art und Weise, wie Menschen arbeiten, lernen und sich unterhalten, grundlegend verändert. Ohne funktionierende Telekommunikationsdienste fühlen wir uns isoliert und von unserem sozialen Netzwerk abgeschnitten, was zu Einsamkeit, Hilflosigkeit und emotionalem Stress führen kann. In Ausnahmesituationen wie medizinischen Notfällen oder Naturkatastrophen wird die Koordination von Hilfsmaßnahmen erschwert — mit lebensbedrohlichen Konsequenzen. Auch der Zugang zu wichtigen Informationen wie Nachrichten, Wetterberichten und öffentlichen Warnungen wird unterbrochen, was die Entscheidungsfindung in kritischen Momenten negativ beeinflusst. Insgesamt zeigt sich, dass eine gestörte Telekommunikationsinfrastruktur nicht nur den täglichen Komfort und die soziale Interaktion beeinträchtigt, sondern auch ernsthafte Risiken für Sicherheit und Unversehrtheit mit sich bringt.

Großveranstaltungen wie Musikfestivals, Sportevents, Messen, Konferenzen und Demonstrationen ziehen oft zehntausende Menschen an, die in kurzer Zeit auf begrenztem Raum zusammenkommen. Dies stellt eine enorme Belastung für die bestehende Mobilfunkinfrastruktur dar und führt immer wieder zu Engpässen, Überlastungen und Ausfällen, welche weitreichende Folgen haben können: Notfallkommunikation wird erschwert, Informationsflüsse stocken und die gesamte Erlebnisqualität der Veranstaltung leidet.

Auf geplanten Veranstaltungen werden deshalb manchmal mobile Funkmasten zur Unterstützung der vorhandenen Basisstationen eingesetzt. Dies kann effektiv und transparent für Besuchende Überlastungen im Mobilfunknetz verhindern. Aufgrund von hohen Kosten, logistischen und technischen Herausforderungen sowie regulatorischen Hürden kann dies nicht auf allen Großveranstaltungen erfolgen. Vor allem nicht auf (meist kurzfristig anberaumten) Demonstrationen, auf denen in manchen Ländern eine erschwerte oder gar absichtlich gestörte Kommunikationsfähigkeit der Teilnehmenden sogar von staatlicher Seite gewünscht sein kann. Wie Menschen deshalb bei Protesten in autoritären Staaten alternative Telekommunikationsdienste benutzt haben, beschreibe ich im Kapitel *Bestehende Anwendungen*.

Smartphone-ad-hoc-Netzwerke

Eine Alternative zu bestehenden Telekommunikationsdiensten, die keine zusätzliche Hardware erfordert, da sie die meisten Menschen bereits besitzen, ist ein Smartphone-ad-hoc-Netzwerk (SPAN). Dieses Konzept basiert auf der Schaffung eines verteilten, mobilen Netzwerks, das aus den Smartphones der Teilnehmenden der Großveranstaltung besteht. Im Gegensatz zu traditionellen Netzwerken, die auf zentrale Basisstationen und verwaltende Infrastruktur angewiesen sind, ermöglichen Ad-hoc-Netzwerke eine direkte Kommunikation zwischen den Geräten. Die integrierten Funkmodule der Smartphones werden genutzt, um ein Peer-to-Peer-Netzwerk zu bilden, das unabhängig von jeder Netzwerkinfrastruktur ist. Jedes Gerät im Netzwerk fungiert dabei als Router, der Datenpakete an andere Geräte weiterleitet. So können auch Geräte miteinander kommunizieren, die nicht direkt miteinander verbunden sind.

Im Bereich mobiler Ad-hoc-Netzwerke (MANET) wird schon seit den 70er Jahren und bis heute intensiv geforscht. Sie sind widerstandsfähiger gegen Ausfälle und Störungen, sind flexibel und skalierbar, verursachen geringere Betriebskosten und können den Datenschutz sowie die Privatsphäre durch erschwerte Überwachungsmaßnahmen erhöhen. Insbesondere das Routing mit begrenzten Topologieinformationen sowie die limitierten Systemressourcen stellen jedoch komplexe Herausforderungen dar. Aber auch die begrenzte Reichweite und Stabilität der Funkverbindungen, niedrigere Datenübertragungsraten, hohe Latenzen sowie eine erhöhte Anfälligkeit für Angriffe und Störungen stellen signifikante Nachteile eines Smartphone-ad-hoc-Netzwerks dar.

Aktuell ist mobile Peer-to-Peer-Kommunikation abseits von direkten Datentransfers wie Apples *AirDrop* nicht verbreitet. Apps wie *FireChat* haben SPANs in der Vergangenheit bereits mit kurzzeitigem Erfolg realisiert. Sie entstanden meist aus derselben Motivation, wie ich sie hier beschrieben habe. Kommunikationsanwendungen können allerdings nur dann nachhaltig erfolgreich sein, wenn sie von einer ausreichend großen Anzahl von Menschen genutzt werden. Es ist sicherlich sehr schwer, mit populären Anwendungen wie *WhatsApp* zu konkurrieren, die ihren Nutzer*innen zudem viel mehr Funktionen bieten können, weil sie auf konventionelle Telekommunikationsinfrastruktur zurückgreifen. Der Vorteil einer Anwendung, die unabhängig von jener ist, ergibt sich erst nach deren Ausfall oder einer potenziellen Gefährdung durch jene.

Obwohl sie für jede Person auf einer Großveranstaltung äußerst hilfreich wäre und die nötigen Hard- und Softwaretechnologien bereits seit mehreren Jahren existieren, gibt es derzeit keine weit verbreitete Kommunikationsanwendung, die ein Smartphone-ad-hoc-Netzwerk umsetzt.

Ziel

Ich möchte die grundsätzliche Umsetzbarkeit einer solchen Anwendung demonstrieren und dabei vor allem das essentielle Routingproblem thematisieren: Wie können Nachrichten sinnvoll in einem SPAN vermittelt werden? Die Beantwortung dieser Frage ist nicht direkt zu erschließen und hat bereits viele verschiedene Lösungsvorschläge hervorgebracht. Ich möchte die Hypothese prüfen, ob ein komplexerer Ansatz, der stets nur eine Kopie der Nachricht im Netzwerk weitervermittelt, effizienter als ein einfacherer Ansatz ist, der Nachrichten vervielfältigt, damit mindestens eine das Ziel erreichen wird.

Zu diesem Zweck werde ich in dieser Arbeit zwei exemplarische Routingverfahren auswählen und miteinander vergleichen sowie die Entwicklung einer iOS App dokumentieren, die diese implementiert. Mein ursprüngliches Ziel, so die beiden Routingverfahren unter realen Bedingungen testen und evaluieren zu können, habe ich nicht erreicht. Ich habe dennoch eine Menge interessanter Erkenntnisse zur Umsetzung eines iPhone-ad-hoc-Netzwerks gewonnen.

Methodik

Begriffe

Geräte, die Teil des Netzwerks sind, werden als Knoten bezeichnet und anhand einer Adresse identifiziert. Sind sie an einer Nachrichtenübermittlung beteiligt, werden sie einzeln als Relay und gemeinsam als Route bezeichnet. Ein Hop beschreibt eine Übermittlung. Knoten, die Ursprung einer Nachricht sind, werden als Quelle bezeichnet und ein Knoten, der Adressat einer Nachricht ist, heißt Ziel. Zwei Knoten, die miteinander verbunden sind, sind Kontakte. Nachrichten können in einzelne Datenpakete aufgeteilt werden, der Einfachheit halber sei aber angenommen, dass jedes Paket genau eine komplette Nachricht umfasst.

Umfang

Einerseits soll die Softwareanwendung End-to-end-Kommunikation — also einen Nachrichtenaustausch zwischen zwei nicht direkt miteinander verbundenen Knoten — über ein dezentrales, selbstorganisiertes, robustes, skalierbares Funknetzwerk demonstrieren, das sich aus eng vermaschten, mobilen Knoten zusammensetzt. Eine ausgereifte Form der App würde sich an Teilnehmende einer Großveranstaltung oder Betroffene einer (Natur-)Katastrophe richten, die — aufgrund von überlasteter Mobilfunkinfrastruktur — SMS-ähnliche Textnachrichten über einen alternativen Telekommunikationsdienst austauschen wollen.

Andererseits soll die Softwareanwendung eine wiederholbare Testung der zwei gewählten Routingverfahren unter realen Bedingungen ermöglichen. Die Algorithmen werden oft nur mit Computersimulationen evaluiert. Diese liefern zwar saubere Messdaten, die sich gut zum Vergleich mit anderen Algorithmen eignen, spiegeln aber nur bedingt die Realität wider. So sorgen z.B. Eigenschaften der Übertragungstechnologie dafür, dass das Routingverfahren nicht so effizient operieren kann, wie es seine Beschreibung vermuten lässt. Im Vergleich der beiden Routingverfahren setze ich den Fokus auf ihre Zuverlässigkeit und Energieeffizienz. Letztere ist für eine mobile Anwendung besonders wichtig.

Im Rahmen dieser Arbeit konzentriere ich mich auf das Routing als Grundvoraussetzung für den Nachrichtenaustausch und schränke mein Systemmodell daher wie folgt ein: Zur Vermeidung von Interoperabilitätsproblemen setze ich homogene Netzwerkknoten voraus — Apple iPhones, auf denen meine Softwareanwendung installiert und geöffnet ist. Die Anwendung enthält keine Funktionalität zur Verwaltung von Kontakten und Nachrichten (außer das Versenden und Empfangen) und auch keinerlei Merkmale, die Sicherheit und Privatsphäre schaffen. Ich schließe also die Existenz böswilliger Akteure komplett aus und gehe davon aus, dass alle Nutzer*innen absolut vertrauenswürdig sind und das System ausschließlich für den intendierten Zweck nutzen. Meine App setzt zudem keinen Fokus auf Datenratenerhöhung, Latenzreduzierung und Lastenverteilung. All diese Aspekte sind zweifellos wichtig in einem Kommunikationsdienst — insbesondere Datensicherheit und -schutz sollten von Anfang an bedacht werden —, stellen aber noch einmal ganz eigene Herausforderungen dar. Sie erfordern meist spezielle Lösungen aufgrund der Art des Netzwerks und würden den Rahmen dieser Arbeit schlicht sprengen.

Aufbau

Zunächst werde ich in *Grundlagen und verwandte Arbeiten* das verzögerungstolerante Netzwerk einführen, die Entscheidung für die Funktechnologie *Bluetooth Low Energy* begründen und ihre wichtigsten Eigenschaften erläutern. Danach thematisiere ich die Routingverfahren. Nach einem Überblick über bestehende Anwendungen erkläre ich meine *Implementierung* sowie die Schwierigkeiten, die ich dabei hatte. Ich gehe auf die Entwicklungsumgebung, das zentrale Framework *Core Bluetooth*, den Aufbau der Anwendungslogik und die Benutzungsoberfläche ein. Im Abschnitt *Evaluierung* erläutere ich meinen Plan sowie die gescheiterte Testdurchführung und schließe mit einem theoretischen Vergleich der Energieeffizienz beider Routingverfahren ab. Meine Erkenntnisse fasse ich anschließend noch in einem *Fazit* zusammen.

Grundlagen und verwandte Arbeiten

Verzögerungstolerante Netzwerke

Da die Knoten mobil sind, sind alle Verbindungen intermittierend und als unzuverlässig zu betrachten. Knoten können auch für längere Zeit (Sekunden bis Stunden) kontaktlos sein und müssen Pakete daher entsprechend zwischenspeichern können (*Store-Carry-Forward*). Daraus folgt eine äußerst asynchrone Kommunikation mit hohen Latenzen, die sich von klassischen paketvermittelnden Netzwerken wie dem Internet unterscheidet, das nur geringe Latenzen auf der Route von der Quelle zum Ziel toleriert. Die Netzwerke werden daher als *Delay-Tolerant Network* (DTN) bzw. *Intermittently Connected Mobile Network* (ICMN) bezeichnet. Mobile Knoten haben zudem limitierte Energiekapazitäten im Vergleich zu stationären Geräten, weshalb die Energieeffizienz so wichtig ist.

Die Mobilität ist aber nicht nur nachteilig. So können z.B. menschliche Bewegungsmuster und Beziehungen dabei helfen, Daten von einem Knoten über längere Strecken transportieren zu lassen. Dieses *Data Mule* Konzept kann helfen, Bereiche mit geringer Knotendichte zu überbrücken (z.B. der Weg zwischen Zeltplatz und Bühnen auf einem Musikfestival).

Nachrichten sollen über mehrere Knoten hinweg transportiert werden können, nicht nur direkt von einer Quelle zum Ziel (*Multi-Hop/End-to-End* im Gegensatz zu *Single-Hop/Point-to-Point*). Dabei sollten die Designziele eines verteilten, dezentralisierten und selbstkonfigurierenden (Ad-hoc-)Netzwerks eingehalten werden. Ein (logisches) Overlay-Netz mit einer vermaschten Topologie kann dies ermöglichen. Konkret habe ich nach Routingverfahren gesucht, die für eine **verbindungslose Ende-zu-Ende-Paketvermittlung an Unicast-Adressen in einem vermaschten, verzögerungstoleranten, mobilen Ad-hoc-Netzwerk** geeignet sind. Die Kennwerte dieses Netzwerks sind zum Teil bedingt durch das Übertragungsverfahren *Bluetooth Low Energy* und gestalten sich wie folgt:

- Großer Durchmesser aufgrund der geringen BLE Übertragungsdistanzen, ggf. verringert aufgrund der hohen Knotendichte auf Großveranstaltungen
- Irregulärer und (aufgrund der beschränkten Anzahl aktiv verbundener BLE Geräte) niedriger Grad
- Bisektionsweite, Symmetrie und Konnektivität sind stark abhängig von der tatsächlichen Knotendichte, aber voraussichtlich eher hoch
- Die Skalierbarkeit schätze ich ebenfalls als hoch ein, wobei diese stark vom verwendeten Routingverfahren abhängt

Hintergrundausführung

Peer-to-Peer-Netzwerke profitieren von einer hohen Verfügbarkeit ihrer Knoten, da jeder potentiell dazu genutzt werden kann, um Nachrichten weiterzuvermitteln. In einem vollständig verteilten System mit gleichwertigen (und hochmobilen) Knoten kommt erschwerend hinzu, dass alle Geräte prinzipiell ständig erreichbar sein müssen, wenn sie Nachrichten, die für sie bestimmt sind, zeitnah empfangen wollen. Es lässt sich nicht vorhersagen, wann Übertragungen stattfinden. Die maßgebliche Anforderung an das Übertragungsverfahren war daher, dass es im Hintergrund funktioniert — also auch dann, wenn die App nicht aktiv genutzt wird.

Grundsätzlich bietet das iOS Betriebssystem verschiedene Mechanismen, die es Apps ermöglichen, Aufgaben im Hintergrund auszuführen. Einige dieser sogenannten *Background Modes* sind vom Internet oder einem physischen Ortswechsel abhängig oder sie sind nur für ganz spezifische Anwendungen wie VoIP zugelassen. Wieder andere Mechanismen ermöglichen es zwar, einen beliebigen Code für eine Ausführung im Hintergrund einzuplanen, allerdings niemals dauerhaft und stets gesteuert vom Betriebssystem — basierend auf verschiedenen Faktoren und außerhalb der Kontrolle des App-Entwickelnden. Diese Einschränkungen verhindern eine freie Wahl des Übertragungsverfahrens. Die einzige Möglichkeit, mit der eine App auch im Hintergrund dauerhaft Funksignale empfangen (und versenden) kann, sind die Nutzung zweier *Background Modes*, die *Bluetooth Low Energy* nutzen. Hierfür muss die App zwar gestartet worden sein, das Gerät kann aber gesperrt sein oder eine andere Anwendung im Vordergrund ausführen. Die beiden Modi entsprechen den beiden Rollen, die ich im folgenden Kapitel zu diesem Funkstandard erläutere.

Andere Funktechnologien in modernen Smartphones sind unter anderem IEEE 802.11 (WLAN), *Ultra Wide Band* (UWB) und *Near Field Communication* (NFC). Um diese zu verwenden, muss sich eine App im Vordergrund befinden. NFC ist zudem nur für einen Datenaustausch über sehr kurze Distanzen (wenige Zentimeter) geeignet. Mobilfunktechnologien wie 5G sind keine Option, da das Betriebssystem keinen direkten Zugriff auf sie ermöglicht. Eine ideale Basis für eine Peer-to-Peer-Anwendung wäre Apples *Multipeer Connectivity* Framework, das unter anderem die Grundlage für den Datentransferdienst *AirDrop* bildet und eine Kombination aus Bluetooth und IEEE 802.11 verwendet. Aber auch dieses ist nur im Vordergrund sinnvoll nutzbar.

Im Android Betriebssystem gibt es ähnliche Einschränkungen, diese habe ich aber nicht weiter untersucht, da ich eine iOS Anwendung entwickeln wollte.

