Mobile Fingerprinting 3.0

Studien-/ Bachelorarbeit

Studiengang Informatik

OST – Ostschweizer Fachhochschule

Campus Rapperswil-Jona

Frühjahrssemester 2021

Autoren:	Lukas Volk
	Severin Marti
Betreuer:	Prof. Beat Stettler
Projektpartner:	onway ag, 8004 Zürich
Experte:	Martin Willi
Gegenleser:	Prof. Stefan Richter

Abstract

Reisedaten von Passagieren dienen den Transportunternehmen zu identifizieren, welche Strecken viel genutzt werden und helfen, die Fahrpläne entsprechend zu optimieren. Momentan werden diese Reisedaten durch manuelle Fahrgastbefragungen ermittelt. Mobiltelefone und Peripheriegeräte wie Kopfhörer oder Smartwatches sind heute weit verbreitet. Diese senden fortlaufend Funksignale aus, die potentiell für eine Zählung der Passagiere genutzt werden könnten. In Vorarbeiten hat sich gezeigt, dass ein Tracking mittels WiFi mittlerweile unmöglich ist. In dieser Arbeit wurde erforscht, ob und wie ein Tracking passiv per Bluetooth möglich ist.

Im faradayschen Käfig des ICOM und in öffentlichen Bereichen wurden die Bluetooth Signale von vier unterschiedlichen iPhone Modellen, drei Android Geräten sowie verschiedenen Peripheriegeräten untersucht und daraus eine Software Suite zum Erfassen, Speichern und Visualiseren der Daten entwickelt. Für das Sniffen der Bluetooth Signale wurde der Ubertooth One verwendet.

Audiodaten werden mit dem BR/EDR Protokoll versandt. Dort werden ausreichende Teile der nicht randomisierten Bluetooth Addressen direkt herausgegeben, was ein Tracking dieser Geräte trivial macht. Bluetooth Low Energy Geräte randomisieren in der Regel ihre Addressen und ändern sie circa alle 15 Minuten. Eine Korrellationslogik wurde implementiert, welche anhand von zeitlichen und räumlichen Daten sowie den in Advertisements enthaltenen 16-Bit Service UUID und Company Ids den Adresswechsel in gesendeten Advertisements detektiert. Bei neueren iPhones ist so ein Tracking möglich, die untersuchten Android Geräte sind aber nur detektierbar, wenn eine entsprechende App wie die Swiss Covid App installiert ist. Die Korrelation stösst an Grenzen, wenn ein Adresswechsel bei einer grossen Anzahl von präsenten Geräten erfolgt, zum Beispiel bei Einfahrt in einen Bahnhof.

Lay Summary

Ausgangslage

Reisedaten von Passagieren dienen den Transportunternehmen dazu zu identifizieren, welche Strecken viel benutzt werden und helfen, die Fahrpläne entsprechend anzupassen. Momentan werden diese Daten durch manuelle Fahrgastbefragungen ermittelt.

Mobiltelefone und Peripheriegeräte, wie zum Beispiel Kopfhörer oder Smartwatches, sind heute weit verbreitet. Diese senden fortlaufend Signale aus. Für die Automatisierung der Datenerhebung sollen diese Signale verwendet werden. Um WiFi anzubieten, befinden sich bereits in vielen Bussen und Zügen Antennen, welche um Bluetooth-Funktionalität erweitert werden können.

In Vorarbeiten hat sich gezeigt, dass ein Tracking mittels WiFi mittlerweile unmöglich ist. Dies, da in neueren Versionen Privatsphärenschutzmethoden vorhanden sind. Als weitere weit verbreitete Wireless-Anwendung bot sich Bluetooth als nächstes Fokusgebiet an.

Bei Bluetooth muss prinzipiell zwischen BD/EDR, auch Classic genannt, und Low Energy unterschieden werden. Diese sind unterschiedliche Protokolle und untereinander nicht kompatibel. Moderne Mobilgeräte unterstützen in der Regel beide.

Vorgehen/Technologien

Es wurde, nach einer Evaluation von Kosten und Nutzen diverser Lösungen, die Verwendung des Ubertooth One als Bluetooth Empfänger beschlossen. Für diese USB-Antenne und die zugehörige Software ist der Programmcode frei verfügbar (Open Source).

Damit wurden Daten von Mobil- und Peripheriegeräten verschiedener Hersteller im faradayschen Käfig des ICOM in Wireshark erfasst und analysiert. Dabei wurden übliche Szenarien, die im ÖV auftreten können, im abgeschirmten Raum nachgespielt.

So konnten verschiedene Ansätze für ein komplett passives, anonymes Tracking erarbeitet werden. In Folge wurden diese Ideen näher betrachtet, darunter ein bestehender Ansatz in Classic und zwei experimentelle Ideen in Low Energy.

Um diese Untersuchungen effektiv durchzuführen und zu verifizieren, wurde zusätzliches Tooling entwickelt, darunter Erweiterungen an der Ubertooth Firmware, eine Client Applikation, eine Server Applikation, und Korrelationsalgorithmen.

Ergebnisse

Audiodaten werden mit dem BR/EDR Protokoll versandt. Dort werden für Fingerprinting ausreichende Teile der nicht randomisierten Bluetooth Adressen, welche Geräte identifizieren, direkt mitgeschickt. Ein Tracking dieser Geräte ist bei aktiver Datenverbindung deshalb unproblematisch.

Für die Kommunikation mit nicht verbundenen Geräten werden bei Bluetooth Low Energy so genannte Advertisements geschickt. Darin ist die Adresse des Absenders enthalten. Geräte randomisieren in der Regel ihre Adressen und ändern sie circa alle 15 Minuten. Um diese Änderungen nachzuvollziehen, wurden Algorithmen entwickelt. Diese detektieren den Adresswechsel in gesendeten Advertisements anhand von zeitlichen und räumlichen Daten sowie den enthaltenen Service- und Company-Ids.

Bei neueren iPhones ist so ein Tracking möglich. Die untersuchten Android Geräte sind nur detektierbar, falls eine entsprechende App, wie beispielsweise die SwissCovid App, installiert ist. Die SwissCovid App sendet kontinuierlich Advertisements, welche über die Service-Id als solche identifiziert werden können.

Das Nachvollziehen von Adresswechseln stösst an Grenzen, wenn ein Adresswechsel bei einer grossen Anzahl von präsenten Geräten erfolgt, zum Beispiel bei der Einfahrt des Zuges in einen Bahnhof. Der Grad an Indeterminismus steigt durch das plötzliche Auftreten einer Vielzahl von Geräten zu stark.

Die genaue Bestimmung der Anzahl von Geräten im Abdeckungsraum ist nicht möglich. Neue iPhones zum Beispiel werden doppelt erfasst, wenn die SwissCovid App installiert ist. Durch die Kombination der Bluetooth-Daten mit Daten aus weiteren Quellen, zum Beispiel aus früheren Fahrgastbefragungen oder aus WiFi-Messungen, sollten aber Hochrechnungen gemacht werden können.

Ausblick

Der Korrelationsalgorithmus könnte durch Daten aus dessen Anwendung weiter optimiert werden. Zusätzliche Geräte-Typen für Messungen und grössere kontrollierte Tests könnten wertvolle Ergebnisse liefern.

Machine-Learning könnte diesbezüglich Anwendung finden. Ein spezialisierter Empfänger könnte die Datenströme besser monitoren. Zudem könnten aktive Verbindungsstrategien für Low Energy Geräte versucht werden.

Die kommende Low Energy-Audio Spezifikation wird neue Ansätze zum Tracken ermöglichen. Bei Verfügbarkeit von LE-Audio in Mobilgeräten sollten die entsprechenden Signale untersucht werden.

Danksagungen

An dieser Stelle möchten wir uns ganz herzlich bedanken bei:

- Prof. Beat Stettler für die hilfreichen Inputs in den Meetings und das Besorgen des zusätzlichen Ubertooth Ones.
- Dem gesamten Team des ICOM für den Zugang zur Antennenmesskammer, das Zurverfügungstellen des Nordic nRF52 Kits und für die Arbeitsplätze in ihrem Büro.
- Dem INS für das Zurverfügungstellen der Android Geräte.
- Karin Batliner für das Zurverfügungstellen ihrer Apple Watch.
- Stefan Marti für das Zurverfügungstellen seiner iPhones und Airpods.
- Nathalie Marti für das Zurverfügungstellen ihres iPhones und ihrer Airpods und das Kochen in stressigen Zeiten.
- Jerome Roy für das Zurverfügungstellen seiner Airpods.
- Lukas Kiefer für das Zurverfügungstellen seiner Airpods.
- Ursula Marti für das Gegenlesen ausgewählter Dokumente.

Inhaltsverzeichnis

Abstract **ii**

Lay Summary **iii**

Danksagungen **v**

Inhaltsverzeichnis **vi**

1 Ausgangslage **1**
1.1 Einleitung . . . 2
1.1.1 Ausgangslage . . . 2
1.1.2 Problembeschreibung . . . 2
1.1.3 Motivation . . . 3
1.1.4 Randbedingungen . . . 4
1.1.5 Einschränkungen . . . 4
1.1.6 Produkt Perspektive . . . 4
1.1.7 Produkt Funktion . . . 5
1.2 Vorherige Arbeiten . . . 6
1.2.1 Einführung . . . 6
1.2.2 Studienarbeit Passenger Tracking . . . 6
1.2.3 Bachelorarbeit Mobile Fingerprinting . . . 7
1.3 Technologieanalyse . . . 9
1.3.1 Bluetooth . . . 9
1.3.2 Spezielle Packets . . . 17
1.3.3 Eigenheiten von iOS 11+ . . . 19

2 Problembeschreibung **26**
2.1 Use Cases . . . 27
2.1.1 Use Case Diagramm . . . 27
2.1.2 Aktoren & Stakeholder . . . 27
2.1.3 Beschreibungen (Brief) . . . 28
2.2 Nicht-Funktionale Anforderungen . . . 29
2.2.1 Adaptability . . . 29
2.2.2 Interoperability . . . 30
2.2.3 Flexibility . . . 30
2.2.4 Performance . . . 30
2.2.5 Durability & Availability . . . 30
2.2.6 Privacy . . . 30
2.2.7 Throughput . . . 31
2.3 Betriebsmittelliste . . . 32
2.3.1 Endgeräte . . . 32
2.3.2 Peripheriegeräte . . . 32
2.3.3 Antennen . . . 32

3 Lösungskonzept **33**

3.1 Tooling . . . 34
3.1.1 Hardware . . . 34
3.1.2 Software . . . 37
3.1.3 Antennenkammer . . . 42
3.2 Domainanalyse . . . 43
3.2.1 Einführung . . . 43
3.2.2 Domain Modell . . . 43
3.3 Ansätze . . . 47
3.3.1 Bluetooth BR/EDR . . . 47
3.3.2 Bluetooth LE . . . 49

4 Umsetzung **55**
4.1 Softwarearchitektur . . . 56
4.1.1 Einführung . . . 56
4.1.2 Systemübersicht . . . 56
4.1.3 Architektonische Ziele & Einschränkungen . . . 56
4.1.4 Logische Architektur . . . 58
4.1.5 Prozesse und Threads . . . 65
4.1.6 Deployment . . . 67
4.1.7 Datenspeicherung . . . 67
4.2 Implementationsdetails . . . 71
4.2.1 Der Monitor . . . 71
4.2.2 Der Correlator . . . 74
4.3 Messungen / Tests . . . 84
4.3.1 Testprotokoll . . . 84
4.3.2 Messung Weg 1 . . . 86
4.3.3 Messung Weg 2 . . . 89
4.3.4 Messung Weg 3 . . . 92
4.3.5 Messung Weg Abend . . . 95
4.3.6 Test 28.05 . . . 98

5 Ergebnisse **100**
5.1 Resultate . . . 101
5.2 Limitationen . . . 102
5.3 Diskussion und Ausblick . . . 104
5.3.1 LE Audio . . . 104
5.3.2 Advertisements . . . 104
5.3.3 Monitor . . . 104
5.3.4 Kombination mit anderen Daten . . . 105
5.3.5 Kontrollierte Tests in grösserem Massstab . . . 105
5.3.6 Machine Learning . . . 105

Abbildungsverzeichnis **I**

Tabellenverzeichnis **II**

Literaturverzeichnis **III**

Glossar **V**

Appendix **I**
Appendix A - Installationsanleitung II
6.1.1 Übersicht III
6.1.2 Systemvoraussetzungen III
6.1.3 Ubertooth III
6.1.4 Monitor IV
6.1.5 Backend & Frontend V
6.1.6 Correlator VI

1 Ausgangslage

1.1 Einleitung

Dieser Abschnitt dient dem Festhalten der Ausgangslage. So werden zuerst die Aufgabenstellung und Motivation der Arbeit beschrieben, dann folgen Zusammenfassungen der Vorgängerarbeiten. Zum Schluss folgt noch die Technologieanalyse.

1.1.1 Ausgangslage

In dieser Arbeit gilt es, die Frage zu beantworten, ob und wie ein Fingerprinting von Mobilgeräten in einem öffentlichen Verkehrsnetz möglich wäre.

Es handelt sich hierbei um die dritte Forschungsarbeit mit diesem Ziel. Bisher wurde das Augenmerk stets auf die Machbarkeit anhand von WiFi gelegt. Es wurde herausgefunden, dass aufgrund der Methoden, die moderene WiFi Netzwerke zur Anonymisierung verwenden, ein Tracking allein über WiFi nur sehr beschränkt möglich ist. Nach diesen Vorgängerarbeiten wurde entschieden, dass neue Ansätze benötigt werden, um dem Ziel näher zu kommen.

So soll die Machbarkeit eines Fingerprintings anhand von Bluetooth Signalen untersucht werden. In einem zweiten Schritt sollen die erarbeiteten Ansätze genauer unter die Lupe genommen und durch die Implementation eines Prototypen verifiziert werden.

Es existieren bereits Lösungen und Ansätze, die speziell die Zählung von Geräten im Empfangsradius eines Access Points bieten. Diese operieren unter den allgemeinen Einschränkungen von Funkschnittstellen. So können die WiFi Probe Request von den Endgeräten getrackt werden. Ist das Interface deaktiviert, werden keine Daten gesendet. Am problematischten ist dabei die Randomisierung von MAC Adressen. Diese dienen in der Regel zum eindeutigen identifizieren der Geräte, wechseln aber periodisch. Um die Adresswechsel der passiven Requests verfolgen zu können und so die Zählung zu implementieren, muss eine Korrelation der randomisierten MAC Adressen zueinander getätigt werden.

Die hier publizierten Forschungsergebnisse sollen dem Industriepartner zur Verfügung gestellt werden, welcher sie dann weiter erforscht und aufgreift. Es wird deshalb angenommen, dass ein Leser bereits ein gewisses Verständnis von Computer Netzwerken, Signalen und Softwareentwicklung mitbringt.

1.1.2 Problembeschreibung

Die zu untersuchende Schnittstelle ist der Bluetooth Standard 5.2, definiert in der IEEE 802.11 zugrunde liegenden Beschreibung der Verwendung von Frequenzen im 2,4 GHz Bereich.

Bei Bluetooth muss prinzipiell zwischen dem Classic und Low Energy Stack unterschieden werden. Massgeblich sind also die Endgeräte auf die das Tracking

abzielt. Moderne Handys können die Signale beider Stacks verarbeiten. Wie genau dem Standard dabei gefolgt wird ist an manchen Stellen den Herstellern der Bluetooth Chips überlassen. In dieser Hinsicht werden vor allem die Peripherie Geräte von No Name Herstellern, die keine offizielle Company ID bei der Bluetooth SIG haben, problematisch. So sind in manchen Feldern des Paketes vom Hersteller schlicht beliebige Daten eingefügt.

Bei den unterschiedlichen Versionierungen von Bluetooth muss weiter beachtet werden, dass das Band, auf dem Bluetooth operiert, je nach Version, entweder 40 oder 80 Kanäle verwendet. Die meisten Antennen sind darauf ausgelegt nur einen Kanal monitoren zu können. So bräuchte man eine korrespondierende Anzahl an Empfängern, wenn man den gesamten Datenfluss mitschneiden möchte.

In Bluetooth Classic sind die Verbindungen auf Piconetze mit wenigen Geräten beschränkt. Die Verbindung läuft über ein direktes Pairing. Die Schwierigkeit besteht also darin, genügend Daten einzufangen, um ausreichend Information über die Verbindung zu erhalten. Besteht keine aktive Verbindung, werden, Ungleich dem LE, keine passiven Signale gesendet. Sind ausreichend Daten gegeben, ist zur Bestimmung der Adresse des Endgeräts ein Bruteforce nötig. Dieser stellt für heutige Prozessoren keine Herausforderung dar. Zur Kalkulation müssen lediglich 2^16 Kombinationen versucht werden. Diese Adressen könnten sich nach dem Standard zwar ändern, dieses Feature scheint jedoch in der Regel nicht genutzt.

In der Low Energy Implementation von Bluetooth existiert eine grössere Anzahl an möglichen Paketen. In erster Linie ist also abzuschätzen, welche Felder dieser Pakete überhaupt in Frage kommen könnten, um ein Tracking zu versuchen. Dabei ist vor allem schwierig, dass die Adressen, mit denen die Pakete versehen werden, randomisiert sind. Die meisten Geräte wechseln ihre Adressen periodisch, allerdings gibt es auch azyklische Events, die zum Adresswechsel führen können. Damit ist nicht vorherzusagen, wann sich die Adressen ändern. In bestimmten Zuständen wird eine Vielzahl an Signalen geschickt, auch ohne aktive bestehende Verbindungen. Es gilt also die Zustände, Begebenheiten für Zustandswechsel und Korrelationen der Adressen zu bestimmen.

1.1.3 Motivation

Infomationen, wie Passagiere die öffentlichen Verkehrsmittel nutzen, sind essentiell, um Fahrpläne optimal auf das Passagierbedürfnis abstimmen zu können. Informationen über die aktuelle Anzahl an Passagieren helfen, im Falle von Störungen die richtigen Ersatzfahrzeuge zu organisieren. Bisher können diese Daten nur aus manuellen Fahrgastbefragungen und aus historischen Werten gewonnen werden. Aus dem Bedürfnis, diese Informationen automatisch und fortlaufend zu sammeln, entstand das Thema dieser Arbeit.

Die Vorarbeiten sind Teil der Motivation, einen neuen Trackingansatz zu erforschen. Bluetooth ist praktisch immer in Mobilgeräten integriert, die

Bevölkerung hat jedoch davon nur ein recht simples Verständnis. Ein primärer Use Case von Bluetooth in Mobilgeräten ist, mit Wireless Kopfhörern Musik zu hören. Das industrielle Interesse an Bluetooth für Sensorik und Services beziehungsweise IoT-Anwendungen ist ungemein grösser. Bluetooth ist eines der am weitesten verbreiteten Signale überhaupt und rückt vor allem durch die heutigen Standards immer weiter ins Rampenlicht.

Es gibt bereits Studien in denen verschiedenste Fähigkeiten, vor allem des Low Energy Protokolls, getestet werden. Darunter auch Ansätze, in denen aktiv Bluetooth Verbindungen aufgebaut werden um die Eigenschaften der Geräte zu erkennen. Ziel der Arbeit ist es, die Nachverfolgung komplett passiv nur als "Zuhörer" zu gestalten.

1.1.4 Randbedingungen

Bei unserer Arbeit handelt es sich um eine Forschungsarbeit mit Machbarkeitsstudien der verschiedenen Ansätze. Deshalb werden die Tests unter möglichst optimalen Bedingungen durchgeführt.

Implementation der Schnittstelle ist herstellerspezifisch. Deshalb ist es uns nicht möglich eine Pauschal-Lösung anzubieten. Es können nur die Auswirkungen und Erfolgsraten der getätigten Experimente beschrieben werden. In der realen Welt sind Abweichungen von den hier präsentierten Ergebnissen zu erwarten.

Der Leser soll dabei bedenken, dass ein Tracking über ein einzelnes Signal warscheinlich nie gut genug sein wird. Mit einer Kombination verschiedener Systeme, die auf weitere Signale abzielen, sollten allerdings genügend Daten vorhanden sein, um effektiv funktionale Nutzungsprofile erstellen zu können.

Alle Versuche in dieser Arbeit, sofern nicht anders beschrieben, werden unter Laborbedingungen getätigt.

1.1.5 Einschränkungen

Dieses Projekt arbeitet mit gewissen Hardware-Einschränkungen. So könnte die entwickelte Software sicherlich performanter gestaltet werden, jedoch mit einem unverhältnismässigen Mehraufwand bei der Hardwarebeschaffung.

1.1.6 Produkt Perspektive

Aus Sicht eines Produkts soll vor allem das Augenmerk auf Resistenz und Nachvollziehbarkeit der entstehenden Datenstreams und daraus generierten Ergebnisse liegen. Weiter muss es möglich sein, die generierten Daten im Nachgang effizient abfragen zu können.

1.1.7 Produkt Funktion

Nutzer werden passiv durch die über Bluetooth gesendeten Daten erkannt, kategorisiert und lokalisiert. Dies soll ohne eine explizite Aufforderung zum Pairing möglich sein.

Entsprechend der Daten sollen dann Nutzungsprofile für die Bahnbetreiber generiert werden können, um so die Auslastungseffizienz der Linien maximieren zu können.

1.1.7.1 Benutzer Charakteristik

Diese Lösung ist explizit für den öffentlichen Transportsektor intendiert. Innerhalb der Fahrzeuge haben die Betreiber sozusagen die Hoheit und können für eine ausreichende Signalabdeckung sorgen.

1.1.7.2 Annahmen

In erster Linie wird angenommen, das jeder oder quasi jeder Teilnehmer im Verkehrsbetrieb über ein Mobiltelefon verfügt.

Zudem wird davon ausgegangen, dass der grösste Teil dieser Handybesitzer ein relativ neues Gerät besitzt, damit es über die neueren Implementationen des Standards verfügt.

Ebenso wird angenommen, dass die Abdeckung innerhalb des Fahrzeugs durch die Betreiber entsprechend Gewährleistet ist.

1.1.7.3 Abhängigkeiten

Das Produkt hängt von einer guten Coverage des Signals in einem vermeshten Netzwerk ab. Zudem ist eine geringe Interferenz ideal.

Die Ergebnisse dieser Arbeit unterliegen einem relativ simplen Monitoring, zumindest bezüglich der Abdeckung an vom Standard verwendeten Kanälen.

Gewisse Limitationen in Schnittstellen oder deren Implementierung sind ebenfalls zu erwarten, so wird sich ein Android oder Apple Endgerät jeweils distinkt verhalten und entsprechend Daten transferieren.

1.2 Vorherige Arbeiten

1.2.1 Einführung

In dieser Sektion werden Erkenntnisse und weitere relevante Punkte aus den zwei Vorgängerarbeiten zusammengefasst.

Die beiden Vorgängerarbeiten [1] [2] haben versucht, anhand von IEEE 802.11 Probe Requests (WLAN Probe Requests) Mobilgeräte zu fingerprinten.

1.2.2 Studienarbeit Passenger Tracking

1.2.2.1 Daten

Es wurde mit Daten der Firma onway AG gearbeitet. Darin enthalten waren unter anderem 175 Millionen Probe Requests, welche während einer Woche auf Buslinien in einer schweizer Grossstadt aufgezeichnet wurden.

1.2.2.2 Ansatz

Mittels Machine Learning wurde versucht anhand der 13 zuverlässig vorkommenden Datenfelder der Payload der Probe Requests Cluster zu bilden und diese einzelnen Geräten zuzuweisen.

1.2.2.3 Ergebnisse

- Es wurde festgestellt, dass 83.5% der gefundenen MAC Adressen randomisiert waren.
- Bei nicht-randomisierten MAC Adressen konnte gezeigt werden, dass die Sequenznummern aus dem Paket Header vorhersehbar zunimmt (Fraefel et al. (2020). pp. 25).
- Bei randomisierten MAC Adressen konnten einzelne Sequenzen identifiziert werden, jedoch waren diese zeitlich stark begrenzt. Fraefel et al. vermuten, dass die sich die Geräte entweder nur kurzzeitig in Reichweite des WLAN Netzwerks waren oder die Sequenznummer bei Wechsel der MAC Adresse bei einer neuen Zahl startet (Fraefel et al. (2020). pp. 26).
- Die Anzahl der Geräte konnte mit einer durchschnittlichen Genauigkeit von 94% bestimmt werden. Ebenfalls konnte der Gerätehersteller erkannt werden (unbekannte Genauigkeit).

- Unterschiedliche Modelle des selben Herstellers konnten nicht zuverlässig unterschieden werden, weshalb einzelne Geräte nicht identifiziert werden konnten.

1.2.2.4 Sonstiges

Fraefel et al. merken an, dass Geräte unmittelbar nacheinander Probe Requests für verschiedene Channels mit inkrementierter Sequenznummer verschicken (Fraefel et al. (20020). pp 27).

1.2.3 Bachelorarbeit Mobile Fingerprinting

1.2.3.1 Daten

Da die in der Studienarbeit Passenger Tracking verwendeten Daten keine Zuweisung von Probe Requests und Geräten erlauben und deshalb keine zuverlässigen Test Sets vorhanden sind, wurden in 108 Messungen neue Daten erhoben.
Es wurden Probe Requests von 4 iPhones mit unterschiedlichen iOS Versionen sowie 9 Android Geräten mit teilweise identischen Versionen gesammelt.

1.2.3.2 Ansatz

Die Probe Request Frames wurden nach Filterkriterien in absteigender Eindeutigkeit in Gruppen aufgeteilt, welche jeweils eine Geräteklasse repräsentieren.

1.2.3.3 Ergebnisse

- Eine Zuteilung von Probe Request Bursts basierend auf dem Timing wurde verworfen. Die Dauer eines Bursts war zu kurz und die Zeit zwischen zwei Bursts nicht vorhersehbar (Schlatter & Schmid (2021). pp. 80).

- iPhones mit iOS-Version 14 oder höher konnten nicht unterschieden werden.

- Android-Geräte konnten unterschieden werden. Die Autoren spekulieren jedoch, basierend auf historischen Trends, dass auch Android-Geräte (mit neueren Android-Versionen) in Zukunft nicht mehr unterscheidbar sein werden.

1.2.3.4 Sonstiges

Die aufgezeichneten Probe Requests unterschieden sich deutlich abhängig vom WLAN-Zustand der Geräte (aktiv/passiv).

1.3 Technologieanalyse

In diesem Kapitel werden die verwendeten Bluetooth Protokolle und PDUs beschrieben.

1.3.1 Bluetooth

1.3.1.1 Frequenz

Bluetooth operiert auf dem lizenzfreien 2.4 GHz ISM-Band. Lizenzfrei bedeutet hier, dass keine Lizenz benötigt wird und demnach jedermann auf diesem Band senden darf.
WiFi sowie weitere, proprietäre Protokolle verwenden dieses ebenfalls.

1.3.1.2 Frequency Hopping

Um Interferenz mit anderen Bluetooth-Geräten oder anderen Signalen wie zum Beispiel WiFi zu vermeiden, benutzt Bluetooth Frequency Hopping.

Dabei wechseln Geräte, basierend auf einem vom Master vorgegebenen Hopping Sequence, mehrere hundert Mal pro Sekunde den verwendeten Kanal. Da die Hopping Sequence pro Link unterschiedlich ist und Geräte jeweils nur für sehr kurze Zeit auf einem gegebenen Kanal sind, wird die Chance für Interferenz minimiert.

Um Interferenz zusätzlich zu vermeiden, kann zudem Adaptive Frequency Hopping (AFH) verwendet werden. Bei AFH werden bekannte “schlechte” Kanäle auf bekannte “gute” Kanäle gemappt.

Abb. 1 zeigt ein Beispiel von Adaptive Frequency Hopping bei Bluetooth LE. Da auf den weiss eingefärbten Kanälen durch die Wifi-Netzwerke viel interferenz zu erwarten ist, werden diese vermieden und stattdessen nur die grün markierten Datenkanäle verwendet.

1.3.1.3 Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR)

BR/EDR (auch Bluetooth Classic genannt) ist das ältere der beiden hier aufgeführten Protokolle. Es ist auf hohen Datendurchsatz und geringen Stromverbrauch ausgelegt.

Die maximale Datenrate beträgt 3Mb/s.

Es werden drei Leistungsklassen mit unterschiedlicher maximaler Sendeleistung unterschieden, wie in Tab. 1 gezeigt.

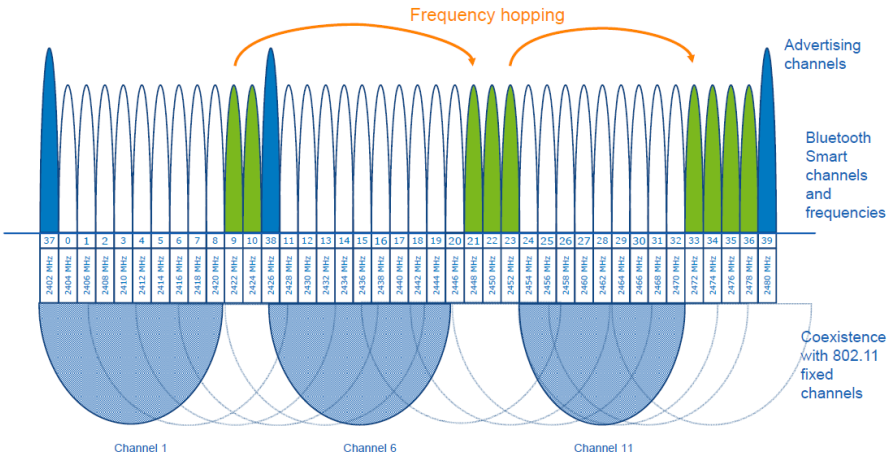

Abbildung 1: Bluetooth LE Channels und Adaptive Frequency Hopping. Quelle: [3]

Tabelle 1: Bluetooth Leistungsklassen. Klasse 1.5 nur bei LE

Klasse	Maximale Sendeleistung	Reichweite
1	100mW / 20dBm	100m
1	100mW / 20dBm	100m
1.5	10mW / 10dBm	20m
2	2.5mW / 4dBm	10m
3	1mW / 0dBm	1m

Die angegebenen Reichweiten sind theoretisch zu erwartende Werte. Unter optimalen Bedingungen können diese erreicht oder sogar überschritten werden, in der Praxis sind sie jedoch meist deutlich niedriger.

