|  |  |  |
| --- | --- | --- |
| **Thesis** |  |  |
| Im Studiengang:  Technische Informatik |  |  |
| im Bereich |  |  |
| Netzunabhängige lokale Peer-To-Peer-Verbindungen auf mobilen Geräten | | |

# Eidesstattliche Erklärung

Ich versichere, dass ich diesen Bericht zum praktischen Studiensemester selbstständig und nur unter Verwendung der angegebenen Quellen- und Hilfsmittel angefertigt habe. Die Stellen, an denen Inhalte aus den Quellen verwendet wurden, sind als solche eindeutig gekennzeichnet. Die Arbeit hat in gleicher oder ähnlicher Form bei keinem anderen Prüfungsverfahren vorgelegen.

|  |
| --- |
| Mannheim, den 30.05.2020 |
|  |
| Alexis Danilo Morgado dos Santos |

# Danksagung

# Zusammenfassung

Abstract auf deutsch

# Abstract

# Inhaltsverzeichnis

Eidesstattliche Erklärung I

Danksagung II

Zusammenfassung III

Abstract IV

Inhaltsverzeichnis V

1 Einleitung 1

1.1 Zielsetzung 1

1.2 Aufbau der Arbeit 2

2 Anforderungsanalyse 3

2.1 Szenarien 3

2.1.1 Szenario 1 – Messenger 3

2.1.2 Szenario 2 – Shared Touchpoint Canvas 4

2.1.3 Szenario 3 – Bouncing Ball 5

2.1.4 Szenario 4 – Score Board Notepad 5

2.1.5 Szenario 5 – Sichtbarkeitsbeschränkter Nachrichtenaustausch 6

2.2 Funktionale Anforderungen 6

2.3 Nichtfunktionale Anforderungen 7

2.4 Gegenüberstellung von Nearby Connections und MQTT 7

3 Software-Design 13

4 Obligatorische Berechtigungen 15

5 Verbindungsaufbau 17

6 Übertragung von Daten 20

6.1 Senden mit Nearfly 20

6.2 Realisierung mit NearbyConnections 21

6.3 Realisierung mit MQTT 23

7 Evaluation 26

7.1 Zeiten für den Verbindungsaufbau 26

7.2 Zeiten für die Datenübertragung 30

7.3 Verifikation der Anforderungen 32

7.4 Nutzung der Nearfly API 34

Fazit 36

Ausblick 36

Anhang A Ausführlichere Diagramme zum Verbindungsaufbau im NeCon-ConnectionMode 37

Quellenverzeichnis 38

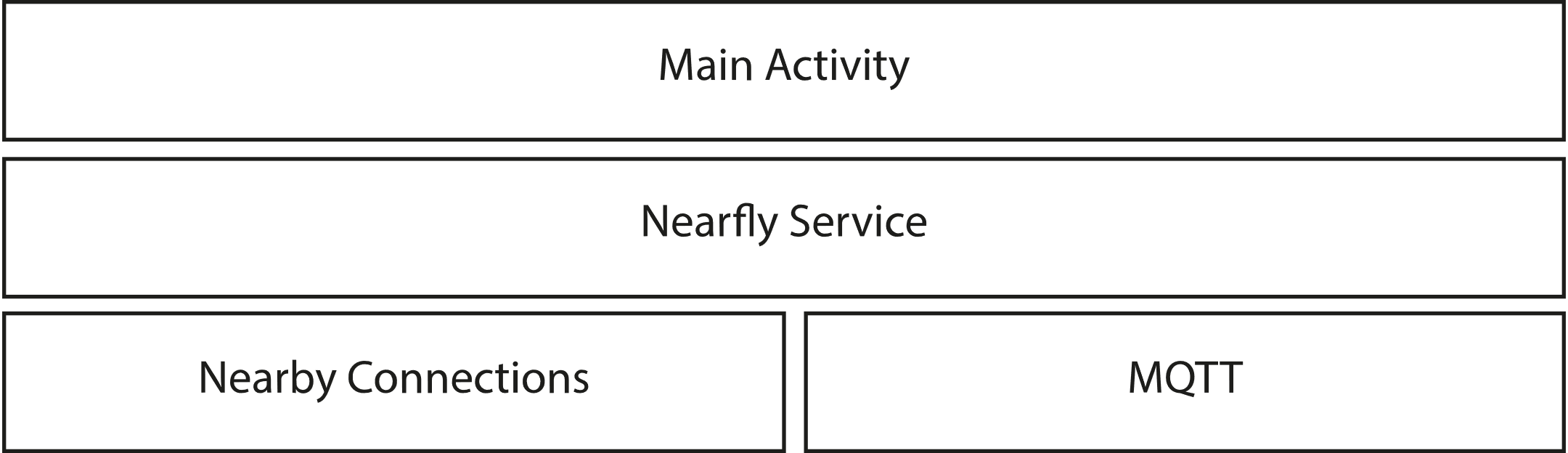
# Einleitung

Die Konnektivität ist der Zeitgeist des 21. Jahrhunderts und als Megatrend die treibende Kraft des graduellen gesellschaftlichen Wandels. Gerade Smartphones, welche der Gesellschaft die Möglichkeit des ständigen Informationsabrufs und unentwegte Disponibilität ermöglichen, tragen eine nicht unwesentliche Rolle zum steigenden Bedarf einer ubiquitären Konnektivität bei. Lange erfüllen diese dem Handy entlehnten mobilen Computer nicht mehr den primären Zweck des telefonieren, sondern werden hauptsächlich zum Nachrichtenaustausch, Browsing, sowie andere Internetgebundene Aktivitäten benutzt [2]. Getrieben wird dieser Trend durch die unzähligen Vorteile, welche die Vernetzung mit sich bringt. Doch existieren auch Situationen, in denen diese Vorteile wünschenswert sind, eine internetgebundene Vernetzung jedoch unangebracht oder schlichtweg nicht möglich ist. Etwa, wenn Multimedia-Daten aus Datenschutzrechtlichen Gründen dezentral transferiert werden sollen, kein Datenvolumen aufgebraucht werden soll oder dringende Wetterwarnungen in Gebieten verbreitet werden sollen, in denen kein Mobilnetz existiert [13]. Im Kontext der Entwicklung wird für die Realisierung dieser vernetzenden Funktionalitäten häufig auf bestehende Technologien, wie etwa MQTT oder die im Juli 2017 von Google veröffentlichte NearbyConnections API, welche das vorhergehende Offline-Szenario erfüllt, zurückgegriffen. Diese werden in Form von Bibliotheken, welche komplexe Sachverhalte abstrahieren und durch eine zu meist anwenderfreundlichen API ansprechbar sind, in das Projekt eingebunden.

## Zielsetzung

Im Rahmen dieser Thesis soll, die NearbyConnection- und die MQTT-Technologie analysiert und eine Android Middleware konzipiert und implementiert werden, welche die Vorzüge beider unterliegenden Technologien in Form einer auf beide Technologien aufsetzenden Ebene vereinheitlicht und dem späteren Android-API Nutzer die Möglichkeit einer generischen Nutzung der vernetzenden unterliegenden Offline-, sowie Online-Lösung bietet (siehe **Abbildung 1**).

Dazu sollen, im Zuge einer Anforderungsanalyse, zunächst die Teilziele konkretisiert und anschließend durch den Einsatz von bekannten Entwurfsmuster eine geeignete Architektur entworfen werden, welche nach der Realisierung validiert und evaluiert wird.



**Abbildung 1:** Abstrahierte drei-Ebenen Darstellung der Nearfly-Bibliothek

## Aufbau der Arbeit

In Kapitel 2 werden im Zuge einer Anforderungsanalyse Szenarien definiert, aus welchen die konkreten Teilziele dieser Thesis ermittelt werden. Des Weiteren werden die anwendungsbezogenen Prozessabläufe beide unterliegenden Technologien näher durchleuchtet und verglichen.

Darauf aufbauend wird in Kapitel 3 durch Einsatz von bekannten Entwurfsmuster eine geeignete Architektur entworfen und im Kapitel 4 vorbereitend die für die Implementierung der Bibliothek im Android-Kontext essenziellen Berechtigungen besprochen.

Kapitel 5 befasst sich mit Implementierung des Verbindungsaufbaus beiden unterliegender Technologien.

Im Kapitel 6 werden geeignete Methoden für den Nachrichtenaustausch der Nearfly-Biblitohek implementiert.

Darauf aufbauend bietet Kapitel 7.4 eine kurze Einführung in die Konkrete Nutzung der Nearfly Bibliothek.

Schlussendlich wird die entwickelte Nearfly-Bibliothek im Kapitel 7 durch Tests hinsichtlich der benötigten Zeiten evaluiert und anschließen die Anforderungen verifiziert, sowie beobachtete Fehler und Mängel kommentiert.

Im Fazit werden danach die Ergebnisse dieser Thesis noch zusammengefasst und ein Ausblick auf Erweiterungsmöglichkeiten gegeben.

# Anforderungsanalyse

Während die Szenarien die Teilziele erfassen, zeigt die Gegenüberstellung der Nearby Connections API und MQTT die zur Verfügung stehenden Mittel und macht deutllich, das eine Assimilieren beider Technologien eine Recht große Anpassung beider APIs, wie etwa die Wahl einer geeigneten Netzwerktopologie für die Nearby Connections API und die Erweiterung der Nachrichtenübertragungsprotokolle fordert.

## Szenarien

Die *Messenger App* soll testen, wie sich das System beim Übertragen von größeren binären Daten verhält, während die *Shared Touchpoint Canvas App*, das System aufgrund der Menge an Touchpoint ausreizen kann und demnach Rückschlüsse auf die Eignung der Nearfly-Bibliothek für Echtzeit-Anwendungen gibt. Als praxisnahes Grenzwert-Szenario für die Übertragungsgeschwindigkeit soll die *Bouncing Ball App* verwendet werden, welche bereits bei vier Nutzer eine Übertragungsgeschwindigkeit von 4\*30FPS verlangt. Das letzte Szenario (Score Board Notepad) soll als Rundenbasiertes Spiel die Ausfallsicherheit und generelle Zuverlässigkeit des Systems bei mittlerer Datenübertragung testen.

### Szenario 1 – Messenger

*Marius ist der Scrum-Master eines fünf-köpfigen Entwicklungsteams für Andorid Anwendungen und damit zuständig für das Beseitigen von Hindernissen (Impediments). Er legt viel Wert auf Datenschutz und möchte am besten keine Daten über das Internet übertragen. Dazu hat dieser eine Gruppe erstellt und fordert seine Teammitglieder auf seiner Gruppe beizutreten. Danach schreibt er an allen einen Begrüßung Text und frag, ob es denn derzeit Probleme geben würde. Während Tommy die Nachricht liest läuft dieser zur Toilette, welche durch die ausgeprägten Wände ein Funkloch ist und antwortet Marius, dass die Kaffee-Pads leer seien. Kurze darauf hängt Tommy ein Foto von den Kaffee-Pads an, mit der Bitte an Marius, neue zu erwerben.*

Das erste Szenario entspricht dem einer klassischen Messenger App, welche es dem Entwickelteam erlaubt offline zu kommunizieren. So soll der Nutzer die Möglichkeit haben, Text- und Multimedia zu versenden. Denkbar ist jedoch auch das Senden von beliebige Binärdaten und damit etwa den Datentransfer von APKs zu ermöglichen. Weiterhin soll durch das Erstellen und Betreten von geschlossenen Benutzergruppen (Chatrooms) eine Isolation innerhalb der App geschaffen werden dürfen, welche die Sichtbarkeit dedizierter Nachrichten auf befugte Nutzer reduzieren. Gesendete Nachrichten sollen zudem im Falle einer getrennten Verbindung nicht verloren gehen, sondern so lange aufbewahrt werden, bis diese versendet werden können.