Bluetooth Low Energy

Überblick

Bluetooth Low Energy (kurz *Bluetooth LE* oder BLE) ist eine frequenzmodulierte Kurzstrecken-Funktechnik, die von der klassischen Bluetooth-Technologie (Bluetooth BR/EDR [Blu24a]) abgeleitet wurde und auch im selben 2,4-GHz-ISM-Band arbeitet, jedoch getrennt von ihr zu betrachten und auch nicht mit ihr kompatibel ist. BLE ist robust, u.a. durch zyklische Redundanzprüfung (CRC) sowie ein adaptives Frequenzsprungverfahren, und mit einem Fokus auf einen geringeren Energieverbrauch vor allem für das *Internet der Dinge* (IoT) vorgesehen. Es ermöglicht bidirektionale Verbindungen und (je nach verwendetem Übertragungsmodus) relativ kurze Übertragungsdistanzen zwischen 10 und 100 Metern sowie eine geringe Datenrate zwischen 125 Kb/s und 2 Mb/s. [Blu24a]

Laut dem Marktforschungsunternehmen *ABI Research* wurden im vergangenen Jahr fünf Milliarden Bluetooth-fähige Geräte verkauft (wovon die meisten auch BLE unterstützen) — mit steigender Tendenz. Bluetooth sei damit der am weitesten verbreitete Funkstandard der Welt. Praktisch alle neuen Smartphones, Tablets und Personal Computer unterstützen BLE. Die Technologie wird von der *Bluetooth Special Interest Group (Bluetooth SIG)* kontinuierlich weiterentwickelt. Insbesondere im Anwendungsbereich Gerätenetzwerke sollen bis 2028 Geräteverkäufe um den Faktor 2,44 ansteigen. [Blu24b]

Während der COVID-19-Pandemie wurde zur Nachverfolgung der Kontaktpersonen *Bluetooth Low Energy* genutzt, um im Hintergrund kontinuierlich, anonym und energieeffizient Daten unter Smartphones auszutauschen. [ApG20]

Verbindungen bestehen immer zwischen sogenannten Centrals und sogenannten Peripherals, wobei erstere primär Daten konsumieren und letztere sie primär bereitstellen. Diese Rollen ähneln dem *Client-Server-Model* in einer Sterntopologie. Es entstehen einzelne abgeschlossene Zellen (Piconets), die gemeinsam eine Zell-Topologie (Scatternet) bilden. Geräte können die Rollen wechseln und auch gleichzeitig bedienen, aber es sind niemals zwei Centrals oder zwei Peripherals miteinander verbunden. Centrals verwalten die Verbindung und Peripherals die Daten. Centrals können mit sieben Peripherals gleichzeitig verbunden sein. Typischerweise ist ein Peripheral immer nur mit einem Central zur selben Zeit verbunden. Geräte identifizieren sich anhand von MAC Adressen im EUI-48 Format. Mehr dazu im Kapitel *Advertisingdaten und Adressrandomisierung*. [App13]

Verbindungsaufbau

Eine unauthentifizierte Verbindung wird grob wie folgt hergestellt [App13]:

- **Advertising:** Ein Peripheral, das sich verbinden möchte, bewirbt seine Services in regelmäßigen Zeitintervallen auf speziellen Advertising-Kanälen
- **Scanning/Discovery:** Ein Central empfängt die Advertisements und ermittelt die Empfangssignalstärke (RSSI). Anschließend kann es noch weitere Informationen vom Peripheral anfordern (*Scan Request* / *Response*)
- **Connection:** Das Central initiiert die Verbindung mit dem Peripheral und schlägt Verbindungsparameter vor, die anschließend verhandelt werden
- Ggf. **Subscription:** Ein Central indiziert, dass es ab sofort Daten einer Notify- oder Indicate-Charakteristik (siehe unten) erhalten möchte

Softwaremodell

Daten werden vom Peripheral in Form sogenannter Charakteristiken (Characteristics) gespeichert und weitergegeben. Charakteristiken enthalten je einen Datenwert und werden in sogenannten Services zusammengefasst. Sie können mit sogenannten Beschreibern (Descriptors) genauer definiert werden. Services, Charakteristiken und Beschreiber werden anhand von 128 bit UUIDs identifiziert. Das Central entscheidet nach erfolgter Verbindung auf welche Charakteristik(en) es zugreifen möchte. Die Eigenschaft einer Charakteristik wird vom Peripheral festgelegt und bestimmt, wie ein Central mit dem Datenwert interagieren kann. Es gibt unter anderem die folgenden Eigenschaften [App24]:

- **Broadcast** (Mitteilungen ohne Verbindung über den Beschreiber)
- **Read** (Lesen durch das Central)
- **Write without response** (Schreiben durch das Central ohne Empfangsbest.)
- **Write** (Schreiben durch das Central mit Empfangsbestätigung)
- **Notify** (Mitteilungen an verbundene Centrals ohne Empfangsbestätigung)
- **Indicate** (Mitteilungen an verbundene Centrals mit Empfangsbestätigung)

Hinzu kommen Eigenschaften, die eine Authentifizierung voraussetzen. BLE ermöglicht einen sicheren Datenaustausch nach einem Kopplungsverfahren (Pairing bzw. Bonding), das nach dem initialen Verbindungsaufbau erfolgen würde. Zur Authentifizierung wird eine Out-of-Band (OOB) Methode genutzt, die eine Interaktion durch die anwendende Person erfordert (z.B. den Abgleich eines Zahlencodes). Der in dieser Arbeit behandelte Anwendungsfall setzt aber voraus, dass auch einander unbekannte Geräte Daten austauschen können. Folglich sind die Eigenschaften für sicheren Datenaustausch nicht relevant. Für Datensicherheit und -schutz wären höhere Protokollschichten verantwortlich.

Bluetooth Mesh

Diese Erweiterung von BLE wurde zum ersten Mal 2017 als Spezifikation von der Bluetooth SIG veröffentlicht und zuletzt 2023 aktualisiert [Blu23]. Sie zielt in erster Linie auf IoT-Anwendungen ab, Anwendungsfelder sind unter anderem die Gebäudeautomation und Industrie (z.B. Bestandsverfolgung). Es gibt aber auch Softwareprojekte, die die Technologie für Chat-Anwendungen nutzen (z.B. github.com/chrisballinger/BLEMeshChat, github.com/fwcd/distributed-chat).

Bluetooth Mesh ist ein robustes, sicheres und sehr ausgereiftes Protokoll. Aufgrund der unterschiedlichen energetischen Ressourcen von BLE Geräten definiert es verschiedene Rollen. Im Freundschaftskonzept beispielsweise können Geräte mit höherer Energiekapazität Nachrichten für Geräte mit geringerer Kapazität empfangen. So sind letztere zwar adressierbar, nehmen aber nicht als Relays am Netzwerk teil. Die Spezifikation listet u.a. die folgenden Designziele:

- **One-to-One-** (Unicast) und **One-to-Many-** (Multicast) **Kommunikation**
- **Weiterleitung von Nachrichten** (Multi-Hop-Kommunikation)
- **Sicherheit** (z.B. Lausch-, MITM-, Replay-, Trash-Can-, Brute-Force-Attacken)
- **Geringe Latenz, Resilienz, Vorwärts- und Rückwärtskompatibilität**

Diese Ziele gehen u.a. zulasten der Energieeffizienz. In ihrer Analyse des Protokolls schreiben Hernández-Solana et al., dass „BLE mit Mesh einige seiner Energiesparziele verliert“ [SPG20]. Im Kapitel Routingverfahren folgen weitere Details zu *Bluetooth Mesh* sowie Gründe, die gegen eine Verwendung sprechen.

Routingverfahren

Kategorisierung

Es gibt viele Peer-to-Peer-Kommunikationsprotokolle, die aufgrund der verwendeten Overlay-Netz-Struktur und relativ hohem Energieverbrauch weniger gut für mobile Anwendungen geeignet sind. Aber auch für ICMNs gibt es bereits über hundert [BTA11] verschiedene Ansätze, wie Pakete vermittelt werden können. Sie lassen sich anhand der folgenden Dimensionen kategorisieren, wobei aufgrund von Mischformen eine trennscharfe Unterscheidung oft nicht möglich ist. Kategorien schließen sich zudem teilweise gegenseitig aus (z.B. kann *Link State Routing* nicht rein reaktiv operieren und isolierte Knoten können Relays nicht geographisch oder an der Quelle wählen). Die Liste ist unvollständig und lässt einige bekannte Routingstrategien — wie nicht adaptive oder zentralisierte Ansätze — aus, da sie nicht zum gewählten Netzwerktyp passen.

- **Operationsmodus**
  - Proaktiv (tabellenbasiert): Routen werden im Vorfeld ermittelt
  - Reaktiv: Routen werden für jedes Paket neu ermittelt
  - Hybrid (z.B. *Backward Learning*): Pakete enthalten Routeninformationen
- **Topologiewissen**
  - *Link State*: Knoten können alle Routen berechnen, weil sie die Verbindungen aller Knoten kennen und miteinander teilen
  - Distanzvektor: Knoten kennen und teilen miteinander eine Teilmenge aller Routen, nämlich solche, die von ihnen ausgehen
  - Isoliert: Knoten teilen keine Routinginformationen miteinander
- **Entscheidungszeitpunkt**
  - Quelle: Die Route wird beim Erstellen des Nachrichtenpakets festgelegt
  - *Hot Potato*: Das nächste Relay wird direkt nach Paketempfang ausgewählt
  - Kontakt: Wenn sich ein Knoten neu verbindet, wird ermittelt, ob er sich als Relay für eines oder mehrere gespeicherte Datenpakete eignet
  - Periodisch: Ereignisunabhängige Entscheidung in bestimmten Intervallen
- **Relayauswahl**
  - Opportunistisch (z.B. Flooding): Pakete werden routenunabhängig an verbundene Knoten übertragen, ggf. mit einfachen Einschränkungen (zufällige Knotenselektion, Duplikatsvermeidung, Lebensdauerkontrolle)
  - Geographisch: Die physische Position der Knoten bestimmt die Routen
  - Geplant (vorhersehbar): Bekannte Mobilitätsmuster werden genutzt
  - Nach Nützlichkeit (wahrscheinlichkeitsbasiert): Pakete werden auf dem (vermutet) bestmöglichen Weg weitergegeben
- **Kopienkontrolle**
  - Vervielfältigungsbasiert: Eine (un)begrenzte Anzahl an Kopien eines Nachrichtenpakets wird vermittelt
  - Weiterleitungsbasiert: Es existiert stets nur eine einzige Kopie eines Nachrichtenpakets im Netzwerk
- **Adressierungsschema**
  - Hierarchisch (Baumstruktur): Organisiert Adressen in mehreren Ebenen
  - Flach: Alle Adressen sind gleichwertig auf einer einzigen Ebene
  - Hybrid: Gleichwertige Adressen innerhalb hierarchisch geordneter Zonen

Darüber hinaus setzen Routingverfahren unterschiedliche Schwerpunkte bzgl. der Dienstgütekriterien. Diese umfassen u.a. **Zuverlässigkeit** (*Delivery probability / ratio*), **Pfadlänge** (Anzahl der Hops), **Energieeffizienz**, **Datenrate** (u.a. bestimmt durch die *Maximum Transmission Unit* MTU), **Latenz** (*Delivery delay*), **Speichergröße** (*Buffer size*), **Fairness**, **Lastenverteilung**, **Resilienz** (Funktionsfähigkeit bei Knotenausfällen), **Skalierbarkeit**, **Sicherheit** und **Datenschutz**.

Im Anhang befindet sich eine *grafische Übersicht dieser Kategorisierung*.

Auswahlkriterien

Knoten sollen möglichst wenig identifizierbare Informationen (wie ihre Position) bereitstellen müssen (sei es aus Datenschutz- oder technischen Gründen) und Verwaltungsdaten sollen in einem ohnehin sehr volatilen System gering gehalten werden. Trotzdem sollen gute Routingentscheidungen getroffen werden.

Infrage kommen daher isoliert, reaktiv agierende Algorithmen, die das optimale nächste Relay zum letztmöglichen Zeitpunkt bestimmen, also entweder, sobald sich ein neuer Kontakt verbindet, oder periodisch. Sie sollten ein flaches Adressierungsschema verwenden, da die dynamische Topologie häufige Änderungen an einer hierarchischen Struktur erfordern würden. Hinsichtlich der Relayauswahl sind nur opportunistische und wahrscheinlichkeitsbasierte Verfahren geeignet, wobei ich mich aus Gründen der Energieeffizienz auf letztere konzentriert habe. Weitere fokussierte Dienstgütekriterien sind Zuverlässigkeit, Pfadlänge, Fairness, Resilienz, Skalierbarkeit und Speicherreduzierung.

Letztlich stellte sich die Frage, ob sich für den Anwendungsfall besser ein vervielfältigungs- oder ein weiterleitungsbasiertes Verfahren eignen würde. Erstere sind einfacher zu implementieren, belasten das Netzwerk aber stärker, da jedes Nachrichtenpaket an mehrere Relays übertragen wird. Letztere agieren zielgerichteter, sind aber auch komplexer umzusetzen und unter Umständen weniger zuverlässig. Auf der Suche nach zwei passenden Vertretern dieser beiden Kategorien habe ich eine Reihe von Routingverfahren untersucht.

Untersuchte Algorithmen

Distributed Post Office [Rea14], Kademlia [MaM02], *Landmarks Based Routing* [Rea14], *Mobile Low-Energy Adaptive Clustering Hierarchy* (LEACH-Mobile) [KiC06], *Sqrt(n) Mesh Routing* [Rea14] und *Zone Based Routing* (ZBR) [AhH11] verwenden u.a. hierarchische Adressierungsschemata wie eine verteilte Hashtabelle (DHT) und sind daher ungeeignet. Die schnell wechselnde Topologie im ICMN erschwert die für DHTs nötige konsistente Verteilung und Lokalisierung von Daten, was zu viel Overhead und ineffizienten Datenzugriffen führt.

Associativity-Based Routing (ABR) [AWD03], *Ad-Hoc On-Demand Distance Vector Routing* (AODV) [AWD03], *Ant-Colony-Based Routing Algorithm* (ARA) [AWD03], *Better Approach to Mobile Ad-Hoc Networking* (B.A.T.M.A.N.) [Aic07], *Destination-Sequenced Distance-Vector Routing* (DSDV) [AWD03], *Dynamic Source Routing* (DSR) [AWD03]*, Delay-Tolerant Link-State Routing* (DTLSR) [VSK21] und *Optimized Link State Routing Protocol* (OLSR) [AWD03] agieren proaktiv mit Topologiewissen, wodurch sie ebenfalls ungeeignet sind.

Epidemic Routing [VaB00] und Bitmessage [War12] wählen Relays u.a. opportunistisch und das Protokoll von Jones et. al. [JLW05] nutzt u.a. Flooding-basiertes *Link-State Routing* zum Austausch von Topologieinformationen. Solche Algorithmen erreichen aufgrund maximaler Vervielfältigung zwar eine hohe Zuverlässigkeit — dies geht jedoch auf Kosten der Energieeffizienz, die für mobile Anwendungen gegenüber der Zuverlässigkeit priorisiert werden muss.

Contact Graph Routing (CGR) [ABB15], *Distance Routing Effect Algorithm for Mobility* (DREAM) [AWD03], *Location Aware Sensor Routing* (LASeR) [HaA16] und MaxProp [BGJ06] nutzen u.a. Positionsdaten oder Zeitpläne zur Relayauswahl. Die Verwendung solcher persönlichen Daten, die Standort und Mobilität der Nutzer*innen im Netzwerk preisgeben, sind für den Anwendungsfall dieser Arbeit (insbesondere hinsichtlich der Nutzung auf Demonstrationen) ungeeignet. Außerdem bewegen sich die Menschen mehr oder wenig zufällig und geben nur wenige Informationen über ihre Bewegungspläne preis, weshalb gar keine gute Datengrundlage für diese Routingverfahren existiert.

Bevor ich die beiden selektierten Algorithmen, *Binary Spray and Wait* (BSaW) [SPR05] als vervielfältigungsbasiertes Verfahren und *Disconnected Transitive Communication* (DTC) [ChM01] als weiterleitungsbasiertes Verfahren detaillierter vorstelle, möchte ich zunächst noch auf das Routing von *Bluetooth Mesh* eingehen und warum ich es als ungeeignet eingeschätzt habe.

Bluetooth Mesh

Alle Angaben basieren auf der *Mesh Protocol Specificatio*n der Bluetooth SIG [Blu23], ich verzichte daher auf wiederholte Quellenverweise. *Bluetooth Mesh* verwendet ein flaches Adressierungsschema und nutzte zunächst ausschließlich ein opportunistisches, vervielfältigungsbasiertes, *Managed Flooding* genanntes Routingverfahren. Dabei erhalten alle Kontakte alle Nachrichten, die sie nicht bereits empfangen haben und deren TTL noch nicht erreicht wurde.

Vor einem knappen Jahr wurde die Spezifikation dann um das sogenannte *Directed Forwarding* Verfahren erweitert. Es handelt sich um einen proaktiven Distanzvektor-Routingalgorithmus, der grundsätzlich weiterleitungsbasiert ist. *Bluetooth Mesh* setzt eine quasi ununterbrochene Weitergabe von Datenpaketen voraus und ist daher nicht verzögerungstolerant. Das Problem instabiler Routen wird anerkannt und soll dadurch reduziert werden, dass mehrere verschiedene Routen von der Quelle zum Ziel (sogenannte Lanes) gesucht bzw. genutzt werden und gefundene Routen regelmäßig überprüft werden (mithilfe sogenannter Echos). Beide Strategien erhöhen die Belastung und den Energieverbrauch des Netzwerks.