Die für uns interessanten Geräte wie Mobiltelefone, Kopfhörer, Smart watches etc sind Geräte der Klasse 2.

1.3.1.3.1 Channels

Bluetooth Classic benutzt 79 1MHz Kanäle von 2'402GHz bis 2'480MHz.

1.3.1.3.2 Bluetooth Address

Jedem Bluetoothgerät wird vom Hersteller eine einzigartige 48-bit Bluetooth Adresse (BD_ADDR) zugewiesen. Abb. 2 zeigt die verschiedenen Teile der Bluetooth Adresse.

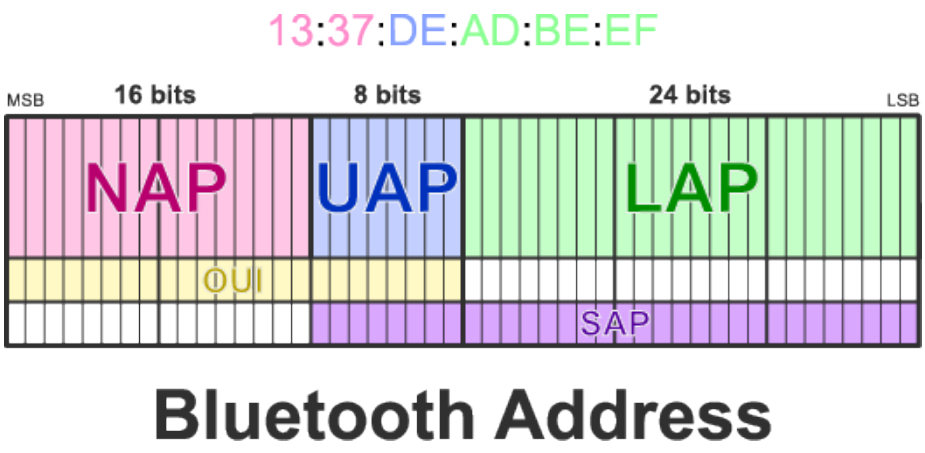

Abbildung 2: Aufbau der Bluetooth Adresse

Non-significant Address Part (NAP)

Der Non-significant Address Part bildet die ersten 2 Bytes der Bluetooth Adresse und repräsentiert die ersten 16 Bit des OUI.

Auf dem Link Layer können Geräte auch ohne diese ersten zwei Bytes eindeutig identifiziert und adressiert werden (00:00:DE:AD:BE:EF ist funktional identisch mit FF:FF:DE:AD:BE:EF).

Der NAP ist nur in Frequency Hopping Synchronization (FHS) Packets enthalten.

Upper Address Part (UAP)

Das dritte Byte der Bluetooth Addresse hat die Bezeichnung Upper Address Part. Er besteht aus den restlichen 8 Bit des OUI.

Der UAP wird, teilweise zusammen mit dem LAP, als Seed für Algorithmen des Bluetooth Standards verwendet (z.B. Header Error Check (HEC), Cyclic Redundancy Check (CRC)).

Der UAP ist in der FHS Payload enthalten. Für Details, wie der UAP aus mitgeschnittenen Packets berechnet werden kann, siehe [4].

Lower Address Part (LAP)

Der LAP besteht aus least significant 4 Bytes der Bluetooth Adresse und wird vom Gerätehersteller zugewiesen.

Organizationally Unique Identifier (OUI)

Der Organizationally Unique Identifier ist eine 3 Byte lange Zahl, die eine Organisation, wie beispielsweise einen Hardwarehersteller, eindeutig identifiziert. Der OUI wird häufig dafür verwendet, Hardwarekomponenten zu identifizieren. MAC-Adressen beinhalten die OUI des Herstellers.

Die ersten drei Oktette der BD_ADDR sind der OUI.

1.3.1.4 Access Code

Jedes Packet beginnt mit einem Access Code. Dieser besteht aus einer Preamble, einem Sync word und optional einem Trailer, vgl. Abb. 3.

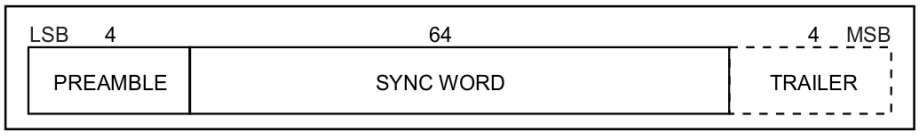

Abbildung 3: Access Code Format. Quelle: [5]

Das Sync Word wiederum wird anhand eines LAP generiert. Sind zwei Geräte verbunden und tauschen Daten aus, so sind sie im Connection State und verwenden den LAP des Masters für das Sync Word. Dadurch kann von einem gegebenen Packet auf die LAP des Masters zurückgeschlossen werden.

1.3.1.5 Piconet

Ein Piconet ist ein ad hoc Netzwerk von mehreren Bluetooth-Geräten. Alle Geräte in einem Piconet haben eine synchronisierte Clock und benutzen die selbe Hopping Sequence. In einem Piconet gibt es einen Master und bis zu 7 aktive Slaves. Es können sich bis zu 255 weitere Slaves im parked State befinden, welche bei Bedarf aktiviert werden können. Bei den für uns interessanten Verbindungen wie Mobiltelefon und Kopfhörer ist das Telefon der Master.

Allen Slaves werden Timeslots zugewiesen in welchen sie Senden können bzw. in welchen sie auf Packets vom Master hören können.
Sämtliche Packets die in einem Piconet versandt werden benutzen den selben Channel Access Code, welcher den LAP des Masters beinhaltet. An wen ein gegebenes Packet gerichtet ist, ergibt sich aus dem Timeslot.

1.3.1.5.1 Pairing

Abb. 4 zeigt den Verbindungsaufbau-Vorgang bei Bluetooth BR/EDR.

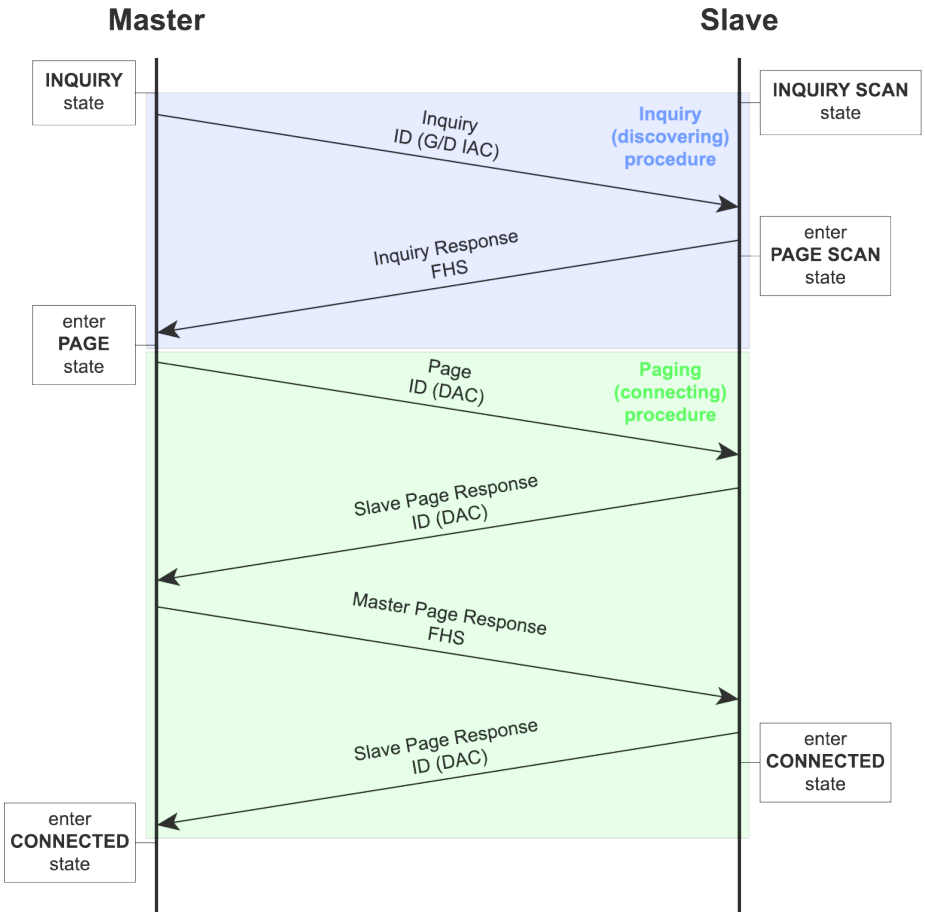

Abbildung 4: Verbindungsaufbau-Vorgang Bluetooth BR/EDR

Der Text über den Pfeilen bezeichnet jeweils den Request/Reply-Typen und darunter den Pakettyp und den Access Code in Klammern.

Abkürzung	Bedeutung
G/D IAC	General/Dedicated Inquiry Access Code
DAC	Device Access Code
FHS	Frequency Hopping Selection

1.3.1.6 Bluetooth Low Energy (LE)

Im Bereich der Internet of Things (IoT) Geräte kam das Bedürfnis nach günstigeren und energieeffizienteren Bluetooth Chips auf. Da viele der Anwendungen die relativ hohen Datenraten von Bluetooth BD/EDR nicht benötigen und stattdessen selteneres Laden oder generell geringeren Stromverbrauch bevorzugen würden, wurde 2010 die Spezifikation für Bluetooth Low Energy (BTLE) in den Standard aufgenommen. Mit BTLE kann der Stromkonsum im Vergleich zu Bluetooth BR/EDR - abhängig vom Anwendungsfall - um einen Faktor 2 bis 100 reduziert werden.

Für heute viel gebrauchte Anwendungen wie Keyboards, Mäuse, Fitness-Tracker, Glühbirnen und Ähnliches bedeutet dies eine deutlich längere Akkulebensdauer.

Da mit dem aktuellen Trend, physische Audioanschlüsse von Mobiltelefonen zu entfernen, wireless Kopfhörer immer beliebter werden und gerade bei diesen eine lange Nutzungszeit ohne Aufladen wichtig ist, wurde anfangs 2020 LE Audio angekündigt. Es soll in Zukunft standard Bluetooth Audio Anwendungen ersetzen.

1.3.1.6.1 Channels

Bluetooth Classic benutzt 40 2MHz Kanäle von 2'402GHz bis 2'480MHz. Dabei sind die Channels 37, 38 und 39, welche über das Frequenzband verteilt sind, als Advertising-Channels designiert. Die verbleibenden 37 Channels sind Daten-Channels.

Abb. 1 zeigt die Advertising-Channels in blau sowie dazwischen die Daten-Channels in grün und weiss.

1.3.1.6.2 Access Address

Anstelle von Access Codes wie sie bei Bluetooth Classic verwendet werden um ein Piconet zu identifizieren, verwendet Bluetooth LE Access Addresses. Jede Verbindung zwischen zwei Geräten benutzt eine unterschiedliche Access Address.

Die Access Address ist ein 32-Bit Wert und wird für jede Verbindung neu generiert. Sie erlaubt keine Rückschlüsse auf beteiligte Geräte. Die zu erfüllenden Anforderungen für Access Addresses sind in der BLUETOOTH CORE SPECIFICATION Version 5.2 auf Seite 2867 [5] zu finden.

Speziell ist die Advertising Access Address. Alle Packets auf Advertising Kanälen (mit Ausnahme von AUX_SYNC_IND und AUX_CHAIN_IND PDUs, welche für uns nicht relevant sind), müssen die Access Address 0x8E89BED6 verwenden.

1.3.1.6.3 Advertisement

Ein BTLE Advertisement besteht aus folgenden Teilen:

- Access Address:
  Immer 0x8E89BED6
- Packet Header, darin:
  - PDU Type, siehe Advertising PDF Types
  - Channel Selection Algorithm
  - Tx Address:
    Ein Bit, gibt an ob die Advertising Address public oder random ist
- Advertising Address, siehe Advertising Address
- Advertising Data
  Zusätzliche Informationen zum Advertisement wie zum Beispiel eine Hersteller-Id, ein Gerätename oder eine Service UUID

1.3.1.6.4 Advertising PDU Typen

Tab. 2 zeigt die verschiedenen Advertising PDU Typen gemäss Bluetooth Core Spezifikation 5.2 [5].

Tabelle 2: Advertising PDU Typen

Typ	Bedeutung	Verwendung
ADV_IND	Connectable undirected advertising	Alle Empfangenden Geräte können sich mit diesem Gerät verbinden
ADV_DIRECT_IND	Connectable directed advertising	Ein spezifisches Gerät kann sich mit diesem Gerät verbinden
ADV_NONCONN_IND	Non-connectable undeficted advertising	Empfangende Geräte können sich nicht verbinden. Sämtliche Informationen sind in diesem Packet enthalten
ADV_SCAN_IND	Scannable undirected advertising	Empfangende Geräte können sich nicht verbinden. Durch einen Scan Request können zusätzliche Informationen empfangen werden

Typ	Bedeutung	Verwendung
SCAN_REQ	Scan Request	Antwort auf ein ADV_SCAN_IND. Aufforderung zu einem SCAN_RSP
SCAN_RSP	Scan Response	Antwort auf ein SCAN_REQ. Enhält zusätzliche beliebige Daten
CONNECT_REQ	Connection Request	Verbindungs-Request

1.3.1.6.5 Advertising Address

Die Advertising Address ist das pendant zur BD_ADDR bei Bluetooth Classic. Sie hat die Form einer MAC-Adresse.

Abb. 5 zeigt die Adress-Typen.

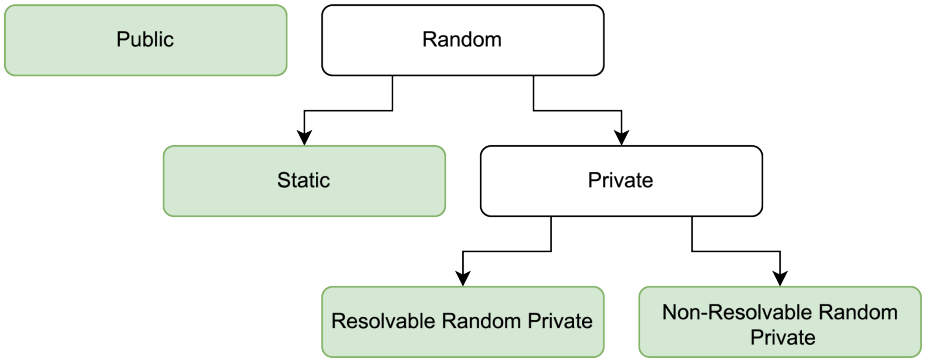

Abbildung 5: Advertising Adress-Typen

Public Adressen müssen bei der IEEE registriert werden und müssen 48-bit extended unique (EUI-48) sein. Sie ändert sich garantiert nie. Da die Registrierung kostenpflichtig ist und eine public Adresse kaum Vorteile gegenüber statischen Adressen hat, wird sie bei Mobilgeräten selten verwendet.

Statische Adressen können beim Start eines Gerätes geändert werden. Während das Gerät läuft, ändern sie sich nicht. Da sie keine Registrierung benötigen, werden sie häufig anstelle von public Adressen verwendet. Die zwei most significant Bits müssen 1 sein.

Private Adressen wurden für den Privatsphärenschutz bei Bluetooth LE eingeführt. Sie können während der Laufzeit des Gerätes geändert werden. Geräteabhängig werden sie nach unseren Messungen circa alle 15 Minuten geändert.

Resolvable Random Adressen erlauben es, gepairten Geräten die gebondet sind, das Gerät trotz geänderter Advertising Address wiederzuerkennen. Beim Bonding, welches optional nach dem Pairing geschehen kann, werden zwischen den zwei Geräten Keys ausgetauscht, anhand welcher die randomisierten Advertising Addresses generiert werden. Dadurch können sich Geräte wiedererkennen und wieder verbinden, ohne dass dies Dritten möglich wäre. Die zwei most significant Bits müssen 01 sein.

Non-Resolvable Random Adressen werden selten verwendet. Sie bieten den höchsten Privatsphärenschutz, da kein Gerät in der Lage ist, anhand einer solchen ein zuvor gesehenes Gerät wiederzuerkennen. Die zwei most significant Bits müssen 0 sein.

1.3.2 Spezielle Packets

Bei Bluetooth BR/EDR werden sämtliche Datenpakete zum Fingerprinten verwendet, bei Bluetooth LE jedoch nur die Advertisements, speziell die Advertising PDU Typen ADV_IND (Connectable undirected advertising), ADV_NONCONN_IND (Non-connectable undirected advertising) und ADV_SCAN_IND (Scannable undirected advertising).

Zwei Packet-Typen sind dabei speziell, da sie häufig und von vielen Geräte gesendet werden und über längere Zeit - ca 15 Minuten - die gleiche Advertising Address behalten. Dies sind die Packets der SwissCovid App und die Advertising Packets von neuen iPhones. Im Folgenden werden diese zwei PDUs genauer beschrieben.

1.3.2.1 SwissCovid App

Die SwissCovid App [6] basiert auf dem Decentralised Privacy-Preserving Proximity Tracing (DP-3T) Projekt[1] [7] und wurde für Contact-Tracing entwickelt. Sie verwendet die Google & Apple Exposure Notifications (GAEN)[2], welche wie folgt funktionieren:

- Geräte generieren täglich einen Temporary Exposure Key (TEK). Die verwendeten TEKs werden gespeichert.
- Anhand des TEK wird regelmässig ein neuer Ephemeral Identifier, auch Rolling Proximity Identifier (RPI) genannt, generiert. Die GAEN-Spezifikation [8] diktiert, dass die RPIs im selben Intervall wie die randomisierte Bluetooth Adresse ändern müssen, um Wireless Tracking zu verhindern. Dies soll alle 10 Minuten geschehen, in unseren Messungen hat das Intervall jedoch zwischen 10 und 12 Minuten variiert.

[1] https://github.com/DP-3T/documents
[2] https://www.google.com/covid19/exposurenotifications/

- Die Geräte senden diese RPI mehrmals pro Sekunde mittels Bluetooth LE Advertisements.
- Geräte im Umfeld registrieren diese Advertisements und speichern die gesehenen RPIs ab.
- Ist eine Person infiziert, so werden die TEKs die in den letzten 14 Tagen verwendet wurden (Kollektiv Diagnose Keys genannt) auf den Diagnoseserver hochgeladen.
- Der Diagnoseserver verteilt die Diagnose Keys auf alle teilnehmenden Geräte, welche dann anhand der TEKs die verwendeten RPIs generieren und diese mit ihrer Liste der gesehenen RPIs vergleichen.

Für die Advertisements benutzt die SwissCovid App Non-connectable Undirected oder Scannable Undirected Advertisements. Abb. 6 und Abb. 7 zeigen, wie diese in Wireshark aussehen. Rot hinterlegt sind Teile der PDU, die sich nicht ändern, während Grüne sich ändern - jedoch, wie erwähnt, gemeinsam. Mit der begrenzten Anzahl an zur Verfügung stehenden Geräten konnte kein genereller Trend, z.B. nach Hersteller, in der Wahl des Advertisement-Typen festgestellt werden.
Da es sich um Advertisements handelt, ist die Access Address immer 0x8e89bed6. Die Advertising Address ist immer Random Non-Resolvable. Die 16-bit Service Class UUID ist immer 0xfd6f und identifiziert die Advertisements als GAEN Advertisements.
Die RPIs und Advertising Adressen haben sich in unseren Messungen Spezifikationsgetreu zusammen geändert. Die SwissCovid App implementiert GAEN - zumindest in diesem Aspekt - also korrekt.

Folgender Filter kann verwendet werden, um in Wireshark nur die GAEN-Packets anzuzeigen: btcommon.eir_ad.entry.uuid_16 == 0xfd6f

Gemäss den Zielen der SwissCovid App sollen Kontakte nur als solche erkannt werden, wenn Geräte höchstens 1.5m entfernt sind. Auch wenn die Sendeleistung generell tiefer ist als bei anderen Advertisements, konnten GAEN Advertisements auch aus über 7 Metern Entfernung noch detektiert werden.

1.3.2.2 Apple Advertisements

Der zweite regelmässig gesehene PDU-Typ ist das Apple Advertisement. Neuere iPhones und Apple Watches senden mehrmals pro Sekunde Advertisements.
Die Autoren mutmassen, dass dies zum schnelleren Wiederverbinden von Geräten so implementiert ist. Alternativ könnten die Advertisements auch dazu dienen, Peripheriegeräte wie Airpods oder Apple Watches einfach zu pairen. Bingt man solche nahe an ein iPhone, so taucht dort automatisch ein Pairing-Popup auf. Die genaue Funktion konnte nicht eruiert werden.
Ebenso unklar ist, welche Kriterien erfüllt sein müssen, damit iPhones diese Advertisements aussenden. Von den getesteten iPhones haben das iPhone

```
Bluetooth Low Energy Link Layer
> Access Address: 0x8e89bed6
> Packet Header: 0x2542 (PDU Type: ADV_NONCONN_IND, TxAdd: Random)
  Advertising Address: 16:c1:b8:b0:45:fb (16:c1:b8:b0:45:fb)
v Advertising Data
  v Flags
      Length: 2
      Type: Flags (0x01)
      000. .... = Reserved: 0x0
      ...1 .... = Simultaneous LE and BR/EDR to Same Device Capable (Host): true (0x1)
      .... 1... = Simultaneous LE and BR/EDR to Same Device Capable (Controller): true (0x1)
      .... .0.. = BR/EDR Not Supported: false (0x0)
      .... ..1. = LE General Discoverable Mode: true (0x1)
      .... ...0 = LE Limited Discoverable Mode: false (0x0)
  v 16-bit Service Class UUIDs
      Length: 3
      Type: 16-bit Service Class UUIDs (0x03)
      UUID 16: Google/Apple Exposure Notification Service (0xfd6f)
  v Service Data - 16 bit UUID
      Length: 23
      Type: Service Data - 16 bit UUID (0x16)
      UUID 16: Google/Apple Exposure Notification Service (0xfd6f)
    v Google/Apple Exposure Notification
        Rolling Proximity Identifier: a792f45b03af1778f0e7886e1999a047
        Associated Encrypted Metadata: 569d6853
  CRC: 0xa5f5ca
```

Abbildung 6: SwissCovid (GAEN) ADV_NONCONN_IND Advertisement in Wireshark

X, iPhone XS Max und das iPhone XR die Signale generiert, das iPhone 6 hingegen nicht. Die iOS-Versionen auf den iPhones X, XS Max und XR waren dabei 14.x, auf dem iPhone 6 12.x. Es ist nicht klar ob das Emittieren des Signals vom Gerätetypen, der iOS-Version oder beidem abhängt.

Abb. 8 zeigt ein Apple Advertisement in Wireshark. Sich ändernde Felder sind grün hinterlegt, die restlichen rot. Wie erwartet ist die Access Address stets 0x8e89bed6 und der Address-Typ Random Resolvable. Auch hier ändern die Advertising Address, die Manufacturer Specific Data, die Länge und der CRC jeweils gemeinsam. Gemäss unseren Tests tun sie dies jeweils etwa alle 15 Minuten. Die Company ID identifiziert die Advertisements als von Apple kommend.

Mit folgendem Filter können in Wireshark nur Apple Advertisements angezeigt werden: btcommon.eir_ad.entry.company_id == 0x004c.

1.3.3 Eigenheiten von iOS 11+

Mit iOS 11 wurde das Deaktivieren von Bluetooth umständlicher gemacht. Das nicht sehr intuitive Verhalten ist vielen Leuten nicht bewusst und soll deshalb hier kurz erklärt werden.

Ist Bluetooth eingeschaltet, so gibt es vermeintlich verschiedene Wege, dieses

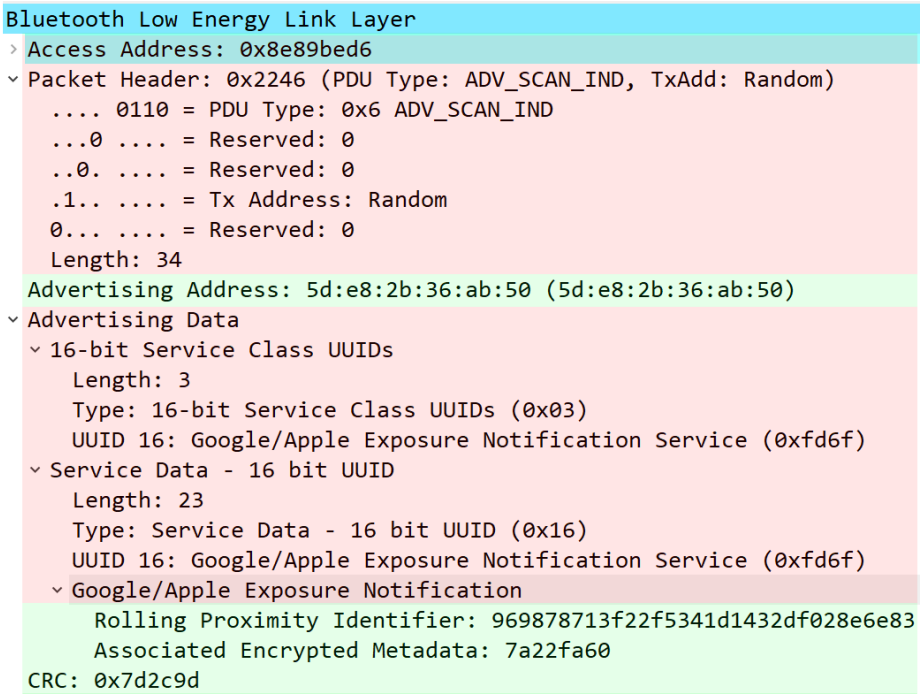

Abbildung 7: SwissCovid (GAEN) ADV_SCAN_IND Advertisement in Wireshark

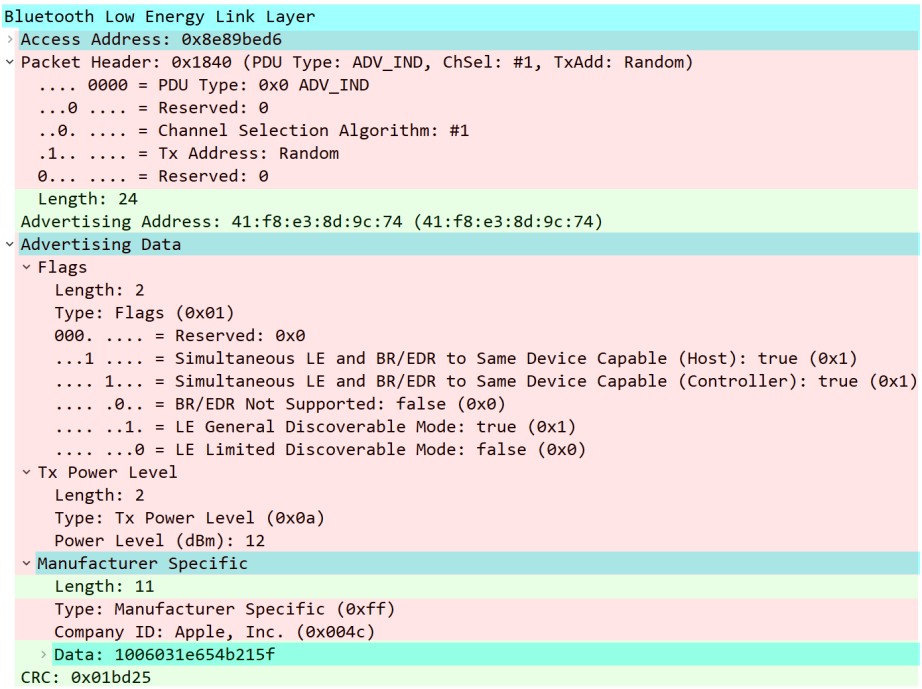

Abbildung 8: Apple ADV_SCAN_IND

wieder zu deaktivieren.

Eine der Möglichkeiten ist über das Control Center, indem man von oben rechts runter-swiped und das Bluetooth-Symbol antippt, wie in Abb. 9 gezeigt. Diese Methode deaktiviert Bluetooth jedoch nicht, sie trennt nur bestehende Verbindungen. Apple Pencils und Apple Watches können weiterhin mit dem iPhone kommunizieren. Auch werden weiterhin Bluetooth LE Advertisements geschickt. Zusätzlich wird die volle Funktionalität automatisch am nächsten Tag um 5 Uhr wiederhergestellt.

Wirklich deaktiviert wird Bluetooth nur, wenn es in den Einstellungen ausgeschaltet wird wie in Abb. 10 gezeigt.

Wird der Flugmodus aktiviert, so würde man erwarten, dass Bluetooth ebenfalls deaktiviert wird. Das ist nicht zwingend der Fall wie in Abb. 11 zu sehen ist. Stattdessen merkt sich das iPhone, ob Bluetooth beim letzten Mal als der Flugmodus aktiviert war, eingeschaltet gewesen ist und übernimmt diesen Zustand. Hat man Bluetooth also nie manuell deaktiviert, während man im Flugmodus gewesen ist, bleibt es weiterhin eingeschaltet.