Da das Entwicklungsteam hauptsächlich Bilder oder Text über das Smartphone übertragen will, zeigt **Tabelle 1** die durchschnittliche Datengröße, der auf drei verschiedenen Smartphones erfassten Multimedia-Daten. Ein bis wenige Seiten große PDFs weisen dabei eine durchschnittliche Größe von 60-800 KByte auf, während Fotos, welche von den Smartphones geschossen werden, teilweise stark von Bilddetail und Smartphone-Modell abhängen. Durchschnitt lassen sich diese im Größenbereich 2-3MByte einordnen. Screenshots hingegen haben bei älteren Smartphones einige Hundert KBytes, umfassen jedoch bei neueren Smartphones vergleichbar den Fotos einen Größenbereich von 1 bis 3 MByte. Da das Team Android Anwendungen entwickelt und dadurch APKs zwischenzeitlich im Rahmen von Systemtests versandt werden müssen, zeigt **Tabelle 1** weiterhin die nach Bosshell durchschnittliche Größe (11,5 MByte) von Android-Anwendungen im Play Store.

**Tabelle 1:** Empirisch ermittelte durchschnittliche Größe von Multimedia-Daten

|  |  |
| --- | --- |
| Datatype | Average file size |
| Documents (PDF) | 60 - 800 KBytes |
| Photos (JPG) | 1 - 5 MByte (4032x3024) |
| Screenshot (PNG) | 400 KByte bis 3 MByte |
| APK | 11.5 MB | 34.3 MB | 38 MB[4] |

### Szenario 2 – Shared Touchpoint Canvas

*Tom, Beni und Jim sitzen im Büro und zeichnen gemeinsam mithilfe ihrer Smartphones auf einer Berührungspunkte-basierten virtuellen Leinwand. Da jeder Benutzer maximal 10 Berührungspunkte setzen kann, die recht schnell wieder ausgeblendet werden, kommt Tom eine Idee. Er fordert seine Freunde auf, gemeinsam mit ihm ein Auto zu zeichnen. Jeder der Freunde sucht sich dazu ein Bereich des Autos aus, das dieser zeichnen möchte und Beni gibt das Start-Signal damit die Berührungspunkte möglichst zeitgleich eingeblendet werden.*

Als kollaboratives Spiel, umfasst das zweite Szenario eine Leinwand, welche gemeinsam von allen beteiligten Benutzern durch antippen bemalt werden kann. Berührt ein Benutzer die Leinwand, entsteht ein kolorierter Berührungspunkt (Touchpoint), welcher nach Ablauf einer gewissen Zeit verschwindet. Da alle gemeinsam Zeichnen wollen, müssen die das System die Daten möglichst zeitnah empfange.

### Szenario 3 – Bouncing Ball

*Mary, Robert und Frederik befinden sich während der großen Pause im Klassenzimmer und versuchen durch Schwenken einer virtuellen Plattform gemeinsam eine Kugel zu balancieren, welche zu Beginn vom Spiel willkürlich angestoßen wurde. Da das Verhalten der Kugel, die Summe der Aktionen aller Spieler ist, müssen alle jeweils die Bewegungen der Mitspieler kompensieren. Nachdem die Kugel erfolgreich zentriert wurde, wird diese erneut einem Impuls ausgesetzt. Dies wiederholt sich einige Male bis die Kugel Letzten Endes nach vier Minuten aus der Plattform fällt und die persönliche Bestzeit auf alle 3 Geräten angezeigt wird.*

Beim dritten Szenario sehr viele Daten innerhalb kürzester Zeit übertragen werden. Dazu müssen die Neigungsdaten aller Spieler kontinuierlich erfassen und die Kugel entsprechend der Summe aller erfassten Neigungsdaten bewegt werden. Zeitgleich fordert das Spiel eine möglichst kleine Latenz, um dem Geschick der einzelnen Spieler nicht entgegen zu wirken und dadurch ein durchgehend synchrones Spielerlebnis zu gewährleisten. Zuletzt dürfen unterschiedliche Systeme, welche die Nearfly-Bibliothek verwenden nicht interferieren, wenn dies vom Entwickler nicht ausdrücklich gewünscht ist.

### Szenario 4 – Score Board Notepad

*Steffan, Mark und Ricky spielen gemeinsam in der Mittagspause Papierwerfen, dazu versucht jeder je Runde mit zehn zerknüllten Papierbällen die in der Ecke stehende Mülltonne zu treffen. Derjenige der als letztes dran war, ist zuständig für das Notieren der Punkte desjenigen Spielers, welcher gerade dran ist. Mark beginnt mit der Rolle des Schreibers, während die zwei anderen Mitspieler kontrollieren, ob Mark die Punkte richtig vergibt. Ricky startet und trifft die Mülltonne. Der Ball berührt zuvor jedoch die Wand, sodass Mark Rickys Punktestand nur um eins inkrementiert. Daraufhin trifft Ricky die Mülltonne ohne, dass der Ball etwas berührt und erhält hierfür von Mark zwei Punkte. Acht weitere Versuche später, wird Rickys Runde beendet. Nun ist Steffan dran und sein Vorgänger Ricky notiert Punkte. Nach Ablauf von 20 Runden, in denen jeder jeweils 10-mal werfen durfte wird das Spiel beendet und der Gewinner bestimmt.*

Beim rundenbasierten Spiel müssen Daten zur Steuerung des Spiels innerhalb dynamischer Zeitintervalle versandt werden. Dabei muss sowohl die Neuzuordnung des Schreibers wie auch das Ermitteln des aktiven Spielers und der Zuschauer erfolgen. Weiterhin muss das System eine einer Smartphone-übergreifende Synchronisation der Punktetabelle gewährleisten.

### Szenario 5 – Sichtbarkeitsbeschränkter Nachrichtenaustausch

*Leonard, Mandy und Kevin befinden sich im Klassenzimmer und spielen in der großen Pause das „Bouncing Ball“-Spiel, während im benachbarten Zimmer, die Lehrer den bevorstehenden Unterricht vorbereiten und sich gegenseitig Bilder und Artikel über die Nearfly Messenger App senden.*

Als Kombination zweier Szenarios zeigt das fünfte Szenario, das der Datenaustausch parallellaufender Anwendungen unterschiedliche Szenarien unsichtbar für die jeweils andere Anwendung ist.

## Funktionale Anforderungen

Die aufgestellte Anforderungsmatrix (siehe

Tabelle 2) zeigt die Korrelation der für die o.g. Szenarien ermittelten Anforderungen. So ist etwa klar festmachbar, dass das Versenden von textbasierten Nachrichten eine fundamentale Anforderung ist, welche in allen der definierten Szenarien zum Senden von Text, Status oder Koordinatendaten benötigt wird und durch die Messenger-App, um das Senden von Binären-Daten erweitert wird. Des Weiteren ergibt aus der Messenger-App aufgrund der angeforderten Isolation für Raum und Chat die Anforderung, dass unterschiedlicher Nachrichten innerhalb derselben Anwendung Trennbar sein sollen. Im Falle eines ungewollten Verbindungsabbruch sollen die Nachrichten zudem gehalten werden und bei einem erneuten Verbindungsaufbau gesendet werden. Dies setzt natürlich voraus, dass die Verbindung im Falle eines Verbindungsabbruchs automatisch wiederaufgebaut wird. Die Echtzeit Fähigkeit, welche im zweiten, sowie dem dritten Szenario benötigt wird, führt zudem zur Anforderung einer möglichst Latenzarmen Datenübertragung, welche genug Durchsatz für ein flüssiges Spielerlebnis (25 FPS) bietet. Zuletzt entsteht aus Szenario fünf, die Anforderung, dass der Nachrichtenaustausch unterschiedliche Anwendungen, welche die Nearfly-Bibliothek verwenden nicht interferieren dürfen, wenn dies vom Entwickler nicht ausdrücklich gewünscht ist.

Tabelle 2: Anforderungsmatrix der Nearfly-Bibliothek

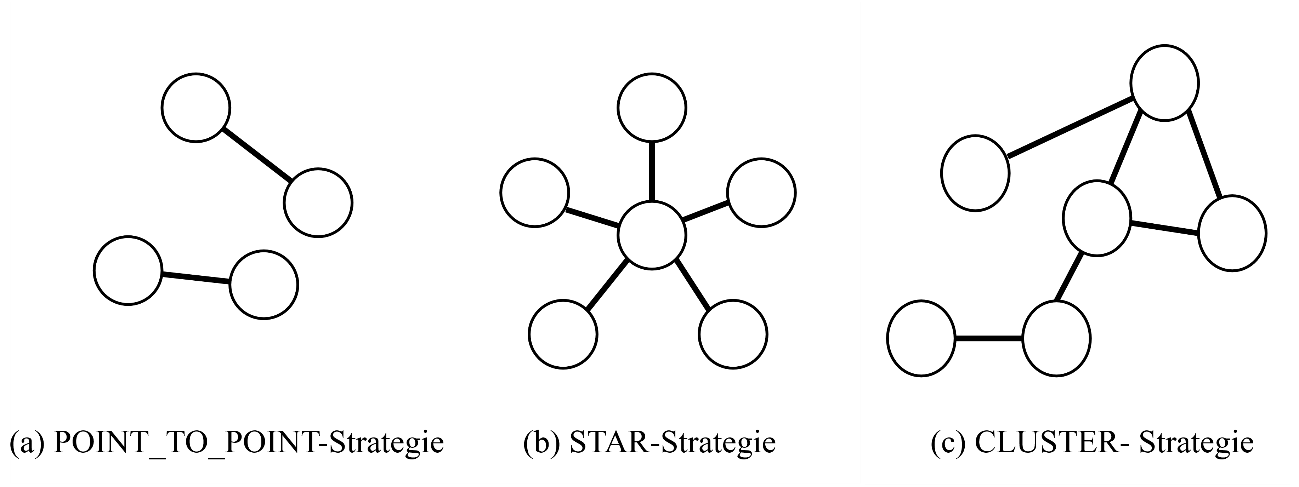
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Funktionale Anforderungen** | Messenger | Shared Touchpoint Canvas | Bouncing Ball | Score Board Notepad | Sichtbarkeitsbeschränkung |
| Textbasierte Nachrichten senden | x | X | x | x |  |
| Multimedia-Nachrichten senden | x |  |  |  |  |
| Priorisierung der Nachrichtentypen | x |  |  |  |  |
| Trennbarkeit unterschiedlicher Nachrichten innerhalb derselben App | x |  |  |  |  |
| Möglichkeit, dass Nachrichten bei getrennter Verbindung gehalten werden | x |  |  |  |  |
| automatischer Verbindungsaufbau | x |  |  |  |  |
| Die Nachrichten müssen den Absender enthalten | x | x | x | x |  |
| Verzögerungsarmes Empfangen der Nachrichten |  | x | x |  |  |
| Echtzeitfähigkeit für ein flüssiges Multiplayer-Spiel-Verfalten bei 30 FPS pro Spieler |  |  | x |  |  |
| Sichtbarkeit der Nachrichten nur innerhalb derselben App | x | x | x | x | x |
| Wechselt der Nutzer von Online- in den Offline-Modus sollen keine Daten keine Daten verloren gehen |  |  |  |  |  |