Sowohl die Designziele als auch die beiden angewendeten Routingverfahren sind inkompatibel mit den für diesen Anwendungsfall gewählten *Auswahlkriterien* und spiegeln den Fokus des Protokolls auf IoT-Anwendungen wider. Aufgrund der Komplexität des Protokolls (insbesondere hinsichtlich der beinhalteten Sicherheitsmechanismen) habe ich es als unrealistisch eingeschätzt *Bluetooth Mesh* bzw. dessen Referenzimplementierung von *Nordic Semiconductor* (github.com/NordicSemiconductor/IOS-nRF-Mesh-Library) in der vorhandenen Zeit um die gewählten Routingverfahren *Binary Spray and Wait* und *Disconnected Transitive Communication* zu erweitern.

Binary Spray and Wait

Alle Angaben basieren auf der Arbeit von Spyropoulos et al. [SPR05], weshalb ich auf wiederholte Quellenverweise verzichte. Das vervielfältigungsbasierte Routingverfahren *Spray and Wait* wurde speziell für den Einsatz in einem ICMN entwickelt und 2005 vorgestellt.

Die Vermittlung teilt sich in zwei Phasen auf: In der Spray-Phase werden so viele Kopien eines Datenpakets an Relays verteilt, dass „wenigstens eine davon das Ziel schnell findet (mit hoher Wahrscheinlichkeit)“ [SPR05]. Wird in dieser ersten Phase das Ziel noch nicht angetroffen, können Knoten mit einer Kopie des Datenpakets dieses in der darauffolgenden Wait-Phase noch direkt an den Zielknoten übertragen.

Neben der Anzahl an Kopien ist die Heuristik, die in der Spray-Phase angewendet wird, von entscheidender Bedeutung. Die Autor*innen empfehlen eine Methode, die sie *Binary Spray and Wait* (BSaW) nennen und in der Knoten (mit mehr als einer Kopie einer Nachricht) die Hälfte ihrer Kopien an einen anderen Knoten (der noch keine Kopie dieser Nachricht hat) übertragen und die andere Hälfte (für weitere Übertragungen) behalten. Haben Knoten nur noch eine Kopie einer Nachricht, wechseln sie in die Wait-Phase.

Der einfach zu implementierende Algorithmus soll in vielfältigen Szenarien einsetzbar und bei gleich bleibender Zuverlässigkeit mit einem Fokus auf die Latenz und Skalierbarkeit effizienter als Flooding sein, da deutlich weniger Kopien desselben Nachrichtenpakets versendet werden. Ist die Lebenszeit (TTL) eines Pakets lang genug, entstehen beim Flooding mindestens so viele Übertragungen wie Knoten im Netzwerk vorhanden sind. *Spray and Wait* hingegen kann in einem Netzwerk aus 100 Knoten mit nur maximal acht Kopien/Übertragungen eine maximale Latenz einhalten, die drei mal so hoch wie die optimale Latenz ist — wie sie mit einem theoretischen, Orakel-basierten Routingverfahren wäre, das stets die beste Route kennt.

Spray and Wait agiert rein reaktiv mit isolierten Knoten in einem flachen Adressierungsschema. Es verteilt Datenpakete vervielfältigungsbasiert und opportunistisch an Kontakte, sobald sie sich verbinden, limitiert aber die Gesamtanzahl der Kopien bzw. Übertragungen. Die Autor*innen sehen ihren Algorithmus als eine Kombination aus „der Geschwindigkeit von epidemischem Routing mit der Einfachheit und Sparsamkeit direkter Übertragung“ [SPR05] und setzen ihren Fokus auf Zuverlässigkeit, Latenz, Energieeffizienz (damit indirekt auch auf die Pfadlänge) sowie die Skalierbarkeit.

Mithilfe von theoretischen Beweisen und Softwaresimulationen zeigen sie, dass *Spray and Wait* „(i) unter geringer Last [...] viel weniger Übertragungen und vergleichbare oder geringere Latenzen als Flooding-basierte Schemata hat, (ii) unter hoher Last signifikant bessere Latenzen und geringere Übertragungen als Flooding-basierte Schemata hat, (iii) äußerst skalierbar ist [...] und (iv) es im laufenden Betrieb einfach angepasst werden kann, um bestimmte Dienstgüteanforderungen zu erfüllen, sogar in unbekannten Netzwerken. [Sie] zeigen auch, dass *Spray and Wait*, während es nur eine Handvoll Kopien pro Nachricht verwendet, vergleichbare Latenzen zu einem [theoretischen] Orakel-basierten, optimalen Schema erreichen kann, das die Latenz minimiert, weil es die niedrigstmögliche Anzahl an Übertragungen verwendet“ [SPR05].

Die Autor*innen belegen, dass BSaW optimal hinsichtlich seiner Latenz im Vergleich zu allen *Spray and Wait* Varianten ist, wenn die zufällige Verteilung der Knoten unabhängig und identisch ist. Außerdem zeigen sie, wie sich die minimale Anzahl an Kopien für eine bestimmte erwartete Latenz (relativ zur optimalen) berechnen und approximieren lässt. Anhand der Zeitabstände zwischen Knotenkontakten ist das sogar möglich, wenn die Netzwerkgröße unklar ist.

Hinsichtlich der Skalierbarkeit merken die Autor*innen an, dass „bei steigender Anzahl an Knoten im Netzwerk der [Anteil] an Knoten, die Relays werden müssen, damit *Spray and Wait* dieselbe Leistung relativ zum Optimum erreichen kann, tatsächlich kleiner wird“ [SPR05].

Disconnected Transitive Communication

Dieses weiterleitungsbasierte Verfahren wurde 2001 von Chen und Murphy für mobile Ad-Hoc-Netzwerke entwickelt, die sich „kontinuierlich in viele getrennte Cluster rekonstruier[en]“ [ChM01]. Da alle weiteren Angaben auf dieser Arbeit basieren, verzichte ich auf wiederholte Quellenverweise. Es scheint prädestiniert für ein Netzwerk, das aus BLE Piconets besteht. Der Name des Verfahrens beschreibt letztlich die Zielsetzung verzögerungstoleranter Netzwerke, Ende-zu-Ende-Kommunikation ohne eine durchgehende Verbindung zu ermöglichen.

Der Algorithmus versucht, die Nachricht immer „näher“ an das Ziel zu bringen, wobei die Nähe eines Knotens definiert ist als die „Wahrscheinlichkeit, früher mit dem Ziel in Kontakt zu kommen als [das aktuelle Relay]“ [ChM01]. Sie wird auch als Nützlichkeit (Utility) eines Knotens bezeichnet. Um diese zu bestimmen, bedient sich DTC u.a. an Informationen aus der Anwendungsschicht.

DTC agiert wie *Spray and Wait* reaktiv mit isolierten Knoten in einem flachen Adressierungsschema. Im Gegensatz zum replizierenden Verfahren verteilt es Datenpakete weiterleitungs- und wahrscheinlichkeitsbasiert. Die Entscheidung, ob ein Knoten als Relay ausgewählt wird, erfolgt periodisch. Dieses Wiederentdeckungsintervall (*Rediscovery Interval* RDI) verkürzt sich bei hoher Knotenmobilität und verlängert sich bei seltenerer Veränderung der verbundenen Knoten.

Die Vermittlung umfasst zwei bis drei Phasen: Die Nützlichkeitsanfrage (*Utility probe*), die Nützlichkeitssammlung (*Utility collection*) und ggf. die Nachrichtenübermittlung (*Message redistribution*). Vereinfacht ist der Ablauf wie folgt: Der Knoten, der eine Nachricht weiterleiten möchte, sendet im Wiederentdeckungsintervall eine Nützlichkeitsanfrage an alle verbundenen Knoten (als Broadcast), woraufhin diese ihre individuelle Nützlichkeit für die (direkte) Übermittlung dieser Nachricht an das Ziel berechnen und zurücksenden. Ist die Nützlichkeit eines verbundenen Knotens höher als die eigene, wird der Knoten diesen Kontakt als Relay auswählen und die Nachricht weiterleiten.

Neben der Adresse eines Knotens definieren die Autor*innen „fünf Merkmale [...], die genutzt werden können, um die Verlässlichkeit der Nützlichkeitsberechnung [eines Knotens] zu erhöhen. Dazu gehören die Liste mit zuletzt [verbundenen Knoten], die Liste mit am häufigsten [verbundenen Knoten], die zukünftigen Pläne [...], das Energielevel und das Wiederentdeckungsintervall“ [ChM01]. Zusammenfassend lässt sich sagen, dass ein Knoten dann eine hohe Nützlichkeit aufweist, wenn er in der Vergangenheit (häufig) mit dem Ziel verbunden war, sich in (naher) Zukunft mit ihm verbinden wird, aufgrund seiner Energiereserven noch lange aktiv sein wird und grundsätzlich momentan aktiver ist (impliziert durch ein kürzeres Wiederentdeckungsintervall).

Im Anwendungsfall dieser Arbeit existieren keine Mobilitätspläne, denn die Knoten bewegen sich mit Menschen und daher mehr oder weniger zufällig. Grundsätzlich lassen sich aber auch aus den vergangenen Kontakten brauchbare Vorhersagen für zukünftige Kontakte treffen. Der Algorithmus *Probabilistic Routing Protocol using History of Encounters and Transitivity* (PROPHET) [VSK21] arbeitet ebenfalls nach diesem Prinzip. Darüber hinaus existieren auch noch andere, komplexere Routingverfahren wie BUBBLE Rap [HCY11], die versuchen, bekannte soziale Bewegungsmuster auszunutzen.

Um ihre Präsenz in der Reichweite anderer Knoten anzukündigen, sieht das Routingverfahren das regelmäßige Versenden sogenannter Hallo-Nachrichten vor. Dieser Aspekt macht den Algorithmus in Teilen proaktiv. „Jede Hallo-Nachricht [enthält] eine boolesche Variable, die kennzeichnet, ob das Wiederentdeckungsintervall des Sendenden zurückgesetzt wurde, seitdem die letzte Hallo-Nachricht übermittelt wurde [...] [So] kann effektiv die Dynamik des Netzwerks ermittelt werden“ [ChM01].

Die Autor*innen haben für ihr Verfahren keine Simulationen durchgeführt.

Bestehende Anwendungen

Bei der Recherche geeigneter Routingverfahren habe ich ebenfalls etablierte Peer-to-Peer-Anwendungen untersucht. Sie definieren meist nur höhere Schichten im Protokollstapel und sind auf bekannte Internetprotokolle angewiesen, die nicht ideal für verzögerungstolerantes Routing sind und zudem eine Verwendung im Rahmen dieser Arbeit erschweren. Dazu zählen zum Beispiel die Anwendungen Bitmessage, Jami, Matrix, Ricochet, SimpleX, Tox und XMPP. Das sogenannte *Internet Protocol Support Profile* (IPSP) macht IPv6 über BLE zwar nutzbar, wird aber nicht direkt vom iOS Framework *Core Bluetooth* unterstützt und müsste erst recht aufwendig implementiert werden. Im Folgenden möchte ich auf fünf Anwendungen etwas detaillierter eingehen.

Serval

Nach einem Erdbeben in Haiti im Jahr 2010 begannen australische und neuseeländischen Wissenschaftler*innen um P. Gardner-Stephen mit der Entwicklung einer „Open-Source-Mesh-Kommunikationsplattform, [die bei] Nichtverfügbarkeit jeglicher unterstützender Infrastruktur [...] viele Funktionalitäten, die von modernen Smartphones erwartet werden, nachbilden“ sollte [GCL13]. Die Forscher*innen entwickelten und testeten verschiedene Hard- und Softwareanwendungen zu diesem Zweck [GCL13]. Für das Routing wird u.a. B.A.T.M.A.N. [Ope24] verwendet, die Funkübertragung zwischen Smartphones nutzt IEEE 802.11 [GCL13]. Eine Prototypanwendung für iOS verwendet zudem Apples *Multipeer Connectivity* Framework [Gar17].

Mit der Ankündigung der Stilllegung des Hauptfinanziers *Shuttleworth Foundation* im Jahr 2022 [Shu22] scheint auch jede Entwicklungsarbeit an Serval eingestellt worden zu sein. Die Projektwebsite (servalproject.org) ist nicht mehr erreichbar und die letzten Aktualisierungen im GitHub-Repository (github.com/servalproject) liegen mehr als zwei Jahre zurück.

FireChat

Weltweite Bekanntheit erreichte die kostenlose *Closed Source* App des amerikanischen Unternehmens *Open Garden* [Nar15]. 2014 wurde erst eine iOS App und wenig später eine Android App veröffentlicht [Yu_14]. *FireChat* ermöglichte zunächst nur einen öffentlichen Austausch von Text- und Bilddaten, wobei die Nachrichten mithilfe von Bluetooth, IEEE 802.11 und Apples *Multipeer Connectivity* Framework [App15] im gesamten Mesh-Netzwerk verteilt wurden [Yu_14]. Später war dann auch eine private und verschlüsselte Ende-zu-Ende-Kommunikation möglich [Ope19].

Viele Menschen nutzten die App bereits kurz nach ihrer Veröffentlichung zur Umgehung von Netzsperren bei Protesten im Iran, Irak und in Hongkong (allein in Hongkong wurde sie an einem Tag 100.000 Mal heruntergeladen) [Put14]. Später war sie laut Unternehmensangaben „in den Top-10-Social-Networking-Apps-Listen im [Apple] App Store und Google Play [Store] in 124 Ländern“ und wurde nicht nur bei Protesten, sondern auch nach „Naturkatastrophen in Ecuador und Kashmir sowie auf abseits der Zivilisation stattfindenden Veranstaltungen wie dem *Burning Man* [Festival] und der *Summit at Sea* [Konferenz]“ genutzt [Ope19].

Seit 2020 ist die App nicht mehr in Apples *App Store* verfügbar und die Website des Unternehmens (opengarden.com) ist unerreichbar. Die Gründe, die zur Stilllegung dieser viralen App geführt haben, sind unklar. Wahrscheinlich haben u.a. bekannte und funktionsreichere *Instant Messaging* Apps wie *WhatsApp* und *Facebook Messenger* eine weite und anhaltende Verbreitung, wie sie Chat-Apps im Allgemeinen, aber Peer-to-Peer-Anwendungen im Speziellen zum Bestehen benötigen, erschwert.

Berty und Briar

Berty (berty.tech) und Briar (briarproject.org) sind zwei erwähnenswerte, nach wie vor existierende, mobile Peer-to-Peer-Anwendungen, die eine internetunabhängige Kommunikation über BLE bzw. IEEE 802.11 ermöglichen — derzeit allerdings nur in einem sogenannten *Single Hop Mesh*, also nur für Direktverbindung vom Sendenden zum Empfangenden [Alt21] [Ber23]. Um Nachrichten über mehrere Knoten zu vermitteln, sind auch sie auf das Internet angewiesen. Beide Anwendungen sind *Open Source*, wobei Briar nicht für iOS verfügbar ist.

Bridgefy

Die Software des 2014 gegründeten mexikanischen Unternehmens Bridgefy (bridgefy.me), wird sowohl als SDK bereitgestellt, um andere Anwendungen um Peer-to-Peer-Funktionalität zu erweitern, als auch als *Instant Messaging* App für iOS und Android angeboten. Letztere erlangte u.a. Bekanntheit bei den Protesten in Hongkong 2019/2020 [Des19] und zählt laut des Beschreibungstexts im *Apple App Store* mehr als 9,5 Millionen Nutzende. Bridgefy implementiert genau die Funktionalität, die den Kern dieser Arbeit bildet: Eine Ende-zu-Ende-Kommunikation über BLE. Für eine weitere Betrachtung war sie dennoch ungeeignet, da die Software *Closed Source* und die Nutzung des SDKs kostenpflichtig ist. Hinzu kommt, dass das Unternehmen in der Vergangenheit wiederholt in der Kritik stand, falsche Angaben, insbesondere hinsichtlich der Sicherheit seines Produkts, gemacht zu haben [AEP22].

Nach diesem Überblick über die Grundlagen und verwandten Arbeiten möchte ich nun auf den praktischen Teil meiner Arbeit eingehen, die Entwicklung meiner Anwendung **bleep**.

Implementierung

Entwicklungsumgebung

Ich traf die Entscheidung, die Anwendung in Swift (Version 5.10.1) zu programmieren, da diese Sprache von Apple für die Entwicklung auf seinen Plattformen aktuell empfohlen wird und gut dokumentiert ist. Außerdem wollte ich mich im Umgang mit dieser Sprache und ihren Frameworks *SwiftUI* und *SwiftData* verbessern. *Core Bluetooth* ist ein weiteres Framework von Apple, das ich intensiv genutzt habe und auf das ich im folgenden Kapitel genauer eingehe.