Es ist anzumerken, dass die Bluetooth Adresse für LE neu randomisiert wird, wenn in oder aus dem Flugmodus gewechselt wird.

Diese Verhalten sind nützlich für diese Arbeit, da iPhone-Benutzer dadurch weniger wahrscheinlich Bluetooth deaktiviert haben und somit trackbar sind.

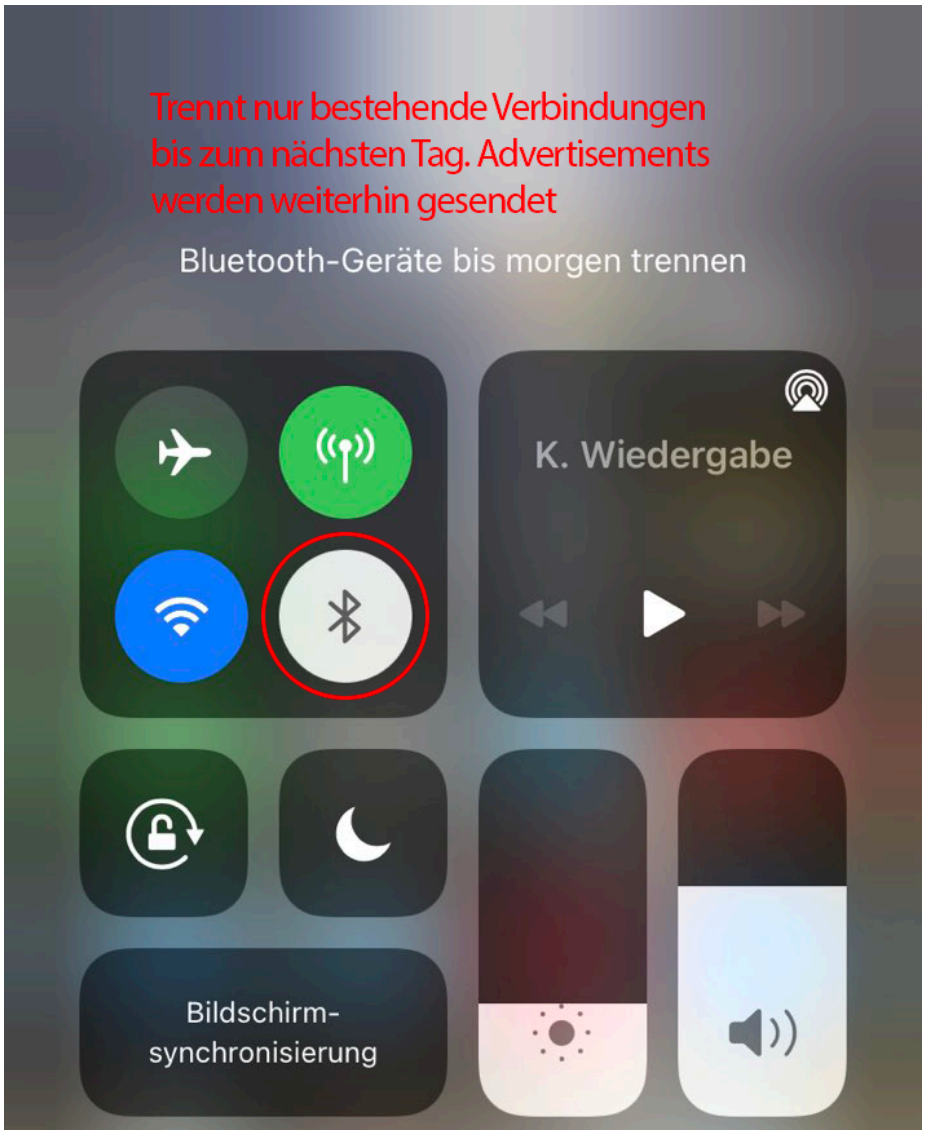

Abbildung 9: Bluetooth Option im Control Center. Deaktiviert nur bestehende Verbindungen bis 5:00 am nächsten Tag

Abbildung 10: Bluetooth Einstellungen. Deaktiviert Bluetooth vollständig

Abbildung 11: Flugmodus ist an, Bluetooth aber auch

2 Problembeschreibung

2.1 Use Cases

Als Use Cases haben wir identifiziert:

- Monitoring der passiven Bluetooth Signale
- Potentielles Verbinden zu identifizierten Geräten
- Lokalisierung eines Geräts im Raum
- Genauere Identifikation eines Geräts durch verschiedene Daten

2.1.1 Use Case Diagramm

Abb. 12 zeigt das Use Case Diagramm des Projektes.

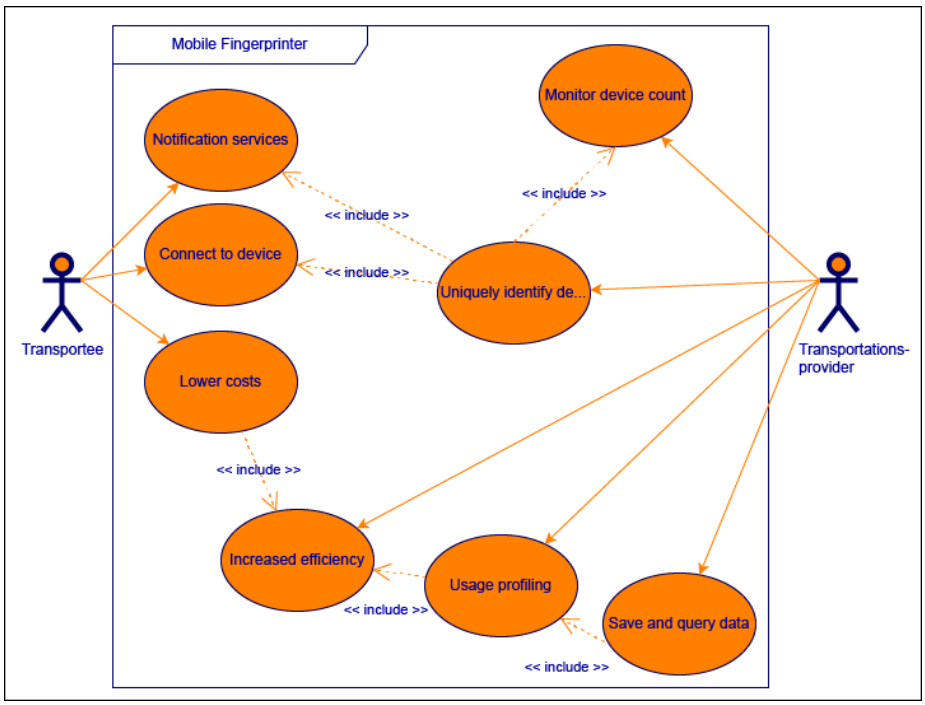

Abbildung 12: Usecases mit den zugehörigen Aktoren

2.1.2 Aktoren & Stakeholder

Folgend ist die Beschreibung der im System zu erwartenden Aktoren und der Interessen der verschiedenen Stakeholder.

2.1.2.1 Aktoren

Die Aktoren in unserem System können essentiell reduziert werden auf die im Empfangsradius befindlichen Personen und deren Mobilgeräte und die ÖV-Betreiber.

2.1.2.1.1 Transportee als Aktor

Durch das Betreten des Empfangsradiuses der in den Fahrzeugen verbauten Antennen werden sie zu unserer Aktoren, die es zu monitoren gilt.

2.1.2.1.2 Transportationprovider als Aktor

Diese sind vorprogrammiert, da sie regelmässig in das System eingreifen müssen, um den korrekten Ablauf zu gewährleisten.

2.1.2.2 Stakeholders

2.1.2.2.1 Transportee als Stakeholder

Erwartet einen vertraulichen Umgang mit seinen Daten.

Erhofft sich als Endnutzer des ÖV bezahlbare Preise der Betreiber.

2.1.2.2.2 Transportationprovider als Stakeholder

Am wichtigsten ist für die Betreiber Störungen zu minimieren. Solange dies gewährleistet ist, soll die Effizienz der Linie maximiert sein.

2.1.3 Beschreibungen (Brief)

- Monitor device count
  - Aktoren: Transportationprovider
  - Type: Primary
  - Beschreibung: Transportationprovider überwacht und loggt die Netzwerkaktivität der User/Geräte im Umfeld seiner Antennen
- Uniquely identify device
  - Stakeholder: Transportationprovider
  - Type: Secondary
  - Description: Der Transportationprovider möchte sicher sein, dass ein Gerät eindeutig einem Besitzer zugeordnet ist, damit dieses nicht mehrfach gezählt wird.
- Locate device
  - Stakeholder: Transportationprovider
  - Type: Secondary

  - Description: Transportationprovider möchte eine ortliche Bestimmung der Geräte relativ zu den Antennenstandorten haben. So werden dann durch zusätzliche Antennen Metadaten Fahrtenprofile ermöglicht.
- Save and query data
  - Aktoren: Transportationprovider
  - Type: Primary
  - Beschreibung: Transportationprovider möchte die generierten Daten persistent speichern und abfragen können.
- Usage profiling
  - Aktoren: Transportationprovider
  - Type: Primary
  - Beschreibung: Transportationprovider möchte Auslastungsprofile der Linien anhand der Daten generieren können.
- Increase efficiency
  - Aktoren: Transportationprovider
  - Type: Primary
  - Beschreibung: Transportationprovider will über die Auslastungsprofile die Effizienz der Linien steigern
- Lower cost
  - Aktoren: Transportationprovider, Transportee
  - Type: Primary
  - Beschreibung: Die Gäste wollen ihre Kosten um von A nach B zu kommen möglichst gering halten und die Betreiber ihre Betriebskosten.
- Connect to device
  - Aktoren: Transportee
  - Type: Secondary
  - Beschreibung: Die Gäste wollen sich mit ihren Peripheriegeräten verbinden können.
- Notification services
  - Aktoren: Transportee
  - Type: Primary
  - Beschreibung: Die Gäste wollen gewisse Services nutzen, welche über Bluetooth implementiert sind. So z.B. die meisten Corona Warn Apps.

2.2 Nicht-Funktionale Anforderungen

2.2.1 Adaptability

Der Anspruch der Arbeit ist es ein adaptierbares Framework zum Fingerprinten via Bluetooth zu schaffen. Es sollen die grundlegenden Ansätze ausgearbeitet werden und eine klare Dokumentation des Prozesses und der benötigten Betriebsmittel festgehalten werden, um die Lösung für jedermann zugänglich und implementierbar zu machen.

2.2.2 Interoperability

Hier geht es um den Austausch verwendeter Schnittstellen bzw. deren Implementation. Nur wenn klar dokumentiert ist welche Schnittstellen unter Verwendung welcher Implementation für die Arbeit gebraucht wurden, kann eine entsprechende Methodik extrapoliert werden um die angewandeten Mittel zu ersetzen.

2.2.3 Flexibility

Die Ansätze beruhen auf dem momentanen Stand der Technik, in diesem Fall dem Bluetooth-Standard. Darum ist die Flexibilität der erarbeiteten Ansätze nur begrenzt gegeben. Neben Konstanten wie beispielsweise die Signalstärke, kommen aber auch viele Faktoren ins Spiel, die auf der Implementation des Standards beruhen. Zum Beispiel das angewendete Protokoll. Darum ist die zukünftige Flexibilität nur beschränkt gegeben.

2.2.4 Performance

Bei der Erstellung des Prototypen liegt das Augenmerk auf der Funktionalität, nicht auf der Performanz. Darum muss, um Performanz zu garantieren, das System in diese Richtung optimiert werden. Denkbar wären hier ein Eventbasiertes System, welches die Datenverarbeitung in die Cloud auslagert.

2.2.5 Durability & Availability

Durch die eingebauten logging Mechanismen und hohe Fehlertoleranz von Linuxbasierten Systemen sollte der Prototyp hochgradig resistent gegenüber Software Bugs oder falschem Input sein. Die Availability ist darum zu gewährleisten. Es wird angenommen, dass der Prototyp an einer unzugänglichen Stelle montiert wird. Dadurch ist ebenfalls das Durchhaltevermögen des Prototypen anzunehmen, ausgenommen zufälliger technischer Probleme.

2.2.6 Privacy

Die private Handhabung von Daten ist dadurch gewährleistet, dass der Prototyp eine direkte Leitung zu dem verarbeitenden System besitzt und keine Daten über einen offenen Port gesendet werden müssen.

2.2.7 Throughput

Bei der Entwicklung des Prototypen könnte es zu Problemen im Durchsatz kommen, da eine grosse Menge an Daten zu erwarten ist. Es ist fraglich ob mit der bestehenden Infrastruktur eine real-time Datenverarbeitung möglich ist.

2.3 Betriebsmittelliste

Dieser Abschnitt dient zum Festhalten der verwendeten Geräte.

Folgend findet sich die Liste der verwendeten Geräte.

2.3.1 Endgeräte

Tabelle 3: Verwendete Endgeräte

Name	Typ
Samsung Galaxy S20+	Mobiltelefon
Samsung Galaxy S8	Mobiltelefon
iPhone 6	Mobiltelefon
iPhone X	Mobiltelefon
iPhone XS Max	Mobiltelefon
iPhone XR	Mobiltelefon
Dell XPS 13	Laptop
Ninebot KickScooter ES1	E-Scooter

2.3.2 Peripheriegeräte

Tabelle 4: Verwendete Peripheriegeräte

Name	Typ
Samsung Galaxy Buds	Kopfhörer
Sony WH-1000XM2	Kopfhörer
Airpods Gen 2	Kopfhörer
Airpods Pro	Kopfhörer
Apple Watch Series 5	Smart Watch
Fossil Explorer Gen. 4	Smart Watch
United Ears 3 BOOM	Lautsprecher

2.3.3 Antennen

- Ubertooth-One
- Nordic nRF development kit

3 Lösungskonzept

3.1 Tooling

Hier soll ein Einblick in die verwendeten Tools gegeben werden. Verschiedene Hardware- und Software-Tools wurden angetestet. Manche dieser Werkzeuge wurden dauerhaft verwendet, andere nach einer Testphase wieder verworfen.

3.1.1 Hardware

Zuerst soll das Augenmerk auf die bestehenden Hardware-Lösungen im Bereich Bluetooth gelegt werden. Ohne zuverlässig ein Signal empfangen zu können, bringen auch die besten Alogrithmen wenig. Dabei wollen wir besonders die verwendeten Lösungen in den Vordergrund rücken, aber auch umfangreichere Tools, die nicht zum Tragen gekommen sind, anschauen.

3.1.1.1 Ubertooth One

Die schlussendlich hauptsächlich zum Einsatz gekommene Bluetooth-Antenne ist der Ubertooth One von Great Scott Gadgets.

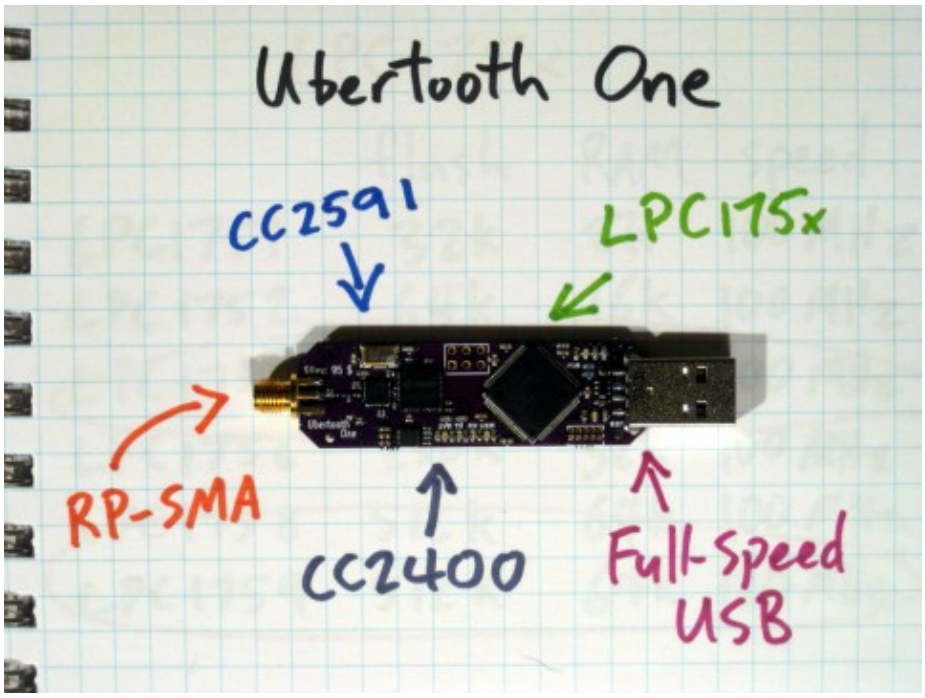

Abbildung 13: Der Ubertooth One mit Komponenten. Quelle: [9]

Das Project Ubertooth ist eine Open Source Kollaboration. Der Ubertooth ist im Handel als fertiges Produkt erhältlich, kann aber auch selbst zusammen-

gelötet werden. Entsprechende Anleitungsschritte und verwendete Komponenten sind hier[3] einsehbar. Die Firmware, auf die der Ubertooth als Software Defined Radio angewiesen ist, ist auf Github[4] veröffentlicht.

Generell ist der Ubertooth One mit Linux-Systemen kompatibel. Er benötigt zur Nutzung die libbtbb, eine in C geschriebene Library zur Decodierung von Bluetooth Packets, und die Ubertooth Host-Tools. Diese müssen selbst kompiliert werden. Danach wird die Firmware auf den Ubertooth geflasht und schon ist dieser bereit verwendet zu werden.

Dank dem integrierten ARM Prozessor ist der Ubertooth relativ kostengünstig und performant. Die Verwendung der Binaries ist elegant gestaltet, die Packet-Daten können direkt in eine Pipe auf dem Betriebssystem geschrieben werden. Diese kann in Wireshark ausgelesen werden. Aktuelle Versionen von Wireshark haben das Bluetooth-Format integriert und können die Bluetooth Pakete anzeigen.

Der Ubertooth kann Senden und Empfangen. Zudem ist in der bestehenden Software einiges an Funktionalität eingebaut. So kann zum Beispiel mit dem Befehl ubertooth−btle −f versucht werden, einer Bluetooth Low Energy Access Address über die Kanäle zu folgen.

Das Senden selbst generierter Pakete war im Zuge dieser Arbeit nicht notwendig.

Limitierend ist beim Ubertooth anzumerken, dass dieser, wie auch die meisten anderen günstigen Lösungen, nur einen Kanal zu einem gegebenen Zeitpunkt analysieren kann. Ebenso muss man sich auf entweder Bluetooth Classic oder Bluetooth Low Energy festlegen.

Um die Diagnose und das Development zu erleichtern, sind ein paar weitere Utilities in den Host-Tools enthalten. Die LEDs geben Aufschluss über den Zustand des Gerätes.

3.1.1.2 Nordic nRF Development Kit

Das nRF52840 Development Kit[5] von Nordic Semiconductor wurde vom ICOM für diese Arbeit zur Verfügung gestellt.

Nordic Semiconductor ist im Bluetooth Marktsegment durch ihre Development Kits, ihre Trainings und vor allem ihre Bluetooth Chips bekannt. Die zugehörige nRF Connect Software ist lizenzfrei beziehbar.

Das Kit ist definitiv eine Lösung die dem Ubertooth in nichts nachsteht, stellenweise sogar besser implementiert ist. Die Daten und deren Visualisierung in Wireshark sind identisch. Wie auch der Ubertooth ermöglicht auch das Development Kit nur das Monitoring auf einem Kanal.

[3] http://ubertooth.sourceforge.net/hardware/one/
[4] https://github.com/greatscottgadgets/ubertooth
[5] https://www.nordicsemi.com/Software-and-tools/Development-Kits/nRF52840-DK

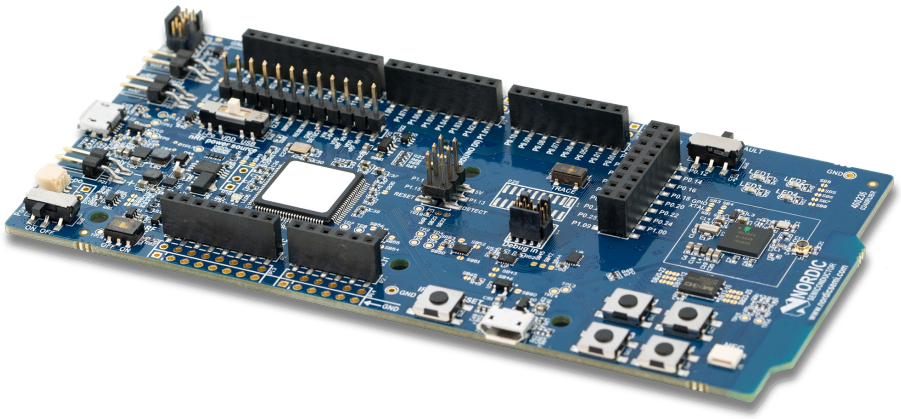

Abbildung 14: Nordic Dev-kit. Quelle [10]

Schlussendlich wurde jedoch der Ubertooth One gewählt, da er bereits mit für unsere Zwecke hilfreichen Tools ausgestattet ist. Mit dem Development Kit hätten diese zuerst selbst implementiert werden müssen.

3.1.1.3 BBC micro:bit

Weiter haben wir aufgrund eines vielversprechenden Github Projektes den BBC Microbit ausprobiert.

Abb. 15 zeigt den micro:bit.

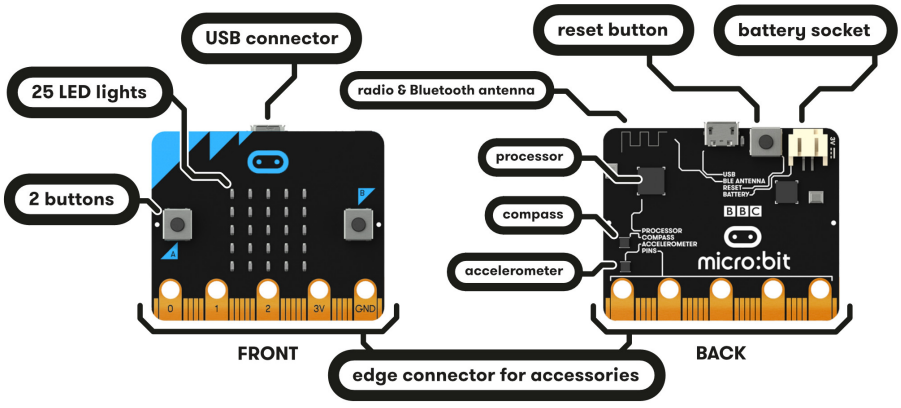

Abbildung 15: Microbit. Quelle: [11]

Eine Übersicht ist auf deren Homepage[6] zu finden.

Der Microbit ist die kostengünstigste Lösung die wir evaluiert haben. Die Antenne und Firmware sind nur für Low Energy ausgelegt und der Prozessor ist nicht besonders leistungsfähig. Hier tritt auch die Limitation auf, dass nur ein Kanal gleichzeitig beobachtet werden kann.

Man muss sich bewusst sein, dass diese Plattform für Lernzwecke konzipiert ist. So kann man mit dem Microbit schnell und einfach experimentieren. Nach ein paar Versuchen musste festgestellt werden, dass die Fähigkeiten des Microbits für die Zwecke der Arbeit nicht ausreichend sind. Vor allem die fehlende Classic Funktionalität und geringe Rechenleistung sind dabei problematisch.

3.1.1.4 Ellisys Bluetooth Analyzer

Die teuerste und funktionsreichste Lösung, die auch angeschaut wurde, ist das Enterprise von Ellisys.

Abb. 16 zeigt den Ellisys Bluetooth Analyzer.

Abbildung 16: BluetoothAnalyzer. Quelle: [12]

Eine genauere Beschreibung des Geräts befindet sich auf der Homepage von Ellisys[7].

Der Bluetooth Analyzer verfügt angeblich über die Möglichkeit, gleichzeitig alle Kanäle und die verschiedenen Protokoll-Implementationen zu capturen.

Damit wäre es das am besten geeignete Gerät um den Standard zu untersuchen. Problematisch sind hierbei jedoch die Kosten des Geräts, diese würden den regulären Rahmen einer solchen Arbeit deutlich sprengen. Weitere Tests mit einem solchem Analysegerät wären am effektivesten.

3.1.2 Software

Nachdem die nutzbaren Hardware Lösungen erarbeitet worden sind, soll nun das Augenmerk auf die Software gelegt werden.

[6] https://www.microbit.org/get-started/user-guide/overview/
[7] https://www.ellisys.com/products/bex400/

3.1.2.1 Visual Studio Code

An erster Stelle soll hier die verwendete IDE erwähnt werden, in der die gesamte Codebase sowie die Dokumentation entstanden ist. Ein weiteres Open Source Projekt, welches aber von einem der Big Player vorangetrieben wird. Der unter Entwicklern populäre Texteditor Visual Studio Code[8].

Abb. 17 zeigt das Programmfenster von Visual Studio Code.

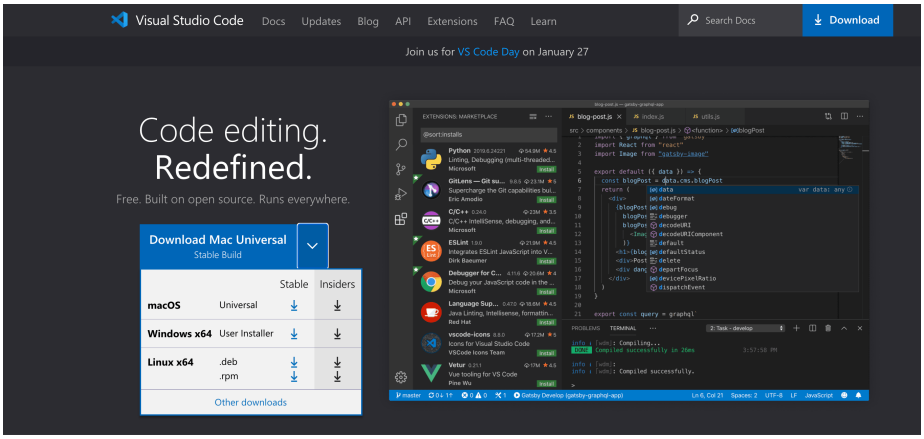

Abbildung 17: Visual Studio Code. Quelle: [13]

Ein von Microsoft lanciertes Programm, das wunderbar erweiterbar und mächtig ist, wenn man es zu benutzen weiss. Wer sich dazu noch auf der Commandozeile auskennt, braucht ausser den Hardware Komponenten nichts mehr um rapide Applikationen zu entwickeln, debuggen und deployen.

3.1.2.2 Wireshark

Wireshark[9] ist eines der bekanntesten und beliebtesten Tools um Pakete zu analysieren. Über Kommandozeilenoptionen des Ubertooth können Pakete direkt in diverse, Wireshark kompatible Formate gespeichert werden. Heutzutage sind in Wireshark grosse Teile des Bluetooth Protokollstacks direkt implementiert.

Abb. 18 zeigt das Programmfenster von Wireshark mit einem Capture.

In Wireshark können verschiedene Operationen mit die Paketdaten ausgeführt werden. So können diese, zur maschinellen Weiterverarbeitung, in andere Formate wie beispielsweise Json, exportiert werden.

Zudem können Pakete direkt in Wireshark gefiltert werden.

[8] https://code.visualstudio.com/
[9] https://www.wireshark.org/

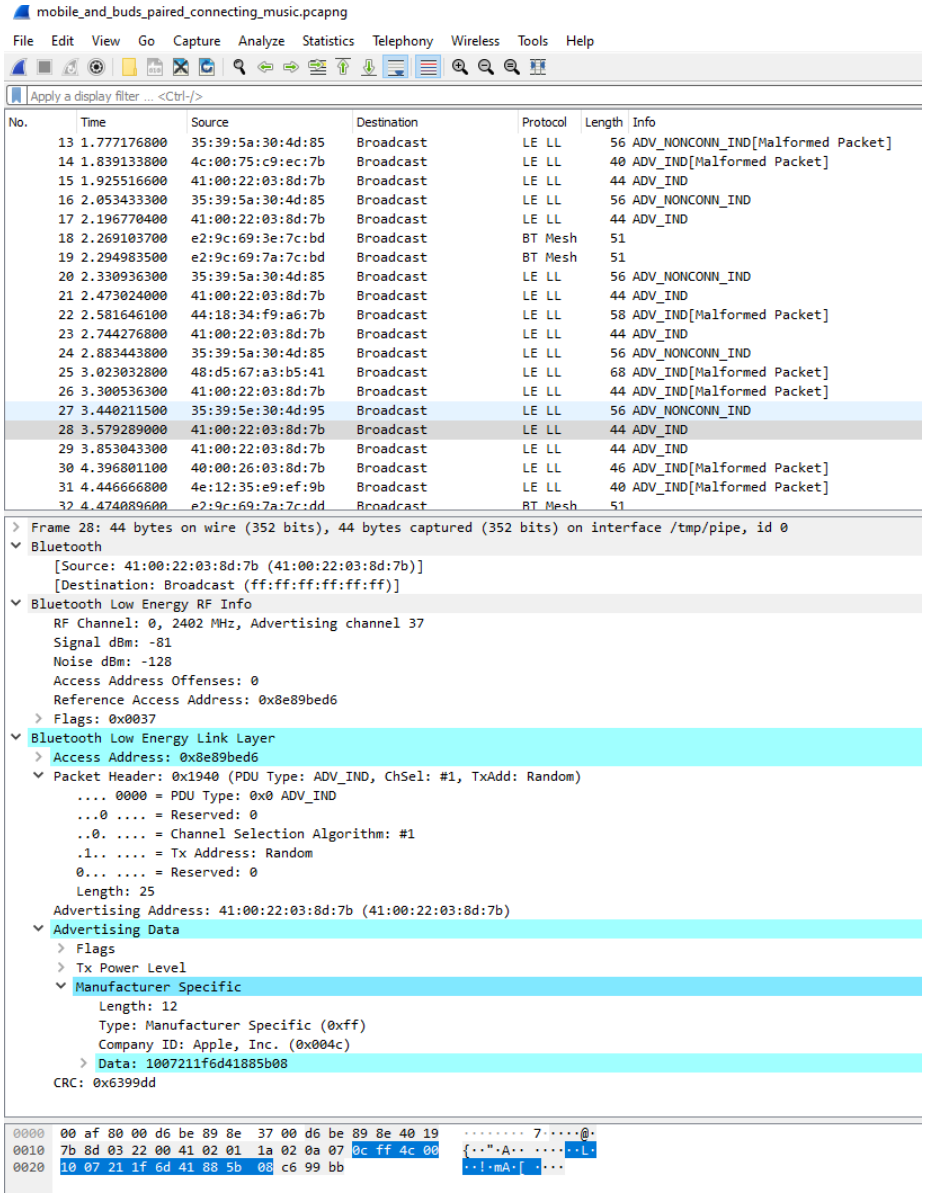

Abbildung 18: Wireshark Programmfester mit Capture

In dieser Arbeit besonders bewährt haben sich folgende Filter:

```
## Spezifische Verbindungen
btle.advertising_address == 50:df:98:ed:c1:61

## Bluetooth Service UUID - hier GAEN (Covid Apps)
btcommon.eir_ad.entry.uuid_16 == 0xfd6f

## Hersteller - hier Apple Geräte
btcommon.eir_ad.entry.company_id == 0x004c
```

3.1.2.3 blue_hydra

Wie der Name vermuten lässt, handelt es sich hier um eine Software aus dem Securitybereich.