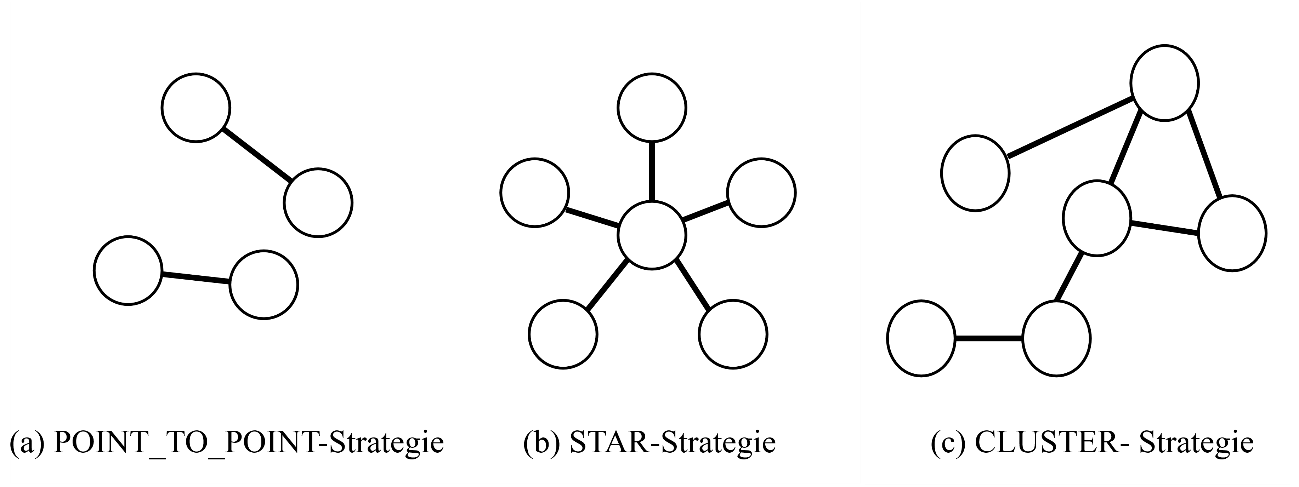
## Nichtfunktionale Anforderungen

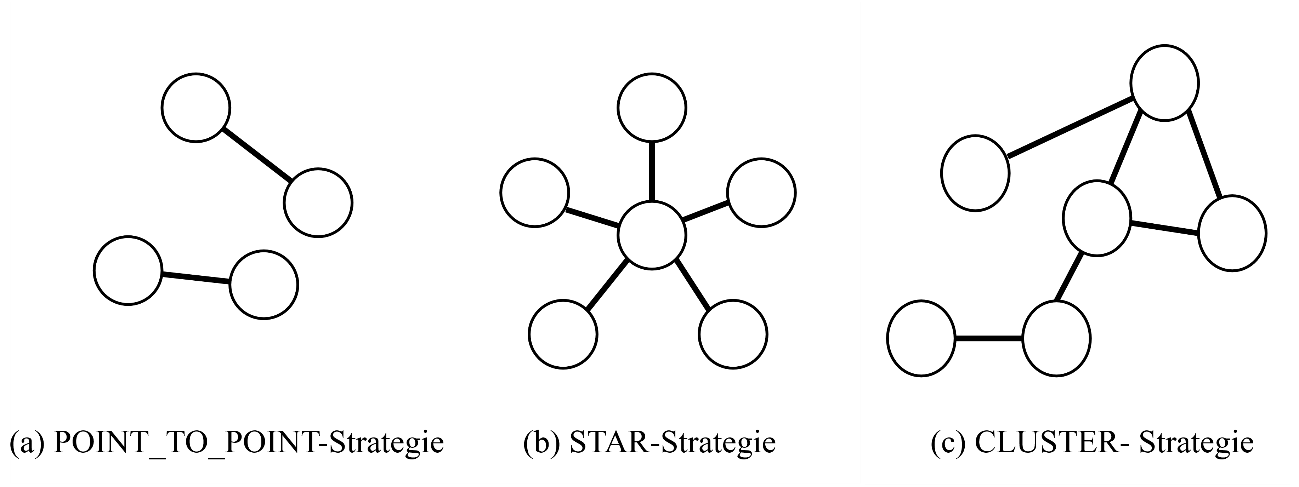
Die zu entwickelnde Bibliothek sollte eine Kompatibilität zu allen Android Versionen ab dem API Level 24 (Android 7.0) aufweisen. Weiterhin sollte das System sowohl im Nearby-Modus wie auch im MQTT-Modus ein ähnliches Verhalten in Bezug auf den Verbindungsaufbau, das Senden und dem Empfangen der Nachrichten aufweisen.

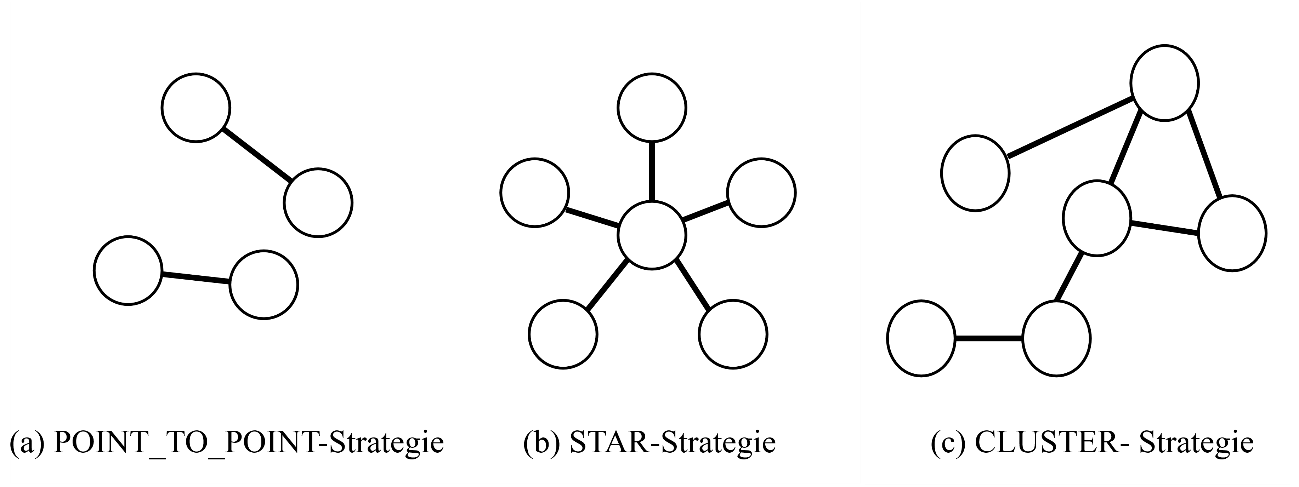
## Gegenüberstellung von Nearby Connections und MQTT

Nearby Connections wird vom Nearby Connection Team als eine *Peer-To-Peer-API* mit hoher Bandbreite und kleiner Latenz beschrieben, welche durch eine optionale Verschlüsselung einen sicheren Datentransfer zwischen den verbundenen Knoten ermöglicht [11]. Verbindungen werden dabei durch Nutzung von Bluetooth, Bluetooth-Low-Energy (BLE) und automatisch angelegten WiFi Hotspot hergestellt [11] und sorgen damit für eine beinahe vollständig automatische offline Vernetzung. Um dies zu ermöglichen besitzt die API die Berechtigung sowohl WiFi wie auch Bluetooth einzuschalten und diese beim Beenden wieder in ihren Ausgangszustand zu versetzen [11].

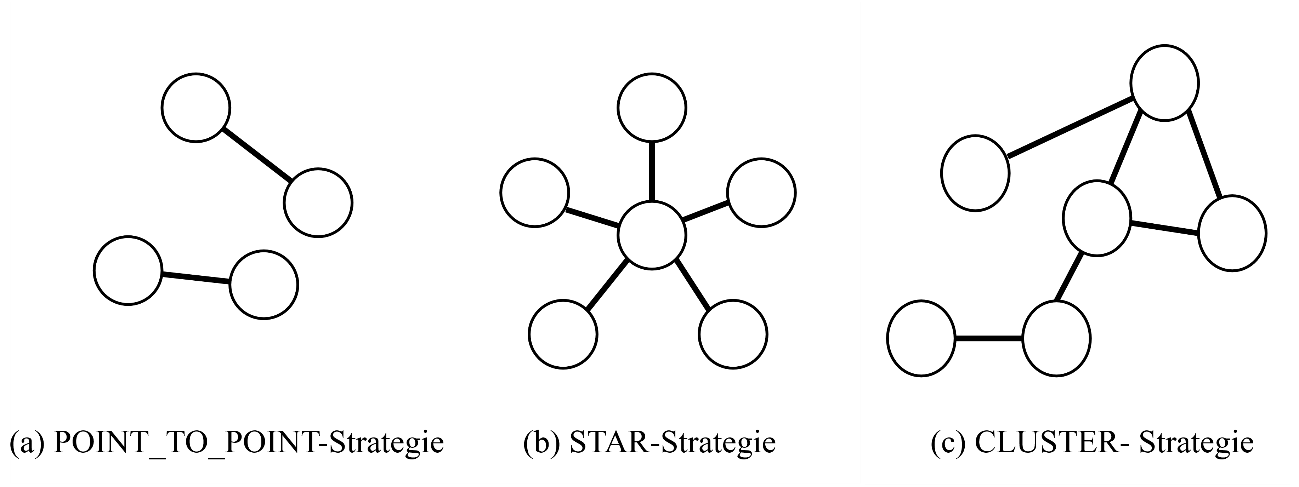
Vor dem eigentlichen Verbindungsaufbau muss eine Netzwerktopologien gewählt werden, welche sich ebenfalls auf die maximale Übertragungsrate im Netzwerk auswirkt. Nearby Connections bietet drei verschiedene Topologien (genannt Strategien), wie in 

**Abbildung 2** gezeigt, an. Die Erste nennt sich CLUSTER-Strategie (****

**Abbildung 2**a) und erlaubt jedem Knoten im Netzwerk beliebige Verbindungen anzunehmen. Die damit entstandene N-M Topologie unterstützt jedoch nur Bluetooth und ist damit in der Bandbreite erheblich eingeschränkt. Zudem ist zu beachten, dass die realistische maximale Anzahl gleichzeitig verbundener Geräte von Bluetooth auf 3-4 limitiert ist [6]. Durch die STAR-Strategie (****

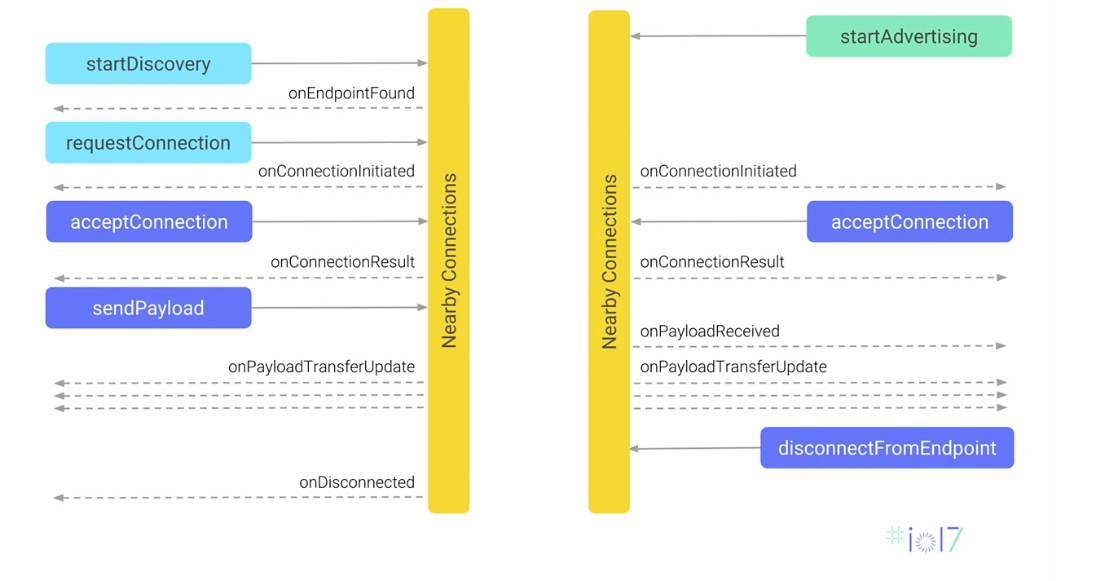
**Abbildung 2**b) hingegen wird eine 1-N Topologie ermöglicht, welche zusätzlich zum Bluetooth ein WiFi -Hotspot verwenden kann und damit deutlich mehr Bandbreite zulässt. Sobald ein Netzwerk zustande gekommen ist, versucht die API das Netzwerk auf ein WiFi -Hotspot upzugraden [8]. Das Netzwerk kann dann bis zu 7 Knoten aufnehmen [6]. Zuletzt existiert die POINT\_TO\_POINT-Strategie (****

**Abbildung 2**c), welche im Verhalten der STAR-Strategie entspricht, mit dem Unterschied das jedem Knoten nur noch eine Verbindungsknoten gestattet wird.