Apples IDE Xcode ermöglichte es, die Anwendung immer wieder relativ schnell auf iPhones zu testen und den Programmablauf z.B. über Log-Nachrichten direkt in der IDE zu überwachen. Auch die Erstellung der grafischen Benutzungsoberfläche war in Xcode über die integrierte Vorschau sehr einfach möglich. Um meine Software auf mehr als drei Geräten installieren zu dürfen, war eine kostenpflichtige Registrierung für das *Apple Developer Program* nötig.

Zur Versionsverwaltung habe ich Git mit einem *Remote Repository* auf GitHub verwendet (github.com/simon-nunez-aschenbrenner/bleep-eval). Notizen und Diagramme habe ich in Apples Freeform Software und mit PlantUML erstellt.

Core Bluetooth

Apples „*Core Bluetooth* Framework stellt die Klassen zur Verfügung, die [...] Apps benötigen, um mit BLE [...] Funktechnologie zu kommunizieren“ [App24]. Es nutzt intensiv Delegation (im deutschen Sprachraum auch als Stellvertreter-Entwurfsmuster bekannt). So wird beispielsweise beim Verbindungsaufbau eines Central mit einem Peripheral dem Betriebssystem die Referenz einer *CBPeripheralDelegate* Instanz mitgeteilt. Aktualisiert ein Peripheral dann das Datum einer Indicate-Charakteristik, ruft das Betriebssystem die folgende Methode in der *CBPeripheralDelegate* Instanz auf.

```
optional func peripheral(
    _ peripheral: CBPeripheral,
    didUpdateValueFor characteristic: CBCharacteristic,
    error: (any Error)? )
```

Im Rumpf dieser Methode kann dann definiert werden, wie mit dem Datenwert (ein Attribut der *CBCharacteristic* Instanz) weiter verfahren wird.

Statt einer detaillierten Beschreibung des Frameworks verweise ich auf die entsprechende *Apple Developer Documentation* [App24] sowie die Dateien *ConnectionManager.swift*, *CentralManager.swift* und *PeripheralManager.swift* in meinem Quellcode, die *Core Bluetooth* verwenden.

An dieser Stelle möchte ich vielmehr einige Einschränkungen erwähnen, die die Implementierung erheblich erschwert haben. Da die Ausführungen die Netzzugangsschicht und Point-to-Point-Übertragungen betreffen, schreibe ich im Folgenden von sendenden und empfangenden Geräten. Die Datenpakete enthalten entweder eine Mitteilung oder eine Antwort und werden als Datenwert einer Charakteristik übertragen. Detaillierte Informationen über die übermittelten Daten finden sich im Kapitel *Mitteilungen und Antworten*.

Advertisingdaten und Adressrandomisierung

Apple schränkt stark ein, welche und wie viele Daten in Advertising-Paketen enthalten sein dürfen. Dies limitiert die Möglichkeiten von Empfängern, einschätzen zu können, ob ein Sender relevante Daten für sie bereit hält und ob es sich lohnt, sich mit ihm zu verbinden. Obwohl Bluetooth seit Version 5 Advertising-Pakete mit bis zu 255 Byte Größe unterstützt [Oev22], limitiert *Core Bluetooth* diese für Apps auf 28 Byte, wovon ein einziger beworbener Service bereits 18 Bytes (128 Bit UUID zzgl. 2 Byte Header-Information) beanspruchen würde. Das Framework erlaubt zusätzliche 10 Byte (8 Byte Nutzdaten) für einen Gerätenamen, die in der *Scan Response* übertragen werden. Diese insgesamt 38 Byte werden allerdings auch nur dann übertragen, wenn sich die App gerade im Vordergrund befindet. Läuft die App im Hintergrund, können ihre Services nur dann von einem Central entdeckt werden, wenn es explizit nach ihnen sucht. Daher kann weder die Service-UUID noch der Gerätename dafür verwendet werden, Informationen über den Netzwerkknoten zu kodieren. [App24]

„Die BLE Protokolle zwingen Geräte ihre öffentliche MAC Adresse regelmäßig zu ändern“ [JVA21]. iOS setzt diese Adressrandomisierung „zum Schutz der Privatsphäre und zur Wahrung des Datenschutzes“ [App21] folglich ebenfalls um. Damit sich bekannte Geräte identifizieren können, muss die sogenannte private Adresse während eines authentifizierten Verbindungsaufbaus „vom jeweils anderen Gerät aufgelöst werden können. Die private Adresse wird unter Verwendung des IRK Schlüssels [...] generiert, der beim Koppeln ausgetauscht wird.“ [App21]. Wie bereits erläutert, ist ein Kopplungsprozess in diesem Anwendungsfall nicht sinnvoll. Es gibt zwar Ansätze, wie von Jouans et al. [JVA21], mit denen auch unauthentifizierte Geräte identifiziert werden können, er ist aber nicht fehlerfrei und könnte nicht mit langen Kontaktpausen umgehen. Daher ist er (zusätzlich zum Implementierungsaufwand) nicht für diesen Fall geeignet.

Aufgrund des eingeschränkten Advertisings und der Adressrandomisierung kann (unabhängig vom verwendeten Routingverfahren) erst nach der Übermittlung einer Mitteilung und nur vom Empfänger entschieden werden, ob jener diese (weiter)verarbeiten oder ignorieren sollte (weil er sie z.B. bereits zuvor erhalten hat). Anschließend sollte er seine Entscheidung dem Sender über die Antwort mitteilen, damit dieser den Status der Mitteilung rückwirkend anpassen kann (z.B. Anpassung der Kopienanzahl bei BSaW). Neben einer höheren Komplexität und größerem Fehlerpotenzial (z.B. wenn die Antwort nicht empfangen wird), ist die Belastung des Netzwerks um ein Vielfaches höher, da zum einen zu jeder Mitteilung Antworten gesendet werden müssen, zum anderen aber muss der Sender stets alle Mitteilungen an alle verbundenen Empfänger übermitteln. Letztendlich wird so jede durch das Routingverfahren gewonnene Effizienz aufgrund von Flooding auf der Netzzugangsschicht zunichtegemacht.

In der Realität ist es sogar noch dramatischer: Nachdem ein Peripheral alle Mitteilungen übertragen hat und sich das Central daraufhin von ihm trennt, verbinden sich die beiden sofort wieder aufs Neue und der Prozess wiederholt sich so lange, bis die beiden Geräte außer Reichweite sind. Dies blockiert Kontakte, bei denen tatsächlich neue Nachrichten ausgetauscht werden könnten.

Sendebereitschaft

Für die Art des Datentransfers scheint ein *Publish-Subscribe-Pattern* und davon abgeleitet eine Indicate-Charakteristik am Besten geeignet. Dies entspricht auch der Empfehlung von Apple; wenn sich der Wert einer Charakteristik häufig ändert, sollte keine Read-Charakteristik verwendet werden [App13]. Allerdings gibt es (im Gegensatz zu Reads) bei Indications keine Methode, die nach erfolgreicher Datenübertragung in der *CBPeripheralManagerDelegate*-Instanz des Peripherals aufgerufen würde, wodurch die nächste Datenübertragung angestoßen werden könnte. Die Methode der *CBPeripheralManager*-Instanz, an die die zu übertragenden Daten übergeben werden, gibt zwar einen booleschen Wert zurück, der angibt, ob die Datenübertragung erfolgreich war. Dieser ist jedoch im realen Gebrauch absolut keine Garantie dafür, dass nun auch die nächste Datenübertragung initiiert werden kann — meistens ist sie fehlgeschlagen, „weil die zugrunde liegende Warteschlange, die zum Übertragen des aktualisierten Charakteristik-Werts verwendet wird, [noch] voll ist“ [App24]. Die Methode *peripheralManagerIsReady(toUpdateSubscribers:)* ermöglicht zwar anschließend eine Wiederaufnahme, allerdings wird sie vom Betriebssystem nur im Fehlerfall aufgerufen. [App24]

Problemlösungen

Zur Vermeidung von unnötigen Übertragungen verwende ich eine zufällige ID als Gerätenamen des Peripherals, die sich ändert, sobald das Peripheral neue Nachrichten bereithält. Centrals merken sich die ID aus vergangenen Kontakten und können so identifizieren, ob sich ein Verbindungsaufbau lohnt. Auch wenn der Gerätename eine ausreichend große Netzwerkadresse kodieren könnte, ist es nicht sinnvoll, ihn für diesen Zweck zu nutzen, da sich Centrals sonst nur ein einziges Mal mit einem Peripheral verbinden würden und nicht wieder, sobald es neue Nachrichten bereithält. Außerdem würde es das Privatsphäre-Ziel von Bluetooth komplett umgehen. Es handelt sich nur um eine Zwischenlösung, da sie nur funktioniert, wenn die App auf dem Peripheral im Vordergrund läuft.

Als weitere Maßnahme wird das Advertising auf Geräten blockiert, die gerade Mitteilungen senden oder empfangen. Sonst würden Centrals nach der ersten erhaltenen (neuen) Mitteilung mit einer neuen ID Advertising-Pakete senden. Diese würden dann unter anderem auch von dem gerade noch selbst sendenden Peripheral empfangen, das sich folglich als Central verbinden würde. Die beiden Geräte würden sich dann gegenseitig ohne Ende Mitteilungen schicken, die beide bereits erhalten haben.

Jede Nachricht hat eine festgelegte TTL, um sie nach Ablauf dieser Lebenszeit komplett von der Weitervermittlung auszuschließen. So wird die Menge der zu sendenden Mitteilungen auf allen Geräten begrenzt.

Schließlich reduziere ich die Anzahl der Antworten, indem ich grundsätzlich ein Scheitern der Übertragung oder eine Ablehnung der Mitteilung annehme und Antworten nur im positiven Fall erfolgen müssen. Eine verbindungsbehaftete Indicate-Charakteristik für das Versenden der Mitteilungen ermöglicht mir (im Unterschied zur verbindungslosen Broadcast-Charakteristik) zudem, die für die Zeit der Verbindung sicher gleichbleibende, öffentliche MAC Adresse des Peripherals zu nutzen, um nur eine Antwort, zielgerichtet an jenes, zu übermitteln.

Um fehlgeschlagene Übertragungen effizient wiederholen zu können, verwende ich eine temporäre Datenstruktur, die einmalig beim Verbindungsaufbau mit allen sendbaren Mitteilungen gefüllt wird. Wird eine Mitteilung erfolgreich übermittelt, wird sie in dieser Datenstruktur entsprechend markiert. Nach einer gescheiterten Übertragung wird über die Datenstruktur iteriert und mit der nächsten ungesendeten Mitteilung fortgefahren. Wenn alle Mitteilungen erfolgreich übermittelt wurden, wird die Datenstruktur verworfen und vom Peripheral eine spezielle Nachricht gesendet, die das Ende der Übertragungen markiert. Daraufhin trennt das Central die Verbindung.

Anwendungslogik

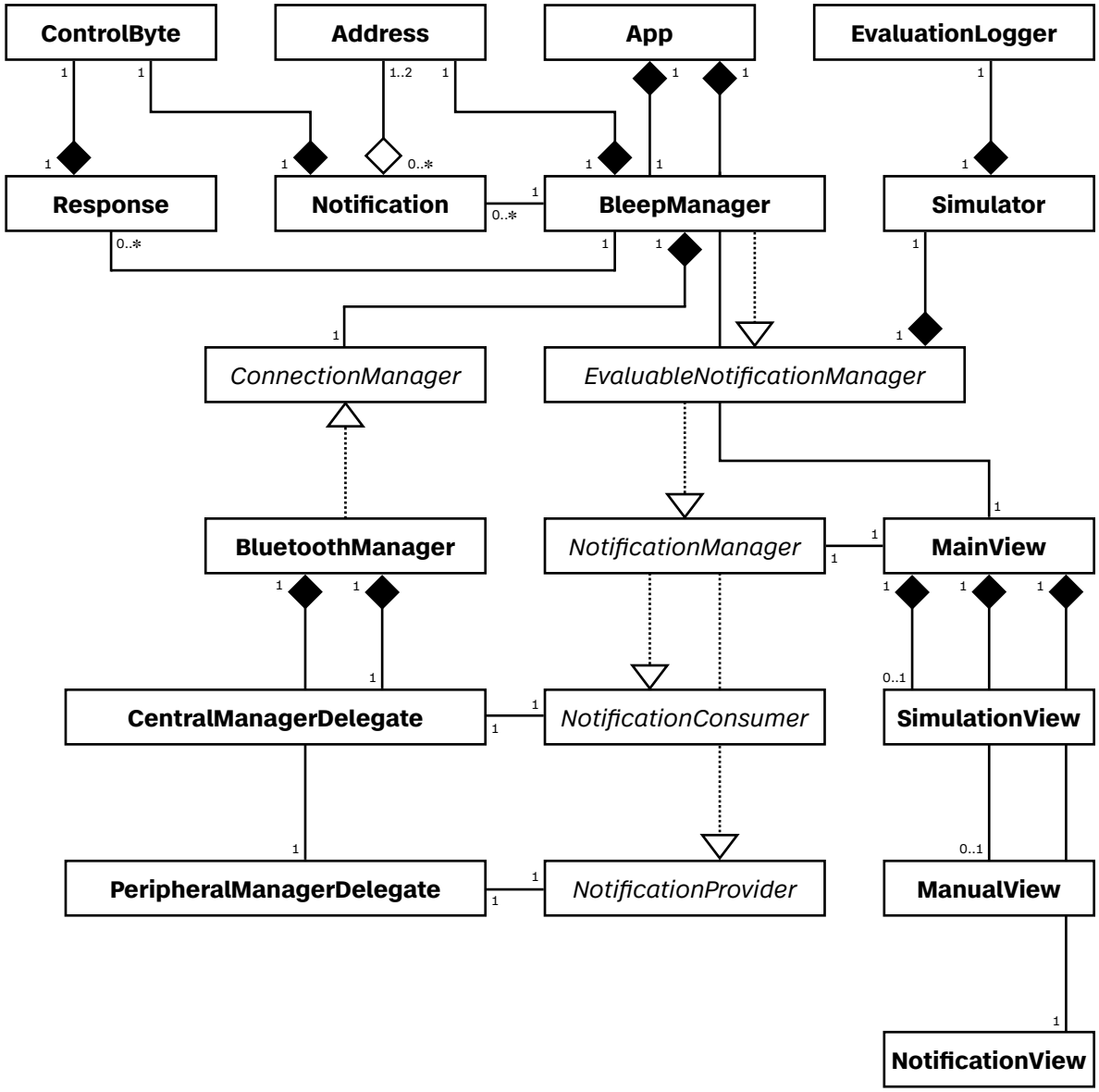


ABBILDUNG 1: EINFACHES UML KLASSENDIAGRAMM

Für einen Überblick der Anwendungslogik, sind im obigen UML Diagramm alle implementierten Strukturen und Klassen (**hervorgehoben**) sowie Protokolle (Schnittstellen, *kursiv*) mit ihren Assoziationen und Multiplizitäten dargestellt. Die durchgezogene Linie einer Assoziation ist ggf. mit einer Raute dekoriert. Ausgefüllte Rauten beschreiben Kompositionen, nicht ausgefüllte Rauten eine Aggregation. Die Pfeile beschreiben eine Schnittstellenrealisierung.

Der Quellcode ist unter github.com/simon-nunez-aschenbrenner/bleep-eval einsehbar. In der Datei *App.swift* findet sich der Eintrittspunkt der Anwendung. Im Folgenden gehe ich auf einige Aspekte meiner Anwendung detaillierter ein.

Mitteilungen und Antworten

```
@Model class Notification { … }
@Model class Address { … }
class Response { … }
struct ControlByte { … }
```

Zur Übermittlung von Nachrichten werden im Netzwerk Mitteilungen und Antworten ausgetauscht. Mitteilungen werden in der Klasse Notification definiert. Diese bildet auch das Datenmodell für das Persistieren mithilfe von *SwiftData*. Das Framework wurde von Apple erst vor einem Jahr als Nachfolger von *CoreData* vorgestellt. Es soll die Datenpersistenz insbesondere im Zusammenspiel mit *SwiftUI* vereinfachen und verwendet ebenfalls eine deklarative Syntax. Die Implementierung gestaltete sich für mich trotzdem als sehr herausfordernd, weil sich zu diesem neuen Framework noch nicht so viele Informationen fanden und ich lange mit der Beseitigung von Programmierfehlern beschäftigt war.