Abbildung 19: blue-hydra. Quelle: [14]

Auf das Blue Hydra Projekt sind wir im Zuge der Analyse verschiedener Hacking Tools gestossen. Dabei handelt es sich um ein Open Source Projekt um Bluetooth Geräte zu erkennen. Der Source Code kann auf GitHub[10] gefunden werden.

Es wurde auf der bluez library gebaut, welche den Bluetooth Stack in den meisten Linux Distributionen implementiert. Die Applikation ist in Ruby geschrieben und verwendet Sqlite zur Speicherung einer Datenbank an gesehenen Geräten. Diese Software ist auch zur Verwendung mit dem Ubertooths entwickelt. So können beliebig viele Uberteeth an die blue_hydra Applikation angehängt werden und so das Monitoring durch das Sniffen verschiedener Kanäle verbessert werden.

Es war vor allem in der Ideenfindungsphase ein wertvolles Tool und ist für das BR/EDR Protokoll Tracking von Classic Geräten gut geeignet.

3.1.2.4 Bettercap

Das selbst ernannte Schweizer Taschenmesser für Wifi, Bluetooth und um Wireless-Netzwerke zu erforschen und anzugreifen. Es bietet ein Web UI, mit dem die Bluetooth-Funktionen des Betriebssystems direkt verwendet werden können. So kann es die vorhanden Low Energy Geräte enumerieren.

[10] https://github.com/ZeroChaos-/blue_hydra

Abbildung 20: bettercap. Quelle: [15]

Eine Enumeration der sichtbaren LE Geräte war nur begrenzt nützlich, vor allem ist Bettercap[11] durch die Schnittstelle mit dem OS limitiert. Gewisse Pakete werden von den handelsüblichen, in Mainboards verbauten Bluetooth Empfängern ignoriert.

Alles in allem eine schöne Software, die sich bestimmt mit Anpassungen einsetzen liesse.

3.1.2.5 BtleJack

Der BtleJack stellt an sich den Anspruch, das neue Low Energy Sackmesser zu sein. Hierbei handelt es sich um das Projekt, welches zum Testen des micro:bit geführt hat.

Dieses weitere Open Source Projekt verspricht viele Anwendungen im LE Stack. Die Software kommt mit einer Firmware, welche auf dem micro:bit installiert werden muss.

Man soll mit dem BtleJack existierende und neue Verbindungen sniffen und in ein pcap File exportieren können. Weiter hat es die Fähigkeit, eine bestehende Verbindung zu blockieren oder bei fehlender Sicherung sogar übernehmen zu können.

[11] https://github.com/bettercap/bettercap

Der BtleJack hat sich nach dem Testing als nette Idee herausgestellt. Es ist durchaus möglich, Verbindungen zu detektieren. Das Übernehmen ist allerdings für moderne Mobilgeräte nicht denkbar. Verbindungen effektiv auf Knopfdruck blockieren zu können ist ebenso eher die Ausnahme. Der limitierende Faktor hier ist der schwache Prozessor des micro:bit. Der Source Code, sowie Bedienungsanleitung sind auf ihrer Github Seite[12] zu finden.

3.1.3 Antennenkammer

Die Antennenkammer des ICOM stellt einen faradayschen Käfig dar. Das heisst, dass keine Signale von aussen hinein- oder von innen herrausdringen können. Dies ist nötig um wenig Interferenz und Reflektion beim Ausmessen von Antennenstrahlungsmustern zu gewährleisten.

Abb. 21 zeigt die Antennenmesskammer.

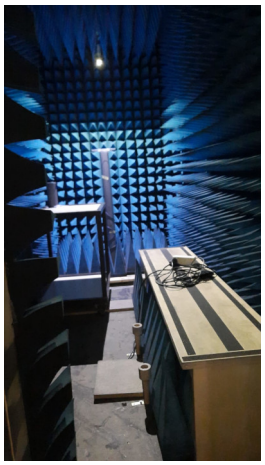

Abbildung 21: Antennenkammer ICOM

Für dieses Projekt war die Antennenkammer nützlich, um die Flut an Signalen die über Bluetooth gesendet werden zu minimieren. Aufgrund des Frequency Hopping Spread Spectrum (FHSS) Multiplexing von Bluetooth werden unglaublich viele kleine Chunks an Daten gesendet. Sinn aus diesen Daten zu machen, ist ohne genau zu wissen wonach man sucht unmöglich.

So wurde nach ersten Versuchen beschlossen, möglichst alle Versuche in der Kammer zu machen um eine saubere, wissenschaftliche und reproduzierbare Arbeitsweise zu gewährleisten.

[12] https://github.com/virtualabs/btlejack

3.2 Domainanalyse

3.2.1 Einführung

Diese Sektion vermittelt das Verständnis der Problemdomäne des Projektteams. Hierin werden die verschiedenen Dokumente, welche zur Lösung der Aufgabenstellung erstellt werden gesammelt und beschrieben.

3.2.2 Domain Modell

Zuerst wollen wir einen groben Überblick über das Problem und die zu erstellenden Software Pakete im Domain Modell geben.

3.2.2.1 Strukturdiagramm

Abb. 22 zeigt das Strukturdiagramm.

3.2.2.2 Wichtige Konzepte

Zur korrekten Analyse der Domain muss man ein grundlegendes Verständnis über die Funkschnittstelle Bluetooth und Netzwerksverbindungen allgemein mitbringen. Bluetooth sendet Pakete, um mit einem anderen Gerät zu kommunizieren. Unter dieser Bandbreite an Paketen zielen wir auf bestimmte Pakete ab. Die BT Classic Pakete, Pakete die eine Verbindung beschreiben und die passiven Advertisements die ab Bluetooth 4.0 versendet werden. Dazu wird eine Antenne mit einem Software Defined Radio eingesetzt. Über diese Antenne empfangen wir einen Kanal und können dort die Pakete abgreifen. Dieser Sniffer besitzt Prozessoren für die verschiedenen Arten von Paketen auf die abgezielt wird. Der Processor formatiert also aus dem Bytestream unser Daten Modell, die sog. Fingerprints, mit zusätzlichen Metainformationen bezüglich der Antenne, aktuellen Zeit und Position. Diese Informationen werden dann gebündelt über HTTP an den einen Application Programming Interface (API)-Controller gesendet und dieser kümmert sich anschliessend um die persistente Speicherung per dediziertem Service. Anschliessend kann man die Rohdaten noch über eine einfache Visualisierung betrachten und auf gewisse Weisen miteinander korrelieren.

3.2.2.2.1 Packet

Das Packet steht representativ für den gesamten möglichen Traffic auf der Bluetooth Schnittstelle. Es beinhaltet Informationen, die Geräte und Funktionen beschreiben. Ein passives Tracking anhand der Rohdaten eines Paketes ist in dieser Arbeit zu untersuchen. Insbesonders die Eignung der Bluetooth Pakete dazu.

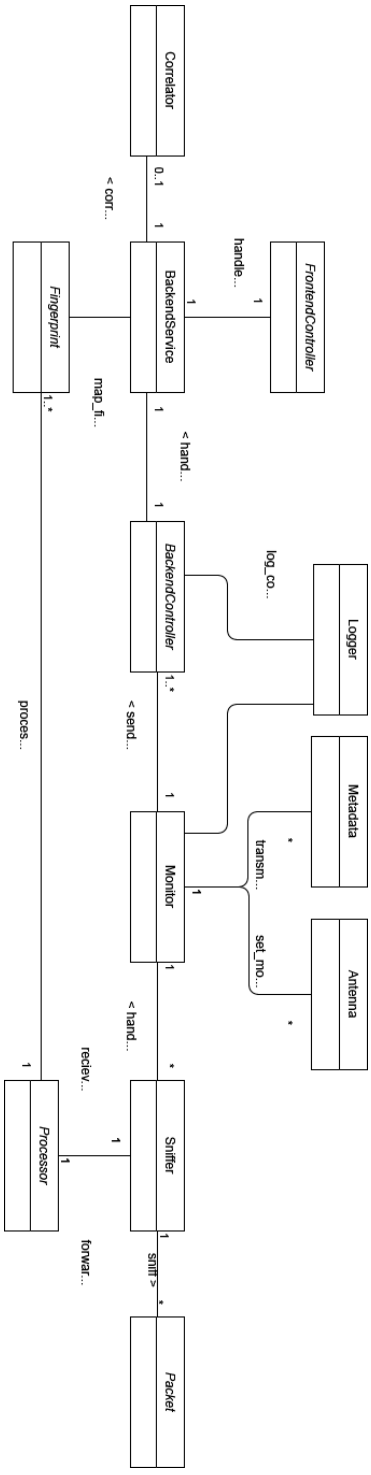

Abbildung 22: Strukturdiagramm

3.2.2.2.2 Fingerprint

Die Fingerprints stellen die zu persistierenden Daten dar. Sie werden vom Sniffer erstellt, sobald dieser detektiert, dass ausreichend Traffic von einem Gerät ausgeht. Bluetooth Classic Geräte sind verhältnismässig einfacher gut zu Fingerprinten als Low Energy Geräte.

3.2.2.2.3 Sniffer

Dieser Service dient dazu, Pakete einzufangen und anschliessend dem korrespondierenden Prozessor zu übergeben. Der Sniffer ist abhängig von der eingesetzten Hardware. Dazu wird in dieser Arbeit der Ubertooth verwendet, aufgrund der dafür bereits bestehenden Software. Der Sniffer ist prinzipiell der Punkt, an dem für die Hardware Antenne ein SDR(Software Defined Radio) erstellt wird.

3.2.2.2.4 Processor

Bei dem Prozessor handelt es sich um einen Service, der den vom Sniffer generierten Bytestream ausliest und zu den vordefinierten Fingerprints prozessiert. Dazu werden in diesem Projekt Linux-Pipes erstellt und ausgelesen.

3.2.2.2.5 Monitor

Der Monitor besteht aus mehreren Threads, die die Client Application überwachen und einen RequestHandler für die Kommunikation über HTTP aufruft.

3.2.2.2.6 Backend Controller

Diese Controller stellen die Endpunkt dar, an die die Fingerprints geschickt werden um anschliessend gespeichert zu werden. Auf dem Controller ist ebenfalls die Validierung der ankommenden Daten implementiert.

3.2.2.2.7 Backend Service

Dieser Service steht bereit um die vom Controller ankommenden Daten zu persistieren. Er wird vom den Backend, als auch von den Frontend Controllern aufgerufen. Der Service implementiert zusätzliche Domain Logik bezüglich der angemessenen Speicherung von einkommenden Daten.

3.2.2.2.8 Frontend Controller

Dient mit Zugriff auf den Service dazu, die Daten zu visualisieren. Dazu können die entsprechenden Endpunkte angesprochen werden.

3.2.2.2.9 Correlator

Diese Klasse dient dazu, die Zuweisung verschiedener Advertisements aufeinander zu machen um verschiedene Geräte zu unterscheiden. Es werden einfache Ansätze, wie eine simple zeitliche und räumliche Zuordnung, sowie weitergehende graphentheoretische Ansätze geprüft.

3.2.2.2.10 Systemoperationen

Die wichtigsten Systemoperationen sind:

- Sniff
- Correlate
- Log
- Save

3.2.2.3 Limitationen

Limitierend ist in Betracht zuziehen, dass Bluetooth Classic (=<3.0) und Bluetooth Low Energy (>=4.0) zwar beide als Bluetooth bezeichnet werden, jedoch unterschiedlich implementiert sind. Darum muss das Domain Model für beide Protokolle implementiert werden. Gezeigt ist also nur das grundsätzliche Verständnis des zu behandelnden Problems. Ebenso sind die Möglichkeiten für OEMs im Rahmen von Bluetooth Low Energy grösser. Darum müssen Anpassungen an proprietäre Protokolle beachtet werden. Ebenso konnte Bluetooth 5.0 aufgrund mangelnder Adoption zum Zeitpunkt der Arbeit nicht untersucht werden.

3.3 Ansätze

Da Bluetooth zwei unterschiedliche, inkompatible Protokolle - namentlich Bluetooth BR/EDR und Bluetooth LE - beinhaltet, müssen auch mindestens zwei Methoden gefunden werden, Devices zu fingerprinten.

Die in Betracht gezogenen Ansätze sind folgend aufgeführt.

3.3.1 Bluetooth BR/EDR

Verbundene Bluetooth BR/EDR-Geräte bilden so genannte Piconets. Darin gibt es einen Master, in unserem Fall ist das jeweils das Mobiltelefon, und Slaves. In diesem Ansatz fingerprinten wir die Master-Geräte der Piconets über den Significant Address Part (SAP) ihrer Bluetooth Adresse (BD_ADDR).

Sobald zwei Geräte verbunden sind, beginnt jedes Bluetooth Packet mit dem Channel Access Code, welcher den LAP des Piconet Masters im Sync Word enthält. Für unseren Fingerprint benötigen wir zusätzlich noch den UAP des Masters. Dieser kann entweder bruteforced werden oder aber berechnet:

Alle Paket Header enthalten einen Header Error Check (HEC), eine 8 Bit Zahl, welche die Integrität des Packets sicherstellen soll. Mit Ausnahme von FHS Pakets im Master Response State wird der HEC vom UAP des Masters abgeleitet. Da der HEC Algorithmus umkehrbar ist, kann der UAP anhand des UAP ermittelt werden. Erschwerend kommt das Data Whitening hinzu. Jedes Paket wird vor dem Versand mit einem Whitening Word, welches anhand 6 Bit der Master Bluetooth Clock berechnet wird, gescrambled um DC Bias (Gleichwerte) zu minimieren. Da nur 6 Byte als Seed für das Whitening Word verwendet werden, gibt es auch nur $2^6 = 64$ Möglichkeiten für die dewhitened Pakets und somit maximal 64 Mögliche UAPs pro Paket (angenommen, das Paket wurde ohne Bitfehler übertragen). Berechnet man die UAP-Kandidaten für mehrere Packets und bildet davon die Schnittmenge, lässt sich der UAP i.d.R. sehr schnell und zuverlässig berechnen.

Das Ubertooth Package beinhaltet bereits eine Utility, die genau dies tut. Mit ubertooth−rx −z wird versucht, den SAP aller Master in Reichweite der Antenne zu ermitteln.

Um zu überprüfen, ob ein identifiziertes Master-Gerät weiterhin anwesend ist, kann entweder geschaut werden, ob weiterhin Packets mit dem entsprechenden LAP empfangen werden, oder es kann mittels einem L2CAP echo Request das Gerät gepingt werden, z.B. l2ping 00:00:S1:DE:B0:0B. Die ersten zwei Bytes sind hierbei irrelevant. Die Round Trip Time eines l2pings befand sich bei Tests jeweils bei etwa 200ms. Dadurch wäre für eine Implementation eine dedizierte Bluetooth-Schnittstelle für die Pings notwendig.

Weitere Informationen und Details zu diesem Ansatz können im Ubertooth Blog

Post 'Discovering the Bluetooth UAP'[13] [4] gefunden werden.
blue_hydra[14] ist ein Projekt, welches ein entsprechendes Tracking mit dem Ubertooth bereits implementiert hat.

Für die Implementation wurden die vorhandenen Ubertooth-Tools verwendet und gegebenenfalls der Sourcecode von ubertooth-rx angepasst.

Weitere Informationen zu Bluetooth BR/EDR befinden sich in Sektion 1.3.1.3.

3.3.1.1 Einschränkungen

Damit Geräte gefunden und erkannt werden können, müssen diese entweder im Inquiry State sein oder aktiv Daten übertragen.

Da wir den NAP nur in Ausnahmefällen, und deshalb nicht die gesamte EUI-48 kennen, kann es sein, dass zwei Geräte den gleichen Fingerprint erhalten. Da wir aber trotzdem noch mindestens 24 Bit für den Fingerprint verwenden, sind 2^32, also ca. 16.8 Millionen, LAPs möglich. Eine Kollision ist deshalb unwahrscheinlich und kann einfach durch Beizug der Positionen der Geräte aufgelöst werden.

Mit den momentan vorhandenen Geräten sollte dieser Ansatz recht gut funktionieren, da die Mehrheit der Bluetooth Kopfhörer den Audiodaten über Bluetooth Classic senden. Mit dem aufkommenden LE Audio wird die Zahl der Geräte die Bluetooth Classic aktiv verwenden jedoch in Zukunft stark abnehmen.

3.3.1.2 Urteil

Dieser Ansatz ist zielführend und im Prototypen in leicht abgeänderter Form implementiert:

Da der UAP nicht zuverlässig bei allen Geräten bestimmt werden kann, wird nur der LAP verwendet. Der UAP wird nur als zusätzliche Information ausgezeichnet. Die 24 Bit des LAP, zusammen mit der Antenne auf welcher das Packet empfangen wurde, sind ausreichend um Kollisionen auszuschliessen. Wird ein LAP während 5 Minuten nicht gesehen, so wird beim nächsten Auftreten dessen ein neuer Fingerprint angelegt. Es werden ausserdem nur LAPs an das Backend reportet, wenn diese während mindestens 60 Sekunden auf der selben Antenne gesehen wurden.

Ein gefundenes Gerät wird nicht angepingt, da dafür eine weitere Bluetooth-Schnittstelle benötigt würde.

Für ein Tracking kann dann verwendet werden:

- Der LAP
- Der UAP, falls vorhanden
- Der Zeitpunkt der ersten Sichtung (pro Antenne)

[13] http://ubertooth.blogspot.com/2014/06/discovering-bluetooth-uap.html
[14] https://github.com/pwnieexpress/blue_hydra

- Der Zeitpunkt der letzten Sichtung (pro Antenne)
- Die aufzeichnende Antenne
- Die Position der aufzeichnenden Antenne

3.3.2 Bluetooth LE

3.3.2.1 Access Address

Da BLE Privacy Features wie MAC Address Randomization unterstützt, können wir einen Ansatz wie bei Bluetooth Classic nicht verwenden und Geräte können nicht direkt durch ihre MAC Adresse identifiziert werden.

Stattdessen werden in diesem Ansatz Verbindungen durch ihre Access Adressen gefingerprintet.

Seite 2866 der Bluetooth Core Spezifikation 5.2 [5] spezifiziert:

> It is intended that each Link Layer connection between any two devices, each BIS, and each periodic advertising train has a different Access Address.

und

> The Link Layer shall generate a new Access Address each time that it enables a periodic advertising train
> The Link Layer shall generate a new Access Address for each Connected Isochronous Stream.

Es könnte möglich sein, die Access Address der ACL Verbindung oder des CIS Streams zu verfolgen.

Weitere Tests und mehr LE-Datenverkehr sind notwendig um diesen Ansatz zu verifizieren.

3.3.2.1.1 Einschränkungen

Damit Geräte gefunden werden müssen sie mit einem anderen Gerät verbunden sein und Daten müssen über den entsprechenden Channel gesendet werden.

3.3.2.1.2 Urteil

Für diesen Ansatz waren nicht genug langlebige Verbindungen vorhanden. Da nur vereinzelte Verbindungen gefunden wurden und eine Zuweisung eines Fingerprints dieser Art auf einen Fingerprint von beispielsweise Bluetooth BR/EDR nicht möglich ist, wurde diese Idee verworfen.

Im Prototypen ist er noch implementiert. Sämtliche Access Addresses die während mindestens 60 Sekunden auf der selben Antenne gesehen werden, werden ans Backend rapportiert.

Sobald Geräte mit Bluetooth LE-Audio verfügbar und verbreitet sind, sollte dieser Ansatz neu überprüft werden. Es sollte dann ausreichend Datenverbindungen existieren um Geräte zuverlässig erkennen zu können.

3.3.2.2 GATT Profiling

Die Studie Fingerprinting Bluetooth-Low-Energy Devices Based on the Generic Attribute Profile[15][16] hat gezeigt, dass BLE Geräte teilweise Anhand ihrer Generic Attribute Profile (GATT) fingerprinted werden können.
Wir würden einen entsprechenden Ansatz selbst implementieren.

Es wird auf den drei dedizierten Advertisement Channels gehorcht. Sobald ein Advertisement eintrifft, wird versucht eine Verbindung mit dem Zielgerät herzustellen und die Attribute des GATT Profils zu eruieren.

3.3.2.2.1 Einschränkungen

Das Zielgerät muss discoverable sein. Es ist uns derzeit nicht bekannt, bei welchen Geräten dies per Default der Fall ist. Die einleitend erwähnte Studie konnte bei iPhones nur ca 5.5% der Geräte sinnvoll fingerprinten. Der Ansatz müsste mit einer weiteren Metrik kombiniert werden.

3.3.2.2.2 Urteil

Da keine offensichtlichen Mängel an der referenzierten Studie gefunden werden konnten, wurde entschieden, den Ergebnissen zu vertrauen. Aufgrund des beschränkten Nutzens der Resultate und der präsenz von vielversprechenderen Ansätzen wurde diese Idee nicht weiter verfolgt.

3.3.2.3 Advertisements

In diesem Ansatz fingerprinten wir Bluetooth LE Advertisements.

Um mit nicht verbundenen Geräten zu kommunizieren, werden bei Bluetooth Advertisements verwendet. Bei Bluetooth LE werden für diese die Advertising Channels 37, 38 und 39 verwendet. Um alle Advertising Channels zu monitoren, müssten drei Bluetooth-Schnittstellen verwendet werden.
In den Advertisements enhalten ist dabei immer eine Advertising Address. Diese ist bei Mobiltelefonen randomisiert und ändert sich periodisch.

Wir limitieren die inkludierten Advertising Addresses auf diejenigen, die länger als eine Minute gesehen wurden.

In Messungen hat sich ergeben, dass die Adressänderung bei den langlebigen Advertising Addresses ca. alle 10-15 Minuten geschieht. Werden die gesehenen

[15] https://hal.inria.fr/hal-02359914/document

Advertising Addresses aufgezeichnet, so kann geschaut werden, ob nach verschwinden einer Advertising Address eine neue innerhalb weniger Sekunden aufgetaucht ist. Die Zuverlässigkeit dieses Ansatzes hängt dabei stark von der Anzahl der präsenten Geräte und dem Adressänderungsintervall ab. Dazu einige Kalkulationen:

Den Zeitraum, in welchem eine neue Advertising Address auftauchen muss, um als Nachfolger einer verschwundenen Advertising Address betrachtet zu werden, beschränken wir auf 5 Sekunden. Den Adressänderungsintervall setzen wir auf 15 Minuten fest, wie diese bei den Apple Advertisements der Fall ist (siehe Sektion 3.3.2.3.1). In den 15 Minuten zwischen Adresswechseln ergibt dies dann

$$15 * 60/5 = 180$$

distinkte Timeslots für Adresswechsel.

Die Wahrscheinlichkeit, dass n Geräte alle in unterschiedlichen Timeslots die Adresse wechseln, ergibt sich dann aus

$$\begin{aligned}\overline{p}(n) &= \frac{180}{180} * \frac{179}{180} * \frac{178}{180} ... \\ &= \frac{180 * 179 * 178 * ... * (180 - n + 1)}{180^n} \\ &= \frac{180!}{180^n * (180 - n)!}\end{aligned}$$

und die Wahrscheinlichkeit, dass zwei Geräte die Adresse im gleichen Zeitslot wechseln und damit ein Nachfolger nicht eindeutig festgestellt werden kann aus

$$p(n) = 1 - \overline{p}(n)$$

Die resultierenden Wahrscheinlichkeiten sind in Tab. 5 zu sehen. Wie darin ersichtlich ist, ist ab 17 Geräten eine Kollision zu erwarten.Anders ausgedrückt, können wir mit nur der zeitlichen Korrelation 16 Geräte unterscheiden.

Tabelle 5: Wahrscheinlichkeit, dass Geräte im selben Time Slot die Adresse wechseln bei Adresswechsel alle 15 Minuten

Geräte	Wahrscheinlichkeit für Kollision
2	0.556%
3	1.66%
4	3.299%
5	5.448%
6	8.075%
7	11.139%

Geräte	Wahrscheinlichkeit für Kollision
8	14.595%
9	18.39%
10	22.471%
11	26.778%
12	31.253%
13	35.836%
14	40.47%
15	45.1%
16	49.675%
17	54.148%
18	58.479%
19	62.631%
20	66.575%
21	70.289%
22	73.756%
23	76.963%
24	79.907%

3.3.2.3.1 Apple

Neuere iPhones senden mehrmals pro Sekunde Advertisements aus. Die Advertising Address ändert sich dabei ca. alle 15 Minuten. Es ergeben sich also die Kollisionswahrscheinlichkeiten aus Tab. 5.

Mehr Details zu den Apple Advertisements befinden sich in Sektion 1.3.2.2.

3.3.2.3.2 SwissCovid App

Die SwissCovid App basiert auf Google & Apple Exposure Notifications (GAEN) und benutzt LE-Advertisements, um Kontakte nachzuvollziehen. Sie verwendet dabei in Advertisements die 16-bit Service Class UUID 0xfd6f.

Die SwissCovid App und die Advertisements davon sind in Sektion 1.3.2.1 beschrieben. Da sich die Advertising Addresses alle 10 Minuten ändern, können wir weniger Geräte tracken als bei anderen Advertisements.

Setzen wir 120 Timeslots in obige Formel ein, erhalten wir die Wahrscheinlichkeiten in Tab. 6. Anhand der Advertising Addresses der SwissCovid App können wir also 13 Geräte unterscheiden.

Tabelle 6: Wahrscheinlichkeit, dass Geräte im selben Time Slot die Adresse wechseln bei Adresswechsel alle 10 Minuten

Geräte	Wahrscheinlichkeit für Kollision
2	0.833%

Geräte	Wahrscheinlichkeit für Kollision
3	2.486%
4	4.924%
5	8.093%
6	11.923%
7	16.326%
8	21.207%
9	26.46%
10	31.976%
11	37.644%
12	43.36%
13	49.024%
14	54.547%
15	59.85%
16	64.868%
17	69.553%
18	73.866%
19	77.786%
20	81.303%
21	84.419%
22	87.146%
23	89.503%
24	91.515%

3.3.2.3.3 Weitere Advertisements

Geräte senden neben den aufgeführten SwissCovid App- und Apple-Advertisements auch eine vielzahl weiterer Advertisements aus. Diese konnten nicht alle einzeln untersucht werden. Sie wechseln auch nicht zwingend alle 10-15 Minuten die Adresse, sondern können dies auch deutlich häufiger oder seltener tun. Daraus resultiert, dass mehr Geräte angezeigt werden als vorhanden. Für das beste Ergebnis beim Fingerprinting und Tracking sollte man sich auf SwissCovid- oder Apple-Advertisements beschränken und eventuell weitere häufig vorkommende Advertisements speziell erfassen.

Isolierte Messungen mit Mobiltelefonen und Peripheriegeräten haben gezeigt, dass ca. $\frac{2}{3}$ der gesehenen Advertisements von SwissCovid Apps oder Apple-Geräten stammen.

3.3.2.3.4 Einschränkungen

Android-Geräte und ältere iPhones sind mit dieser Methode momentan nicht detektierbar, sofern darauf keine entsprechende App, wie beispielsweise die SwissCovid App, installiert ist.

Ab 14 Geräten mit SwissCovid Apps bzw. 17 Apple Geräten ist zu erwarten, dass nicht mehr alle Adresswechsel anhand der zeitlichen Korrelation zuverlässig erkannt werden können.

Neue iPhones werden doppelt erfasst, wenn wie auch die SwissCovid App installiert haben.

Es werden unter auch Peripheriegeräte wie Kopfhörer oder Smart Watches erfasst. Zudem können Geräte, abhängig von den installieren Apps, mehrfach erfasst werden.

3.3.2.3.5 Urteil

Dieser Ansatz ist zielführend und im Prototypen inklusive Verfolgung über mehrere Antennen implementiert.