**Abbildung 2:** Netzwerktopologien (Strategien), die von Nearby Conenctions angeboten werden.

Um das Erkennen der Geräte untereinander zu ermöglichen, verwendet die Nearby Connection API ein Advertising/ Discovery-Verfahren. Das *Advertising* ist ein passiver Zustand, bei dem ein Gerät gefunden werden kann, während das *Discovering* eine aktive Suche nach *Advertisern* initiiert, welche den Batteriekonsum deutlich steigert. Ein Gerät kann dabei sowohl einen, wie auch beide Zustände zeitglich einnehmen [11], was jedoch zu *trashing* führen kann[7] Sendet ein Gerät im *Discovering*-Zustand eine Verbindungsanfrage, welche vom *Advertiser* bestätigt wird, wird eine symmetrische Verbindung (siehe **Abbildung 3**) zwischen beiden Knoten initialisiert. Eine Authentifizierung ist standardmäßig nicht vorimplementiert, kann jedoch nachträglich vom API Nutzer z.B. als Popup-Anfrage, welche den Endpunkt-Namen des Gerätes nennt und die Verbindung nur nach erfolgreicher Bestätigung instanziiert, implementiert werden. Sind mindestens 2 Geräte verbunden können entsprechend **Abbildung 3**, durch Angabe eines Empfängers in Form einer *EndpointName*, Daten ausgetauscht werden. Der Datenaustausch verläuft dabei stehts verschlüsselt [11] und wird in 3 Nachrichtentypen unterschieden. Diese sind einerseits Byte-Nachrichten, welche auf 32KByte limitiert sind und andererseits die zwei unlimitierten Datentypen File und Stream [11].



**Abbildung 3:** Nachrichtenverlauf der Nearby Connections API

MQTT hingegen wird in der OASIS Spezifikation als ein leichtgewichtiges, leicht benutzbares Client Server *Pubish/Subscribe* Nachrichten-Protokoll beschrieben, welches wenig Bandbreite benötigt und einen kleinen *Code*-*Footprint* hinterlässt [3]. Im Gegensatz zu Nearby Connections setzt MQTT eine TCP[[1]](#footnote-1) Verbindung zum Server (Broker) voraus, welcher die Verbindungen und Nachrichtenübermittlung verwalteten und die Clients voneinander entkoppelt, sodass diese nie direkt miteinander verbunden sind. [9]

Hat ein Client die Verbindung zum Broker hergestellt, kann dieser Nachrichten publishen, sowie Subskriptionen auf gewisse *Topics[[2]](#footnote-2)* (z.B. „Test/topic“) durchführen. Dabei müssen Nachrichten stehts ein *Topic*, wie auch ein Byte Payload unter 256MByte [14] haben, sodass der Broker die Nachrichten an Subskribierte Clients weiterleiten kann. [9].

Die MQTT Spezifikation sieht dabei keine Verschlüsselung der Nachrichten vor, um das Protokoll möglichst simple zu halten [3], jedoch wird dies in vielen Implementierungen, wie auch der zu verwendenden Paho-Client-Bibliothek unterstützt.

Zudem erhalten Nachrichten Angaben wie *Quality-Of-Service* (QoS) und ein *Retain-Flag*, welche dafür sorgt, dass der Server die letzte Nachricht speichert und diese fortan an allen *Subscriber* des zugehörigen Topics beim erfolgreichen Verbindungsaufbau, sendet [3].

**Tabelle 3:** Überblick über Unterschiede zwischen Nearby Connections und MQTT [11]

|  |  |  |
| --- | --- | --- |
| Technologie | MQTT | Nearby Connections |
| Netzwerktopologie | Server/Client | Cluster, Star oder Point-to-Point |
| Verbindung | Persistent zum Server | Persistent zu den einzelnen Peers |
| Verwendetes Protokoll | TCP/ IP | Bluetooth, BLE, WiFi Direct |
| Übertragungsgeschwindigkeit  (theoretisch) | WiFi: max. 250 Mbit/s | Bluetooth 3.0: max. 24 Mbit/s  WiFi Direct: max. 250 Mbit/s |
| Nachrichtenaustausch | Publish (1: N) | Bidirektional One-Way (1: 1) |
| Authentifikation | Namen und Passwort-Feld | Verbindungsanfrage an Nutzer  (Optional implementierbar) |
| Nachrichtenverschlüsselung | TLS möglich | durchgehend |
| Payload-typ | Byte | Byte, Stream oder File |
| Adressant | Topic-Subscriber | EndpointName |
| Max Packet-Größe | Maximal 256MB (Seitens des Brokers einstellbar) | Bytes: 32KB  Files&Stream: unbegrenzt |

Auch ist der Vergleich beider Technologien im Bezug auf die Datenrate sinnvoll, da Nearby Connections im Bezug zu MQTT, nicht ständig auf das WiFi-Protokoll zurückgreift, sondern als Hybrid-Technologie zunächst durch Bluetooth kommuniziert und erst bei günstiger Gelegenheit wechselt. Im Grunde schaffen alle Chips ab Bluetooth Version 3.0 durch Kombination von Bluetooth und WLAN-Übertragungstechnik einen zusätzlichen Highspeed-Kanal, welcher eine theoretische maximale Übertragungsgeschwindigkeit von 24Mbit/s, z.B. zum Übertragen von Fotos, Videos oder Musik zulässt [10]. Ab Version 4.0 ist in den Chips zusätzlich Bluetooth Low Energy (BLE) integriert, welcher eine deutlich weniger Strom konsumiert, jedoch nur maximal 220kBit/s und eine eingeschränkte Reichweite von 10 Metern besitzt [15]. Mit Version 4.2 ist die BLE-Datenrate im Vergleich zur Vorgängerversion, um Faktor 2.5 erhöht und die Übertragung sicherer, während mit Bluetooth 5.0 je nach Anforderung einen verdoppelter Bruttodurchsatz von 2Mbit/s oder eine Reichweite von bis zu 200m durch BLE erzielen werden kann. [16].

Der tatsächliche Durchsatz liegt jedoch, wie Evan Fallis und Petros Spachos durch das Testen von vier verschiedenen Smartphones mit Bluetooth 4.0 und 4.1 zeigt, mit 1 Mbit/s weit unter den theoretischen Angaben [1]. Auch zeigen die Messungen dieser beiden, das die WiFi-Übertragung via Smartphone mit einem durchschnittlichen Durchsatz von 28.95 Mbit/s und Extrema von 25.7 Mbit/s, sowie 37.8 Mbit/s im clientseitigen Übertragen von TCP-Paketen[1], die theoretische Angabe bis zu 250 Mbit/s (im Kontext der Smartphones) weit verfehlt.

# Software-Design

Wie bereits bekannt soll die Nearfly-Bibliothek ein Wrapper der Nearby Connection- und MQTT-API sein. Für das Benutzen der Nearfly-API soll die Nearfly-Bibliothek zwei Möglichkeiten anbieten. Zum einen kann ein der NearflyClient instanziiert und die Methode, wie gewöhnlich aufgerufen werden. Zum anderen kann die Nearfly-API über einen Android-Service (genannt NearflyService), welcher selbst als Wrapper-Klasse (wie in **Abbildung 4**) den NearflyClient innehält, benutzt werden. Das Ansprechen der Nearfly API über den NearflyService, soll dabei jener Weg sein, welcher in dieser Thesis bevorzugt benutzt wird und hat den Vorteil, dass der Service nach dem einmaligen Starten solange, wie die eigentliche App im Hintergrund läuft. Wird innerhalb derselben Anwendung die *Activity* gewechselt, bleibt die Verbindung zum Netzwerk bestehen. Da mehrere Activities auf den NearflyService zugreifen können, soll die Kopplung zwischen den Activities und dem NearflyService durch Anbieten der Möglichkeit zur An- und Abmeldung beliebiger NearflyListener, verringert werden. Die NearflyListener ermöglichen dabei das Reagieren auf bestimmte Events seitens der Nearfly-Bibliothek, wie etwa das Erhalten von Nachrichten.

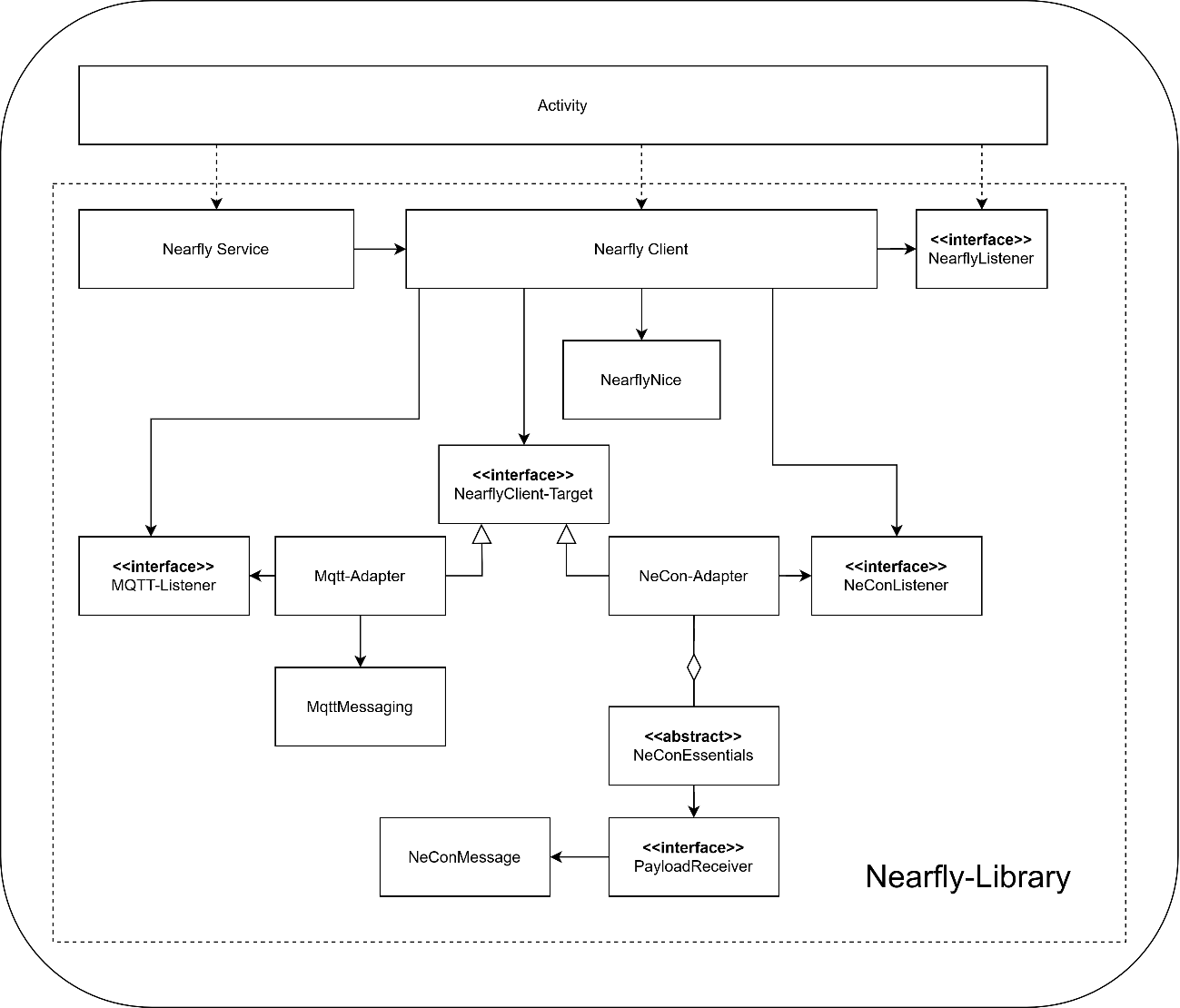
Der NearflyClient soll die unterliegenden NearbyConnections- und MQTT-API über dedizierte Adapter, welche die Anfragen per Delegation an die jeweiligen APIs weiterleiten und den NearflyClientTarget implementieren, ansprechen. Das NearflyClientTarget-Interface definiert die Methoden, wie das Senden von Nachrichten, welche der NearflyClient benötigt.