Byte #	**0**	**1 … 32**	**33 … 64**	**65 … 96**	**97 … 104**	**105 … 523**
Mitteilung	*ControlByte* (1 Byte)	ID (Hash) (32 Byte)	Zieladresse (Hash) (32 Byte)	Quelladresse (Hash) (32 Byte)	Versand-Zeitstempel (8 Byte)	Nachricht (0…419 Byte)
Antwort	*ControlByte* (1 Byte)	ID (Hash) (32 Byte)				

TABELLE 1: AUFBAU DER SERIALISIERTEN DATEN VON MITTEILUNG UND ANTWORT

Der relevante Quellcode befindet sich in den Dateien *NotificationModel.swift* und *AddressModel.swift*. Mitteilungen sind 105 bis 524 Byte groß, beschränkt durch die maximale Paketgröße, die *Core Bluetooth* erlaubt. Sie enthalten die Nutzdaten mit bis zu 419 Byte (z.B. 419 ASCII Zeichen) und folgende Metadaten:

- **ID:** Ein mit SHA-256 gehashter 16 Byte String, wobei die ersten 8 Byte der Quelladresse entsprechen und die folgenden 8 Byte eine beim Erstellen generierte, zufällige, *zero padded*, vorzeichenlose 64 Bit Ganzzahl sind
- **Ziel-/Quelladresse:** Eine mit SHA-256 gehashte vorzeichenlose 64 Bit Ganzzahl (Ich verwende Hashwerte für die Adressen, da ich ursprünglich geplant hatte, gewisse Sicherheitsmerkmale zu implementieren)
- **Versand-Zeitstempel:** Eine 64 Bit Gleitkommazahl, die das Zeitintervall (*TimeInterval*) in Referenz zu 00:00:00 UTC am 1.1.2001 angibt
- **ControlByte:** Mitteilungen und Antworten steuern den Programmablauf mithilfe der Informationen, die in ihrem *ControlByte* kodiert sind

Routingverfahren (0…3)		**Typ**	**Empfangskontrolle** (0…3)		**Kennzahl** (0…15)
2	Binary Spray and Wait (BSaW)	**Mitteilung**	0	Niemand / Ende	
			1	Jeder	Kopienanzahl (1…16) (kodiert als 0…15)
			2	Nur Ziel	
		Antwort	1	Akzeptiert	
			2	Akzeptiert vom Ziel	
3	Disconnected Transitive Communication (DTC)	**Mitteilung**	0	Niemand / Ende	
			1	Hallo	RDI-Reset seit l. Hallo (0…1)
			2	Ziel / Höh. Nützlichk.	Nützlichkeitsschwellenwert (0…15)
			3	Ziel / Nützlichk.anfr.	
		Antwort	2	Akzeptiert	
			3	Nützlichkeitsantwort	Nützlichkeit (0…15)
7 … 6		**Bit #**	**5 … 4**		**3 … 0**

TABELLE 2: AUFBAU DES CONTROLBYTE FÜR BSAW UND DTC

Übertragung

```
protocol ConnectionManager {
    var maxNotificationLength: Int! { get }
    init(notificationManager: NotificationManager)
    func advertise()
    func publish(_ data: Data) -> Bool // Mitteilung/Nützlichkeitsanfrage
    func publish(_ data: Data, to id: String) -> Bool // Mitteilung mit DTC
    func write(_ data: Data, to id: String) -> Bool // Antwort
    func disconnect(_ id: String)
    func disconnect()
    func reset()
}
class BluetoothManager: ConnectionManager { … }
class PeripheralManagerDelegate: CBPeripheralManagerDelegate { … }
class CentralManagerDelegate: CBCentralManagerDelegate, CBPeripheralDelegate
    { … }
```

In den Dateien *ConnectionManager.swift*, *PeripheralManager.swift* und *CentralManager.swift* finden sich die Klassen, die die Logik für die BLE Kommunikation auf der Netzzugangsschicht definieren. Sie regeln den Verbindungsaufbau, versenden Daten und geben empfangene Daten an die höhere Protokollschicht weiter, die in *Vermittlung* näher beschrieben wird. Der dort definierte *NotificationManager* greift nur auf das Interface *ConnectionManager* zu, wodurch die Adaptierung anderer Übertragungsverfahren als BLE einfach umzusetzen wäre.

Zwischenzeitlich hatte ich eine abgewandelte Form des *Apple Notification Center Service* (ANCS) implementiert. Dieser hat eigentlich „zum Ziel [BLE] Zubehör Zugriff auf [iOS] Mitteilungen zu geben“ [App14a]. Das Zusammenspiel der drei Charakteristiken *Notification Source*, *Control Point* und *Data Source* sowie die Formatierung der von ihnen verwalteten Daten schienen mir aber bestens geeignet, um viele Nachrichten in kurzer Zeit zuverlässig zu übermitteln. Der spezifizierte Ablauf ist auch prädestiniert dafür, zunächst nur die Metadaten einer Nachricht über die *Notification Source* Charakteristik zu übertragen und je nach Antwort des Centrals auf der *Control Point* Charakteristik danach erst den Nachrichteninhalt über die *Data Source* Charakteristik zu senden. Die Umsetzung gestaltete sich jedoch als äußerst kompliziert, weshalb ich den Ablauf deutlich vereinfacht habe und aktuell nur noch zwei Charakteristiken verwende: *Notification Source* und *Notification Response*.

Vermittlung

```
protocol NotificationProvider {
    var blocked: Bool { get set } // Verh. Advertising während Datenaustausch
    func transmit() // Zum Reinitiieren einer gescheiterten Datenübertragung
    func receiveResponse(_ data: Data, from id: String) -> Bool
}
protocol NotificationConsumer {
    var blocked: Bool { get set } // Verh. Advertising während Datenaustausch
    func receiveNotification(_ data: Data, from id: String)
}
protocol NotificationManager: NotificationProvider, NotificationConsumer {
    var address: Address! { get } // Adresse dieses Netzwerkknotens
    var contacts: [Address] { get }
    // inbox array enthält an diesen Netzwerkknoten adressierte Mitteilungen
    var inbox: [Notification] { get }
    var maxMessageLength: Int! { get }
    func send(_ message: String, to destinationAddress: Address)
}
protocol EvaluableNotificationManager: NotificationManager {
    // Routingverfahren kann über type Attribut jederzeit geändert werden
    var type: NotificationManagerType! { get set }
    var lastRSSIValue: Int8? { get set } // Hilft rssiThreshold zu bestimmen
    var rssiThreshold: Int8 { get set } // Simulationsparameter
    var notificationTimeToLive: TimeInterval { get set } // Routingparameter
    var initialRediscoveryInterval: TimeInterval { get set } // DTC-Parameter
    var countHops: Bool { get } // Simulationsparameter
    var storedHashedIDsCount: Int { get } // Anzahl empfangener Mitteilungen
    var simulator: Simulator! { get }
    func setInitialNumberOfCopies(to value: UInt8) throws // BSaW-Parameter
    func reset() // Zurücksetzen aller Parameter auf ihre Default-Werte
}
class BleepManager: EvaluableNotificationManager { … }
```

Die wesentliche Vermittlungslogik ist in *NotificationManager.swift* definiert und am einfachsten dem *Aktivitätsdiagramm* im Anhang oder den *Sequenzdiagrammen* im Kapitel *Theoretischer Vergleich* entnehmbar.

Simulation

```
class Simulator { … }
class EvalutionLogger { … }
```

In der Datei *Evaluation.swift* befindet sich der Quellcode, der den Versand von Nachrichten simuliert und alle ein- und ausgehenden Nachrichten in Form einer CSV Datei protokolliert. Die folgenden Daten werden erfasst: Gerätename, Nummer des Testlaufs, Geräteadresse, aktueller Zeitstempel, Nachrichtentyp (Mitteilung / Antwort), Nachrichten ID (Hash), verwendetes Routingverfahren, Empfangskontrolle, Kennzahl, Quelladresse (Hash), Versand-Zeitstempel, Zieladresse (Hash), Empfangs-Zeitstempel sowie die Anzahl zurückgelegter Hops.

Für den simulierten Nachrichtenversand verwende ich das Dispatch Framework von Apple, um den Hauptthread in den Warteintervallen nicht zu blockieren. Diese Nebenläufigkeit war trivial mit einer *DispatchSourceTimer* Instanz umzusetzen. Über die Benutzungsoberfläche lassen sich das Warteintervall (Frequency) in Sekunden und die zufällige Abweichung (Variance) sowie die Zieladressen einstellen. Beim Versand wird dann eine davon zufällig ausgewählt.

Benutzungsoberfläche

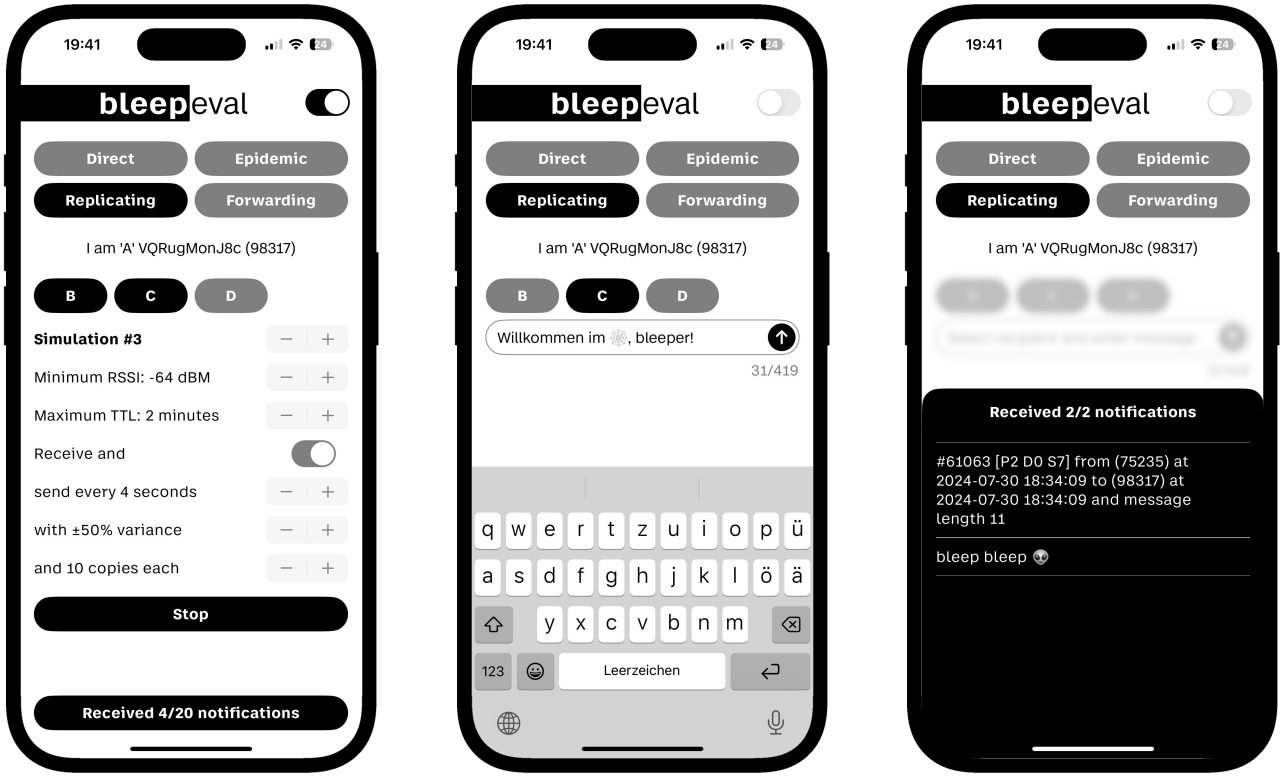


ABBILDUNG 2: DREI SCREENSHOTS DER APP-BENUTZUNGSOBERFLÄCHE

Ich habe einen eingeschränkt funktionsfähigen Prototypen entwickelt, der einerseits ein einfaches Chat-Interface bietet, das die grundlegende Funktionalität meiner App **bleep** präsentiert, als auch ein Interface, mit der ein Nachrichtenversand simuliert werden kann, sodass vergleichbare Tests durchgeführt und die Routingverfahren evaluiert werden können. Ein Schalter oben rechts wechselt zwischen den beiden Ansichten. Das Routingverfahren kann eingestellt und die Parameter ggf. justiert werden.

- **Direct** (Nebenprodukt der Entwicklungsarbeit): Nachrichten werden nur direkt von der Quelle zum Ziel übertragen (*Single-Hop*, ähnlich zur Funktionalität von Berty und Briar)
- **Epidemic** (Nebenprodukt der Entwicklungsarbeit): Nachrichten werden an alle verbundenen Knoten übermittelt, die die Nachricht bisher noch nicht erhalten haben (ähnlich zu *Managed Flooding* in *Bluetooth Mesh*)
- **Replicating:** Implementierung des Binary Spray and Wait Routingverfahrens
- **Forwarding:** Implementierung des DTC Routingverfahrens

Unter der eigenen Adresse finden sich Buttons, mit denen die Zieladresse(n) ausgewählt wird bzw. werden. Diese Adressliste ist noch hartkodiert. Das Chat-Interface bietet ein Textfeld, in das beliebige Unicode-Zeichen eingegeben werden können. Unter dem Senden-Button (mit Pfeilsymbol) findet sich eine Anzeige der tatsächlichen Nachrichtengröße in Byte, da die UTF-8 kodierte Zeichen einen unterschiedlichen Speicherbedarf haben.

Das Simulations-Interface bietet Einstellungsmöglichkeiten für Simulations- und Routingparameter sowie ein kleines Hilfs-Interface zur Ermittlung eines RSSI-Schwellenwerts (nicht abgebildet). Unten befindet sich ein Button, um einen Testlauf (mit Verzögerung) zu starten und zu beenden. Im Anschluss ist das Testprotokoll aufrufbar und kann über ein iOS Interface direkt aus der App z.B. per *AirDrop* an ein anderes Apple Gerät gesendet werden (nicht abgebildet).

Beide Interfaces zeigen am unteren Bildschirmrand eine Zusammenfassung erhaltender Nachrichten (Notifications). Durch Klicken erweitert sich diese Ansicht und zeigt die Inhalte derjenigen Nachrichten, die an das Gerät adressiert waren (also nicht diejenigen, die das Gerät weitergeleitet hat). Durch Klicken auf einen Nachrichten-Eintrag werden dessen Metadaten angezeigt.

Die Dateien *MainView.swift*, *SimulationView.swift* und *ManualView.swift* beinhalten den relevanten Quellcode der Benutzungsoberfläche. Mit der deklarativen Syntax von *SwiftUI* war sie relativ schnell erstellt, aber die korrekte Implementierung der Abhängigkeiten und Interaktionen mit der Anwendungslogik war zwischenzeitlich herausfordernd.

Evaluierung

Plan

Um die Routingverfahren unter möglichst realistischen Bedingungen testen zu können, habe ich mehrere gebrauchte iPhones erworben und ausgeliehen. Ich habe den Bluetooth 4.2 Standard von 2014 als Mindestanforderung identifiziert, da damals u.a. die BLE Nutzdatenmenge (MTU) erhöht wurde [Ern14], was den Einsatz in einer Chatanwendung sinnvoller machte. Prinzipiell wären also eine ganze Reihe von Modellen, konkret ab iPhone 6 (2014) [App14b], geeignet gewesen. Eine effiziente Anwendungsimplementierung erforderte aber schließlich iOS 17 als Mindestanforderung an das Betriebssystem, weshalb ich ein iPhone 8 (2017) nicht mehr zum Testen nutzen konnte. Letztlich habe ich die folgenden fünf Geräte zum Testen verwendet: Ein iPhone XS (2018), zwei iPhone SE (2020), ein iPhone SE (2022) und ein iPhone 14 Pro (2022).

Zwei Geräte haben mithilfe des Simulators einen hochfrequenten, aber nicht unrealistischen Nachrichtenaustausch durchgeführt und so die Kommunikation zweier Menschen simuliert. Die Geräte wurden von Personen im Testfeld bewegt, waren aber stets außerhalb der direkten Übertragungsreichweite, sodass die Nachrichten über die weiteren, dort zwischen ihnen verteilten Geräte vermittelt wurden. Mithilfe des Simulators sollten für die verschiedenen Routingverfahren (und unterschiedlichen Simulations- und Routingparameter) jeweils mehrere Testdurchläufe unter möglichst gleichen Bedingungen durchführbar sein, um die Ergebnisse anschließend mitteln zu können.

Um einen direkten Nachrichtenaustausch zu verhindern und gleichzeitig kein Versuchsgelände mit mehreren 100 Metern Durchmesser zu benötigen, wurde der Verbindungsaufbau aller beteiligten Geräte mit einem Schwellenwert für die Empfangssignalstärke (RSSI) künstlich beschränkt. Hierfür wurde die Empfangssignalstärke auf der halben Minimaldistanz, die die beiden mobilen Geräte einhalten, gewählt. So haben sich nur noch Geräte miteinander verbunden, die sich innerhalb dieser halben Minimaldistanz zueinander befanden. Einige Meter Varianz sind mitzubedenken, da die Empfangssignalstärke aufgrund natürlicher Phänomene schwankt. Die Nutzung eines Schwellenwerts könnte — ebenso wie artifiziell erzeugte Nachrichten — das Ergebnis verfälschen. Denn das Übertragungsmedium wird zum einen mit (relativ zur simulierten Knotendichte unrealistisch) vielen Advertising-Paketen gefüllt, zum anderen müssen diese trotzdem von Centrals verarbeitet werden, auch wenn keine Verbindung aufgebaut und das Kommunikationsprotokoll frühzeitig abgebrochen wird.

Während dieser Tests hat jedes Gerät alle ein- und ausgehenden Nachrichten in Form einer CSV Datei protokolliert. Anschließend konnten diese Tabellen mit einem simplen Shell Skript in einer einzigen Datei zusammengefasst werden und der oder die Weg(e) jeder Nachricht anhand ihrer individuellen ID durch das Netzwerk verfolgt und ausgewertet werden. Während einer Simulation inkrementierten Relays zudem die Anzahl der zurückgelegten Hops und speicherten diesen Zahlenwert im Nachrichtenfeld der Mitteilung — mit dem Ziel eine schnelle, erste Auswertung der Routingverfahren zu ermöglichen.