Für ein Tracking kann dann verwendet werden:

- Die Advertising Address
- Die 16-bit Service Class UUID
- Die Company ID
- Der Zeitpunkt der ersten Sichtung (pro Antenne)
- Der Zeitpunkt der letzten Sichtung (pro Antenne)
- Die aufzeichnende Antenne
- Der RSSI
- Die Position der aufzeichnenden Antenne

4 Umsetzung

4.1 Softwarearchitektur

4.1.1 Einführung

In diesem Abschnitt soll Aufschluss über die eingesetzte Software zum Bluetooth-Fingerprinting gegeben werden.

Die Software ist so wie sie in diesem Dokument beschrieben ist vorhanden, eingesetzt und geprüft. Ein Verständnis für das Systems sollte mit diesem Dokument und den zusätzlichen Beschreibungen in der Arbeit ausreichen, um dieses Projekt weiterzuentwickeln.

4.1.2 Systemübersicht

Einleitend wird in Abb. 23 das Packagediagramm der Gesamtarchitektur und deren Abhängigkeiten präsentiert.

4.1.3 Architektonische Ziele & Einschränkungen

Es sollen weiter die Hauptarchitekturkriterien beschrieben werden und wie diese erfüllt wurden.

4.1.3.1 Effiziente Datenspeicherung

Die Daten werden relational in einer Datenbank gespeichert. Das ist ein wichtiges Kriterium, da eine riesige Menge an Paketen jede Sekunde gesendet wird. Eine Speicherung des gesamten Objekts beziehungsweise Pakets würde zu einer grossen Menge unnötiger Daten führen. Deshalb ist ein Datenmodell konzipiert worden, welches nur die Felder, welche signifikante Information enthalten, aus den Paketen abspeichert. Die Relationen werden über die Empfangsgeräte getätigt. Da sich das Projekt noch in der Development Phase befindet, wird hierzu eine einfache Sqlite Datenbank verwendet. Der bestehende Code kann allerdings durch eine einfache Anpassung im Service auf eine Produktionsdatenbanklösung umgeschaltet werden.

4.1.3.2 Performantes & Erweiterbares Backend

Eine komplett asynchrone Architektur des Backends, welches die ganzen Daten verarbeiten muss, spielt ebenso in den Punkt Performanz hinein. Das Backend ist durch ASP.NET API-Controller umgesetzt. Da es sich bei C# um eine kompilierte Sprache handelt, ist die Exekution des Codes hier sehr schnell. Ausserdem wurde der gesamte Backend-Teil des Projekts als asynchrone Tasks modelliert. Diese schöpfen die Compute Resourcen der Infrastruktur bestmöglich aus.

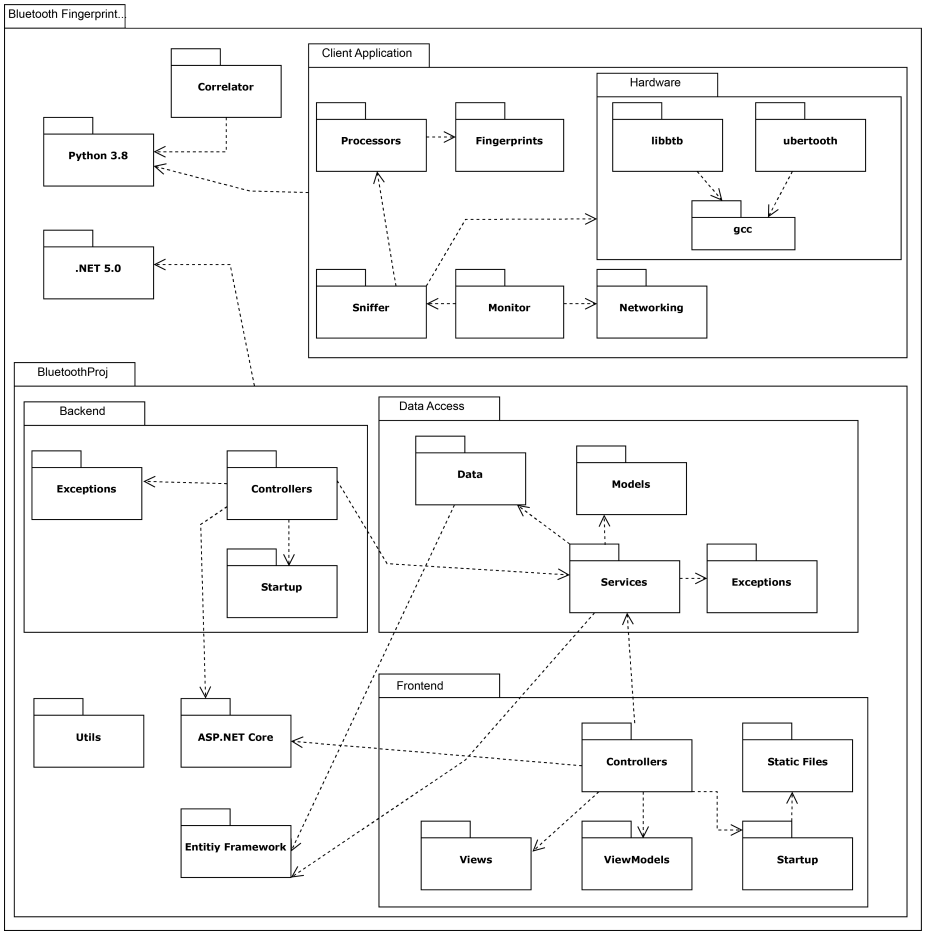

Abbildung 23: Paketübersicht, mit den gesamten Abhängigkeiten

Ebenso ist es möglich, das Backend einfach für weitere Bluetooth Standards zu erweitern. Dazu müssen nur zusätzliche Controller geschrieben werden.

Diese Erweiterungen wären auch sehr einfach zu testen, aufgrund der OpenAPI-Implementation durch die Controller. Diese stellen im Development Build direkt eine Swagger [17] Dokumentation der API bereit.

4.1.3.3 Entkopplung

Die Komponenten werden alle einzeln entwickelt und schlussendlich über die Rest Schnittstelle zusammengeführt. Somit kann jedes Subsystem seine eigenen Erweiterungen bekommen, ohne die Operabilität des Gesamtsystems zu stören. Ausserdem hat das den Vorteil, dass nur die Clients ortsgebunden sind. Solange Internet-Routing vorhanden ist, können die Daten transferiert werden.

4.1.3.4 Rapid Development

Ein wichtiges Ziel war es, dass ein Prototyp schnell ausgereift werden kann. Dazu wurden weit verbreitete Technologien eingesetzt. Python [18] als Programmiersprache der Client Software ist bekannt für seine rapiden Entwicklungszyklen. Dadurch, dass eine entsprechende C-Schnittstelle bereits durch den Ubertooth gegeben war, war es nur noch eine Frage der Erweiterung und Integration dieser in die Software.

C# löst heutzutage Java als kompilierte Backend Sprache weitestgehend ab, vorallem auch wegen des guten Supports und der umfangreichen Dokumentation. Die eingesetzten C# Frameworks existieren in entsprechender Form schon lange und werden oft zum schreiben von APIs eingesetzt.

4.1.4 Logische Architektur

Abb. 24 zeigt die Aufteilung des Systems auf die verschiedenen Tiers der Applikation.

Dabei wird von einem Client der Daten an den Application Server sendet asugegangen. Der Applications Server hört auf eingehende HTTP Methoden an definierten Endpunkten und reagiert entsprechend. Die Backend Controller validieren die Requests bevor Sie diese in SQL umwandeln und an die Datenbank weitersenden.

So wird die Struktur des Projekts gut ersichtlich mit den States, welche das System durchläuft. Zudem lässt sich auf die Kopplung des Systems schliessen. So ist sichtbar, welche Software Packete eigenständig agieren. Signifikant ist vorallem die Entkopplung der Datenbank auf einen Vendor Lock-In. Um die Datenbank zu ändern benötigt es lediglich einen SQL-Connector der Datenbank in Entity Framework.

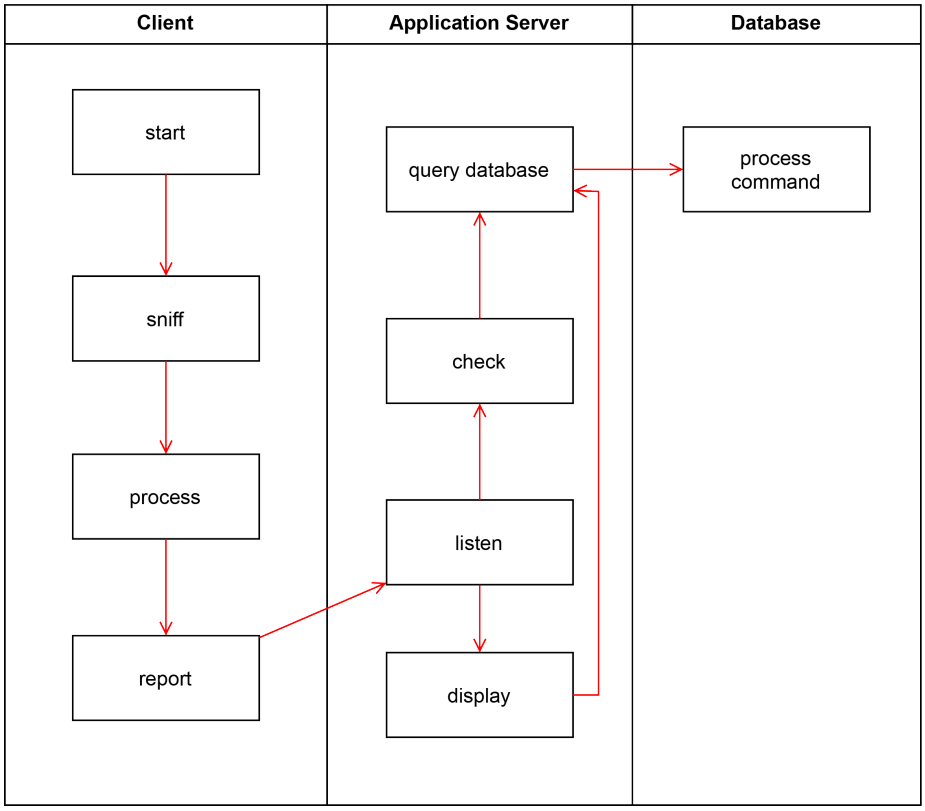

Abbildung 24: Logische Aufteilung des Systems und dessen Zustände

4.1.4.1 Client Application

Hierbei handelt es sich um die Applikation, welche auf dem Client Gerät bzw. Router laufen soll.

Es handelt sich hierbei um ein Linux Betriebssystem (getestet mit Ubuntu 20.04 LTS) mit einer installierten Python Laufzeit, wessen Versionierung um mit dem Code kompatibel zu sein mindestens >= 3.8 sein muss und der C-Toolchain um die benötigten Binaries und Libraries für den Ubertooth kompilieren und installieren zu können.

4.1.4.2 BluetoothProj

Das ist eine C# Solution welche Serverseitig läuft. An diese Applikation sendet der Client seine eingefangenen Fingerprints. Prinzipiell besteht die Solution aus drei Teilen, welche alle auf dem neusten dotnet 5.0 SDK [19] gebaut wurden. Die startbaren Frontend und Backend Applikationen und der Data Access Lib, welche beide Projekte benötigen.

Die Applikationen basieren auf dem ASP.NET Framework [20] um die Controller und Services per Dependency Injection zu mappen.

Die Library verwendet das Entity Framework um die Datenbank zu manipulieren.

4.1.4.2.1 Klassenstruktur BluetoothProj

Data Access

Abb. 25 zeigt die Struktur der Daten und den programmatischen Zugriff darauf.

Die Entitätsklassen werden vom Entity Framework zu einem Datenbankschema übersetzt.

Backend Server

Abb. 26 zeigt das Klassendiagramm des Backend-Projekts.

Hier ist wichtig zu beachten, dass die Datenbank erstellt wird, falls sie nicht vorhanden ist. Das Backend verwendet ASP.NET 5.0 als Framework sowohl für die Dependency Injection als auch für die Controller.

Frontend Server

Abb. 27 zeigt das Klassendiagramm des Frontend-Projekts. Darin nicht beinhaltet sind die statischen JavaScript Dateien, welche ebenfalls von diesem Server geservt werden.

Unter diesen Dateien befindet sich vorallem die Verbindung zu der Googlemaps API. Der entsprechende MapController liefert das HTML und bindet die gmaps.js Datei ein. Das Frontend verwendet ASP.NET 5.0 als Framework

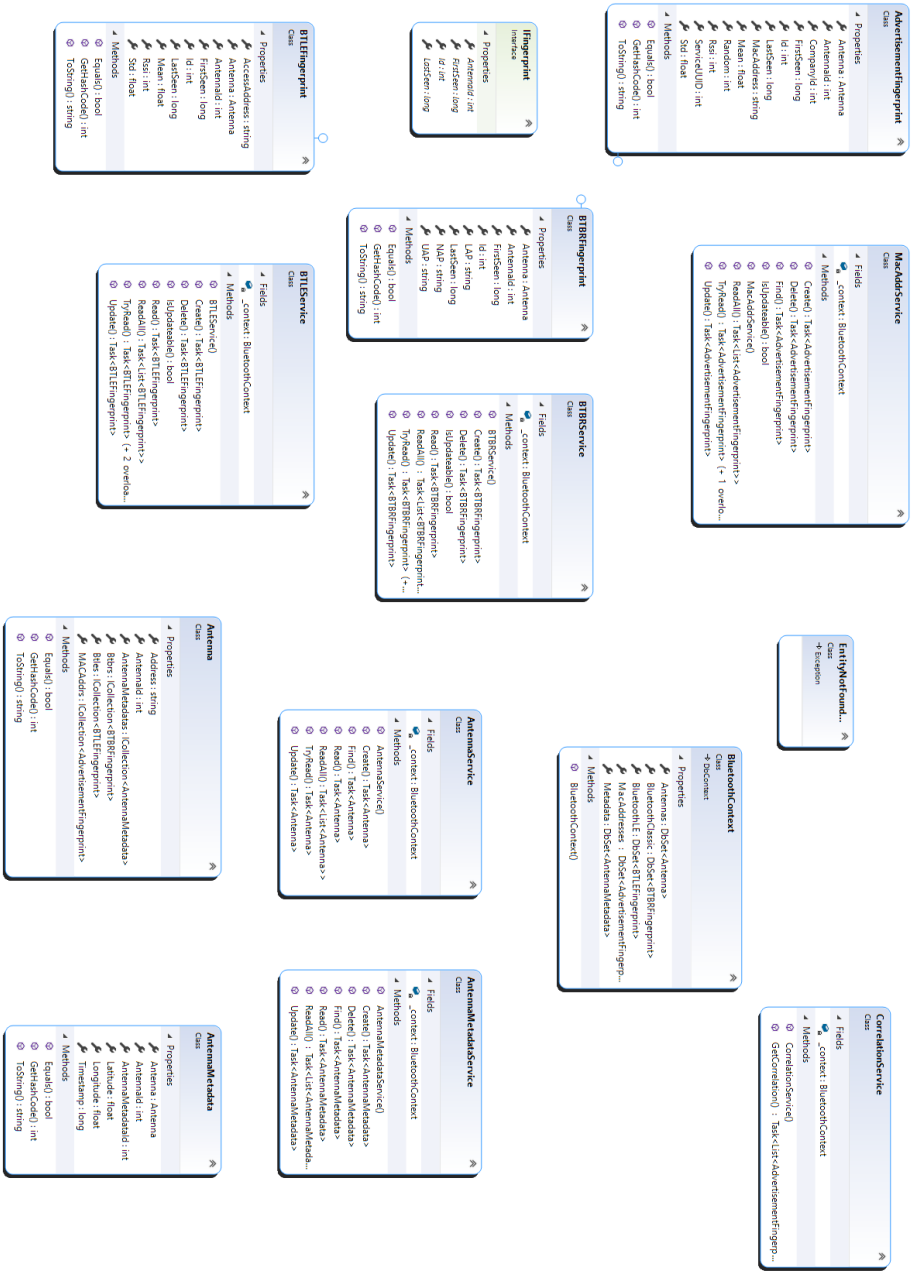

Abbildung 25: Das Klassendiagramm des DataAccess Projekts

Abbildung 26: Das Klassendiagramm des Backend-Projekts

Abbildung 27: Das Klassendiagramm des Frontend-Projekts

für die Dependency Injection als auch für die MVC-Controller und um die statischen Files HTML, JS & CSS Files zur Verfügung zu stellen.

4.1.4.2.2 Schnittstellen

Funk 2,4GHz

Wie bereits erwähnt ist die Schnittstelle, die von dem System gesnifft wird um Nutzdaten zu erzeugen, die Funkschnittstelle im 2,4GHz Bereich basierend auf der von der Bluetooth SIG herausgegebenen Spezifikation. Aktuell ist Bluetooth Classic und Low Energy implementiert. Falls es in Zukunft weitere Spezifikationen der Schnittstelle geben sollte, müsste der Code des Ubertooth angepasst werden, damit das erstellte SDR die neuen Pakettypen erkennen könnte.

Rest

Die Rest Schnittstelle [21] wird zur Kommunikation zwischen Client und Server verwendet. Eine aktuelle Spezifikation der Schnittstelle lässt sich beim Ausführen der Development Version des Projekts erkennen. Dazu wird OpenAPI verwendet und automatisch generiert.

Rest bietet sich aufgrund des ASP.NET Frameworks und seiner einfachen Implementation und guten Performanz an. Alle am Server ankommenden Requests werden asynchron verarbeitet und dann ebenfalls asynchron an die Datenbank weitergeleitet.

4.1.4.2.3 Wichtige interne Abläufe

Client

Der Client wählt sich beim starten der clientseitigen Applikation zuerst beim Server mit der/den Mac-Addresse/n der Antenne/n ein.

Server

Am wichtigsten für den Server ist, dass der Startup problemlos ablaufen kann. Das heisst, dass die Dependency Injection, die bei ASP.NET eingesetzt wird, ohne Fehler abläuft. Darin werden die Einstellungen per Umgebungsvariablen oder die in den appsettings.json festgehaltenen Werte von der Applikation gelesen und der Server und dessen Verbindungen entsprechend konfiguriert. So ist vor allem wichtig, dass der Port, die Routen Endpunkte und die Datenbankverbindung korrekt erstellt sind.

4.1.4.3 Wichtige Abläufe

In diesem Abschnitt soll ein Walktrough der quintessentiellen Abläufe des Systems gegeben werden.

4.1.4.3.1 Client Beispiel

In Sektion 4.2.1.2.3 wird der gesamte Programmfluss des Clients vom Starten bis zum Beenden erläutert.

4.1.4.3.2 Server Beispiel

Hier wird der Programmfluss des Servers bei erhalten eines Fingerprints abgebildet.

Abb. 28 zeigt den Programmfluss des Servers beim Erhalt eines Fingerprints.

Der Controller stellt einen Endpunkt für Post-Requests bereit. Dieser wird beim Starten auf den Server gemappt und steht als Singleton Listener bereit um Requests programmatisch aufzulösen.

Bei einem einkommenden Request bezüglich eines Fingerprints wird zuerst am Request Body validiert, ob ein korrekter Fingerprint vorhanden ist. Anschliessend kommt die Domain Logik ins Spiel und entscheidet, wie die Daten behandelt werden sollen.

Kommt ein neuer Fingerprint, mit einer bisher ungesehener MAC Adresse, an wird dieser direkt abgespeichert. Kennt das Backend den Fingerprint bereits, so muss entschieden werden, ob dieser als neu zu behandeln ist oder ob der existierende Eintrag geupdated werden soll.

Ist die zeitliche Distanz zwischen dem alten Eintrag und dem neuen gross genug, so wird der Eintrag neu angelegt. Ist das nicht der Fall, wird geupdated.

4.1.5 Prozesse und Threads

4.1.5.1 Client Prozesse

Die Client Software ist auf Python gebaut. Aus der Sprache resultieren gewisse Einschränkungen. So kann man nicht einfach die verschiedenen Funktionsaufrufe als Tasks oder der ähnlichem deklarieren.

Deshalb werden in der Client Applikation geschickt Prozesse geforkt und synchronisiert, damit keine Fehler durch das Threading entstehen. So bestehen die Sniffer, Prozessoren & der RequestHandler aus ihren eigenen Threads. Zusätzlich gibt es noch einen Watcher-Thread für Logging und Abbruch.

Da die Sniffer-, bzw. Prozessoren-Threads jeweils ihre eigene Pipe auslesen und nur max. 1 Subprozess erstellt wird, ist deren nebenläufiger Zugriff gesichert. Der RequestHandler besitzt eine Queue die er nebenläufig abarbeitet. Die Queue ist Threadsicher.

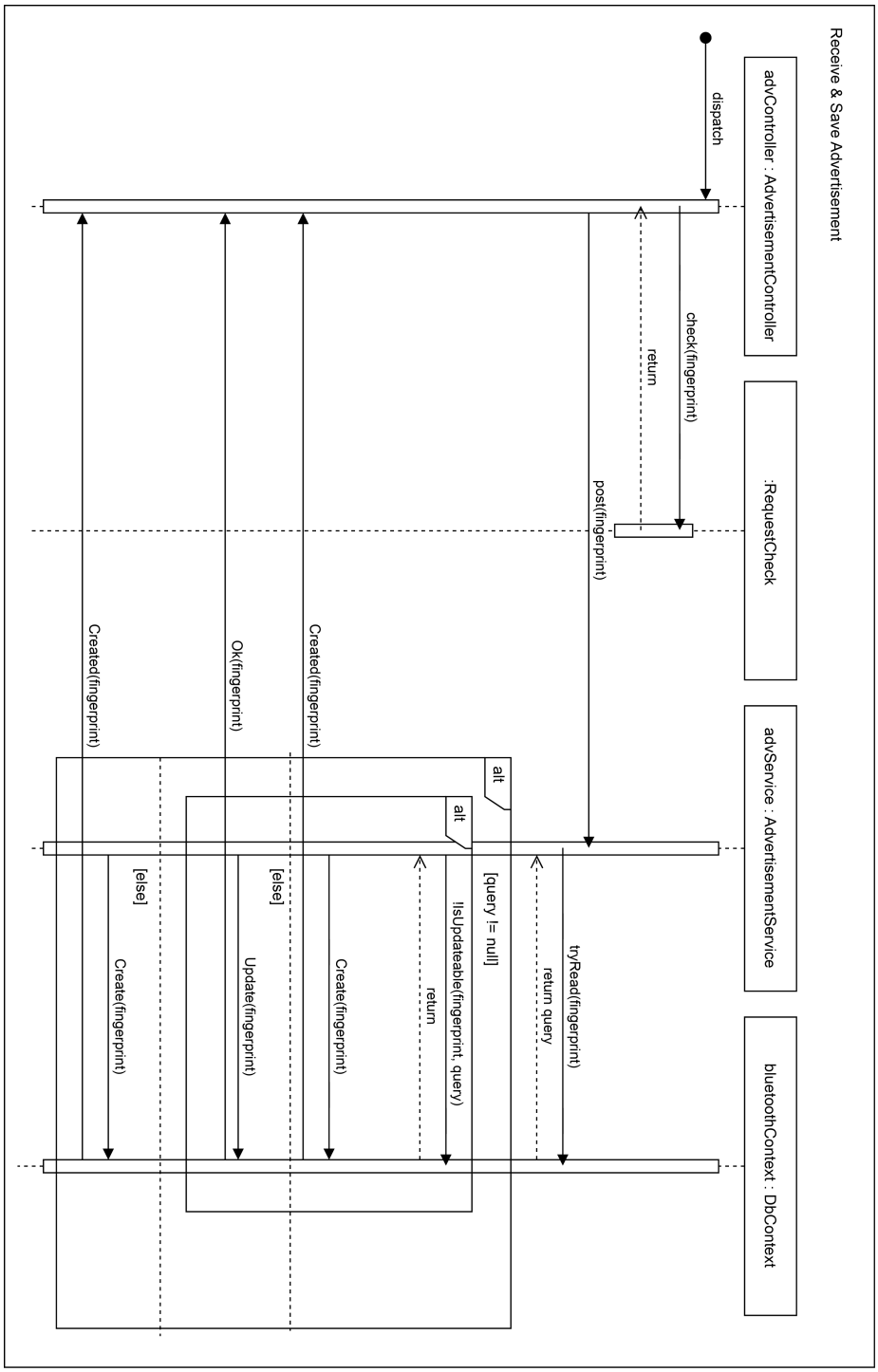

Abbildung 28: Der Programmfluss des Backends bei Erhalt eines Post-Requests am Advertisements Endpunkt.

4.1.5.2 Server Threads

Beim Server ist gewährleistet, dass keine Race Conditions oder Data Races auftreten. Dies wird durch das in C# implementierte async/await Model kontrolliert.

Alle Aufrufe an die Kontroller oder die Datenbank sind als asynchrone Tasks modelliert. Jegliche Art von direktem Aufruf ist eine statische Methode.

Zudem wirft das ADO.NET Framework, welches den Datenbankzugriff kontrolliert, bei Concurrency Problemen Exceptions und rollt die Transactions automatisch zurück.

4.1.6 Deployment

Abb. 29 zeigt, wie das Deployment der Applikationen auf die Router und in die Cloud aussehen.

Angedacht ist, dass die Router, auf welchen die Clientsoftware laufen sollen, eine x86 oder ARM Prozessor Architektur bestizen, mit einem Linux Betriebssystem. Weiter ist die C Toolchain für die Kompilation der Ubertooth-Tools und der Firmware nötig. Dies wird aber nicht zwingend benötigt, sofern immer die gleiche Architektur zum Tragen kommt. Die Router müssen zwingend bei der Initialisierung mit dem Internet verbunden sein.

Das übrige System bietet sich an für ein Deployment in einer Cloud. So hat man eine hohe Skalierbarkeit und kann viele Sicherheitsmechanismen einarbeiten. So z.B. einen Application Gateway, der auf multiple Instanzen der Applikation Routet und gleichzeitig die Datenbank hinter einem privaten Subnetz versteckt.

4.1.7 Datenspeicherung

Hier sollen die Bezeichnungen der Fingerprints und anderer Nutzdaten für den Betrieb der Applikation festgelegt werden.

Aus dem Modell in Abb. 30 wird schnell ersichtlich, das die Antennen, von denen wir empfangen, das Herzstück der Mappings darstellen.

Die Daten, welche in Tabellen festgehalten werden, besitzen alle ihre eigene seriell generierte Id als Primary Key. Das bietet sich an, da so die Fähigkeiten des Entity Framework am effizientesten genutzt werden können. Weiter gilt die Id der Antenne als Foreign Key der weiteren Tabellen um die Daten so einzigartig zuordnen zu können. Ebenso werden Metadaten der Antennen gesammelt um so Position und Zeitabschnitt genau bestimmen zu können.

Bei der massiven Anzahl an Paketen die über Bluetooth geschickt werden erscheint es sinnvoll, in einer relationalen Datenbank nicht das gesamte Paket zu loggen, sondern es auf die essentiellen Daten, welche den höchsten Grad

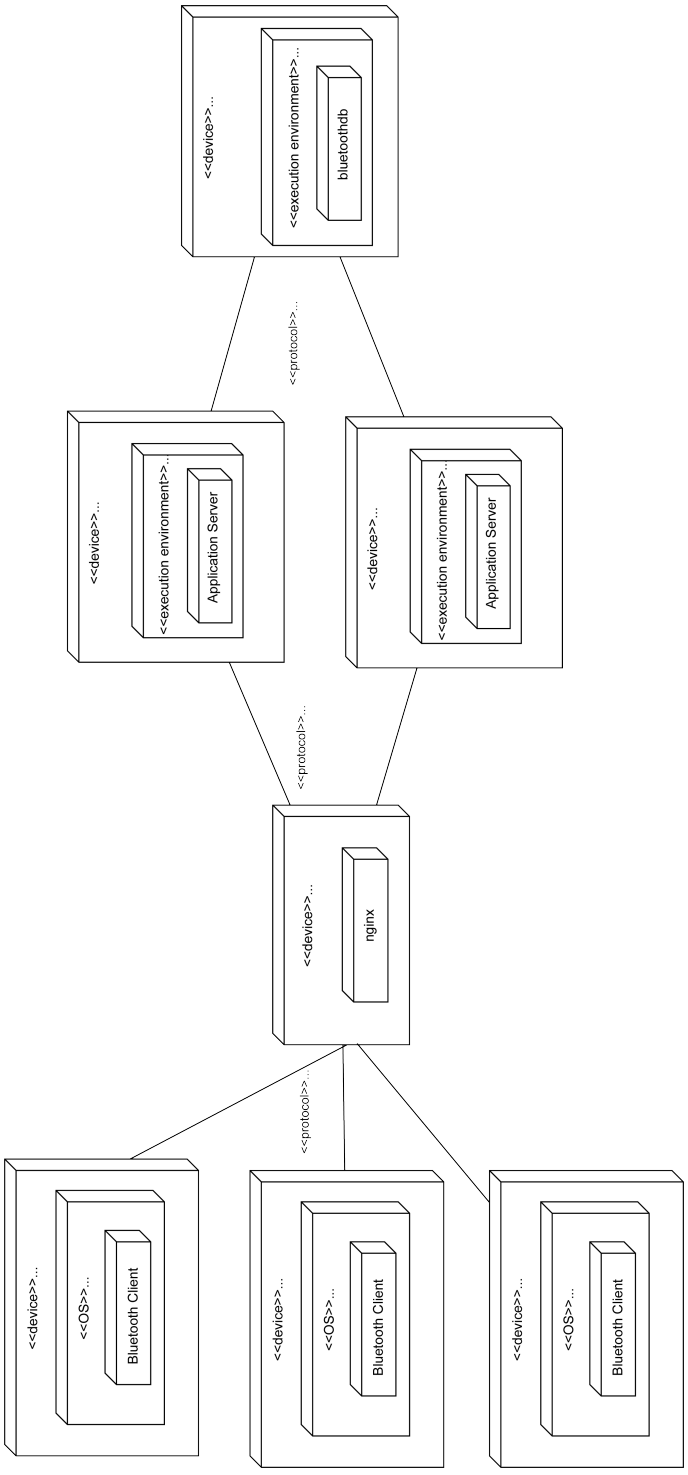

Abbildung 29: So sollte ein Deployment der Applikation auf die Router und in die Cloud aussehen.