Die auf die Nearby Connections (NeCon) API aufbauenden Funktionalitäten sollen in zwei verschiedenen Komponenten (wie in **Abbildung 4**) ausgelagert werden. Während die NeConEssentials-Komponente die Grundfunktionalitäten für eine korrekt funktionierende Nearby Connections API, wie etwa das Versetzen der Endpunkte in den *Advertising*-/*Discovering*-Zustand oder das Verwalten von entdeckten, akzeptierten sowie schwebenden (pending) Verbindungen, bietet. Soll die NeConAdapter-Komponente hauptsächlich die Funktionalitäten anbieten, welche speziell für die Nearfly-Bibliothek benötigt werden. Dazu gehört unteranderem das autonome Aufbauen eines lokalen Peer-to-Peer Netzwerkes aus den nahliegenden Knoten, sowie die für die Datenübertragung notwendigen Funktionalitäten. Zwischen der NeConAdapter- und der NeConEssentials-Komponente soll dabei die Abhängigkeit einer Basis-/Kind-Klasse bestehen.

Um auf eintreffende Nachrichten zu reagieren, soll die NeConEssentials-Komponente weiterhin eine Abhängigkeit zur PayloadReceiver-Komponente, welche eine Erweiterung des von Nearby Connections API angebotenen primitiven PayloadCallback-Listerners ist, besitzen.

Auch sollen die Nutzung der MQTT API über zwei Komponenten verteilt werden. Während die MqttMessaging-Komponente jene ist, die das Funktionieren von MQTT sicherstellt, werden in der MqttAdapter-Komponente die für die Nearfly-Bibliothek benötigten Funktionalitäten, wie etwa das Senden von Multimedia umgesetzt.

Gleichend der Kommunikation zwischen Nearfly-Biblitohek und Activity, sollen die Adapterklassen durch die Nutzung dedizierte Interface-Komponenten, dem NearflyClient den Eingang von Nachrichten oder Statusänderungen signalisieren.



**Abbildung 4:** System-Architektur der Nearfly API

# Obligatorische Berechtigungen



Apps brauchen in Android statische und dynamische Berechtigung für alle Operationen, welche andere Apps, das Betriebssystem oder den Benutzer nachteilig beeinflussen können [5]. Die Berechtigungen werden dabei in drei Risikogruppen (genannt *protection level*) unterteilt. *Normal-*Berechtigungen können in das Manifest eingetragen werden und werden daraufhin einmalig beim Installieren der App abgefragt. *Signature*-Berechtigungen hingegen müssen nicht explizit abgefragt werden, wenn diese bereits einer App gleicher Signatur gewährt wurden. Zuletzt existieren die *Dangerous*-Berechtigungen, wie etwa das Lesen von Benutzerkontakten [5]. Diese müssen ab Android 6.0 (API level 23) jeweils explizit von der Anwendung durch implementieren einer Abfrage, zur Laufzeit abgefragt werden [5]. Während MQTT eine *Normal*-Berechtigung zum Öffnen von Internet Sockets (wie in **Tabelle 4**) braucht, erfordert Nearby Connections inkl. der Möglichkeit zum Senden von binären Daten insgesamt acht verschiedene Berechtigungen. Zu den *Normal*-Berechtigungen zählen zwei Berechtigungen zum *Discovern*, sowie Koppeln von Geräten via Bluetooth (wie in **Tabelle 4**) und zwei Berechtigungen zum Auslesen der WiFi-Informationen, sowie Ändern des Wi-Fi-Konnektivitätsstatus, welche für den Hotspot verwendet werden (wie in **Tabelle 4**). Die restlichen vier obligatorischen Berechtigungen gehören zum *Protection-Level* *Dangerous* und umfassen zwei Berechtigungen zum Abfragen vom ungefähren (FINE\_LOCATION in **Tabelle 4**), wie auch dem genauen Standort (COARSE\_LOCATION in **Tabelle 4**) des Gerätes und zwei Berechtigungen für das Lesen und Schreiben von Daten außerhalb des Anwendungsspezifischen Verzeichnisses. Hierbei ist zu beachten, dass die Leseberechtigung automatisch beim Erteilen der Schreibberechtigung gewährt wird. Dies gilt ebenso für das Ersteilen der FINE\_LOCATION-Berechtigung, welche den Zugriff auf den Dienst, welcher durch COARSE\_LOCATION benutzt werden kann, inkludiert. Damit zeigt sich das die Neafly-Bibliothek insgesamt neun Berechtigungen benötigt, von denen vier zur Laufzeit abgefragt werden müssen, jedoch dem Endnutzer aufgrund der genannten Abhängigkeiten nur als zwei Abfragen angezeigt werden.

**Tabelle 4:** Obligatorische Berechtigungen zur Nutzung der Nearfly-Bibliothek

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | BLUETOOTH | BLUETOOTH\_ADMIN | ACCESS\_WIFI\_STATE | CHANGE\_WIFI\_STATE | ACCESS\_COARSE\_LOCATION | ACCESS\_FINE\_LOCATION | INTERNET | READ\_EXTERNAL\_STORAGE | WRITE\_EXTERNAL\_STORAGE | ACCESS\_MEDIA\_LOCATION | ACCESS\_COARSE\_LOCATION |  |
|  |
| Nearby Connections | X | X | X | X | X | X |  | (X) | (X) | X | X |  |
| MQTT |  |  |  |  |  |  | X |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| (X) optional | □ | Normal permission | | | |  | ■ | Dangerous permission | | | | |

# Verbindungsaufbau



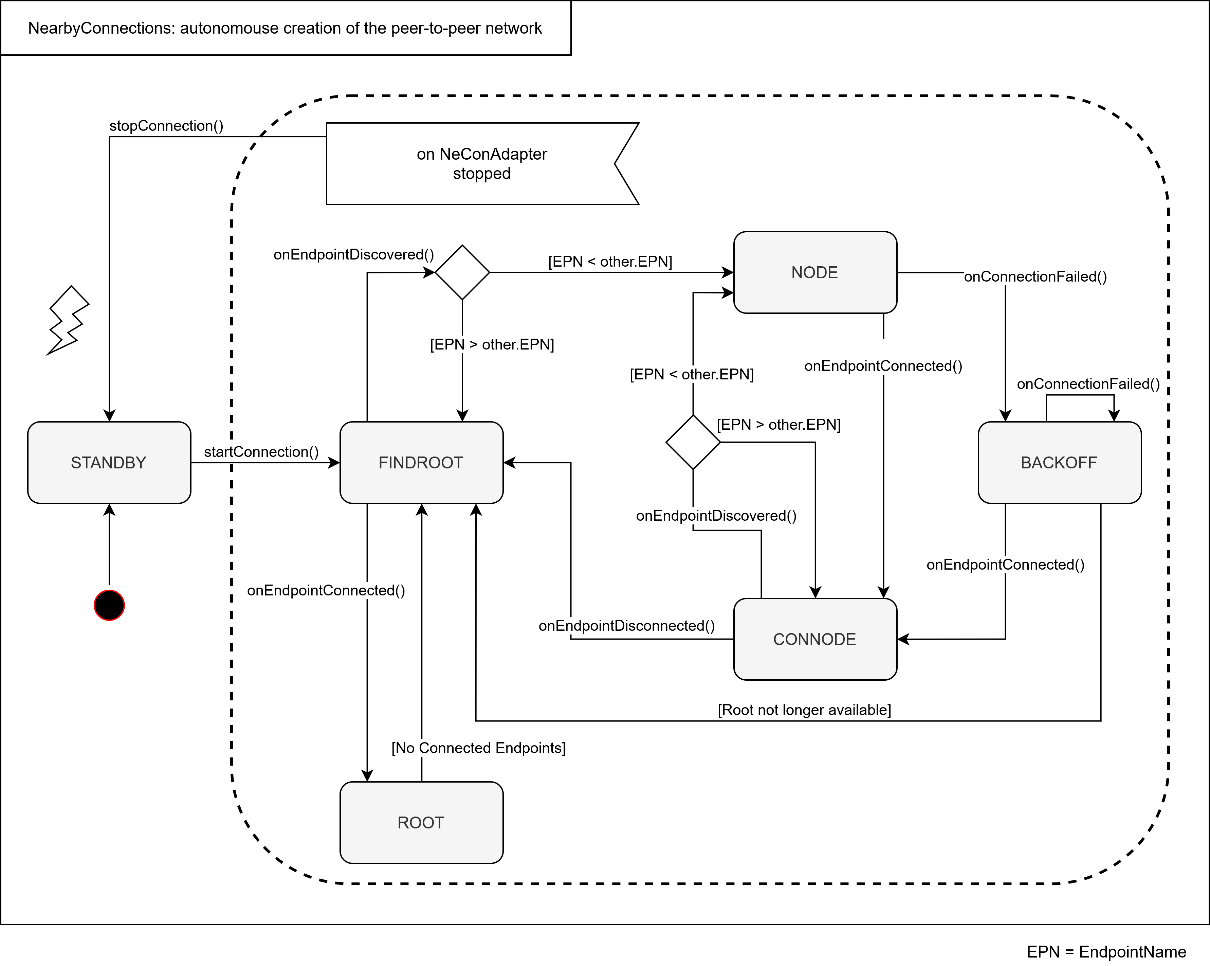
Der Entwickler muss sich vor dem Verbindungsaufbau auf die zu benutzende Technologie (durch Setzen des ConnectionMode) festlegen[[3]](#footnote-3). Entsprechend der Anforderung, einer Sichtbarkeitsbeschränkung der Nachrichten außerhalb derselben Anwendung, soll dabei vor dem Verbinden eine für die Anwendung eindeutige textbasierter Identifikator (genannt Room-String) festgelegt werden. Anwendungen, die denselben Room-String verwenden, können dabei miteinander kommunizieren. Technisch wird dies seitens MQTT über das Eintragen des Room-Strings als Top-Level Topic beim *subscriben* und *publishen* realisiert. Nearby Connections bietet hierfür eine sogenannte SERVICE\_ID an. Diese wird beim Starten vom *Advertising* oder *Discovering* angegeben und fortan zur Sichtbarkeitsbeschränkung von Knoten ungleicher SERVICE\_IDs verwendet. Nach Eintragung des Room-Strings kann nun die Verbindung aufgebaut werden. Durch die Server/Client Architektur genügt dabei seitens MQTT das Aufrufen der korrekt parametrisierten Connect-Methode, durch welchen eine indirekte Verbindung zu allen anderen Broker-Clients entsteht (direkt nur zum Server). Anders sieht es bei Nearby Connections aus, welche als Peer-to-Peer API das sukzessive Aufbauen eines Netzwerkes durch Anfragen der zu verbindenden Knoten fordert. Zur Bewältigung diese Probleme wurden zwei Möglichkeiten gefunden. Die erste Möglichkeit ist das Erstellen eines Overlay-Netzwerkes mithilfe einer *Shorcut*-freien zirkularen distributed Hash-Tabelle, welche mit einer Komplexitätsklasse von O (1) sehr gut skalierbar ist. Jedoch führt dies dazu, dass zwangläufig die CLUSTER Topologie gewählt werden muss, die wie bereits erwähnt auf Bluetooth limitiert ist und damit zu einem vergleichsweise niedrigeren Datentransfer führt. Zusätzlich entsteht relativ viel Overhead beim Verbindungsaufbau, wie auch beim Austreten eines Knotens (*Peer Churn)*. Um die Möglichkeit des *WiFi-Direct* nutzen zu können, muss also ein sternenförmiges Netzwerk angestrebt werden. Dies hat den Vorteil einer overheadärmeren, stärkere Datenübertragung, die besonders bei niedriger Teilnehmeranzahl (drei bis vier) gut funktioniert. Diese zweite Möglichkeit birgt jedoch auch nicht unwesentliche Schwächen. Zum einen wird der zentralen Knoten (Root-Knoten) stärker belastet und die Last nichtmehr verteilt. Zum andren bildet der Root-Knoten als verbindende Einheit aller Netzwerkteilnehmer, einen *Single point of failure*. Fällt der Root-Knoten weg, muss das Netzwerk wiederaufgebaut werden. Auch ist zu beachten, dass das Netzwerk, wie bereits bekannt, trotz WiFi-Hotspot durch Bluetooth auf sieben Teilnehmer beschränkt wird.