Neben der Energieeffizienz sollte auch die Zuverlässigkeit der Routingverfahren bestimmt werden. Die Zuverlässigkeit lässt sich als Anteil zugestellter Nachrichten im Vergleich zu der Anzahl versendeter Nachrichten darstellen.

Durchführung

Die Evaluation ist daran gescheitert, dass die Implementierung des Datenaustauschs den Skalierungseffekten nicht gewachsen war. Mit *Binary Spray and Wait* konnten immerhin eine Handvoll Nachrichten zugestellt werden (im übrigen auch dann, wenn die App im Hintergrund war), letztlich aber lag die Zuverlässigkeit im niedrigen einstelligen Prozentbereich. Dies liegt deutlich unter den erwarteten Ergebnissen und deutet auf Fehler in der Implementierung hin, die eine weitere Auswertung nicht sinnvoll erscheinen lassen. Mit DTC konnten gar keine Nachrichten vermittelt werden.

Die *Rohdaten* eines Testdurchlaufs befinden sich im Anhang. Dabei waren drei Geräte beteiligt. Zwei Knoten haben ungefähr alle 5 Sekunden Nachrichten an vier Adressen verschickt und mit *Binary Spray and Wait* weitervermittelt. Innerhalb von ca. 93 Sekunden wurden 35 individuelle Nachrichten verschickt, wovon zwei 22 Mal erfolgreich übermittelt wurden. Eine Nachricht wurde direkt vom Ziel empfangen. Die andere war an das vierte, nicht beteiligte Gerät adressiert, wurde aber von den beiden erreichbaren empfangen. Die protokollierten Kopien- und Hop-Anzahlen zeigen, dass diese zweite Nachricht einige Male auf nicht direktem Wege empfangen wurde und das Routingverfahren somit erfolgreich angewendet wurde. Insgesamt gab es erwartungsgemäß drei Antworten. In Relation zu den 22 Übermittlungen zeigt sich bereits die Ineffizienz.

Die genauen Gründe, warum und ab welcher Belastung die Anwendung nicht mehr fehlerfrei funktionierte, konnte ich in der vorhandenen Zeit nicht mehr ermitteln. Mit hoher Wahrscheinlichkeit betreffen sie aber die Netzzugangsschicht und sind den diversen Behelfslösungen geschuldet, um die Anwendung trotz der Einschränkungen von *Core Bluetooth* zum Laufen zu bringen.

Theoretischer Vergleich

Ohne Messdaten lassen sich keine Aussagen zur Zuverlässigkeit treffen. Ein theoretischer Vergleich der Energieeffizienz beider Algorithmen ist aber möglich. Die Energiekosten für eine BLE Übertragung $E_{Uebertragung}$ sowie die Anzahl aller Kontakte K und aller Nachrichten N — in einem bestimmten Zeitraum — sind durch äußere Faktoren bestimmt. Sie können daher für beide Verfahren als gleich angenommen werden und als Grundlage für eine Gegenüberstellung dienen. K ist bestimmt durch die Knotenanzahl, -dichte und -mobilität und N ist ein Indikator für die Frequenz, in der diese Knoten Nachrichten versenden. Die Anzahl der Knoten ist unerheblich, da auf der Netzzugangsschicht jeder neue Kontakt einem neuen Knoten gleicht. Der Gesamtenergieverbrauch eines Verfahrens ist die Summe an Übertragungen pro Nachricht bzw. pro Kontakt.

Kosten einer Übertragung

Eine detaillierte Betrachtung der energetischen Kosten für die Übertragung eines Datenpakets auf der Netzzugangsschicht würde den Rahmen dieser Arbeit sprengen. Erschwerend kommt hinzu, dass Apple keine Angaben hinsichtlich des Energieverbrauchs oder der konkreten Steuerung der BLE Kommunikation auf Betriebssystem- bzw. Hardwareebene veröffentlicht. Es ist davon auszugehen, dass die Arbeitsphasen des Funkmoduls anhand einer Reihe von Faktoren laufend angepasst werden, um Energie zu sparen. Ich werde daher den Energieverbrauch auf einer abstrakten Ebene betrachten und gehe außerdem zur Vereinfachung davon aus, dass alle Übertragungen erfolgreich sind bzw. dass Übertragungen, die aufgrund der im Kapitel *Sendebereitschaft* beschriebenen Problematik scheitern, keine zusätzliche Energie verursachen.

Die Kosten für die Übertragung eines Datenpakets setzen sich im Wesentlichen aus der Sendeenergie E_{Senden} (für die Ausstrahlung der Daten), Empfangsenergie $E_{Empfangen}$ und der Verarbeitungsenergie $E_{Verarbeiten}$ (für die Bearbeitung der Daten) zusammen. Letztere sind sowohl für die Sende- als auch für die Empfangsseite zu bedenken. Hinzu kommt der Energieverbrauch $E_{Standby}$ zwischen zwei Übertragungen auf der Empfangsseite, während diese neuen Signalen lauscht. Energie, die beim Starten der Funkmodule aus einem eventuellen Schlafmodus verbraucht wird, vernachlässige ich. Die Übermittlung eines Datenwerts einer Indicate- oder Write-Charakteristik umfasst mindestens zwei Datenübertragungen: den Datenwert selbst, sofern er in ein Paket passt, wovon ich zur Vereinfachung ausgehe, sowie die Empfangsbestätigung in Rückrichtung. Der Energieverbrauch einer Übertragung lässt sich daher grob wie folgt zusammenfassen:

$$E_{Uebertragung} = 2 \cdot (E_{Senden} + E_{Empfangen} + E_{Standby} + 2 \cdot E_{Verarbeiten})$$

Der Ablauf der Implementierung beider Algorithmen ist zum besseren Verständnis hier noch einmal in einem Sequenzdiagramm zusammengefasst:

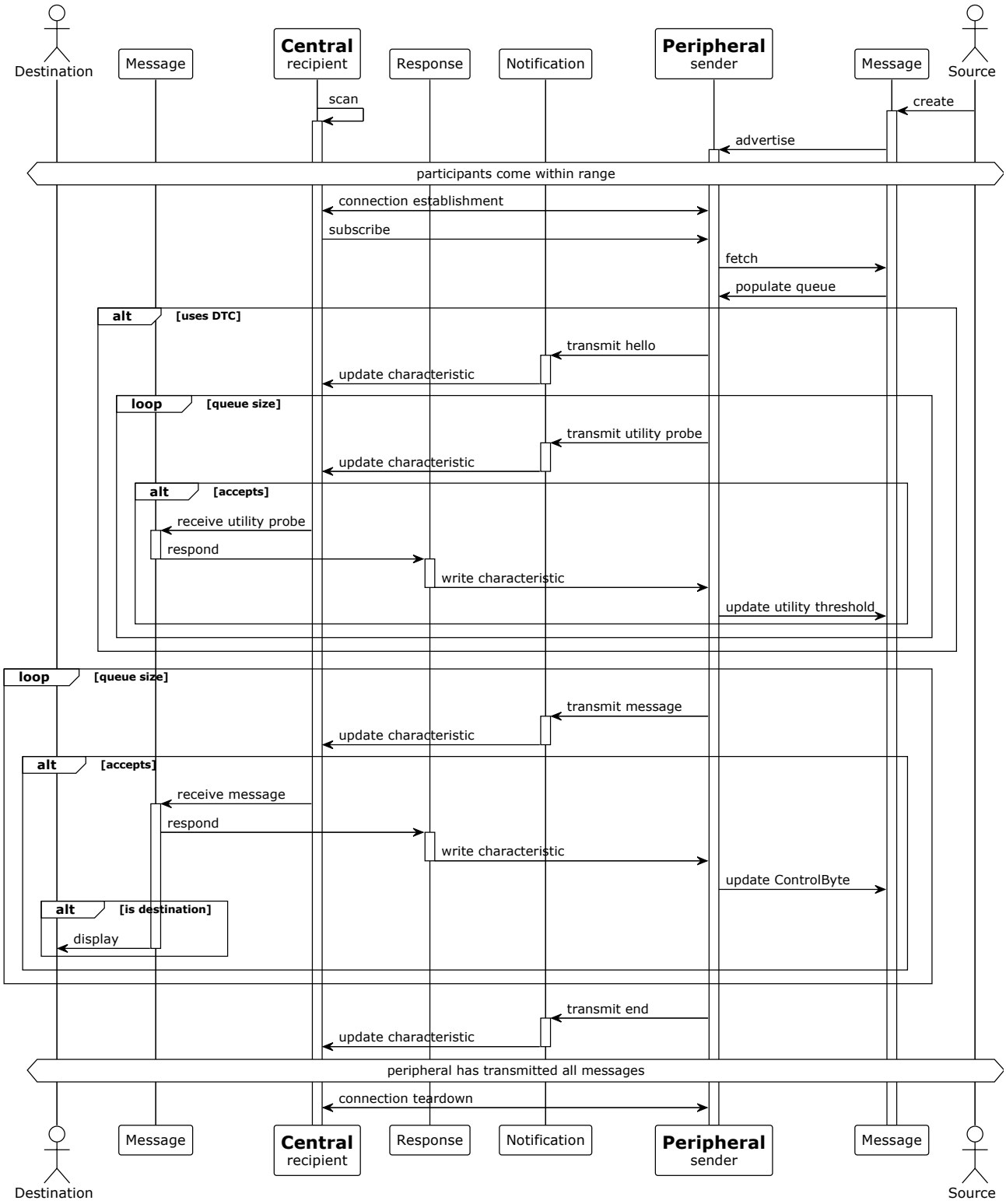


ABBILDUNG 3: SEQUENZDIAGRAMM DER BSAW UND DTC IMPLEMENTIERUNG

Im Folgenden werde ich den Gesamtenergieverbrauch der beiden Verfahren BSaW und DTC anhand einer Betrachtung nach Kontakten bzw. Nachrichten ermitteln.

Betrachtung nach Kontakten

Jeder Kontakt verbraucht Energie für den Verbindungsaufbau und -abbau auf Seiten des Centrals und des Peripherals. Diese Kosten fasse ich mit $E_{Verbindung}$ zusammen. Jedes Peripheral überträgt immer $m \geq 0$ Mitteilungen für alle Nachrichten, die noch nicht ihre TTL erreicht haben, an ein verbundenes Central und erhält von diesem $a \geq 0$ Antworten für akzeptierte Nachrichten zurück. Hinzu kommt eine Mitteilung, die das Ende markiert und im DTC Verfahren eine Hallo-Mitteilung. Mitteilungen und Antworten sind Übertragungen und kosten jeweils $E_{Uebertragung}$. Der Energieverbrauch eines Kontakts zweier Knoten im *Binary Spray and Wait* Routingverfahren ist daher wie folgt:

$$E_{Kontakt_{BSaW}} = E_{Verbindung} + \underbrace{(1 + m_{BSaW}) \cdot E_{Uebertragung}}_{Mitteilungen} + \underbrace{a_{BSaW} \cdot E_{Uebertragung}}_{Antworten}$$

$$= E_{Verbindung} + (1 + m_{BSaW} + a_{BSaW}) \cdot E_{Uebertragung}$$

Wobei m_{BSaW} die Anzahl gesendeter Mitteilungen und a_{BSaW} die Anzahl zurückgesendeter Antworten ist. Letzterer ist als einziger Parameter durch den Algorithmus bestimmt. Es gilt $0 \leq a_{BSaW} \leq m_{BSaW}$. Wird eine Mitteilung mit mehr als einer Kopie (angegeben in der Kennzahl des *ControlByte*) empfangen oder ist der Empfänger Adressat der Mitteilung, wird er eine Antwort senden.

Der Energieverbrauch der ersten und zweiten DTC Phase ähnelt dem von *Binary Spray and Wait*. Abgesehen von der Hallo-Mitteilung ist die Anzahl der übermittelten Mitteilungen (bzw. Nützlichkeitsanfragen) m_{DTC1} neben der TTL nun aber auch durch das Wiederentdeckungsintervall bestimmt. Es kann angenommen werden, dass $m_{DTC1} \leq m_{BSaW}$ ist. Nur solche Kontakte, deren Nützlichkeit besser ist als der Schwellenwert in der Nützlichkeitsanfrage, senden eine Antwort. Die Anzahl dieser Antworten ist $a_{DTC2} \geq 0$. Eignet sich der Kontakt dann als Relay für eine oder mehrere Nachrichten, erhält dieser eine entsprechende Anzahl an Mitteilungen m_{DTC3} und wird jede davon mit einer Antwort bestätigen. Es gilt $0 \leq m_{DTC3} \leq a_{DTC2} \leq m_{DTC1}$, wobei m_{DTC3} als sehr klein angenommen werden kann.

$$E_{Kontakt_{DTC}} = E_{Verbindung} + \underbrace{(2 + m_{DTC1}) \cdot E_{Uebertragung}}_{Erste\ Phase} + \underbrace{a_{DTC2} \cdot E_{Uebertragung}}_{Zweite\ Phase} + \underbrace{m_{DTC3} \cdot 2 \cdot E_{Uebertragung}}_{Dritte\ Phase}$$

$$= E_{Verbindung} + (2 + m_{DTC1} + a_{DTC2} + 2 \cdot m_{DTC3}) \cdot E_{Uebertragung}$$

Für den Vergleich der beiden Routingverfahren eignet sich die Betrachtung der Kontakte allerdings nicht. Zur Vereinfachung vernachlässige ich die Hallo- und Ende-Mitteilungen, da sie im Vergleich zu m_{BSaW} bzw. m_{DTC1} unbedeutend sind.

Auf den ersten Blick scheint DTC mehr Übertragungen zu benötigen. Allerdings ist $m_{DTC1} \leq m_{BSaW}$ und vor allem m_{DTC3} sehr klein. In welchem Verhältnis a_{BSaW} und a_{DTC2} zueinander stehen ist schwer einzuschätzen, beide sind jedoch kleiner gleich m_{BSaW}. Wenn $m_{DTC3} = 0$ gelten für beide Algorithmen die folgenden Obergrenzen des Gesamtenergieverbrauchs bei K Kontakten:

$$E_{BSaW} \leq K \cdot (E_{Verbindung} + 2 \cdot m_{BSaW} \cdot E_{Uebertragung})$$

$$E_{DTC} \leq K \cdot (E_{Verbindung} + 2 \cdot m_{BSaW} \cdot E_{Uebertragung})$$

Betrachtung nach Nachrichten

Sinnvoller als die Frage nach Übertragungen pro Kontakt ist die Erörterung, wie viele davon pro Nachricht durch das jeweilige Verfahren verursacht werden. Zur Vereinfachung vernachlässige ich im Folgenden $E_{Verbindung}$ sowie weiterhin die Kosten für die Hallo- und Ende-Mitteilungen, da diese nur einmal pro Kontakt entstehen und sich diese Kosten daher im Normalfall unter den vielen verschiedenen Nachrichten aufteilen, die pro Kontakt ausgetauscht werden.

Im Falle von *Binary Spray and Wait* werden Mitteilungen nie mehr als $c + 1$ Antworten erhalten, wobei $c > 0$ die für alle Nachrichten definierte initiale Kopienanzahl ist. Eine zusätzliche Antwort ist dann noch durch den adressierten Empfänger in der Wait-Phase möglich. Zur Vereinfachung vernachlässige ich diese jedoch und gehe von c Antworten aus. $k_{BSaW} > 0$ ist die Anzahl der Kontakte, die von Knoten mit einer Kopie der Nachricht innerhalb der TTL hergestellt werden.

$$E_{BSaW} \leq N \cdot (\underbrace{k_{BSaW} \cdot E_{Uebertragung}}_{Mitteilungen} + \underbrace{(c+1) \cdot E_{Uebertragung}}_{Antworten}) = N \cdot (k_{BSaW} + c) \cdot E_{Uebertragung}$$

Im Falle von DTC könnten prinzipiell alle Knoten, die in der ersten Phase eine Mitteilung (bzw. Nützlichkeitsanfrage) erhalten haben, eine (Nützlichkeits-)Antwort zurücksenden. Somit können in diesen Phasen bis zu $2 \cdot k_{DTC}$ Übertragungen stattfinden, wobei $k_{DTC} > 0$ die Anzahl der Kontakte beschreibt, die von Relayknoten der Nachricht vor Ablauf der TTL hergestellt werden. Für diese Knoten folgen auch noch die beiden Übertragungen (Mitteilung und Antwort) der dritten Phase. Dies Anzahl der Relayknoten bzw. Hops ist mit $h \geq 0$ ausgedrückt. Die Obergrenzen des Gesamtenergieverbrauchs zum Vermitteln von N Nachrichten lassen sich für beide Routingverfahren daher wie folgt definieren:

$$E_{DTC} \leq N \cdot (\underbrace{k_{DTC} \cdot 2 \cdot E_{Uebertragung}}_{Erste\ und\ zweite\ Phase} + \underbrace{h \cdot 2 \cdot E_{Uebertragung}}_{Dritte\ Phase}) = N \cdot 2 \cdot (k_{DTC} + h) \cdot E_{Uebertragung}$$

Angenommen alle Knoten, die die Nachricht übertragen, haben ungefähr gleich viele Kontakte k, ist $k_{BSaW} \approx c \cdot k$, da bei einer typischen Knotendichte eine Nachricht innerhalb ihrer TTL immer c-mal vervielfältigt wird. Außerdem kann dann $k_{DTC} \approx h \cdot k$ angenommen werden. Die Ungleichungen lassen sich somit wie folgt vereinfachen:

$$E'_{BSaW} \leq N \cdot (c \cdot k + c) \cdot E_{Uebertragung} = N \cdot c \cdot (k + 1) \cdot E_{Uebertragung}$$

$$E'_{DTC} \leq N \cdot 2 \cdot (h \cdot k + h) \cdot E_{Uebertragung} = N \cdot 2 \cdot h \cdot (k + 1) \cdot E_{Uebertragung}$$

Bei hoher Knotenmobilität und relativ großen Distanzen zwischen den Kommunikationspartner*innen, wovon in den in dieser Arbeit beschriebenen Anwendungsfällen auszugehen ist, kann von $c < h/2$ ausgegangen werden. Ob Nachrichten, die im *Disconnected Transitive Communication* Verfahren vermittelt werden, tatsächlich mehr als doppelt so viele Hops zurücklegen, wie *Binary Spray and Wait* bei vergleichbarer Zuverlässigkeit an initialen Kopien benötigt, müsste in der Praxis (mit einer recht großen Knotenanzahl) überprüft werden. Der geplante Test mit fünf Knoten hätte daher vermutlich gar keine messbaren Unterschiede zwischen den beiden Routingverfahren ergeben.