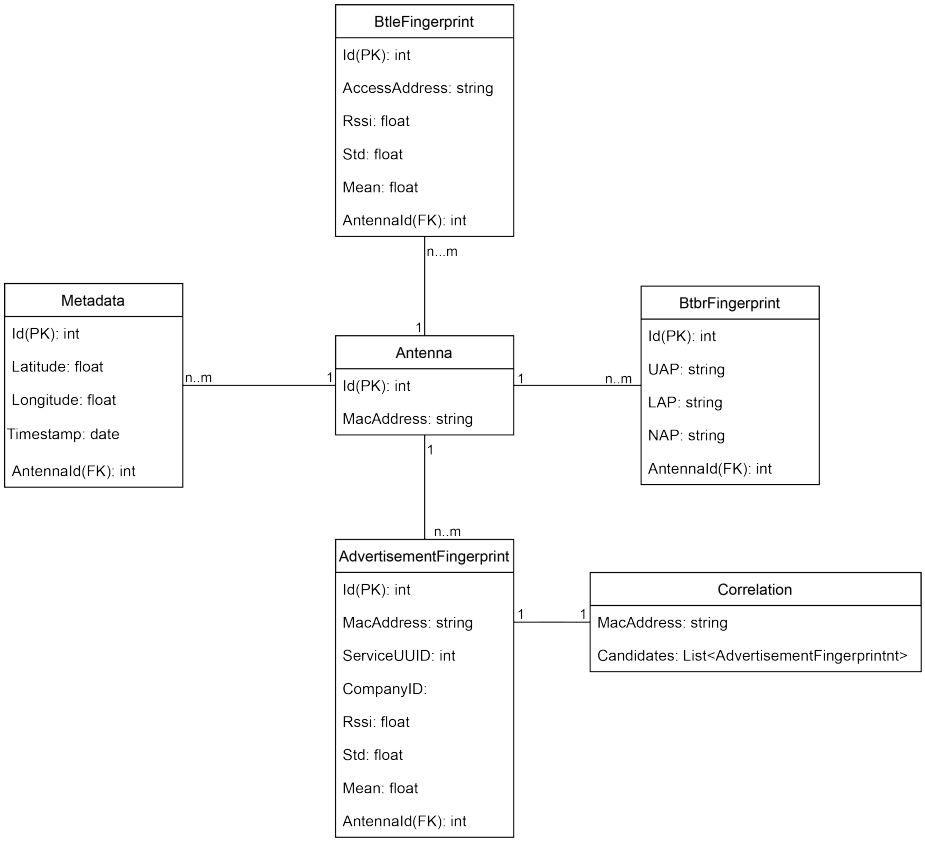

Abbildung 30: Das im Projekt verwendete Datenmodell zur Speicherung der über Bluetooth ermittelten Paketdaten

an Uniqueness innerhalb der Pakete haben, zu reduzieren um effizient und vor allem kostengünstiger das Fingerprinting machen zu können.

Die Korrelation ist an diesem Punkt eine transienter Eintrag, welcher über Logik erreicht wird um ein bluetoothfähiges Gerät anhand der passiv erfassten Advertisements trotz der geänderter randomiserter MAC Addresser verfolgen zu können.

4.2 Implementationsdetails

In Softwarearchitektur wurde bereits der Aufbau der verschiedenen Komponenten beschrieben. Hier sollen nun noch gezielt Implementationsdetails im Monitor und Correlator beschrieben werden.

4.2.1 Der Monitor

Die Monitor-Implementation beinhaltet folgende Komponenten:

- Ubertooth Firmware & Host-Tools
- Monitoring Python Scripts

Die Ubertooth Firmware sorgt dafür, dass die Pakete gesnifft und über die USB-Schnittstelle gesandt werden. Dort werden die Daten von den Host-Tools ausgelesen und verarbeitet. Die Host-Tools verarbeiten die Daten und schreiben die gewonnenen Fingerprints in ein Pipe-File, welches von den Python Scripts gelesen wird. Die Python Scripts filtern und updaten dann die Fingerprints und senden sie ans Backend.

Die für das Monitoring verwendeten Host-Tools sind ubertooth-rx und ubertooth-btle. Für das Auslesen bzw. Setzen von Einstellungen wird zudem ubertooth-util verwendet. In Letzterem mussten keine Anpassungen vorgenommen werden.

Im Folgenden werden Implementationsdetails und Angepasste Codeteile dieser Komponenten beschrieben.

4.2.1.1 Ubertooth Firmware & Host-Tools

Die Firmware wird auf den Ubertooth One geflasht.

Der vorhandene Firmware-Quellcode wurde so angepasst, dass die neuen Modi unterstützt werden. Auf Basis des LE Promiscuous Modus, welcher aktive Verbindungen snifft und beginnt zu folgen, wurde der LE Monitor Modus ergänzt. Dieser funktioniert grösstenteils identisch mit dem Promiscuous Modus, jedoch wird nicht gefolgt, da für diese Arbeit eine Übersicht über alle vorhandenen Verbindungen relevant ist und nicht etwaige Details einer einzelnen.

In der 'libubertooth' sind die Callbacks implementiert, über die der Ubertooth Daten über die USB-Schnittstelle an den Host schickt. Sie wurde um Callbacks für das Monitoring erweitert.

Die Logik für das Verarbeiten der über USB empfangenen Daten auf dem Host ist in der Shared Library libbtbb implementiert. Hier wurden die meisten Änderungen vorgenommen. Das Ziel war es, möglichst nur die Daten in die Pipe zu

schreiben, die auch notwendig sind. Dafür wurden die Indikatoren in monitor.c wie folgt definiert:

```
struct indicator_btbr {
    uint16_t flags;
    uint8_t uap;
    uint32_t lap;
    uint32_t timestamp;
} ind_btbr;

struct indicator_btle {
    uint32_t aa;
    uint32_t timestamp;
    int32_t rssi;
} ind_btle;

struct indicator_btle_adv {
    uint8_t type;
    uint8_t random;
    char mac[6];
    uint32_t timestamp;
    int32_t rssi;
    uint16_t service_uuid;
    uint16_t company_id;
} ind_btle_adv;
```

Empfängt ein Host-Tool ein Bluetooth Paket über USB, so werden daraus die für die Indikatoren benötigten Informationen ausgelesen und die structs anschliessend in die Pipe geschrieben. Im BTLE Advertisement Modus (ubertooth−btle btle−adv) werden alle Pakets verworfen, wenn sie nicht vom Typ ADV_IND, ADV_NONCONN_IND oder ADV_SCAN_IND sind.

4.2.1.2 Python Scripts

4.2.1.2.1 networking.py

In diesem File ist der Request-Handler als Singleton implementiert. Er verwendet eine Queue für das Tracken zu versendender Requests. Die Queue ist Threadsafe und kann deshalb problemlos verwendet werden. Hostname und Port des Backends werden aus der Datei network.conf ausgelesen.

Die Methode make_post_request wird aufgerufen um Pakets in die Send-Queue einzufügen. Sie nimmt als Argument einen Endpoint, einen dict mit den Key-Value Paaren des json-Contents des Requests sowie optional zwei Callbacks für erfolgreich versendete Requests (Status Code 2xx) und nicht erfolgreich versendete Requests (Status Code != 2xx).

Können 5 aufeinanderfolgende Requests nicht erfolgreich versandt werden, so kommt ein Backoff-Timer ins Spiel. Es wird 10 Sekunden gewartet bevor erneut versucht wird zu senden. Nicht erfolgreich gesendete Requests werden wieder am Ende der Queue eingefügt.

4.2.1.2.2 sniffer.py

In dieser Datei wird die Haupt-Logik des Monitoring implementiert. So wird für jeden Modus (btle-adv, btle oder btbr) ein Sniffer instanziert, welchem ein Processor übergeben wird. Der Sniffer startet dann einen Watcher-Thread, der das entsprechende Ubertooth-Tool startet und sicherstellt, dass es noch läuft. Zudem wird der Processor gestartet, indem dessen Methode start aufgerufen wird.

Die start-Methode startet dann einen weiteren Thread in welchem aus der Pipe, in welche das Ubertooth-Tool die Indicators schreibt, ausgelesen wird und die Daten verarbeitet.

Der Processor verwaltet eine Map, in welchem die Fingerprints als Values enthalten sind. Als Key werden bei Btbr der LAP, bei Btle Advertisement die Advertising Address und bei Btle die Access Address verwendet. Für die Map wird ein defaultdict verwendet. Dieser erstellt bei fehlendem Key automatisch ein Key-Value-Paar mit dem gegebenen Key und einem Default-Value. Das erlaubt es, Code wie folgenden zu schreiben auch wenn noch kein Key data.lap vorhanden ist:

```
self._fingerprints = defaultdict(BtbrFingerprint)
self._fingerprints[data.lap].update(data)
```

Liest der Processor einen Indicator aus der Pipe, so wird in der Map der entsprechende Value geupdated bzw. eingefügt. Dabei werden laufend auch die Standardabweichung, und das Mean berechnet.

Der Processor besitzt ein Read-Only Property result. Wird auf dieses zugegriffen, so wird die Map der Fingerprints aktualisiert. Konkret bedeutet dies, dass alle Fingerprints, die seit dem letzten Zugriff nicht mehr gesehen wurden, entfernt werden. Anschliessend wird eine Liste aller Fingerprints die länger als eine konfigurierbare Zeit gesehen wurden zurückgegeben. Während dieser Operation muss die Map gelockt werden, um Race Conditions zu verhindern.

4.2.1.2.3 monitor.py

In diesem File werden die verschiedenen Teile der Python-Implementation zusammengefügt.

Wird das Modul ausgeführt, so wird folgendes ausgeführt:

1. Ein Logger konfiguriert. Dieser schreibt (per default Verbose) Logging-Informationen in die Konsole sowie in ein Logfile. Dafür wird die logging Library verwendet.

2. Die übergebenen Argumente werden geparsed und rudimentär geprüft, unter anderem auch, ob genug viele Uberteeth angeschlossen sind. Dafür wird die argparse Library verwendet.

3. Der Request-Handler wird instanziert.

4. Es wird ein Request an den Antenna-Endpoint geschickt um die zu verwendende Antennen-Id zu erhalten. Er sendet dabei seine Mac-Adresse.

5. Sobald eine Antennen-Id vorhanden ist, werden die Sniffer und Processor instanziert und gestartet.

6. Es wird ein Thread für das Location-Reporting gestartet. Daraus wird alle 2 Sekunden ein Request mit der aktuellen Position der Antenne ans Backend geschickt.

7. Es wird ein Thread für das Fingerprint-Reporting gestartet. Daraus wird alle 15 Sekunden von allen vorhandenen Processors das result-Property ausgelesen und für alle darin enthaltenen Fingerprints ein entsprechender Request via dem Request-Handler versandt.

Im Prototypen wird nicht die tatsächliche Mac-Adresse verwendet. Stattdessen ist diese in der Funktion get_antenna_id hardcoded.
Die Position wird über eine Mock-Funktion get_location bestimmt. Ist entsprechende Hardware zum bestimmen der GPS-Koordinaten vorhanden, müsste diese Funktion angepasst werden.

Im Sequenzdiagramm in Abb. 31 ist der Programmfluss des Monitors dargestellt.

4.2.1.3 Bekannte Bugs

Der Ubertooth sendet teilweise beim Starten der Host-Tools keine Daten oder hört abrupt auf Daten zu senden. Es werden dann keine Fingerprints an den Monitor gesandt. In diesem Fall müssen die Host-Tools bzw. der Monitor neu gestartet werden. Da dieses Verhalten auch bei unveränderter Ubertooth-Soft- und Firmware auftritt, liegt das Problem wohl darin. Die genaue Ursache des Problems konnte nicht festgestellt werden.

4.2.2 Der Correlator

Der Correlator verarbeitet die in der Datenbank vorhandenen Fingerprints und produziert Ketten von Fingerprints die zum selben Gerät gehören.

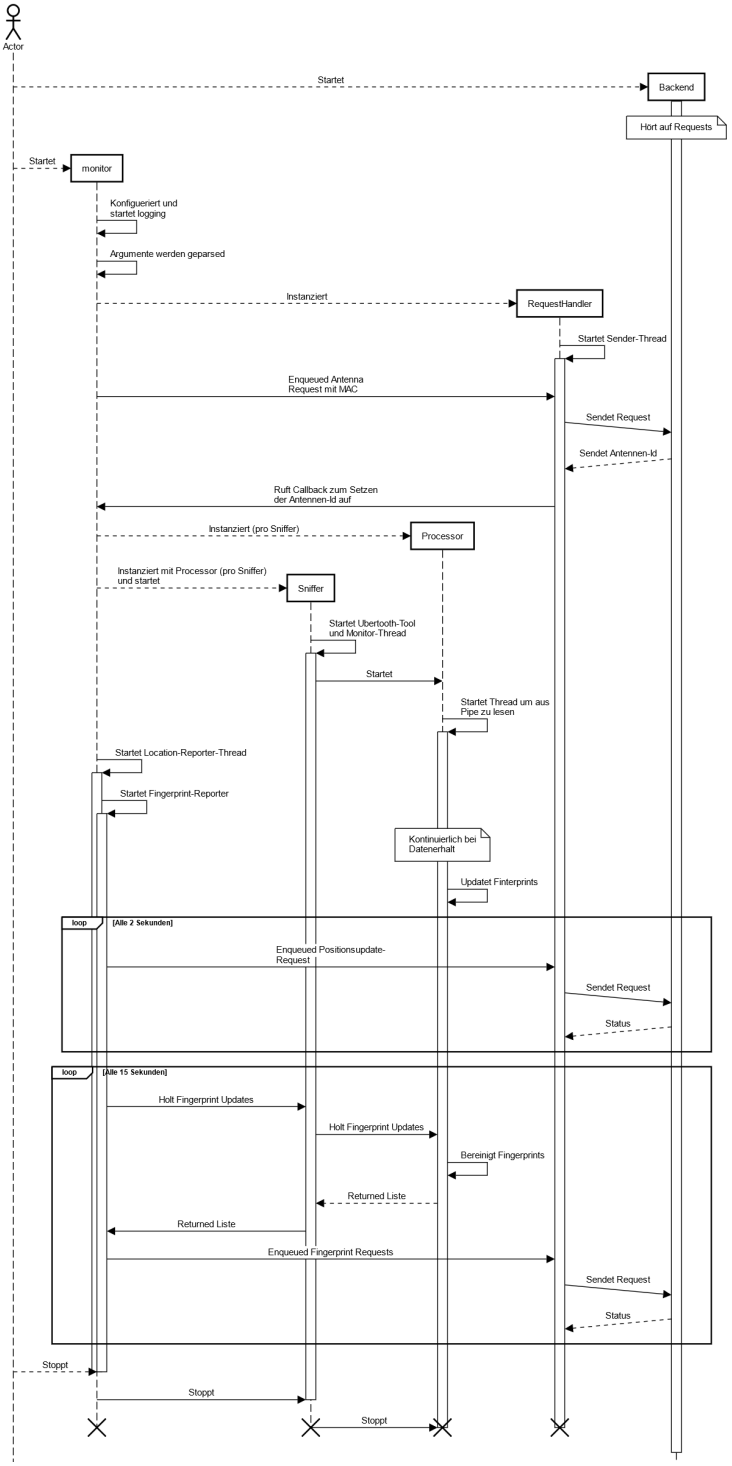

Abbildung 31: Sequenzdiagramm Monitor

Für den Correlator existieren Tests, welche mit

```
pip3 install pytest
python3 -m pytest -v
```

aus dem Verzeichnis correlation ausgeführt werden können.

Die Logik dieser Korrelation ist im Folgenden beschrieben.

4.2.2.1 BTLE

Bei Bluetooth Low Energy müssen grundsätzlich zwei Fälle unterschieden werden: Die Änderung der Advertising Address und das Auftreten der selben Advertising Address auf mehreren Antennen. Bei der Korrelation von Bluetooth LE Advertisements müssen zwei Fälle unterschieden werden. Zum einen die Änderung der Adresse, zum anderen das Auftreten der gleichen Adresse auf mehreren Antennen.

Dazu ein Beispiel in Abb. 32.

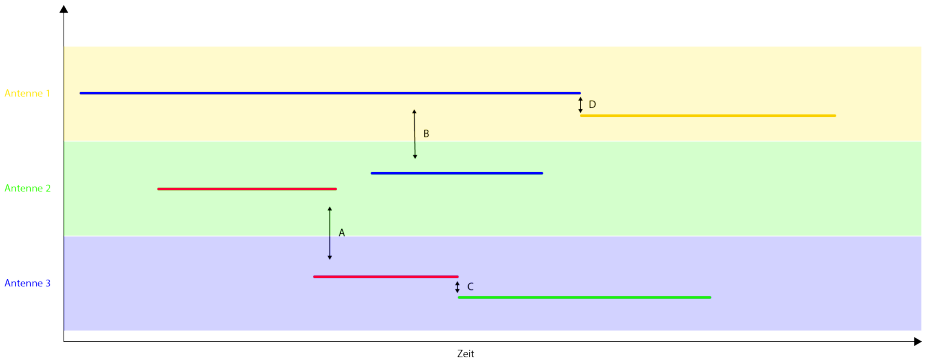

Abbildung 32: Korrelationsbeispiel. Die Linien repräsentieren Fingerprints. Fingerprints mit gleicher Farbe haben die selbe Advertising Address.

Es sind die aufgezeichneten Fingerprints von drei Antennen visualisiert. Es sind zwei Geräte gecaptured worden. Die Linien repräsentieren Fingerprints, Fingerprints mit gleicher Farbe haben die selbe Advertising Address. In den Fällen A und B wurde jeweils die selbe Advertising Address auf mehreren Antennen gesehen. Im folgenden wird dieser Fall 'Antennenwechsel' genannt. Bei C und D hat die Advertising Address geändert. Aufgrund der weiteren Daten im Advertisement kann festgestellt werden, ob diese Fingerprints zum selben Gerät gehören. Dieser Fall wird im folgenden 'Adressänderung' genannt.
In diesem Beispiel sollte der Correlator nun zwei Geräte detektieren: blau/gelb und rot/grün.

4.2.2.2 Adressänderung

Ein Fingerprint wird dann als Nachfolger eines anderen Vorgänger-Fingerprints - und damit zum selben Gerät gehörend - betrachtet, wenn folgende Kriterien erfüllt sind:

- Die Service Class UUIDs sind identisch
- Die Company ID ist identisch
- Der Nachfolger wurde frühestens beim letzten Auftreten des Vorgängers zum ersten Mal gesehen
- Der Nachfolger wurde spätestens 5 Sekunden nach dem letzen Auftreten des Vorgängers zum erstem Mal gesehen
- Die Distanz zwischen den Antennenpositionen zu den Zeitpunkten des letzten Auftretens des Vorgängers und dem ersten Autreten des Nachfolgers ist maximal 500m

Unsere isolierten Tests in der Antennenkammer des ICOM haben gezeigt, dass Geräte bei Adressänderung innerhalb von 5 Sekunden Advertisements mit neuer Adresse zu versenden beginnen.
Die Distanz von 500m wurde gewählt, da eine Korrelation auch in fahrenden Zügen möglich sein muss. 500m/5s entspricht 360km/h. Zudem wird damit ausgeschlossen, dass zwei Fingerprints von unterschiedlichen Geräten mit identischen Adertising Addresses aber grosser Entfernung als das selbe Gerät erkannt werden.

Zum Detektieren einer Adressänderung wird wie folgt vorgegangen:

1. Sämtliche Fingerprints in der Datenbank werden nach dem first_seen Timestamp sortiert ausgelesen und in einer Liste platziert.
2. Für jeden Fingerprint wird anhand des last_seen Timestamps determiniert, an welcher Index der Liste ein Nachfolger frühestens auftreten könnte. Da die Liste bereits nach first_seen sortiert ist, kann dafür der bisect-Algorithmus verwendet werden, welcher eine Komplexität von O(log n) hat.
3. Ab dem determinierten Index und solange das zeitliche Kriterium erfüllt ist, wird für alle Fingerprints anhand oben ausgeführten Kriterien geprüft, ob sie Nachfolger-Kandidaten sind.
4. Die Kandidaten aufsteigend nach der Differenz der RSSI (minus eine Standardabweichung um Schwankungen zu kompensieren) sortiert.
5. Die resultierende, sortierte Nachfolger-Kandidatenliste wird dem Fingerprint zugewiesen.
6. Gibt es nur einen Kandidaten, wird dieser als Nachfolger markiert. Da bei mehreren Kandidaten nicht sichergestellt werden kann, dass der richtige

gewählt wuder, wird in diesem Fall kein Fingerprint als Nachfolger markiert.

Diese Implementation stösst an Grenzen, wenn zum Zeitpunkt eines Adresswechsels viele neue Geräte auftauchen. Konkret wäre dies zum Beispiel bei Einfahrt in einen Bahnhof der Fall. Es konnte keine Lösung für dieses Problem gefunden werden.

4.2.2.3 Antennenwechsel

Zwei Fingerprints werden als zum gleichen Gerät gehörend betrachtet, wenn folgende Kriterien erfüllt sind:

- Die Advertising Addresses sind identisch
- Die Service Class UUIDs sind identisch
- Die Company ID ist identisch
- Der später zum ersten Mal gesehene Fingerprint wurde maximal 15 Minuten nach dem letzten Auftreten des früher Gesehenen zum ersten Mal gesehen
- Gibt es einen Zeitraum, in dem keiner der beiden Fingerprints gesehen wurde, ist die Distanz zwischen den Antennen zu den Zeitpunkten des letzten Auftreten des früher Gesehenen und dem ersten Auftreten des später gesehenen maximal 10 Kilometer, ansonsten maximal 100m

Die erlaubte Distanz von 10km wurde gewählt, um zu erlauben, dass sich ein Gerät mit einem nicht gemonitorten Transportmittel (z.B. mit einem Bus zu einem anderen Bahnhof) fortbewegt hat. Ohne Observationslücke wurden 100m gewählt um für allfällige GPS-Ungenauigkeiten und die Signalreichweite von Bluetooth zu kompensieren.

Zum Detektieren eines Antennenwechsels wird wie folgt vorgegangen:

1. Während der Iteration über alle Fingerprints zum detektieren der Adressänderung wird eine Map mit der Advertising Address als Keys und einer Liste an Fingerprints mit dieser Advertising Address als Value erstellt.
2. Für jeden Key in dieser Map - also für alle Advertising Addresses - wird überprüft ob mehr als ein Fingerprint im zugehörigen Value vorhanden ist. Ist dies nicht der Fall, wurde eine Advertising Address nur auf einer einzigen Antenne gesehen und kann unmöglich die Antenne gewechselt haben.
3. Für alle Advertising Addresses welche auf mehr als einer Antenne gesehen wurden, wird für die zugehörigen Fingerprints paarweise anhand obiger Kriterien geprüft, ob sie zum gleichen Gerät gehören. Schritt 3 und 4 sind detailliert beschrieben in Reduktion der Wechsel.

4. Mittels Graphentheorie wird eine Liste die benötigten Fingerprints ermittelt und gegebenenfalls einem Fingerprint ein auf einer anderen Antenne gefundener Fingerprint zugewiesen.

In der Implementation in Python wird für die in Schritt 1 erwähnte Map ein defaultdict verwendet. Dieser erstellt bei fehlendem Key automatisch ein Key-Value-Paar mit dem gegebenen Key und einem Default-Value, in diesem Fall einer leeren list. Das erlaubt es, Code wie folgenden zu schreiben:

```
record = defaultdict(list)
record[fingerprint.mac].append(fingerprint)
```

auch wenn der defaultdict noch keinen Key 'fingerprint.mac' hat.

4.2.2.3.1 Reduktion der Wechsel

Gegeben ist eine Liste an Fingerprints, welche die selbe Advertising Address haben. Diese Liste ist sortiert nach den first_seen Timestamps der Fingerprints. Es soll nun determiniert werden, wie viele Geräte vorhanden sind und welche Fingerprints einen Informationsgewinn bringen.

Dazu wird Graphentheorie benutzt.

Zuerst werden alle Fingerprints als Nodes in den Graphen eingefügt. Im Pythoncode wird dafür networkx verwendet.

Dann werden die Fingerprints paarweise nach den in Antennenwechsel aufgeführten Kriterien verglichen. Gehören sie zum selben Gerät, wird im Graphen eine Kante zwischen den Fingerprints eingefügt.

Abb. 33 zeigt dazu ein Beispiel.

Die Fingerprint-Liste sei die Liste mit Fingerprints einer gegebenen Advertising Address. Sie ist nach First Seen sortiert. Darunter eine Visualisierung der Daten. Antenne 4 bedinde sich weit entfernt der anderen Antennen, sodass das Distanzkriterium der Antennenwechsel-Korrelation nicht erfüllt ist.

Das paarweise Vergleichen ist notwendig, damit Fingerprint B korrekt mit Fingerprint A verbunden werden kann.

Oben rechts ist der resultierende Graph. Man sieht, dass es darin zwei getrennte Teile gibt. Subgraphen in denen alle Nodes durch Edges erreicht werden können nennt man in der Graphentheorie Komponenten. Ein Komponent entspricht bei uns dann einem Gerät.

Der generierte Graph wird also in die Komponenten zerlegt. Die Komponenten können noch unnötige Nodes enthalten. Im Beispiel sieht man, dass Fingerprint C keinen Informationsgewinn bringt, da die Position des Gerätes bereits durch Fingerprint A bekannt ist. Um den ganzen Pfad des Gerätes nachzuvollziehen reichen die Fingerprints A und B.

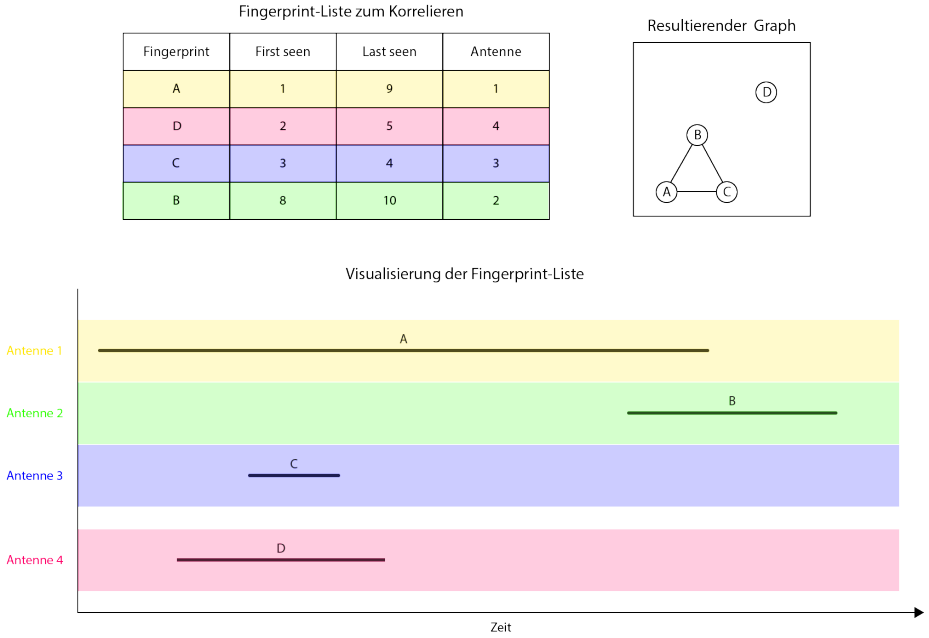

Abbildung 33: Beispiel einer Korrelation mit dem resultierenden Graphen

Um das herauszufinden wird ein Shortest Path Algorithmus verwendet (per Default wird in networkx der Dijkstra-Algorithmus verwendet). Für jeden der Komponenten bestimmen wir als nächstes den als erstes und den als letztes gesehenen Fingerprint. Diese sind die Start- bzw. Endnodes unseres Pfades. Sollte es dafür mehrere Kandidaten geben, wird der am längsten gesehene davon verwendet. Sind immernoch mehrere Kandidaten vorhanden, sind diese mit Ausnahme der Antenne identisch und es wird der erste davon verwendet. Da die Kanten nicht gewichtet sind, ist der kürzeste Pfad einfach derjenige mit dem wenigsten Nodes, also derjenige der die wenigsten Fingerprints benötigt. Es kann davon durchaus mehrere geben, welche allerdings alle gleichwertig sind. Folglich ist es egal, welcher gefunden wird.

Aus der Shortest Path-Berechnung resultiert dann eine Liste mit Fingerprints die zur vollständigen Nachvollziehbarkeit des Pfades eines Gerätes notwendig sind. In jedem Fingerprint des kürzesten Pfades wird dann der jeweils nächste Fingerprint eingetragen und dieser als antennengewechselt markiert. Nicht verwendete Fingerprints werden ebenfalls als antennengewechselt markiert, jedoch nicht in einem anderen Fingerprint eingetragen.

Alle Fingerprints eines Komponenten, ausser dem zuerste gesehenen, sind folglich als antennengewechselt markiert.

4.2.2.4 Zusammenführung Adressänderung und Antennenwechsel

Nachdem die Adress- und Antennenwechsel bestimmt worden sind, können die verschiedenen Geräte und deren Pfade bestimmt werden.