Um Gebrauch vom der WiFi-Direct-Technologie zu machen, wird eine sternenförmige Netzwerk-Topologie verwendet. Da jedem Gerät während des *Discovering*s, der Endpoint-Name aller entdeckter *Advertiser* angezeigt wird, muss dieser über den Endpoint-Namen entscheiden, welches Gerät als Root-Knoten dienen kann. Dazu soll der Endpoint-Name entsprechend **Fehler! Verweisquelle konnte nicht gefunden werden.** aus der maximalen CPU Frequenz in Megahertz () des Gerätes und der Differenz zwischen der absoluten Tageszeit und aktuellen Tageszeit zusammengestellt werden, wobei die Gesamtstunden eines Tages[[4]](#footnote-4) in Millisekunden und die Differenz der Millisekunden ausgehend von 0 Uhr (Mitternacht) meint. Dabei dient der Bewertung des jeweiligen Gerätes. Durch soll sichergestellt werden, dass leistungsfähigere Geräte, welche dem Netzwerk später beitreten, nichtmehr als Root dienen können. ist dabei eine Konstante, welche empirisch ermittelt wurde und festsetzt, wie verspätet ein leistungsfähigeres Gerät dem Netzwerk beitreten kann, ohne auf die Root-Position verrichten zu müssen.

|  |  |
| --- | --- |
|  | () |

Der in **Abbildung 4** gezeigte NeConAdapter soll 6 Zustände (siehe **Abbildung 5**) benutzen. Während der NeConAdapter nicht vom NearflyClient verwendet wird, soll dieser im STANDBY-Zustand verweilen, andernfalls wechselt dieser in den FINDROOT-Zustand. Während des FINDROOT-Zustandes wird sowohl das *Advertising,* wie auch das *Discovering* aktiviert. In dieser Phase prüft jeder Knoten beim Entdecken (onEndpointDiscovered) eines anderen Knotens durch Vergleichen der Endpoint-Namen, ob dieser kleiner ist. Bewahrheitet sich diese Bedingung wechselt dieser in den NODE-Zustand und deaktiviert zeitgleich das Advertising, sodass dieser für andere nicht mehr sichtbar ist. Nach Ablauf der INITIAL\_DISCOVERY\_TIME wird von jedem NODE-Knoten eine Verbindungsanfrage an den Knoten mit dem größten entdeckten Endpoint-Namen gesendet. Um die Wahrscheinlichkeit von zeitgleichen Anfragen an den Root zu senken, müssen die Knoten zuvor eine zufällige Zeit innerhalb eins festgelegten Intervalls (RANDRANGE\_COLAVOID) warten.

Bemerkt ein Knoten A eine eingehende Verbindunganfrage (onConnectionInstantiated) von Knoten B und befindet sich dieser nicht im Verbindungsaufbau mit einem anderen Knoten C, akzeptiert dieser diesen. Nachdem die Verbindung erfolgreich aufgebaut wurde (onEndpointConnection), wechselt Knoten A in den ROOT-Zustand und deaktiviert das *Discovering*, da dieser (als Root-Knoten) nunmehr gefunden werden will. Zeitgleich wechselt Knoten B vom NODE-Zustand in den CONNODE-Zustand. Befindet sich Knoten A gerade hingegen im Verbindungsaufbau, wird die Verbindungsanfrage von Knoten B abgelehnt (onConnectionFailed). Der anfragende Knoten B (im NODE-Zustand) verfällt daraufhin in den BACKOFF-Zustand und wartet innerhalb eines gegebenen Zeitfensters eine zufällige Zeit (BACKOFF\_TIME), bis dieser daraufhin eine erneute Verbindungsanfrage lossendet. Da der Verbindungsaufbau auch dann fehlschlagen kann, wenn der Root-Knoten aufgrund eines Netzwerkaustritts (*peer churn*) nichtmehr erreichbar ist, wechselt Knoten B nach MAX\_FAILATTEMPTS gescheiterten Anfragen in den FINDROOT-Zustand und löscht seinen Zwischenspeicher (u.a. *pending Endpoints*). Haben sich alle Knoten zum Root-Netzwerk verbunden, besteht das Netzwerk. Will nun ein neues Gerät am bestehenden Netzwerk teilnehmen, sieht dieser nur noch den Root-Knoten, welcher als einziger das *Advertising* aktiviert hat und versucht sich wie o.g. zu diesem zu Verbinden. Will ein Knoten dem Netzwerk beitreten und hat aufgrund der eingestellten einen größeren Endpoint-Namen als der aktuelle Root, verbinden sich alle Netzwerkteilnehmen nach und nach, gefolgt vom alten Root, zu diesen. Je nach Leistungsunterschied zwischen aktuellen Root und neuem Root, beträgt diese Zeit aktuell ca. 10 bis 40 Sekunden. Nach Ablauf von 60 Sekunden deaktivieren alle Knoten im CONNODE-Zustand das *Discovering*, um den Akku zu schonen. Trennt der NearflyClient die Verbindung zum NeConAdapter, kehrt der Knoten zum STANDBY-Zustand. Dabei wird das *Advertising* und *Discovering* deaktiviert, sowie bestehende Verbindungen getrennt und aufgeräumt.



**Abbildung 5:** Zustandsdiagramm vom Verbindungsaufbau des NeConAdapters

# Übertragung von Daten



Die Datenübertragung des NearflyClients wird durch einen Kanal, eine Priorität, sowie eine Flag zum Puffern der Nachrichten initialisiert. Dieser Ruft entsprechend dem aktuellen ConnectionMode die jeweilige Adapter-Komponente auf, welche die fragmentierte Datenübertragung durch Nutzung des NeCon-Nachrichtenprotokoll mit unterschiedlichen Feldern ausführen.

## Senden mit Nearfly

Um der Anforderung, welche die Trennbarkeit unterschiedlicher Nachrichten innerhalb derselben Anwendung fordert nachzukommen, muss vor dem ersten Nachrichtenaustausuch mindestens ein Kanal (*Channel*) abonniert werden. Das Prinzip gleicht dabei dem der MQTT Topics. Wird ein Kanal abonniert, kann fortan nach Übergabe eines NearflyListeners auf alle Nachrichten, welche denselben Kanal adressieren reagiert werden. Umgekehrt werden nicht abonnierte oder deabonnierte Kanäle ignoriert. Der NearflyListener unterscheidet hierbei drei Nachrichtentypen. Zum einen können Status-Events (onLogMessage) der Adapter-Komponenten, wie die Zustandsänderungen seitens des NeConAdapters oder der erfolgreiche Verbindungsaufbau wahrgenommen werden. Zum anderen kann auf Multimedia-Nachrichten (onFile) und Binary-Nachrichten (onMessage) reagiert werden.

Besteht eine Verbindung zum Netzwerk, können Daten mithilfe eines der zwei vom NearflyClient angebotenen Methoden (pubIt und pubFile) übertragen werden. Während kleinere Byte-Daten bis zu 30KBytes (MAX\_PUBIT\_MESSAGE\_SIZE) durch pubIt versandt werden können, müssen größeren multimediale binäre Daten mithilfe der pubFile-Methode übertragen werden. Dabei entsteht die Grenze aus der bereits aus dem Kapitel 2.4 bekannten Limitierung für Byte-Nachrichten der Nearby Connections API von 32KBytes und einer großzügigen Toleranz, um größere zukünftige Erweiterungen des NeCon-Protokolls (welcher im folgenden Unterkapitel 6.2 besprochen wird) zu kompensieren. Zusätzlich zur zwingend erforderlichen Angabe des Empfangskanals, können die Methoden mit einem Prioritätslevel von -20 (niedrigste Priorität) bis 19 (höchstmögliche Priorisierung), sowie ein Retain-Flag parametrisiert werden.

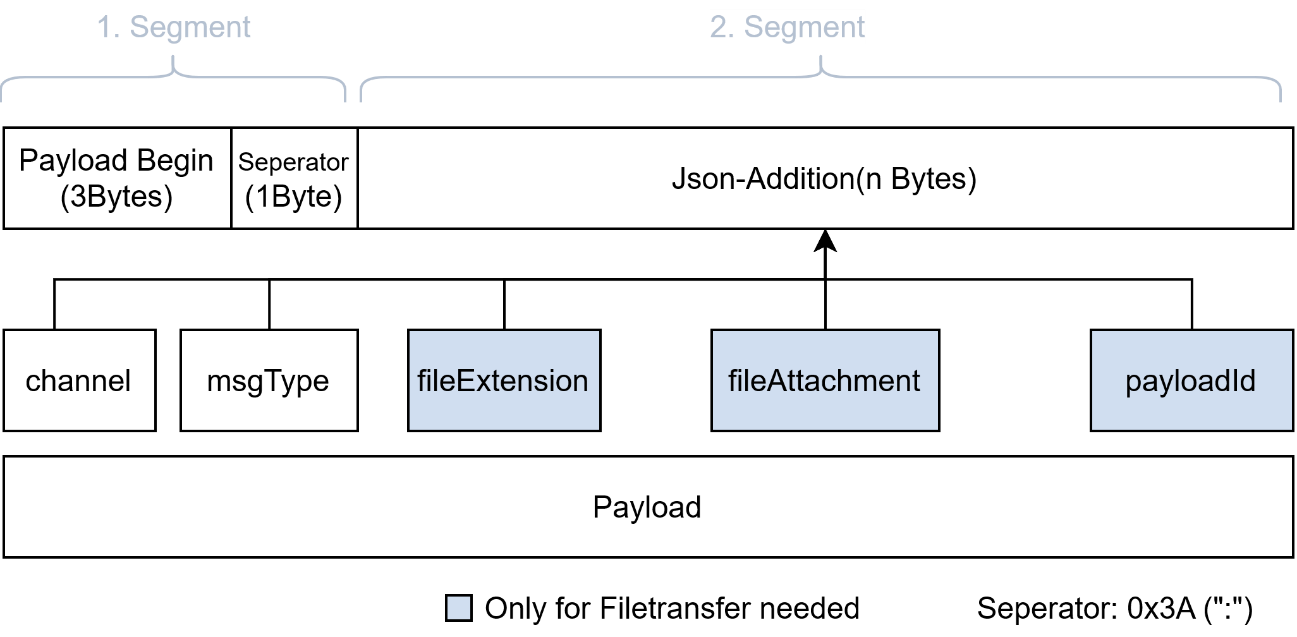
Ein gesetztes Retain-Flag meint dabei, das Nachrichten selbst im Verbindungslosen Zustand „gesendet“ werden können. Dazu gelangen diese zunächst in einen prioritätenbasierten Sendepuffer und werden erst dann gesendet, wenn eine Verbindung zur unterliegenden Technologie besteht und der ModeSwitcher pausiert ist. Ist das Retain-Flag nicht gesetzt, wird der Sendevorgang im verbindungslosen Zustand abgebrochen.