Abschließend lässt sich trotzdem sagen, dass *Disconnected Transitive Communication* seine theoretischen weiterleitungsbasierten Effizienz-Vorteile aufgrund des ineffizienten Datenaustauschs auf der Netzzugangsschicht nicht ausspielen kann. Der weniger komplexe, vervielfältigungsbasierte *Binary Spray and Wait* Algorithmus ist wegen einer vergleichbaren Energieeffizienz und einer einfacheren Implementierung für das Routing in einem auf BLE Übertragungen basierten mobilen Ad-hoc-Netzwerk vorzuziehen.

Fazit

Zusammenfassung

Wenn die Mobilfunkinfrastruktur auf einer Großveranstaltung überlastet ist, können Menschen mit ihren Smartphones über ein mobiles Ad-hoc-Netzwerk kommunizieren. Für einen Nachrichtenaustausch über mehrere Knoten hinweg eignen sich Peer-to-Peer-Routingalgorithmen, die verzögerungstolerant sind. Mit *Bluetooth Low Energy* können Daten auch im Hintergrund drahtlos übertragen werden. Bestimme Eigenschaften dieser Technologie erschweren allerdings den Einsatz in einem mobilen Ad-hoc-Netzwerk, vor allem hinsichtlich effizienter Routingverfahren.

Anhand meiner iOS App **bleep** habe ich die grundsätzliche Umsetzbarkeit eines iPhone-ad-hoc-Netzwerks auf Großveranstaltungen demonstriert und die zahlreichen Hindernisse bei der Implementierung des Routings beschrieben. Die Anwendung war letztlich nicht voll funktionsfähig und konnte nicht dazu genutzt werden, die Routingverfahren *Binary Spray and Wait* und *Disconnected Transitive Communication* unter realen Bedingungen zu evaluieren. Die Hypothese, dass für den Anwendungsfall dieser Arbeit ein weiterleitungsbasiertes Verfahren (DTC) effizienter als ein vervielfältigungsbasierter Algorithmus (BSaW) ist, habe ich dennoch anhand theoretischer Vergleiche widerlegen können.

Erkenntnisse

Bluetooth Mesh zeigt, dass Peer-to-Peer-Routing mit BLE möglich und relevant ist. Die verwendeten Routingalgorithmen sind nicht besonders effizient, aber vermutlich so gut, wie es das Übertragungsverfahren erlaubt. Für einen Einsatz in einem verzögerungstoleranten Netzwerk müsste das Protokoll zwar angepasst werden, dennoch schätze ich es jetzt als die beste Grundlage für eine App wie **bleep** ein. Nicht zuletzt wegen all der Sicherheitsmerkmale, die bereits integriert und essentiell für eine Kommunikationsanwendung sind.

Ich habe erfahren, wie komplex die Entwicklung verteilter Systeme ist. Sie hat doppelt so viel Zeit in Anspruch genommen, als ich ursprünglich geplant hatte. Vermutlich wäre eine Anpassung einer bestehenden *Bluetooth Mesh* Implementierung an den Anwendungsfall effektiver gewesen statt der Suche nach geeigneter Routingalgorithmen und deren eigener Implementierung — wenngleich beides sehr interessant und aufschlussreich war.

Abkürzungsverzeichnis

ABR	Associativity-Based Routing
ANCS	Apple Notification Center Service
AODV	Ad-Hoc On-Demand Distance Vector Routing
ARA	Ant-Colony-Based Routing Algorithm
ASCII	American Standard Code for Information Interchange
B.A.T.M.A.N.	Better Approach to Mobile Ad-Hoc Networking
BLE	Bluetooth Low Energy
BR	Basic Rate
BSaW	Binary Spray and Wait
CGR	Contact Graph Routing
COVID-19	Coronavirus Disease 2019
CRC	Cyclic Redundancy Check
CSV	Comma-separated values
DREAM	Distance Routing Effect Algorithm for Mobility
DTC	Disconnected Transitive Communication
DTH	Distributed Hash Table
DSDV	Destination-Sequenced Distance-Vector Routing
DSR	Dynamic Source Routing
DTLSR	Delay-Tolerant Link-State Routing
DTN	Delay-Tolerant Network
EDR	Enhanced Data Rate
EUI	Extended Unique Identifier
ICMN	Intermittently Connected Mobile Networks
ID	Identifikator
IDE	Integrated Development Environment
IEEE 802.11	[Normen für Wireless Local Area Networks (WLAN)]
IoT	Internet of Things
IPSP	Internet Protocol Support Profile
IPv6	Internet Protocol Version 6
IRK	Identity Resolving Key
ISM	Industrial, Scientific and Medical
LASeR	Location Aware Sensor Routing
LEACH	Low-Energy Adaptive Clustering Hierarchy
MAC	Media Access Control
MANET	Mobile Ad-Hoc Network
MITM	Man-in-the-Middle
MTU	Maximum Transmission Unit
NFC	Near Field Communication
OLSR	Optimized Link State Routing Protocol

OOB	Out-of-Band
PROPHET	Probabilistic Rout. Prot. using History of Encounters & Transitivity
RDI	Rediscovery Interval
RSSI	Received Signal Strength Indication
SDK	Software Development Kit
SHA-256	Secure Hash Algorithm [mit Ausgabe von acht 32 Bit Hashwerten]
SIG	Special Interest Group
SMS	Short Message Service
SPAN	Smartphone Ad-Hoc Network
TTL	Time-to-Live
UML	Unified Modeling Language
UTC	Coordinated Universal Time
UTF-8	8-Bit Unicode Transformation Format
UUID	Universally Unique Identifier
UWB	Ultra Wide Band
VoIP	Voice over Internet Protocol
ZBR	Zone Based Routing

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ABB15] Araniti, G., Bezirgiannidis, N., Birrane, E., Bisio, I., Burleigh, S., Caini, C., Feldmann, M., Marchese, M., Segui, J. & Suzuki, K. (2015). Contact Graph Routing in DTN Space Networks: Overview, Enhancements and Performance. In: Hu, R. Q. (Hrsgb.), IEEE Communications Magazine, Volume 53, Issue 3 (S. 38-46). Institute of Electrical and Electronics Engineers, New York. https://doi.org/10.1109/MCOM.2015.7060480

[AEP22] Albrecht, M. R., Eikenberg, R. & Paterson, K. G. (2022). Breaking Bridgefy, again: Adopting libsignal is not enough. In: Proceedings of 31st USENIX Security Symposium (S. 269-286). Usenix Association, Boston. https://www.usenix.org/conference/usenixsecurity22/presentation/albrecht

[AhH11] Ahmed, U. & Hussain, F. B. (2011). Energy Efficient Routing Protocol for Zone Based Mobile Sensor Networks. In: Proceedings of 7th International Wireless Communications and Mobile Computing Conference IWCMC 2011 (S. 1081-1086). Institute of Electrical and Electronics Engineers, Istanbul. https://doi.org/10.1109/IWCMC.2011.5982691

[Aic07] Aichele, C. (2007). Mesh: Drahtlose Ad-hoc-Netze. Open Source Press, München. https://d-nb.info/984120211

[Alt21] Alt, N. (2021). Tief in Briar eintauchen beim XMPP-Treffen Berlin. https://nico.dorfbrunnen.eu/de/posts/2021/diving-at-xmpp/

[App13] Apple (2013). Core Bluetooth Programming Guide. https://developer.apple.com/library/archive/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/AboutCoreBluetooth/Introduction.html

[App14a] Apple (2014). Apple Notification Center Service (ANCS) Specification. https://developer.apple.com/library/archive/documentation/CoreBluetooth/Reference/AppleNotificationCenterServiceSpecification/Introduction/Introduction.html

[App14b] Apple (2014). iPhone 6 – Technische Daten. https://support.apple.com/de-de/111954

[App15] Apple (2015). FireChat by Open Garden – iTunes Preview. Archiviert von The Wayback Machine, Internet Archive, San Francisco. https://web.archive.org/web/20150216190323/https://itunes.apple.com/en/app/firechat/id719829352

[App21] Apple (2021). Bluetooth-Sicherheit. https://support.apple.com/de-de/guide/security/sec82597d97e/web

[App24] Apple (2024). Core Bluetooth Framework Reference. https://developer.apple.com/documentation/corebluetooth

[ApG20] Apple, Google (2020). Exposure Notification – Bluetooth Specification, Version 1.2. https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ExposureNotification-BluetoothSpecificationv1.2.pdf

[AWD03] Abolhasan, M., Wysocki, T. & Dutkiewicz, E. (2004). A review of routing protocols for mobile ad hoc networks. In: Kanhere, S. (Hrsgb.), Ad Hoc Networks, Volume 2, Issue 1 (S. 1-22). Elsevier, Amsterdam. https://doi.org/10.1016/S1570-8705(03)00043-X

[Ber23] Berty Technologies (2023). Wesh protocol. https://berty.tech/docs/protocol

[BGJ06] Burgess, J., Gallagher, B., Jensen D. & Levine, B. N. (2006). MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks. In: Proceedings of 25th IEEE International Conference on Computer Communications INFOCOM 2006 (S. 1-11). Institute of Electrical and Electronics Engineers, Barcelona. https://doi.org/10.1109/INFOCOM.2006.228

[Blu23] Bluetooth SIG (2023). Mesh Protocol Specification, Version 1.1. https://www.bluetooth.com/de/specifications/specs/mesh-protocol/

[Blu24a] Bluetooth SIG (2024). Bluetooth Technology Overview. https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

[Blu24b] Bluetooth SIG (2024). 2024 Bluetooth Market Update. https://www.bluetooth.com/2024-market-update/

[BTA11] Boukerche, A., Turgut, B., Aydin, N., Ahmad, M. Z., Bölöni, L. & Turgut, D. (2011). Routing protocols in ad hoc networks: A survey. In: Melodia, T. & Iera, A. (Hrsgb.), Computer Networks, Volume 55, Issue 13 (S. 3032-3080). Elsevier, Amsterdam. https://doi.org/10.1016/j.comnet.2011.05.010

[ChM01] Chen, X. & Murphy A. L. (2001). Enabling Disconnected Transitive Communication in Mobile Ad Hoc Networks. In: Proceedings of Workshop on Principles of Mobile Computing POMC 2001. Association for Computing Machinery, Newport. https://courses.csail.mit.edu/6.885/fall08/papers/ChenMurphy-pomc01.pdf

[Des19] De Silva, M. (2019). Hong Kong protestors are once again using mesh networks to preempt an internet shutdown. In: G/O Media (Hrsgb.), Quartz. G/O Media, New York. https://qz.com/1701045/hong-kong-protestors-use-bridgefy-to-preempt-internet-shutdown

[Ern14] Ernst. N. (2014). Bluetooth 4.2 wird schneller und sicherer. In: Golem Media (Hrsgb.), Golem.de. Golem Media, Berlin. https://www.golem.de/news/neue-spezifikation-bluetooth-4-2-wird-schneller-und-sicherer-1412-110945.html

[Gar17] Gardner-Stephen, P. (2017). The Serval Project Wiki – Serval Chat. Archiviert von The Wayback Machine, Internet Archive, San Francisco. https://web.archive.org/web/20230129133334/https://developer.servalproject.org/dokuwiki/doku.php?id=content:servalchat:main_page

[GCL13] Gardner-Stephen, P., Challans, R., Lakeman, J., Bettison, A., Gardner-Stephen D. & Lloyd, M. (2013). The Serval Mesh: A Platform for Resilient Communications in Disaster & Crisis. In: Proceedings of IEEE Global Humanitarian Technology Conference GHTC 2013 (S. 162-166). Institute of Electrical and Electronics Engineers, San Jose. https://doi.org/10.1109/GHTC.2013.6713674

[HaA16] Hayes, T. & Ali, F. H. (2016). Location Aware Sensor Routing (LASeR) Protocol for Mobile Wireless Sensor Networks. In: IET Wireless Sensor Systems, Volume 6, Issue 2 (S. 49-57). The Institution of Engineering and Technology, Stevenage. https://doi.org/10.1049/iet-wss.2015.0027

[HCY11] Hui P., Crowcroft, J. & Yoneki, E. (2011). BUBBLE Rap: Social-Based Forwarding in Delay-Tolerant Networks. In: Cui, R. (Hrsgb.), IEEE Transactions on Mobile Computing, Volume 10, Issue 11 (S. 1576-1589). Institute of Electrical and Electronics Engineers, New York. https://doi.org/10.1109/TMC.2010.246

[JLW05] Jones, E. P. C., Li, L. & Ward, P. A. S. (2005). Practical Routing in Delay-Tolerant Networks. In: Proceedings of ACM SIGCOMM Workshop on Delay-tolerant networking WDTN 2005 (S. 237–243). Association for Computing Machinery, New York. https://doi.org/10.1145/1080139.1080141

[JVA21] Jouans, L., Viana, A. C., Achir, N. & Fladenmuller, A. (2021). Associating the Randomized Bluetooth MAC Addresses of a Device. In: Proceedings of IEEE 18th Annual Consumer Communications & Networking Conference CCNC 2021 (S. 1-6). Institute of Electrical and Electronics Engineers, Las Vegas. https://doi.org/10.1109/CCNC49032.2021.9369628

[KiC06] Kim, D.-S. & Chung, Y.-J. (2006). Self-Organization Routing Protocol Supporting Mobile Nodes for Wireless Sensor Network. In: First International Multi-Symposiums on Computer and Computational Sciences IMSCCS 2006 (S. 622-626). Institute of Electrical and Electronics Engineers, Hangzhou. https://doi.org/10.1109/IMSCCS.2006.265

[MaM02] Maymounkov, P. & Mazières, D. (2002). Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: Druschel, P., Kaashoek, F., Rowstron, A. (Hrsgb.), Peer-to-Peer Systems IPTPS 2002 Lecture Notes in Computer Science, Volume 2429 (S. 53-65). Springer, Berlin/Heidelberg. https://doi.org/10.1007/3-540-45748-8_5

[Nar15] Narayanan, C. (2015). Open Garden's FireChat app can be a mobile megaphone. In: Living Media India (Hrsgb.), Business Today, Ausgabe 15.2.2015. Living Media India, New Delhi. https://www.businesstoday.in/magazine/features/story/could-off-the-grid-social-network-capitalise-143810-2015-01-24

[Oev22] Øvrebekk, T. (2022). Bluetooth 5 Advertising Extensions. In: Nordic Semiconductor (Hrsgb.), Get Connected Blog. Nordic Semiconductor, Trondheim. https://blog.nordicsemi.com/getconnected/bluetooth-5-advertising-extensions

[Ope19] Open Garden (2019). FireChat. Archiviert von The Wayback Machine, Internet Archive, San Francisco. https://web.archive.org/web/20191223004858/https://www.opengarden.com/firechat/

[Ope24] Open-Mesh (2024). B.A.T.M.A.N. Experience. https://www.open-mesh.org/projects/open-mesh/wiki/Experience

[Rea14] Real (2014). Freedomlayer Research. https://www.freedomlayer.org/research/

[Shu22] Shuttleworth Foundation (2022). Our Thinking – All good things. https://www.shuttleworthfoundation.org/thinking/2022/07/19/thinking-closing/

[SPG20] Hernández-Solana, Á., Pérez-Díaz-De-Cerio, D., García-Lozano, M., Bardají, A. V. & Valenzuela, J.-L. (2020). Bluetooth Mesh Analysis, Issues and Challenges. In: Abbott, D. (Hrsgb.), IEEE Access, Volume 8 (S. 53784-53800). Institute of Electrical and Electronics Engineers, New York. https://doi.org/10.1109/ACCESS.2020.2980795