Aus den vorherigen Schritten ergibt sich eine Liste aller Fingerprints mit folgenden Eigenschaften:

- Ist der Fingerprint der einzige Kandidat für einen Adresswechsel, ist er als Nachfolger markiert. Er ist in Fingerprints für die er ein Nachfolger ist als Nachfolger eingetragen.
- Ist die Advertising Address eines Fingerprints mehrfach in der Liste, so sind alle Fingerprints mit dieser Adresse als antennengewechselt markiert, mit ausnahme des ersten.
- Ist der Fingerprint im kürzesten Pfad des Antennenwechsel-Pfades und nicht der Letzgesehene, ist sein Nachfolger bei ihm eingetragen.

Alle Fingerprints die nicht als Nachfolger oder antennengewechselt markiert sind, können als einzelnes Gerät angeschaut werden.

Abb. 34 zeigt die Logik zum Ausgeben aller Geräte. Wird der Correlator mit correlator.py −a ausgeführt, wird die hier beschriebene Logik ausgeführt und jeweils die Advertising Address ausgegeben. Antennenwechsel werden durch einen Asterisk vor der Adresse denotiert, Adresswechsel durch einen Einzug.

4.2.2.5 BT BR/EDR

Bei Bluetooth BR/EDR enfällt die Adresswechsel-Korrelation welche bei BTLE notwendig ist.

Zwei Fingerprints sollen als zum gleichen Gerät gehörend betrachtet werden, wenn folgende Kriterien erfüllt sind:

- Die Bluetooth Adressen sind identisch
- Der später zum ersten Mal gesehene Fingerprint wurde maximal 15 Minuten nach dem letzten Auftreten des früher Gesehenen zum ersten Mal gesehen
- Gibt es einen Zeitraum, in dem keiner der beiden Fingerprints gesehen wurde, ist die Distanz zwischen den Antennen zu den Zeitpunkten des letzten Auftreten des früher Gesehenen und dem ersten Auftreten des später gesehenen maximal 10 Kilometer, ansonsten maximal 100m

Da Bluetooth Adressen nicht ändern, könnten die Kriterien bezüglich Zeit und Ort auch etwas relaxiert werden, um Reisen auch bei längeren Unterbrechungen der Datenverbindung nachvollziehen zu können.

Die Bluetooth BR/EDR-Korrelation wurde aus Zeitgründen und der weitgehend gleichen Implementation wie BTLE im Prototypen nicht gemacht. Folgend ist ein Vorschlag für eine möglich Implementation:

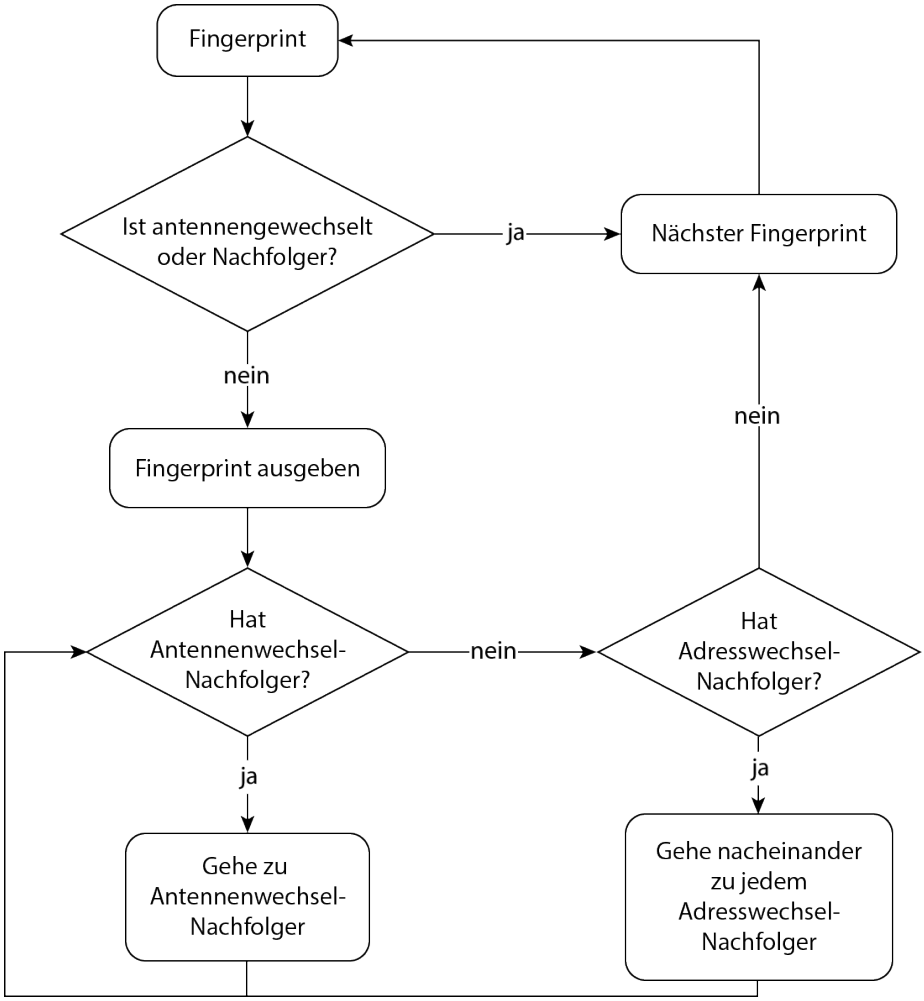

Abbildung 34: Logik zum Ausgeben aller Geräte

1. Sämtliche Fingerprints in der Datenbank werden nach dem first_seen Timestamp sortiert ausgelesen und in einer Map platziert mit der Adresse als Key und einer Liste der Fingerprints als Value.

2. Für jeden Key in dieser Map - also für alle Bluetooth Adressen - wird überprüft ob mehr als ein Fingerprint im zugehörigen Value vorhanden ist. Ist dies nicht der Fall, wurde eine Bluetooth Adresse nur auf einer einzigen Antenne gesehen und kann unmöglich die Antenne gewechselt haben.

3. Für alle Bluetooth Adressen, welche auf mehr als einer Antenne gesehen wurden, wird für die zugehörigen Fingerprints paarweise anhand obiger Kriterien geprüft, ob sie zum gleichen Gerät gehören.

4. Mittels Graphentheorie wird eine Liste die benötigten Fingerprints ermittelt und gegebenenfalls einem Fingerprint ein auf einer anderen Antenne gefundener Fingerprint zugewiesen.

4.3 Messungen / Tests

4.3.1 Testprotokoll

4.3.1.1 Einführung

In dieser Arbeit werden speziell die verschiedenen Eigenschaften von Bluetooth untersucht. Dazu werden diverse Tests, unter verschiedenen Bedingungen durchgefuehrt.

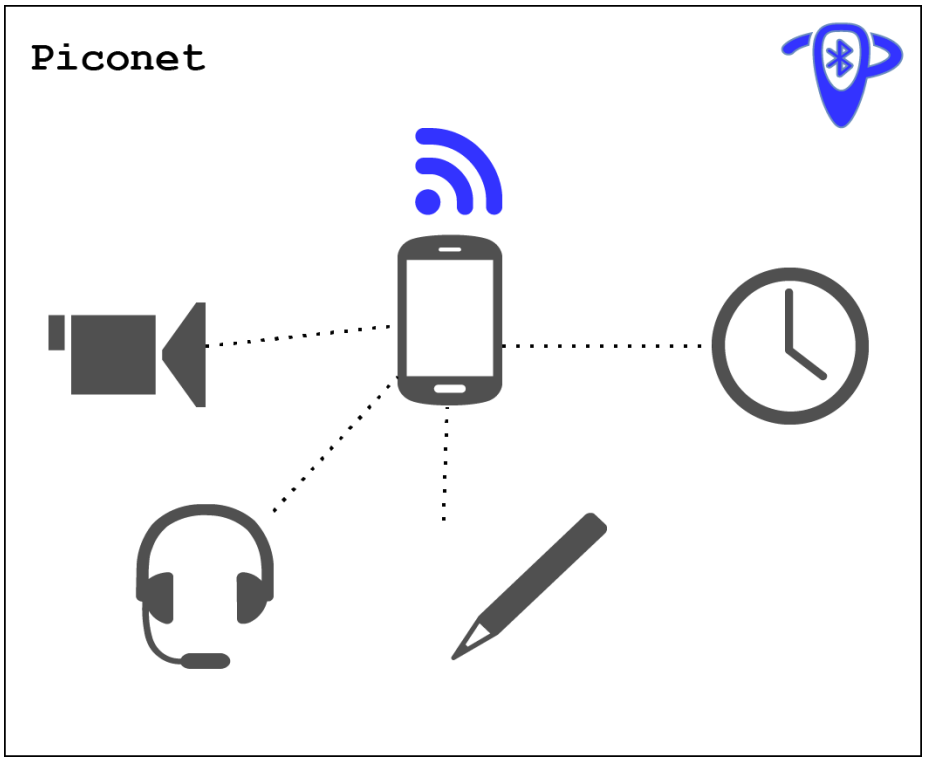

Abbildung 35: Beispiel eines Piconetzes

4.3.1.1.1 Zweck

Um einen korrekten Ablauf der Arbeit zu gewährleisten, ist eine genaue Beschreibung der verwendeten Methodik notwendig. Hier werden die Methoden zur Verifizierung der aufgestellten Hypothesen festgehalten. Dazu gehören sowohl die verwendeten Mittel, eine Beschreibung der Herangehensweise und der erwarteten Ergebnisse sowie deren Resultate und eine Schlussfolgerung.

4.3.1.1.2 Übersicht

Zuerst werden die verwendeten Materialien und die Umstände beschrieben. Weiter werden die zu untersuchenden Begebenheiten genauer spezifiziert. Folgend werden die Ergebnisse präsentiert und diskutiert.

4.3.1.1.3 Grundlegender Versuchsaufbau

Es werden die verschiedenen Signale, welche vorallem das Mobiltelefon, aber auch Peripheriegeräte, über Bluetooth senden, untersucht. Dabei wird auf bestimmte Eigenschaften oder Versionierungen des Standards abgezielt. So ist zu jedem Test festzuhalten, welche Begebenheit untersucht wird.

Als die Empfängerantenne wird stets der Ubertooth One mit der passenden Firmware verwendet. Ebenso soll nach Möglichkeit ein Wireshark Capture zum Überprüfen der Ergebnisse erstellt werden. Bei Tests zu denen ein entsprechendes Custom Tooling besteht wird ebenfalls ein SQLite-Datenbank-File hinterlegt.

4.3.1.1.4 Anmerkung

Die Durchführung der Versuche in der Antennenkammer ein Muss, da sonst viel unnötiger Lärm mit eingefangen wird.

Zu beachten ist ebenso, dass wir grundsätzlich zwischen Versuchen zu Bluetooth Classic und Low Energy unterscheiden müssen. Die Limitation unserer Empfängerantennen ist zu beachten, da wir mit der eingesetzten Hardware auf einen einzigen Kanal während des Monitorings limitiert sind.

Für eine einsehbare Liste aller verwendeter Geräte sehen sie die Betriebsmittelliste in Sektion 2.3.

4.3.1.1.5 Erwartung

In dieser Arbeit wird in erster Linie davon ausgegangen, dass ein Fingerprinting per Bluetooth möglich wäre. Somit werden positive Bestätigungen der genannten Ansätze erwartet.

So wird bei Bluetooth Classic erwartet, dass die letzten 4 Byte der Bluetooth MAC durch passives Monitoring erkannt werden können und ein anschliessender l2ping auf das Device ausführbar ist.

Für Bluetooth LE wird erwartet, dass Devices eindeutig über die Access Addresses, die sie mit den Peripheriegeräten bestimmen, identifizieren lassen. Dabei ist wichtig, das die Access Addressen bei einer zu erwartenden Anzahl Geräte nicht überschneiden.

4.3.2 Messung Weg 1

Datum: 23.05.2021
Personen: Severin Marti
Ort: Feldweg in Brütten

4.3.2.1 Dateien

- Datenbank-File
  - Name: weg_1-2.db
  - Command:
    - Backend: dotnet run
    - Monitor: monitor.py btle-adv
- Wireshark-Capture
  - Name: capture_weg_1.pcapng
  - Command:
    - ubertooth−btle −n −q /tmp/pipe −U1
    - Pipe in Wireshark geöffnet, am Schluss gespeichert

4.3.2.2 Geräte

In Tab. 7 sind die verwendeten Geräte aufgelistet.

Tabelle 7: Geräte Capture Weg 1

Typ	Name	OS Version	SIM	COVID App
Phone	iPhone 6	12.3.1	No	No
Phone	iPhone X	14.0.1	No	No
Phone	Galaxy S20+	11	Yes	Yes
Headphone	Airpods Gen 2	n/a	n/a	n/a
Smart Watch	Apple Watch Series 5	watchOS 7	No	No

4.3.2.3 Zweck

Es sollen Daten gesammelt werden, um damit die Funktionstüchtigkeit des Monitorings zu beurteilen.
Wenn möglich, sollen diese Daten danach verwendet werden um die Korrelations-Implementation zu verbessern.

4.3.2.4 Events

Tab. 8 zeigt sämtliche durch den Tester vorgenommenen Handlungen auf. Ebenfalls erfasst sind Spaziergänger oder Velofahrer, sofern sie sich mehr als 1 Minute in der nahen Umgebung der Antenne aufgehalten haben.

Tabelle 8: Events Capture Weg 1

Event Id	Zeit	Event
	14:23:43	START MEASUREMENT (capture_weg_1.pcapng, weg_1-2.db)
A	14:27:11	Open S20+
B	14:31:45	Open Airpod case
C	14:36:xx	2 Velofahrer (kein iPhone, mit COVID App)
D	14:52:xx	Spaziergänger
E	14:54:50	Close Airpod case lid
F	15:03:08	Turn off BT on S20+
G	15:05:30	Turn on S20+
H	15:10:02	Restart monitor with new id
I	15:11:45	Turn on Apple Watch (flight mode on)
J	15:16:30	Turn off S20+
K	15:16:xx	Spaziergänger
L	15:19:00	Turn off flight mode Apple Watch
M	15:20:30	Turn on iPhone 6 (BT on)
N	15:22:xx	Velofahrer
O	15:33:40	Turn on location sharing on iPhone 6
P	15:36:40	Turn off iPhone 6
Q	15:37:25	Turn on S20+
R	15:39:40	Restart monitor with new id
S	15:42:25	Turn on iPhone x
T	15:42:xx	Spaziergänger
U	15:44:45	Velofahrer (iPhone XR, mit COVID App)
V	15:51:xx	2 Spaziergänger
W	15:55:06	STOP MEASUREMENT

4.3.2.5 Auswertung / Manuelle Korrelation

Die Daten in Tab. 9 kommen aus der Backend Datenbank (weg_1-2.db). Manuell hinzugefügt wurden die Spalten Event und Kommentar, wo versucht wurde, eine Korrelation mit Aktionen und Geräten zu machen.

Tabelle 9: Auswertung Capture Weg 1

Id	MacAddress	Rssi	FirstSeen	LastSeen	Seen for	Antenna	Event	Kommentar (line for reference)
1	70:fc:ab:57:06:f4	-87	14:26:30	14:28:52	142s	1		
2	6f:0f:c8:d0:8f:4e	-88	14:27:15	14:38:30	675s	1	A	S20+
3	49:35:c6:93:24:05	-58	14:31:52	14:34:13	141s	1	B	Airpod case
4	41:b9:ac:83:88:3b	-58	14:34:13	14:49:13	900s	1		Airpod case address change (3)
5	6d:23:78:9b:f4:6d	-71	14:36:50	14:43:16	386s	1	C	Velofahrer
6	7d:c7:98:7e:29:92	-86	14:38:33	14:48:50	617s	1		S20+ address change (2)
7	46:27:3c:89:78:a3	-64	14:43:17	14:45:09	112s	1		Velofahrer address change (5)
8	41:1f:fb:c6:1e:0d	-89	14:45:10	14:50:39	329s	1		
9	49:f3:4f:c8:3c:07	-87	14:48:53	14:59:20	627s	1		S20+ address change (6)
10	53:31:25:ce:ae:ad	-40	14:49:13	14:54:58	345s	1	E	Airpod case address change (4)
11	13:74:27:a1:4a:be	-89	14:52:13	14:53:28	75s	1	D	Spaziergänger
12	5d:5d:eb:ad:30:e9	-87	14:52:20	14:53:28	68s	1	D	Spaziergänger
13	1a:89:b8:3e:ef:1f	-86	14:52:22	14:55:42	200s	1	D	Spaziergänger
14	60:43:f2:be:78:e2	-67	14:59:22	15:03:11	229s	1	F	S20+ address change (9)
15	51:83:68:fd:f5:ef	-78	15:05:33	15:09:46	253s	1	G	S20+
16	51:83:68:fd:f5:ef	-86	15:10:02	15:12:39	157s	2	H	S20+, same address
17	73:81:0c:a3:75:49	-79	15:12:40	15:16:39	239s	2	J	S20+ address change (16)
18	51:96:66:9d:a8:c1	-86	15:12:50	15:39:34	1604s	2	I	Apple Watch unpaired
19	6a:ba:61:ec:99:bf	-90	15:16:16	15:17:47	91s	2	K	Spaziergänger
20	73:cd:9a:f4:06:50	-88	15:22:06	15:23:16	70s	2	N	Velofahrer
21	09:4c:b6:02:09:97	-86	15:22:07	15:23:16	69s	2	N	Velofahrer
22	56:f0:75:90:23:5f	-75	15:37:27	15:39:34	127s	2		S20+ address change (17)
23	51:96:66:9d:a8:c1	-63	15:39:46	15:42:38	172s	3	R	Apple Watch, same address (18)
24	56:f0:75:90:23:5f	-73	15:39:46	15:41:25	99s	3	R	S20+, same address (22)
25	75:9d:bc:a2:64:95	-89	15:41:59	15:43:07	68s	3	T	Spaziergänger
26	26:ae:be:c5:bf:b7	-88	15:41:59	15:43:07	68s	3	T	Spaziergänger
27	7e:df:46:93:12:01	-87	15:42:02	15:43:04	62s	3	T	Spaziergänger
28	56:24:5d:7b:10:55	-58	15:42:35	15:45:27	172s	3	S	iPhone X
29	1a:54:b3:d1:19:53	-97	15:44:38	15:52:35	477s	3	U	Velofahrer iPhone XR (COVID App)
30	41:f8:e3:8d:9c:74	-88	15:44:39	15:52:35	476s	3	U	Velofahrer iPhone XR

4.3.2.6 Kommentar

Die Geräte wurden sehr gut erkannt. Auf Adress-Änderungen können nachvollzogen werden. Der Prototyp funktioniert also.
Das iPhone 6 wurde nie gesehen.

4.3.3 Messung Weg 2

Datum: 23.05.2021
Personen: Severin Marti
Ort: Feldweg in Brütten

4.3.3.1 Dateien

- Datenbank-File
  - Name: weg_1-2.db
  - Command:
    - Backend: dotnet run
    - Monitor: monitor.py btle-adv
- Wireshark-Capture
  - Name: capture_weg_2.pcapng
  - Command:
    - ubertooth-btle -n -q /tmp/pipe -U1
    - Pipe in Wireshark geöffnet, am Schluss gespeichert

4.3.3.2 Geräte

In Tab. 10 sind die verwendeten Geräte aufgelistet.

Tabelle 10: Geräte Capture Weg 2

Typ	Name	OS Version	SIM	COVID App
Phone	iPhone 6	12.3.1	No	No
Phone	iPhone X	14.0.1	No	No
Phone	Galaxy S20+	11	Yes	Yes
Headphone	Airpods Gen 2	n/a	n/a	n/a

4.3.3.3 Zweck

Da in der Messung Weg 1 das iPhone 6 nicht erfasst wurde, werden hier gezielte Tests mit iPhones gemacht.
Ebenfalls will eruiert werden, ob sich Airpods im und ausserhalb des Cases unterschiedlich verhalten.

4.3.3.4 Events

Tab. 11 zeigt sämtliche durch den Tester vorgenommenen Handlungen auf. Ebenfalls erfasst sind Spaziergänger oder Velofahrer, sofern sie sich mehr als 1 Minute in der nahen Umgebung der Antenne aufgehalten haben.

Tabelle 11: Events Capture Weg 2

Event Id	Zeit	Event
A	15:56:46	START MEASUREMENT
B	15:58:xx	Spaziergänger
C	16:00:30	Turn off iphone x (only watch on)
D	16:04:15	Turn on iphone x
E	16:05:30	Turn off watch
F	16:08:50	Turn on iphone 6
G	16:13:45	Open Airpod case
H	16:14:10	Pair Airpods with iPhone 6
I	16:15:40	Close Airpod case lid
J	16:16:24	Open Airpod case lid
K	16:18:00	Take out Airpods, close lid
L	16:22:00	Put away Airpods (lid closed)
M	16:23:29	STOP MEASUREMENT

4.3.3.5 Auswertung / Manuelle Korrelation

Die Daten in Tab. 12 kommen aus der Backend Datenbank (weg_1-2.db). Manuell hinzugefügt wurden die Spalten Event und Kommentar, wo versucht wurde, eine Korrelation mit Aktionen und Geräten zu machen.

Tabelle 12: Auswertung Capture Weg 2

Id	MacAddress	Rssi	FirstSeen	LastSeen	Seen for	Antenna	Event	Kommentar (line for reference)
1	78:3d:93:9d:cb:6e	-47	15:56:46	15:58:31	105s	3		
2	52:3f:71:6d:37:5f	-59	16:01:23	16:04:28	185s	3		
3	50:14:37:35:c7:e7	-42	16:04:25	16:07:51	206s	3	D	iPhone X
4	6e:e1:03:89:fe:69	-47	16:13:48	16:15:40	112s	3	G, I	Airpod case
5	50:df:98:ed:c1:61	-53	16:16:24	16:18:08	104s	3	J	Airpod case
6	47:1b:10:b6:d8:a0	-48	16:18:07	16:22:10	243s	3	K, L	Airpods

4.3.3.6 Kommentar

Das iPhone 6 wurde wiederum nicht erkannt. Dies ist möglicherweise auf die alte Hardware oder die alte iOS-Version zurückzuführen.

Airpods erhalten neue Adressen, sobald sie aus dem Case genommen werden.

4.3.4 Messung Weg 3

Datum: 23.05.2021
Personen: Severin Marti
Ort: Feldweg in Brütten

4.3.4.1 Dateien

- Datenbank-File
  - Name: weg_3.db
  - Command:
    - Backend: dotnet run
    - Monitor: monitor.py btle-adv
- Wireshark-Capture
  - Name: capture_weg_3.pcapng
  - Command:
    - ubertooth-btle -n -q /tmp/pipe -U1
    - Pipe in Wireshark geöffnet, am Schluss gespeichert

4.3.4.2 Geräte

In Tab. 13 sind die verwendeten Geräte aufgelistet.

Tabelle 13: Geräte Capture Weg 3

Typ	Name	OS Version	SIM	COVID App
Phone	iPhone 6	12.3.1	No	No
Phone	iPhone X	14.0.1	No	No
Phone	Galaxy S20+	11	Yes	Yes
Headphone	Airpods Gen 2	n/a	n/a	n/a

4.3.4.3 Zweck

Es sollen nochmals Daten gesammelt werden, um die Korrelation-Implementation zu verbessern.

4.3.4.4 Events

Tab. 14 zeigt sämtliche durch den Tester vorgenommenen Handlungen auf. Ebenfalls erfasst sind Spaziergänger oder Velofahrer, sofern sie sich mehr als 1 Minute in der nahen Umgebung der Antenne aufgehalten haben.

Tabelle 14: Events Capture Weg 3

Event Id	Zeit	Event
A	16:28:25	START MEASUREMENT iPhone 6 still on
B	16:29:00	Turn on S20+
C	16:29:30	Take out Airpods, close lid
D	16:30:25	Turn on Apple Watch
E	16:35:00	Spaziergänger (iPhone 11, no COVID App, hat Mac)
F	16:42:55	Turn on iPhone X
G	16:49:15	Turn off BT S20+
H	16:51:3x	Turn on BT S20+
I	17:02:00	Spaziergänger
J	17:04:05	Turn off monitor
K	17:04:23	Turn on monitor with new id
L	17:06:01	Put away Airpods
M	17:06:58	Turn off BT on S20+

4.3.4.5 Auswertung / Manuelle Korrelation

Die Daten in Tab. 15 kommen aus der Backend Datenbank (weg_3.db). Manuell hinzugefügt wurden die Spalten Event und Kommentar, wo versucht wurde, eine Korrelation mit Aktionen und Geräten zu machen.

iPhones benötigen i.d.R. ca 10-15s zum starten, die Apple Watch >1m.

Tabelle 15: Auswertung Capture Weg 3

Id	MacAddress	Rssi	FirstSeen	LastSeen	Seen for	Antenna	Event	Kommentar (line for reference)
1	5d:e8:2b:36:ab:50	-86	16:29:03	16:39:39	636s	1	B	S20+
2	6e:4e:74:2e:ad:63	-66	16:29:16	16:31:27	131s	1		
3	5e:9c:50:92:69:60	-48	16:29:32	16:37:16	464s	1	C	Airpods
4	6e:14:79:54:fb:43	-51	16:31:24	16:33:03	99s	1	D	Apple Watch
5	72:49:da:4a:e7:27	-88	16:34:49	16:44:14	565s	1	E	Spaziergänger, iPhone 11
6	6c:6a:0b:bf:81:3f	-50	16:37:16	16:52:23	907s	1		Airpods, address change (3)
7	60:01:8a:07:7e:2b	-82	16:39:42	16:49:16	574s	1	G	S20+ address change (1)
8	7b:b2:60:1d:a9:37	-59	16:43:05	17:00:34	1049s	1	F	iPhone X
9	6c:7c:dc:4f:b8:b8	-86	16:51:39	17:02:42	663s	1	H	S20+
10	57:ad:cf:df:7e:61	-53	16:52:23	17:03:57	694s	1		Airpods, address change (6)
11	64:2d:7e:69:ff:ca	-59	17:00:34	17:03:57	203s	1		iPhone X, address change (8)
12	01:31:bd:b1:e1:f3	-89	17:01:58	17:03:01	63s	1	I	Spaziergänger
13	58:18:d4:65:80:7a	-88	17:01:59	17:03:01	62s	1	I	Spaziergänger
14	79:30:b4:71:82:63	-86	17:02:43	17:03:57	74s	1		S20+ address change (7)
15	57:ad:cf:df:7e:61	-46	17:04:21	17:06:07	106s	2	K, L	Airpods, same address (10)
16	79:30:b4:71:82:63	-87	17:04:21	17:06:59	158s	2	K, M	S20+, same address (14)
17	64:2d:7e:69:ff:ca	-84	17:04:21	17:06:51	150s	2		iPhone X, same address (11)

4.3.4.6 Kommentar

Es ist unklar, weshalb die Apple Watch abrupt aufgehört hat Advertisements zu senden.

4.3.5 Messung Weg Abend

Datum: 23.05.2021
Personen: Severin Marti
Ort: Feldweg in Brütten

4.3.5.1 Dateien

- Datenbank-File
  - Name: weg_abend.db
  - Command:
    - Backend: dotnet run
    - Monitor: monitor.py btle-adv
- Wireshark-Capture
  - Name: capture_weg_abend.pcapng
  - Command:
    - ubertooth-btle -n -q /tmp/pipe -U1
    - Pipe in Wireshark geöffnet, am Schluss gespeichert

4.3.5.2 Geräte

In Tab. 16 sind die verwendeten Geräte aufgelistet.

Tabelle 16: Geräte Capture Weg Abend

Typ	Name	OS Version	SIM	COVID App
Phone	iPhone XS Max	14.4.2	Yes	No
Phone	iPhone XR	14.4.2	Yes	Yes
Scooter	Ninebot E-Scooter	n/a	n/a	n/a

4.3.5.3 Zweck

Da in früheren Tests das iPhone 6 nicht erfasst wurde, soll nun bei zwei weiteren iPhones geschaut werden, ob sie erfasst werden.

4.3.5.4 Events

Tab. 17 zeigt sämtliche durch den Tester vorgenommenen Handlungen auf.

Tabelle 17: Events Capture Weg Abend

Event Id	Zeit	Event
A	20:56:58	START MEASUREMENT iPhone XR turned on, airplane mode on
B	21:01:05	Turn off airplane mode on iPhone XR
C	21:01:40	Bring Scooter to monitoring station
D	21:01:40	Bring iPhone XS Max to monitoring station, BT off
E	20:02:40	Turn on BT on iPhone XS Max

4.3.5.5 Auswertung / Manuelle Korrelation

Die Daten in Tab. 18 kommen aus der Backend Datenbank (weg_abend.db). Manuell hinzugefügt wurden die Spalten Event und Kommentar, wo versucht wurde, eine Korrelation mit Aktionen und Geräten zu machen.

Tabelle 18: Auswertung Capture Weg 3

Id	MacAddress	Rssi	FirstSeen	LastSeen	Seen for	Antenna	Event	Kommentar (line for reference)
1	46:6a:8a:37:f7:5e	-88	20:56:58	21:06:00	542s	1	(A)	iPhone XR
2	6b:ee:8b:31:68:42	-57	20:56:58	21:02:41	343s	1	(A)	iPhone XR
3	2c:41:fa:ed:70:52	-77	21:01:08	21:09:13	485s	1	B	iPhone XR COVID App
4	64:48:80:ca:bc:29	-77	21:01:08	21:09:13	485s	1	B	iPhone XR
5	c6:94:91:3e:69:8f	-86	21:01:39	21:04:12	153s	1	C	Scooter
6	43:25:d0:40:50:25	-59	21:02:43	21:09:13	390s	1	E	iPhone XS Max

4.3.5.6 Kommentar

Sämtliche Geräte wurden erfasst.
Schön ersichtlich hier ist, dass der Flugmodus Bluetooth nicht deaktiviert. Um Bluetooth komplett zu deaktivieren, muss es in den Einstellungen deaktiviert werden. Mehr dazu ist im Dokument iPhone zu finden.
Es ist ebenfalls zu sehen, dass die Adressen ändern, wenn der Flugmodus ausgeschaltet wird.