Da MQTT das Einhalten der Reihenfolge der Nachrichten für Nachrichten gleicher QoS garantiert[3] und die NearbyConnection API dieses Garant für gleicher Payload-Typen (Bytes und Files) gibt [11], reicht die Realisierung einer sendeseitigen Priorisierung.

## Realisierung mit NearbyConnections

Während Bytes-Nachrichten von der Nearby Connections API ad-hoc gesendet und empfangen werden, ist dies bei File-Nachrichten anders. Diese werden automatisch zerlegt (*chunked*) und stückweise übertragen. Die Nearby Connection API stellt dabei die empfangene Datei stückweise in einem selbstangelegten „Nearby“-Ordner im Standard-Downloadverzeichnis[[5]](#footnote-5) her. Da die gespeicherten Daten allerdings entsprechend Ihres Identifikators benannt (z.B. -7808006164344755168), der Endnutzer diese Datei aufgrund der fehlenden Dateiendung nicht öffnen kann und das Wiederherstellen der Dateiendung anhand des MIME-Typs erfahrungsgemäß nicht immer mit Erfolg gesegnet ist, muss beim Versenden einer File-Nachricht ebenso eine Byte-Nachricht gesendet werden. Diese beinhaltet die Dateiendung und den File-Identifikator, welcher zur späteren Zuordnung der Datei dient und soll im Folgenden als NeConFileInformation bezeichnet werden. Zusätzlich beinhaltet diese einen textAttachment, welcher dem späteren Entwickler die Möglichkeit zum Anhängen eines zusätzlichen String-Beitextes offenhält.

Ruft der NearflyClient nun die pubFile-Methode auf, sendet der NeConAdapter die Datei und zusätzlich die NeConFileinformation, während das Aufrufen der publish-Methode eine NeConBytesMessage sendet. Dabei wird für beide NeCon-Nachrichten das in **Abbildung 6** gezeigte NeCon-Nachrichtenpotokoll verwendet. Um eine dynamische Headerlänge zu ermöglichen, wird der Header in zwei Segmenten unterteilt. Das erste Segment hat eine fixe Bytelänge und wird aus einem UFT-8 encodierten String[[6]](#footnote-6) gewonnen. Es beinhaltet das Feld für den Beginn der Bytes-Nutzlast, welche zunächst auf 3Bytes begrenzt wird und damit eine maximale Headerlänge von 999Bytes zulässt. Die Größe des zweiten Segments ist dynamisch und beinhaltet ein JSON-Objekt im UTF-8-Stringformat, welcher bei NeConBytes-Nachrichten derzeitig nur den Channel erhält. Im Falle von NeConFileinformation, beinhaltet dieses zusätzlich noch die Dateierweiterung, einen Nutzdaten-Identifikator und den Beitext, welcher der Datei angehängt werden kann.



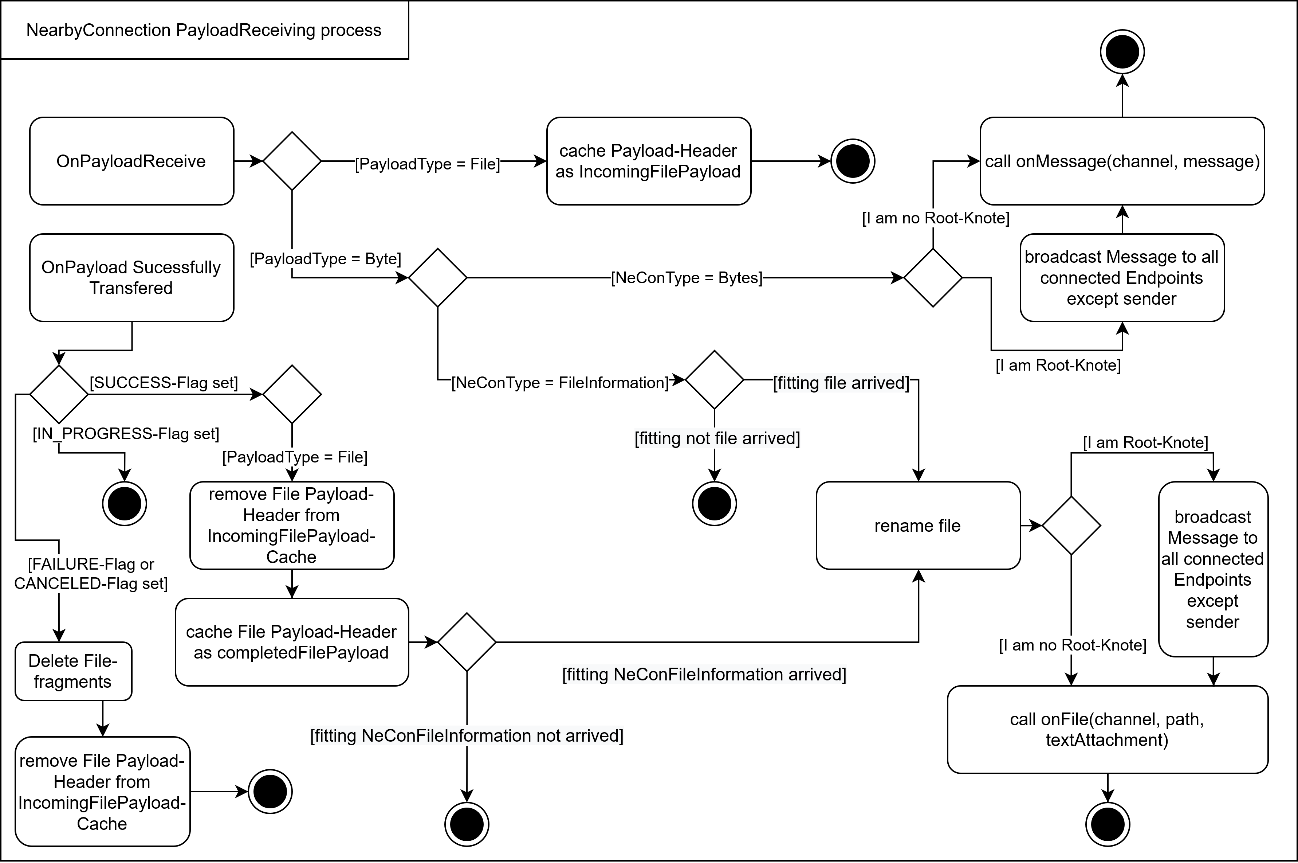
**Abbildung 6:** NeCon-Nachrichtenprotokoll

Empfängt die NearbyConnection API einen Payload-Header (siehe **Abbildung 7**), prüft der PayloadReceiver, ob diese vom Payload-Typ File oder Bytes ist. Handelt es sich hierbei um eine Bytes-Nachricht (nicht zu verwechseln mit der ähnlich benannten NeConByte-Nachricht), wird der NeCon-Nachrichtentyp identifiziert. Die Nachricht wird dem NeCon-Client (onMessage) übergeben. Dieser prüft, ob der entsprechende Kanal der Nachricht abonniert ist und leitet diese bei Erfolg an den NearflyService weiter. Ist dies nicht der Fall, wird die Nachricht verworfen. Ist der aktuelle Knoten als Root-Knoten deklariert, wird die Nachricht zusätzlich an alle verbundenen Knoten, ausgeschlossen dem Sender der Nachricht, gesendet. Anhand des NearflyListeners kann schlussendlich auf die Nachricht reagiert werden. Eine NeConFileInformation-Nachricht hingegen wird zunächst zwischengespeichert, zeitgleich wird kontrolliert, ob sich der Payload-Header der File-Nachricht mit demselben Identifikator, bereits angekommen ist und sich im CompletedFilePayload-Zwischerspeicher befindet. Ist dies nicht der Fall wird passiv auf den Eingang der Datei gewartet.

Kommt nun der Payload-Header einer File-Nachricht an (onPayloadReceive), signalisiert dies den Anfang der Übertragung einer File-Nachricht. Daraufhin wird der Payload-Header (welcher den Identifikator beinhaltet) als IncomingFilePayload zwischengespeichert, um die laufende Datenübertragung zu vermerken. Während der Übertragung wird bei jedem übertragenen Datei-Fragment ein Payload-Header (OnPayloadTransferUpdate) mit gesetztem IN\_PROGRESS-Flag empfangen, durch welchen sich die Anzahl der transferierten Bytes einsehen lassen. Tritt dabei ein Fehler auf, wird dies durch ein gesetztes FAILURE- oder CANCELED-Flag signalisiert. Der initial zwischengespeicherte Payload-Header, samt NeConFileInformation und die fragmentierte-Datei im Download-Verzeichnis müssen daraufhin gelöscht werden. Wurde die Datei vollständig übertragen, enthält der Payoad-Header ein gesetzten SUCCESS-Flag. Der vermerkte Header wird nun aus dem IncomingFilePayload-Zwischenspeicher entfernt und in den CompletedFilePayload-Zwischenspeicher geschoben. Befindet sich nun auch die NeConFileInformation-Nachricht im Zwischenspeicher, wird der Prozess zum Umbenennen der Datei initialisiert.

Fungiert der aktuelle Knoten als Root-Knoten, wird die empfangene Datei zusätzlich an alle verbundenen Knoten weitergeleitet. Ist der Empfangsvorgang erfolgreich abgeschlossen werden Payload-Header, wie auch die NeConFileInformation aus deren Zwischenspeicher gelöscht und die onFile-Methode mit dem Kanal, dem Dateipfad und dem textAttachment aufgerufen.

Aufgrund des beschränkten Einflusses, welcher dem Nearby Connection API Nutzer auf den Prozess der Dateiübertragung gegeben wird, kann dabei der Datenübertragungsprozess seitens des Empfängers nicht abgebrochen werden, wenn der jeweilige Kanal nicht abonniert ist.



**Abbildung 7:** Prozessablauf beim Empfangen einer Nachricht seitens des NeCon-Clients

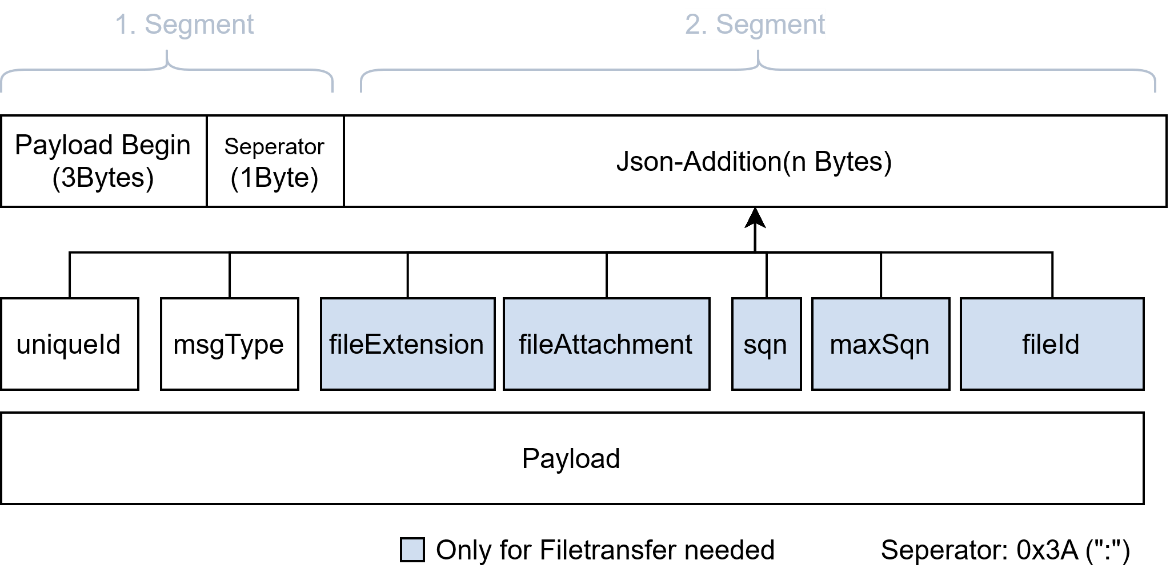
## Realisierung mit MQTT

Da der Nearfly-Kanal auf das MQTT-Topic basiert, kann dieser als Topic (room/kanal) dem Room-String angehängt werden. Um das Übertragung von Multimedia-Daten Mithilfe der MQTT API möglichst der Nearby Connections API gleichzustellen, sollen versandte Daten nach Ankunft ebenfalls im Standarddownload-Verzeichnis im „Nearby“-Ordner abgelegt werden. Dazu muss auch hier zuzüglich der zu übermittelnden Datei die Dateiendung, ein Beitext und der Nachrichtentyp mitversandt werden.