[SPR05] Spyropoulos, T., Psounis, K. & Raghavendra, C.S. (2005). Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. In: Proceedings of ACM SIGCOMM Workshop on Delay-tolerant networking WDTN 2005 (S. 252–259). Association for Computing Machinery, New York. https://doi.org/10.1145/1080139.1080143

[Put14] Putz, U. (2014). Joshua, 17, lehrt China das Fürchten. In: Augstein, R. (Hrsgb.), DER SPIEGEL (online). DER SPIEGEL, Hamburg. https://www.spiegel.de/politik/ausland/hongkong-studenten anfuehrer-koennte-peking-gefaehrlich-werden-a-994550.html

[VaB00] Vahdat, A. & Becker, D. (2000). Epidemic Routing for Partially-Connected Ad Hoc Networks. In: Duke Technical Report CS-2000-06. Duke University, Durham. https://cseweb.ucsd.edu//~vahdat/papers/epidemic.pdf

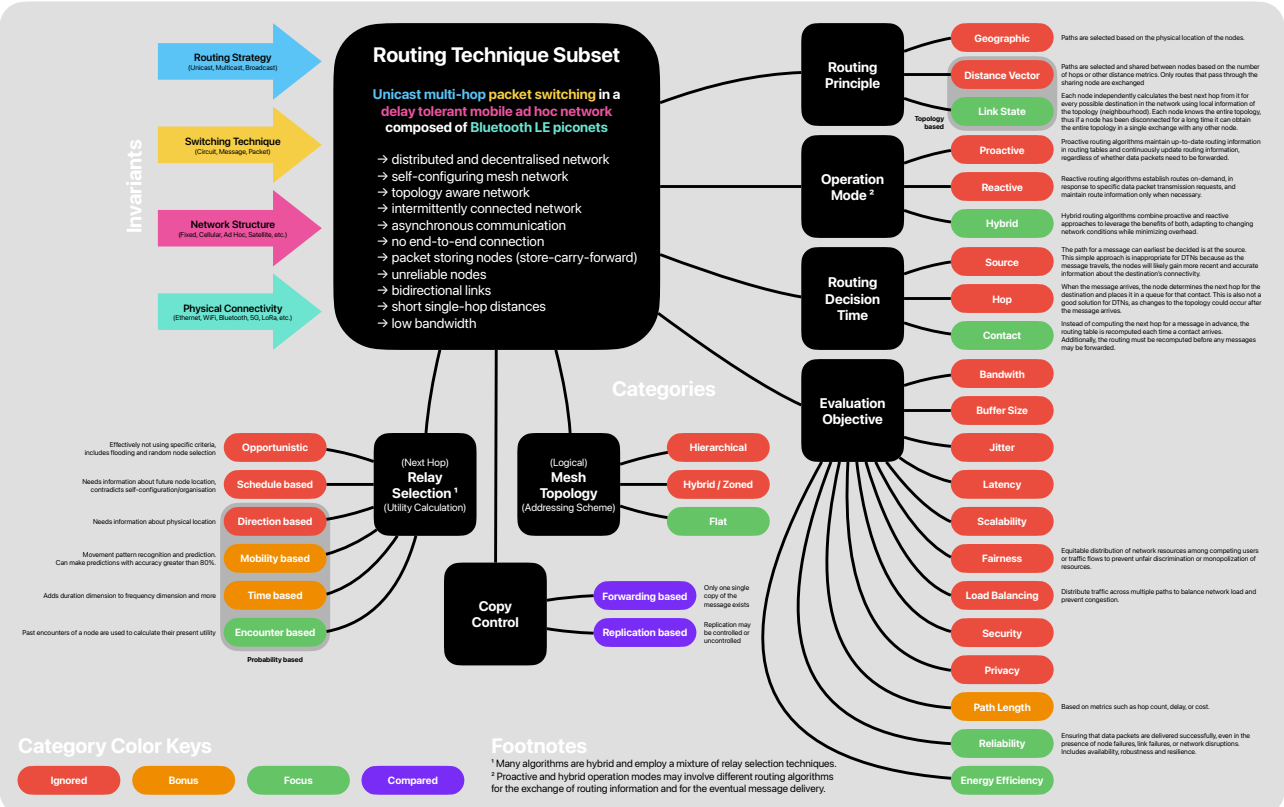
[VSK21] Verma, A., Savita & Kumar, S. (2021). Routing Protocols in Delay Tolerant Networks: Comparative and Empirical Analysis. In: Sezgin, A. & Prasad R. (Hrsgb.), Wireless Personal Communications, Volume 118, Issue 1 (S. 551–574). Springer, Berlin/Heidelberg. https://doi.org/10.1007/s11277-020-08032-4

[War12] Warren, J. (2012). Bitmessage: A Peer-to-Peer Message Authentication and Delivery System. https://bitmessage.org/bitmessage.pdf

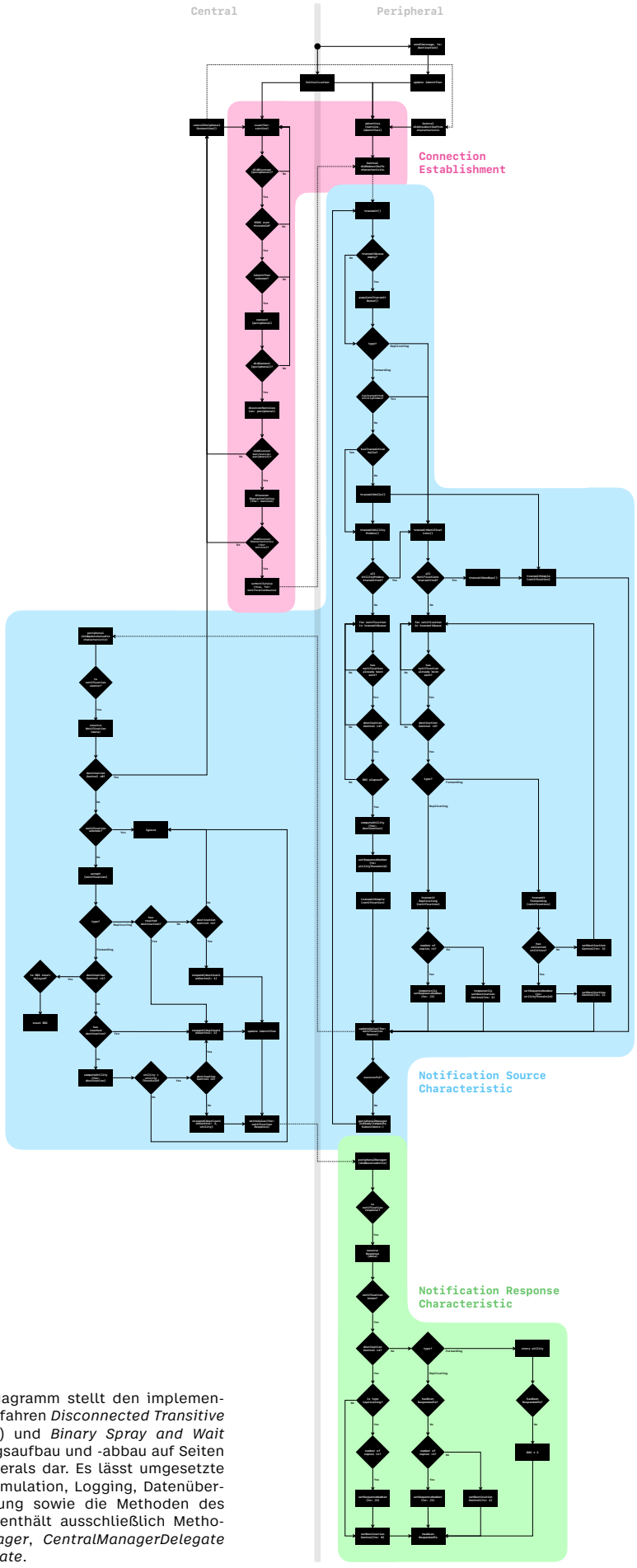
[Yu_14] Yu, A. (2014). How One App Might Be A Step Toward Internet Everywhere. In: National Public Radio (Hrsgb.), All Tech Considered. National Public Radio, Washington. https://www.npr.org/sections/alltechconsidered/2014/04/07/298925565/how-one-app-might-be-a-step-toward-internet-everywhere

Anhang

ANHANG 1: GRAFISCHE KATEGORISIERUNG DER ROUTINGVERFAHREN



ANHANG 2: AKTIVITÄTSDIAGRAMM DER VERMITTLUNGSLOGIK



Anmerkung: Das Aktivitätsdiagramm stellt den implementierten Ablauf der Routingverfahren *Disconnected Transitive Communication* (Forwarding) und *Binary Spray and Wait* (Replicating) inkl. Verbindungsaufbau und -abbau auf Seiten des Centrals und des Peripherals dar. Es lässt umgesetzte Funktionen für Persistenz, Simulation, Logging, Datenüberprüfung und Fehlerbehandlung sowie die Methoden des *BluetoothManagers* aus. Es enthält ausschließlich Methoden der Klassen *BleepManager*, *CentralManagerDelegate* und *PeripheralManagerDelegate*.

ANHANG 3: KOMMENTIERTE ROHDATEN EINES TESTDURCHLAUFS

Log-Adresse	Log-Zeitstempel ↓	Typ	ID (Hash)	Empfangs-kontrolle	Kopien	Quell-Adresse	Versand-Zeitstempel	Ziel-Adresse	Empfangs-Zeitstempel	Hops
A	742232628,99	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	16	A	742232628,99	D		0
C	742232629,24	Mitteilung	shqplBPearvNGq0And/dz/kRVk	1	16	C	742232629,24	A		0
A	742232630,30	Mitteilung	shqplBPearvNGq0And/dz/kRVk	1	8	C	742232629,24	A	742232630,30	1
C	742232630,33	**Antwort**	shqplBPearvNGq0And/dz/kRVk	2					742232630,33	
A	742232630,36	Mitteilung	shqplBPearvNGq0And/dz/kRVk	1	8	C	742232629,24	A	742232630,36	1
C	742232630,36	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	8	A	742232628,99	D	742232630,36	1
A	742232630,39	**Antwort**	6PgBE4OcEKedPuO8mJXwvf5z	1					742232630,39	
A	742232630,42	Mitteilung	shqplBPearvNGq0And/dz/kRVk	1	8	C	742232629,24	A	742232630,42	1
C	742232630,42	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	8	A	742232628,99	D	742232630,42	1
A	742232630,48	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	4	A	742232628,99	D	742232630,48	2
C	742232630,51	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	8	A	742232628,99	D	742232630,51	1
A	742232630,54	Mitteilung	shqplBPearvNGq0And/dz/kRVk	1	8	C	742232629,24	A	742232630,54	1
A	742232630,60	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	4	A	742232628,99	D	742232630,60	2
C	742232630,60	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	8	A	742232628,99	D	742232630,60	1
A	742232630,66	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	4	A	742232628,99	D	742232630,66	2
C	742232630,66	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	8	A	742232628,99	D	742232630,66	1
C	742232630,72	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	8	A	742232628,99	D	742232630,72	1
A	742232630,72	Mitteilung	shqplBPearvNGq0And/dz/kRVk	1	8	C	742232629,24	A	742232630,72	1
C	742232630,78	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	8	A	742232628,99	D	742232630,78	1
A	742232630,78	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	4	A	742232628,99	D	742232630,78	2
B	742232630,83	Mitteilung	shqplBPearvNGq0And/dz/kRVk	1	8	C	742232629,24	A	742232630,83	1
A	742232630,84	Mitteilung	shqplBPearvNGq0And/dz/kRVk	1	8	C	742232629,24	A	742232630,84	1
C	742232630,84	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	8	A	742232628,99	D	742232630,84	1
B	742232630,88	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	8	A	742232628,99	D	742232630,88	1
A	742232630,90	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	4	A	742232628,99	D	742232630,90	2
B	742232630,90	Mitteilung	6PgBE4OcEKedPuO8mJXwvf5z	1	4	A	742232628,99	D	742232630,90	2
A	742232630,92	**Antwort**	6PgBE4OcEKedPuO8mJXwvf5z	1					742232630,92	
C	742232634,97	Mitteilung	7i3L3ZU/mQJgi0xYM9zQceYvw	1	16	C	742232634,97	D		0
A	742232635,09	Mitteilung	EkllYku1lSnziliswr9Y7VGrs2O9	1	16	A	742232635,09	B		0
A	742232639,51	Mitteilung	s5yfQ4my3cdY1zmhpPW5X+Hc	1	16	A	742232639,51	C		0
C	742232642,38	Mitteilung	+zrGEtKd0wzFRnXXobNg13gw	1	16	C	742232642,38	B		0
C	742232645,25	Mitteilung	OAbstmlQ8Uaf9upact9zRlJwF	1	16	C	742232645,25	D		0
A	742232646,45	Mitteilung	OU6ClcsB53AfzeE6BAZ23r3+ht	1	16	A	742232646,45	B		0
A	742232651,12	Mitteilung	WuVbClc8jKm+jNgdiCBlSGEJt	1	16	A	742232651,12	D		0
C	742232652,76	Mitteilung	aPrJAQ0JcWI3RkIeFIVAgMgMR	1	16	C	742232652,76	A		0
C	742232656,54	Mitteilung	BtCnMYbhlzcUuLjAZNsZI912iW	1	16	C	742232656,54	D		0
A	742232657,19	Mitteilung	YuZXzTbSmNpk+8s6JhaZ3eM+	1	16	A	742232657,19	C		0
A	742232661,53	Mitteilung	W8bXCK8pEIKWG3/hVmU63zX	1	16	A	742232661,53	D		0
C	742232662,14	Mitteilung	d78+g7NVlhbRYqwAoZzOcBCW	1	16	C	742232662,14	A		0
A	742232668,96	Mitteilung	DdB4/LtFp6Tcv1nC25Q6bJwIh	1	16	A	742232668,96	D		0
C	742232669,32	Mitteilung	bwE/H6yD2sFOHNPPxisAZ8PF/	1	16	C	742232669,32	A		0
C	742232673,67	Mitteilung	mfeu7pOXHlHqbZGvRxZ1dMRr	1	16	C	742232673,67	D		0
A	742232674,01	Mitteilung	vw0XQlXbltoKBDdZzJOxY7emc	1	16	A	742232674,01	B		0
A	742232680,93	Mitteilung	aDNo6DlYkfRLoK/BXWgzDGTQ	1	16	A	742232680,93	D		0
C	742232681,07	Mitteilung	xNiy2W2+okGgwkMOxG5SCGv	1	16	C	742232681,07	A		0
A	742232685,80	Mitteilung	TgGxBnjHpqSKwFJr3UF1zxqsp	1	16	A	742232685,80	D		0
C	742232687,48	Mitteilung	SuDu8NRgORZthVf3NDtHr9zY	1	16	C	742232687,48	A		0
A	742232689,05	Mitteilung	cq7Wp7VteHQvgJRJBXk9Gq3D	1	16	A	742232689,05	D		0
A	742232692,36	Mitteilung	dBJVttvsZG30pqpF8LhwnZa/W	1	16	A	742232692,36	D		0
C	742232694,42	Mitteilung	uOXrODCZAgUx9tjLp+og1m4Vl	1	16	C	742232694,42	B		0
A	742232699,58	Mitteilung	mskHCx8wIJtRo/KZoHsJbfZmA	1	16	A	742232699,58	B		0
C	742232700,05	Mitteilung	0QEGdtCnXoKRp6QsjZWuZ6F2	1	16	C	742232700,05	A		0
A	742232704,54	Mitteilung	Rsm0aC9Y1gvy5A0RAULCBiiDT	1	16	A	742232704,54	B		0
C	742232706,31	Mitteilung	9vZxvEZzxY1XNusvX37MUX1E2	1	16	C	742232706,31	A		0
A	742232708,70	Mitteilung	iwzhMtYWBQuzJyEQeMGq+QF	1	16	A	742232708,70	D		0
C	742232712,13	Mitteilung	/ly1UUxO499Wuxnl/k4fohrP5n	1	16	C	742232712,13	D		0
A	742232715,85	Mitteilung	e7TQrCaa0qkylIZnlgRsDOozi4	1	16	A	742232715,85	C		0
C	742232717,61	Mitteilung	/fVlxuxjGHsOCtyXqieqWaIA1VJ	1	16	C	742232717,61	A		0
A	742232720,03	Mitteilung	lEu8LMVvDtCnulas57zn3oS+Q	1	16	A	742232720,03	D		0
C	742232722,12	Mitteilung	L8fc+G0RIm7NCy4JlQkyTVVsX	1	16	C	742232722,12	D		0

Legende: Nachricht I | Nachricht II | Mitteilung von Ziel empfangen | Mitteilung von Relay empfangen | 1 Hop zurückgelegt | 2 Hops zurückgelegt
Informationen: 93 Sekunden, 3 Knoten (4 adressierte), 60 Log-Einträge, 35 individuelle Nachrichten (davon zwei 22 Mal erfolgreich übermittelt)
Simulationsparameter: Min. RSSI -72 dBM, Max. TTL 2 Minuten, Nachrichtenversand durch A und C alle 5 Sekunden (± 50%) und initial 16 Kopien