4.3.6 Test 28.05

Datum: 28.05.2021
Personen: Lukas Volk
Ort: Antennenkammer ICOM

4.3.6.1 Dateien

- Datenbank-File
  - Name: bluetooth.db
  - Command:
    - Backend: dotnet run
    - Monitor: monitor.py btle-adv

4.3.6.2 Geräte

In Tab. 19 sind die verwendeten Geräte aufgelistet.

Tabelle 19: Geräte Capture vom 28.05.

Typ	Name	COVID App
Phone	iPhone X	No
Phone	iPhone 6s	No
Phone	Galaxy 6	Yes

4.3.6.3 Zweck

Es soll untersucht werden, welche und wie oft iPhones Advertisements senden

4.3.6.4 Events

Tab. 20 zeigt sämtliche durch den Tester vorgenommenen Handlungen auf.

Tabelle 20: Events Capture vom 28.05.

Zeit	Event
15:40:30	ACTIVE ADVERTISMENT IPHONE 6
15:45:xx	Reactivate Iphone X

4.3.6.5 Kommentar

Das iPhone X geht nach einer Zeit in einen “Schlafmodus” und stoppt, Advertisements zu senden. Wird der Screen aktiviert, beginnen die Advertisements wieder mit neuer Adresse. Dieses Verhalten konnte nicht zuverlässig reproduziert werden.

5 Ergebnisse

5.1 Resultate

In dieser Arbeit wurde untersucht, ob und wie ein passives Fingerprinting und Tracking von Mobilgeräten mit Bluetooth möglich ist. Es wurden dabei Ansätze für Bluetooth BR/EDR und Bluetooth Low Energy ausgearbeitet und basierend darauf ein Prototyp entwickelt.

Bei Bluetooth BR/EDR können Piconet Master-Geräte erkannt und anhand des LAP gefingerprintet werden, sofern sie aktiv Daten senden. Die Master-Geräte sind dabei Mobiltelefone sowie Laptops und ähnliche Geräte. Als weitere Fingerprint-Eigenschaft wird, falls vorhanden, der UAP verwendet. Ein Tracking über mehrere Antennen ist damit möglich.

Bei Bluetooth LE konnte ein Ansatz erarbeitet werden, bei dem aus den LE-Advertisements die Advertising Addresses zusammen mit optionalen Feldern als Fingerprint verwendet werden. Die Advertisements von Apple-Geräten sowie der SwissCovid App sind dabei besonders geeignet. Um die Advertising Address-Wechsel nachzuvollziehen, wurde entsprechende Korrelationslogik implementiert. Diese sollte theoretisch bei einer geringen Anzahl von Geräten (Apple: <17, Covid <14) gut funktionieren. Bei einer grossen Anzahl Geräte oder vielen gleichzeitig erscheinenden Geräten können die Addresswechsel jedoch nicht mehr alle zuverlässig erkannt werden.
Neuere iPhones lassen sich mit dieser Methode gut erkennen. Android-Geräte werden jedoch nur erkannt, falls sie eine App, wie zum Beispiel die SwissCovid App, installiert haben. Peripheriegeräte und sonstige LE-fähige Geräte werden ebenfalls erfasst, sofern sie Advertisements aussenden.

Ein Tracking durch die Datenpakete von Bluetooth LE konnte nicht erfolgreich gemacht werden. Es wurde versucht, die Access Addresses von LE-Verbindungen für den Fingerprint zu verwenden. Bei den Testgeräten war dafür jedoch zu wenig aktiver Datenverkehr vorhanden.

Eine Aussage über die absolute Anzahl an präsenten Geräten lässt sich mit den ausgearbeiteten Ansätzen nicht machen. Auch wurde keine Möglichkeit gefunden, einen BR/EDR-Fingerprint einem LE-Fingerprint zuzuweisen.

Der entwickelte Prototyp besteht aus einer Monitorkomponente, welche mithilfe des Ubertooth One Bluetooth-Sniffers die Fingerprints erstellt. Dabei werden Bluetooth BR/EDR-, LE-Advertisement- und LE-Datenpaket-Fingerprints generiert. Diese werden dann an die Backend-Komponente gesandt, welche sie persistent in einer Sqlite-Datenbank abspeichert. Die Backend-Komponente erlaubt auch einen Blick auf die in der Datenbank gespeicherten Fingerprints über HTML Views. Mit der Korrelator-Komponente können dann die Adresswechsel der Advertisement-Fingerprints erkannt und der Pfad eines Gerätes mithilfe der Google Maps API visualisiert werden.

Abb. 36 zeigt die Visualisierung eines Pfades. Es wurde der Befehl python correlator.py −m 46:27:3c:89:78:a3 −i ausgeführt. Anhand des Textes

am oberen Rand ist ersichtlich, dass Adresswechsel von 51:83:68:fd:f5:ef auf 51:83:68:fd:f5:ef (andere Antenne), 46:27:3c:89:78:a3 und schlussendlich 41:1f:fb:c6:1e:0d erkannt wurden. Der Pfad ist in diesem Fall eine gerade Linie, da für den zugrundeliegenden Test Mock-GPS-Daten verwendet wurden.

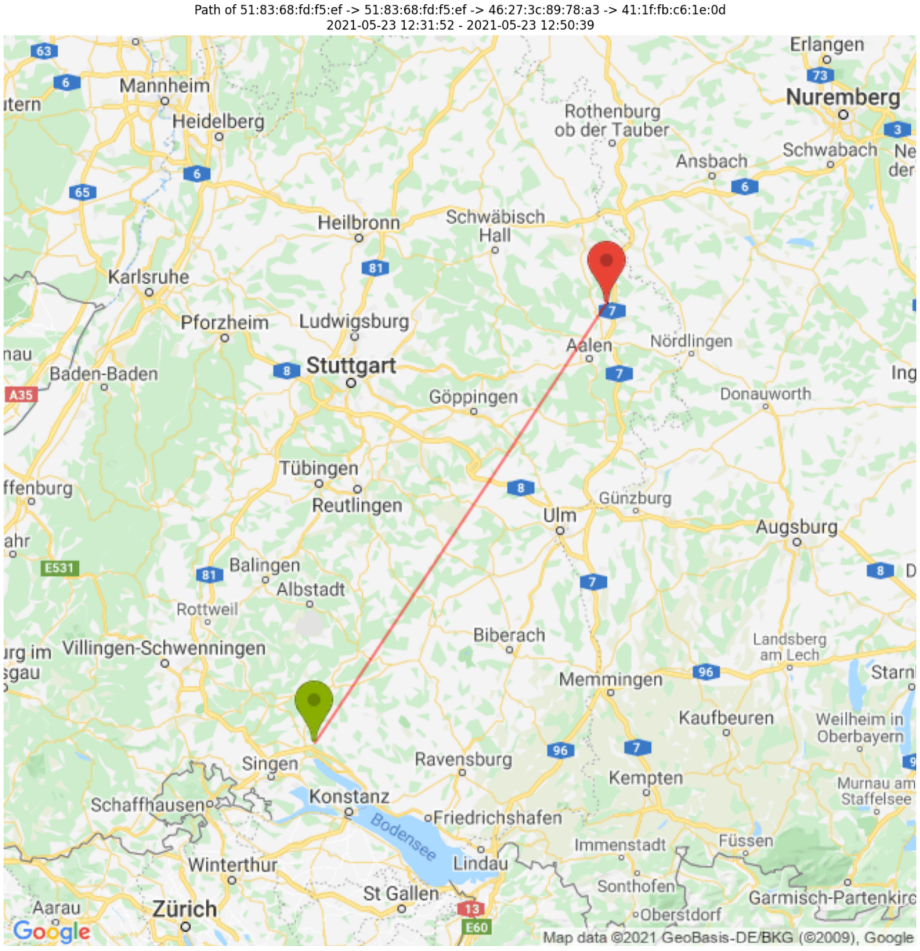

Abbildung 36: Visualisierung eines Pfades. Generiert durch python correlator.py −m 46:27:3c:89:78:a3 −i

5.2 Limitationen

Durch die limitierte Anzahl Testgeräte und das Personenlimit bei Versammlungen aufgrund der andauernden Covid-Pandemie, konnten keine grossen kontrollierte Messungen durchgeführt werden. Die Verifikation der Ergebnisse war in gewissen Bereichen deshalb schwierig bis unmöglich. Die grösseren Messungen

mussten deshalb im öffentlichen Bereich unter nicht-kontrollierbaren Umständen durchgeführt werden. Bei Tests in Zügen zum Beispel war es nicht möglich zu verifizieren, dass die erkannten Adresswechsel und die Anzahl der erkannten Geräte korrekt sind. Es konnten nur manuelle Plausibilitätsabschätzungen gemacht werden. Die in der Antennenkammer oder an isolierten Orten getätigten Messungen konnten verifiziert werden.

5.3 Diskussion und Ausblick

Die Untersuchungen haben gezeigt, dass ein passives Fingerprinting und Tracking von Mobilgeräten über Bluetooth unter gewissen Bedingungen möglich ist.

Die absolute Anzahl an Geräten kann mit keinem der erarbeiteten Ansätze bestimmt werden. Wenn durch die Anwendung der Ansätze entsprechende sinnvolle Grenzwerte definiert werden, können jedoch ordinale Aussagen wie "Wenige," "Durchschnittlich viele" oder "Viele" gemacht werden.

5.3.1 LE Audio

Bereits vor Beginn unserer Arbeit wurde die Bluetooth Core Spezifikation 5.2 [5] publiziert. Diese enthält die Unterstützung für LE Audio. Es wurden jedoch keine Geräte gefunden, welche diese Technologie unterstützen. Sämtliche getesteten Kopfhörer haben die Audiodaten über BR/EDR empfangen. Der nicht erfolgreiche Ansatz über die LE-Datenpakete könnte in Zukunft erfolgreich eingesetzt werden, wenn Geräte LE Audio implementieren. Es sollten dann ausreichend Daten vorhanden und damit ein Fingerprinting sinnvoll möglich sein.

5.3.2 Advertisements

Da nur neuere Apple iPhones über die Advertisements fingerprintbar sind, scheint dies eine neue Entwicklung zu sein. Da Apple oft eine Vorreiterrolle hat, ist es denkbar, dass Android-Geräte-Hersteller in Zukunft nachziehen werden und dadurch auch diese Geräte mit diesem Ansatz trackbar werden.

Die Fingerprints könnten um den Advertisement-Typ erweitert werden, um mehr Geräte unterscheiden zu können. Der Ubertooth liefert diese Daten bereits an die Python-Scripts, sie müssten jedoch noch im Backend und bei der Korrelation verwendet werden.

Die Verwendung der SwissCovid App-Advertisements ist zwar aus rechtlicher Sicht zulässig, jedoch aus ethischer Sicht fragwürdig. Sollten Leute befürchen, dass sie durch die Verwendung einer solchen App getrackt werden, wären sie weniger bereit, diese zu installieren. Das hätte direkte Auswirkungen auf die Gesundheit der Bevölkerung.

5.3.3 Monitor

In dieser Arbeit kamen maximal drei Uberteeth gleichzeitig zum Einsatz. Dadurch waren Messungen auf drei Channel limitiert. Für ein umfassenderes Bild des Bluetooth-Datenverkehrs sollten mehr Antennen verwendet werden.
Für den produktiven Einsatz wäre die Verwendung einer andere Antenne

empfehlenswert. Wie in Sektion 4.2.1.3 erwähnt, stoppt der Ubertooth teilweise plötzlich Daten zu übertragen.

5.3.4 Kombination mit anderen Daten

Für das beste Resultat sollten die durch Bluetooth gewonnenen Daten mit Daten aus anderen Quellen kombiniert werden. Ist zum Beispiel aus Passagierbefragungen historisch oder aus WiFi-Daten aktuell bekannt, wie viele Personen sich in einem Wagon aufhalten, so könnte möglicherweise mit den Trackinginformationen von Bluetooth eine Hochrechnung gemacht werden.

Eine weitere Verbesserung der der Korrelation könnte mithilfe mehrerer Antennen gemacht werden. Hat mehr als eine Antenne eine Advertising Adressee aufgezeichnet, könnten damit das getrackte Gerät im Raum positioniert werden und die Adresswechsel-Kandidaten weiter gefiltert werden.

5.3.5 Kontrollierte Tests in grösserem Massstab

Es sollten grössere kontrollierte Tests durchgeführt werden. Wegen der Covid-Pandemie konnten keine Menschengruppen organisiert werden. Messungen in welchen alle Geräte bekannt waren, waren deshalb auf die zu dem Zeitpunkt verfügbaren Geräte limitiert. Nur dort konnten die Ergebnisse auch sinnvoll validiert werden. Bei Tests im öffentlichen Raum waren mehr Geräte verfügbar, jedoch war unbekannt, wieviele und welche Geräte sich im Umfeld befanden.

5.3.6 Machine Learning

Die Advertisements der SwissCovid App und Apple-Geräte machten etwa zwei Drittel der total empfangenen Advertisements aus. Mittels Machine Learning könnte der verbleibende Drittel untersucht werden. Eine Auswertung wie in der Vorgängerarbeit SA Passenger Tracking [1] könnte in den verbleibenden Advertisements distinktive Eigenschaften finden.

Abbildungsverzeichnis

1 Bluetooth LE Channels und Adaptive Frequency Hopping. Quelle: [3] . . . 10
2 Aufbau der Bluetooth Adresse . . . 11
3 Access Code Format. Quelle: [5] . . . 12
4 Verbindungsaufbau-Vorgang Bluetooth BR/EDR . . . 13
5 Advertising Adress-Typen . . . 16
6 SwissCovid (GAEN) ADV_NONCONN_IND Advertisement in Wireshark . . . 19
7 SwissCovid (GAEN) ADV_SCAN_IND Advertisement in Wireshark . . . 20
8 Apple ADV_SCAN_IND . . . 21
9 Bluetooth Option im Control Center. Deaktiviert nur bestehende Verbindungen bis 5:00 am nächsten Tag . . . 23
10 Bluetooth Einstellungen. Deaktiviert Bluetooth vollständig . . . 24
11 Flugmodus ist an, Bluetooth aber auch . . . 25
12 Usecases mit den zugehörigen Aktoren . . . 27
13 Der Ubertooth One mit Komponenten. Quelle: [9] . . . 34
14 Nordic Dev-kit. Quelle [10] . . . 36
15 Microbit. Quelle: [11] . . . 36
16 BluetoothAnalyzer. Quelle: [12] . . . 37
17 Visual Studio Code. Quelle: [13] . . . 38
18 Wireshark Programmfester mit Capture . . . 39
19 blue-hydra. Quelle: [14] . . . 40
20 bettercap. Quelle: [15] . . . 41
21 Antennenkammer ICOM . . . 42
22 Strukturdiagramm . . . 44
23 Paketübersicht, mit den gesamten Abhängigkeiten . . . 57
24 Logische Aufteilung des Systems und dessen Zustände . . . 59
25 Das Klassendiagramm des DataAccess Projekts . . . 61
26 Das Klassendiagramm des Backend-Projekts . . . 62
27 Das Klassendiagramm des Frontend-Projekts . . . 63
28 Der Programmfluss des Backends bei Erhalt eines Post-Requests am Advertisements Endpunkt. . . . 66
29 So sollte ein Deployment der Applikation auf die Router und in die Cloud aussehen. . . . 68
30 Das im Projekt verwendete Datenmodell zur Speicherung der über Bluetooth ermittelten Paketdaten . . . 69
31 Sequenzdiagramm Monitor . . . 75
32 Korrelationsbeispiel. Die Linien repräsentieren Fingerprints. Fingerprints mit gleicher Farbe haben die selbe Advertising Address. 76
33 Beispiel einer Korrelation mit dem resultierenden Graphen . . . 80
34 Logik zum Ausgeben aller Geräte . . . 82
35 Beispiel eines Piconetzes . . . 84
36 Visualisierung eines Pfades. Generiert durch python correlator.py −m 46:27:3c:89:78:a3 −i102

Tabellenverzeichnis

1 Bluetooth Leistungsklassen. Klasse 1.5 nur bei LE 10
2 Advertising PDU Typen 15
3 Verwendete Endgeräte 32
4 Verwendete Peripheriegeräte 32
5 Wahrscheinlichkeit, dass Geräte im selben Time Slot die Adresse wechseln bei Adresswechsel alle 15 Minuten 51
6 Wahrscheinlichkeit, dass Geräte im selben Time Slot die Adresse wechseln bei Adresswechsel alle 10 Minuten 52
7 Geräte Capture Weg 1 86
8 Events Capture Weg 1 87
9 Auswertung Capture Weg 1 88
10 Geräte Capture Weg 2 89
11 Events Capture Weg 2 90
12 Auswertung Capture Weg 2 91
13 Geräte Capture Weg 3 92
14 Events Capture Weg 3 93
15 Auswertung Capture Weg 3 94
16 Geräte Capture Weg Abend 95
17 Events Capture Weg Abend 96
18 Auswertung Capture Weg 3 97
19 Geräte Capture vom 28.05. 98
20 Events Capture vom 28.05. 99

Literaturverzeichnis

1. A Fraefel, A Shanmuganathan, M Sonder. Studienarbeit passenger tracking. In INS; 2020.
2. J Schlatter, M Schmid. Bachelorarbeit mobile fingerprinting. In INS; 2021.
3. MT Inc. Bluetooth® low energy channels [Internet]. Verfügbar unter: https://microchipdeveloper.com/wireless:ble-link-layer-channels, Letzter Zugriff am: 17.06.2021
4. M Ossmann. Discovering the bluetooth UAP [Internet]. Verfügbar unter: https://ubertooth.blogspot.com/2014/06/discovering-bluetooth-uap.html, Letzter Zugriff am: 17.06.2021
5. CSW Group. Bluetooth core specification. Bluetooth SIG; 2019.
6. ETHZ, EPFL. SwissCovid documentation [Internet]. Verfügbar unter: https://github.com/SwissCovid/swisscovid-doc, Letzter Zugriff am: 17.06.2021
7. ETHZ, EPFL. DP3T - decentralized privacy-preserving proximity tracing [Internet]. Verfügbar unter: https://github.com/DP-3T/documents, Letzter Zugriff am: 17.06.2021
8. Apple, Google. Exposure notification - cryptography specification v1.1.8 [Internet]. Verfügbar unter: https://www.blog.google/documents/60/Exposure_Notification_-_Cryptography_Specification_v1.1.pdf, Letzter Zugriff am: 17.06.2021
9. GS Gadgets. Ubertooth github repository [Internet]. Verfügbar unter: https://github.com/greatscottgadgets/ubertooth, Letzter Zugriff am: 17.06.2021
10. N Semiconductor. nRF52840 DK [Internet]. Verfügbar unter: https://www.nordicsemi.com/Software-and-tools/Development-Kits/nRF52840-DK, Letzter Zugriff am: 17.06.2021
11. ME Foundation. User guide [Internet]. Verfügbar unter: https://www.microbit.org/get-started/user-guide/overview/, Letzter Zugriff am: 17.06.2021
12. Ellisys. Ellisys bluetooth explorer all-in-one bluetooth® analysis system [Internet]. Verfügbar unter: https://www.ellisys.com/products/bex400/, Letzter Zugriff am: 17.06.2021
13. Microsoft. Code editing.redefined. [Internet]. Verfügbar unter: https://code.visualstudio.com/, Letzter Zugriff am: 17.06.2021
14. Zero_Chaos. Blue hydra github repository [Internet]. Verfügbar unter: https://github.com/ZeroChaos-/blue_hydra, Letzter Zugriff am: 17.06.2021
15. BD Team. Bettercap repo [Internet]. Verfügbar unter: https://github.com/bettercap/bettercap, Letzter Zugriff am: 17.06.2021

16. G Celosia, M Cunche. Fingerprinting bluetooth-low-energy devices based on the generic attribute profile. IoT S&P 2019 - 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things. 2019;24–31.
17. S Software. API development for everyone [Internet]. Verfügbar unter: https://swagger.io/, Letzter Zugriff am: 17.06.2021
18. PS Foundation. Python homepage [Internet]. Verfügbar unter: https://www.python.org/, Letzter Zugriff am: 17.06.2021
19. Microsoft. Introduction to .NET [Internet]. Verfügbar unter: https://docs.microsoft.com/en-us/dotnet/core/introduction, Letzter Zugriff am: 17.06.2021
20. Microsoft. What is ASP.NET? [Internet]. Verfügbar unter: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet, Letzter Zugriff am: 17.06.2021
21. M Fowler. Richardson maturity model. Martin Fowler Blog. 2010;

Glossar

Abkürzung	Bedeutung
AFH	Adaptive Frequency Hopping. Siehe FHSS.
API	Application Programming Interface. Programmierschnittstelle.
BA	Bachelorarbeit
BD_ADDR	Die Bluetooth-Adresse bei Bluetooth BR/EDR
BLE	Siehe LE
BR/EDR	Bluetooth BR/EDR. Auch Bluetooth Classic genannt.
BT	Bluetooth
BTLE	Siehe LE
COVID	Coronavirus Disease
CRC	Cyclic Redundancy Check. Eine Checksumme für Fehlererkennung.
EUI-48	Siehe MAC
FHS	Frequency Hopping Synchronization
FHSS	Frequency Hopping Spread Spectrum. Eine Methode um Interferenz zu vermeiden, indem Channels gewechselt werden.
FS	Frühlingssemester
GAEN	Google & Apple Exposure Notifications
GATT	Generic Attribute Profile. Definiert wie Bluetooth LE-Geräte Daten austauschen.
GHz	Gigahertz. SI-Einheit der Frequenz
GPS	Global Positioning System. Navigationssystem zur Geolokation
HEC	Header Error Check. Eine Checksumme im Paket-Header für die Sicherstellung der Datenintegrität.
HSR	Hochschule für Technik Rapperswil
HTTP	Hypertext Transfer Protocol. Protokoll zum Laden von Websites.
ICOM	Institute for Communication Systems
IDE	Integrated Development Environment. Entwicklungsumgebung.
IEEE	Institute of Electrical and Electronics Engineers
INS	Institute for Networked Solutions

Abkürzung	Bedeutung
L2CAP	Logical Link Control Adaption Protocol. Ein Bluetooth-Protokoll des Data Link Layers.
LAP	Lower Address Part. Die drei least significant Bytes dere Bluetooth-Adresse
LE	Bluetooth Low Energy. Ehemals Bluetooth Smart. Energiesparendere Alternative zu Bluetooth Classic
MAC	Media-Access-Control(-Adresse). Identifiziert Gerätee auf einer Datenverbindung
MVC	Model-View-Controller. Ein Software Design Pattern.
NAP	Non-significant Address Part. Die zwei most significant Bytes der Bluetooth-Adresse.
OST	Ostschweizer Fachhochschule
OUI	Organizationally Unique Identifier. Die ersten drei Bytes der Bluetooth Adresse. Wird vom IEEE zugewiesen und identifiziert den Hersteller.
PDU	Protokol Data Unit
POC	Proof of Concept. Minimale Implementierung, um grundlegende Funktionstüchtigkeit zu beweisen.
RPI	Rolling Proximity Identifier
SA	Studienarbeit
SAP	Significant Address Part. Besteht aus UAP und LAP. Teil der Bluetooth-Adresse.
SDR	Software Defined Radio
SQL	Datenbanksprache für relationale Datenbanken
TEK	Temporary Exposure Key
UAP	Upper Address Part. Das dritte Byte der Bluetooth-Adresse.
UI	User Interface. Benutzeroberfläche.
UUID	Universally Unique Identifier. Ein in der Praxis eindeutiger Identifier.

Abkürzung	Bedeutung
WLAN	Wireless Local Area Network. Eine Art von drahtlosen Netzwerken, normalerweise der Standards IEEE-802.11.

Appendix

Appendix A
Installationsanleitung

6.1.1 Übersicht

Das Projekt besteht aus mehreren Komponenten, die separat installiert werden.

- libbtbb (geforked von greatscottgadgets/libbtbb)
  Die Bluetooth baseband decoding library. Wird vom Ubertooth verwendet und muss auf dem capturenden Gerät installiert sein.
- Ubertooth (geforked von greatscottgadgets/ubertooth) Die Ubertooth Firmware und Host-Tools um damit zu interagieren. Benötigt libbtbb und muss auf dem capturenden Gerät installiert sein.
- Monitor
  Parsed die Daten des Ubertooth One und schickt die Fingerprints an das Backend. Verwendet Ubertooth und muss auf dem capturenden Gerät installiert sein.
- Backend und Frontend Speichert die Fingerprints in eine Datenbank. Erlaubt Einblick in die Datenbank über HTML Views.
- Correlator
  Korreliert die Fingerprints in einer Datenbank des Backends. Kann auf einem beliebigen Gerät verwendet werden, benötigt jedoch ein Datenbank-File.

Damit alle Projekte vorhanden sind, klone das Git-Repository mit der −−recursive Option:

```
git clone --recursive \
  ssh://git@gitlab.ost.ch:45022/ins-stud/ba21-mobile-fingerprinting.git
```

Die Anweisungen sind relativ zum root-Verzeichnis dieses Repositories.

6.1.2 Systemvoraussetzungen

Entwickelt und getestet wurde auf Ubuntu 20.04 LTS. Auf anderen Betriebssystemen kann die Funktionalität nicht gewehrleistet werden.

6.1.3 Ubertooth

Beinhaltet die Firmware für den Ubertooth One Dongle sowie die notwendige Host-Software um mit dem Dongle zu interagieren.

6.1.3.1 Abhängigkeiten

Installiere die Abhängigkeiten:

```
sudo apt install cmake libusb-1.0-0-dev make gcc g++ \
  libbluetooth-dev pkg-config libpcap-dev \
  python3-distutils python3-setuptools
```

6.1.3.2 libbtbb

Aus dem root-Verzeichnis des Repos:

```
cd code/libbtbb
mkdir build
cd build
cmake ..
make
sudo make install
```

6.1.3.3 Ubertooth Tools

Aus dem root-Verzeichnis des Repos:

```
cd code/ubertooth/host
mkdir build
cd build
cmake ..
make
sudo make install
```

6.1.3.4 Firmware

Aus dem root-Verzeichnis des Repos:

```
cd code/ubertooth/firmware
make clean all && make
```

Schliesse den Ubertooth One an um die neue Firmware zu flashen, dann flashe mit:

```
ubertooth-dfu -r -d bluetooth_rxtx/bluetooth_rxtx.dfu
```

Stecke den Ubertooth im Anschluss daran einmal aus und wieder ein.

6.1.4 Monitor

Es wird Python 3.8 oder neuer benötigt.
Überprüfe deine Version mit

```
python3 --version
```

und upgrade wenn nötig.

6.1.4.1 Abhängigkeiten

Installiere pip:

```
sudo apt install python3-pip
```

Installiere die benötigten Module:
Aus dem root Verzeichnis des Repos:

```
cd code/ubertooth/monitoring
pip3 install -r requirements.txt
```

Passe den Hostnamen und den Port des Backends in code/ubertooth/monitoring/network.conf an.

Das Monitoring kann nun gestartet werden.

Aus der Hilfe (python3 monitor.py -h):

```
usage: monitor.py [-h] modes [modes ...]

Bluetooth Device Tracker.

positional arguments:
  modes       Operating modes. One or more of btbr, btle, btle-adv. One Ubertooth is required per mode.

optional arguments:
  -h, --help  show this help message and exit
```

Es wird ein Ubertooth pro Modus benötigt.

6.1.5 Backend & Frontend

Überprüfe ob das dotnet CLI Tool installiert ist

```
dotnet --version
```

falls das der Fall ist, kann weitergegangen werden. Ist dies nicht der Fall, siehe dotnet Installationsinstruktionen

Das Starten des Projekt funktioniert schnell & einfach per Tool

```
## Switch to folder
## example using Backend
cd code/BluetoothProj/BluetoothBackend
## Run using CLI Tool directory containing *.csproj
dotnet run
## or use in other working directory
dotnet run --project <path-to-csproj-file>
```

jegliche Abhängigkeiten werden automatisch installiert, gebuildet und geservt.

An diesem Punkt kann zum Beispiel auf https://localhost:5001/api/antenna navigiert werden für das Backend oder auf https://localhost:8001/Map/ für das Frontend.

6.1.6 Correlator

6.1.6.1 Abhängigkeiten

Installiere pip:

```
sudo apt install python3-pip
```

Installiere die benötigten Module:
Aus dem root Verzeichnis des Repos:

```
cd code/correlation
pip3 install -r requirements.txt
```

Der Correlator kann nun gestartet werden.

Aus der Hilfe (python3 correlator.py -h):

```
usage: correlator.py [-h] [-a] [-m MAC [MAC ...]] [-c] [-p] [-i] [-t TYPE] [db_file]

Bluetooth Signal Correlator

positional arguments:
  db_file               The database file to parse.

optional arguments:
  -h, --help            show this help message and exit
  -a, --all             Correlate all database entries and print result.
  -m MAC [MAC ...], --mac MAC [MAC ...]
                        Mac addresses for which to display information. If none of -c, -p, -i are specified, print correlation.
  -c, --correlation     Print correlation of device. Specify device with -m.
  -p, --path            Print path of device. Specify device with -m.
  -i, --image           Create image of path of device. Specify device with -m.
  -t TYPE, --type TYPE  Disply only devices of specified type. Options are: covid, apple.
```