Eine unkomplizierte und daher oft gewählte Möglichkeit, um dieses Problem zu lösen, ist das Senden der gesamten Nachricht als String im JSON-Format. Die binäre Multimedia-Datei wird dabei durch das Base64-Kodierungsverfahren in einen String kodiert und beim empfangen erneut dekodiert. Allerdingt sorgt die Base64-Kodierung durch das Kodieren von drei Byte-Oktetts in vier Zeichen-Oktetts und das anschließende Padding der Daten auf ein vielfachen von vier dafür, das für alle Binäre-Daten im kodierten Zustand das 1.37 Fache [17] der ursprünglichen Datengröße benötigt wird. Um diesen Overhead zu vermeiden, soll daher eine andere Möglichkeit gewählt werden. Diese ergibt sich durch den Einsatz des NeCon-Nachrichtenprotokolls (**Abbildung 6**), das durch den dynamischen String Header, welcher durch das JSON-Segment leicht modifiziert werden kann, String von Payload-Bytes trennt.

Da die Nearby Connections API keine maximale Datengröße für Multimedia-Daten angibt und das MQTT-Protokoll eine maximale Payloadgröße von 256MByte [14] vorsieht, wird der MQTT-Adapter um ein Chunking-Mechanismus, das Pakete ab 2MByte[[7]](#footnote-7) (MAC\_CHUNK\_SIZE) zerlegt, erweitert. Um dies zu tun wird das JSON-Segment, um eine Sequenznummer (sqn), die maximal zu sendenden Fragmenten (maxSqn) und einen Identifikator (fileId) ergänzt. Die fileId soll dabei für die Empfängerseitigen Unterscheidung zeitgleich ankommender unterschiedlicher Multimedia-Fragmente verwendet werden und ist bei jedem Fragment dergleichen Datei identisch, während durch die Sequenznummer doppelte Fragmente, welche durch die eingestellte QoS von eins auftauchen können, erkannt und ignoriert werden können. Damit ergibt sich das in **Abbildung 8** gezeigte JSON-Segment. Soll nun eine Multimedia-Datei größer der MAC\_CHUNK\_SIZE gesendet werden, kreiert der Sender eine zufällige fünfstellige fileId und sendet die Datei-Fragmente, welche daraufhin vom Empfänger zwischengespeichert werden. Empfängt der Empfänger das Datei-Fragment mit der Sequenznummer (maxSqn-1), wird die Datei rekonstruiert, und die zwischengespeicherten Datei-Fragmente gelöscht. Daraufhin wird die onFile-Methode des NearflyServices getriggert.

Um nun auch die die Bytes-Übertragung zu ermöglichen, erhält das JSON-Segment zusätzlich den Nachrichtentyp (msgType) und eine uniqueId (wie in **Abbildung 8** gezeigt), welche verhindern soll, dass der Sender seine eigenen Nachrichten bekommt und zunächst für das Angleichen der Verhaltensweisen in Hinsicht auf das Empfangen beider ConnectionModes verwendet wird.



**Abbildung 8:** Modifizierter NeCon-Nachrichtenprotokoll zur Nachrichtenübertragung seitens MQTT

# Evaluation

Die Evaluation zeigt durch vergleichendes Messen beider aus der Implementierung resultierenden ConnectionModi die markant unterschiedlichen Zeiten, welche im Verbindungsaufbau existieren. Weiterhin zeigt das Vergleichen der Datenübertragung, den vorteiligen Durchsatz der Nearby Connections API, welcher jedoch nicht zum gänzlichen Erfüllen der Echtzeitanforderung reicht. Für die Datenerhebung werden dabei entsprechend **Tabelle 5** das Motorola G5, G6, sowie das Samsung Galaxy S7 edge, das S9 und das LG G2 mini LTE verwendet[[8]](#footnote-8). Alle Smartphones unterstützen dieselben WiFi Protokolle, unterscheiden sich jedoch sowohl in den Android-Versionen wie auch teilweise in den verbauten Bluetooth Chips, welche wesentlich für den Verbindungsaufbau im NeCon-ConnectionMode sind.

**Tabelle 5:** Durchschnittsgröße von Multimedia-Daten

|  |  |  |  |
| --- | --- | --- | --- |
| **Smartphone** | **Android Version** | **WiFi 802.11 protocols** | **Bluetooth** |
| Motorola Moto G6 | v8.0.0 | a/b/g/n | 4.2 |
| Motorola Moto G5 | v8.1.0 | a/b/g/n | 4.2 |
| Samsung Galaxy S9 | v10.0.0 | a/b/g/n | 5.0 |
| Samsung Galaxy S7 edge | v7.0.0 | a/b/g/n | 4.2 |
| LG G2 mini LTE | v7.1.2 | a/b/g/n | 4.0 |

## Zeiten für den Verbindungsaufbau

Zunächst werden die Zeiten untersucht, welche für den Aufbau eines zwei Knoten Netzwerkes benötigt werden. Dies hat den Vorteil das ein besserer Einblick in die benötigten Zeiten eines einzelnen Knotens gewonnen werden kann. Der Test zum Verbindungsaufbau verläuft automatisch, wobei die zufällig generierte INITIAL\_DISCOVERY\_TIME, wie auch die Zeit, welche für das Erreichen der Zustände (NODE, CONNODE) vom System protokolliert werden. Wurde eine Verbindung vom letzten Knoten erfolgreich aufgestellt, wechselt dieser nach fünf Sekunden in den STANDBY-Zustand [[9]](#footnote-9)und trennt damit die Verbindung. Weitere fünf Sekunden später, welche als Toleranzzeit zum Erkennen der getrennten Verbindung dienen, kehrt der Nicht-Root-Knoten in den FINDROOT-Zustand zurück, um erneut am Netzwerkaufbau zu partizipieren. **Abbildung 9** stellt die für den Aufbau dieses kleinsten Netzwerkes benötigte Zeit für von 120 Versuche dar. Das Diagramm ist in vier Segmenten unterteilt, welche jeweils die Versuche mit verschiedenen Smartphone-Kombinationen beinhalten. Zu sehen ist, dass sich die Werte teilweise von Smartphone zu Smartphone relativ stark unterscheiden. Beim Betrachten des Zeiten der Kombination zwischen dem LG und dem S9 fällt auf, das der Root nach durchschnittlich 1500 Millisekunden vom jeweils anderen Knoten gefunden und weiteren durchschnittlichen 1500 Millisekunden, welche die die künstliche Verzögerung der initialen Root-Findungsphase (INITIAL\_DISCOVERY\_TIME) ist, eine Verbindungsanfrage an diesen gesendet wird. Da die Verbindung nach durchschnittlichen 6510 Millisekunden zustande kommt, ergibt sich somit eine durchschnittliche Differenz von Verbindungsanfrage zu Verbindungsaufbau von 3510 Millisekunden. Für S7Edge-LG-Kombination entstehen in etwa dieselben Zeiten, während erstaunlicherweise die Zeiten bei den übrigen Smartphone-Kombinationen mit durchschnittlichen 11600 und 12600 Millisekunden im Verhältnis zur LG-S9-Kombination fast doppelt so hoch ist[[10]](#footnote-10), obwohl die eigentliche Root-Findungsphase bei diesen nach wenigen maximal aber 4 Sekunden beendet wird. Die restliche Zeit brauchen die Knoten für die Verbindungsanfrage.

**Abbildung 9:** Benötigte Zeiten im Verbindungsaufbau[[11]](#footnote-11)

Der Verbindungsaufbau im zwei Knoten Netzwerk erscheint zunächst langsam. Doch gibt der Nearby Connections API Mitentwickler Will Harmon für den Unidirektionalen Verbindungsaufbau (d.h. im Falle eines einzigen Advertisers und eines einzigen Discoverers) bereits eine durchschnittliche Verbindungslatenz von 2-7 Sekunden [7] an. Die künstliche Latenz, welche nach dem Finden des ersten Knotens gesetzt wird, gilt im zwei Knoten Netzwerk zwar als Overhead, ist jedoch ein essenzieller Baustein im Mehrknotens-Netzwerk zum Finden eines geeigneten Roots. Da Smartphones, wie das LG G2 Smartphone aufgrund von zwischenzeitlichen Trennungen der Verbindung und der mangelnden Leistung für das Forwarding von Nachrichten[[12]](#footnote-12), wenn möglich nicht als Root gewählt werden sollen. Auch sorgt der initiale FINDROOT-Zustand, in dem ein Knoten durchschnittlich 1500 Millisekunden sowohl das Advertising, wie auch das Discovering betreibt, für ein Trashing. Insgesamt schaffen einige Smartphones den Verbindungsaufbau im durchschnittlichen Rahmen der von Harmon angegebenen Zeiten, für die übrigen Smartphones konnte auch durch das Entfernen der INITIAL\_DISCOVERY\_TIME kein deutlich besseres Ergebnis erzielt werden, was sich aber auch aus **Abbildung 9** ableitbar ist.

Mit dem Wissen, das bereits zwischen unterschiedlichen Smartphone-Kombinationen unterschiedliche Latenzen entstehen, können nun die Zeiten für den Aufbau eines 3 Knoten-Netzwerkes (**Abbildung 10**) evaluiert werden. Auch dieser Prozess zur Datenerhebung verläuft automatisch, indem dieses Mal Seitens der Root-Knoten die Zeiten vom initialisieren der Verbindung bis zum Knotenbeitritt aller zwei Knoten protokolliert. Wurde das drei Knoten-Netzwerk aufgebaut, trennt der Root-Knoten durch wechseln in den STANDBY-Zustand die Verbindung. Damit das Netzwerke von den Knoten nicht wiederaufgebaut wird, wechseln diese ebenfalls in den STANDBY-Zustand. Ist die aktuelle Minute, eine durch zwei teilbare Zahl, wechseln alle Geräte (einmalig in dieser Minute) in den FINDROOT-Zustand und versuchen erneut das Netzwerk aufzubauen.

Für den Aufbau des Netzwerkes wurden hierbei das G5, das G6 und das S9 verwendet. Auffällig ist, dass die Zeiten für das Verbinden des ersten Knotens zunächst relativ ähnlich den bekannten Zeiten sind. Bis jedoch die zweite Verbindung zustande kommt, vergeht durchschnittlich nochmal die doppelte Zeit. Ein Backoff kommt dabei nur einige Male vor, jedoch selten zwei Mal in Folge. Trotzdem beträgt die Zeit bis zum erfolgreichen Verbindungaufbau durchschnittliche 15 Sekunden.

**Abbildung 10:** Netzwerkaufbau im 3 Knoten Netzwerk

Der Verbindungsaufbau seitens MQTT ist hingegen spürbar schneller, da der Netzwerkaufbau kaum abhängig der Netzwerkgröße ist. Stattdessen ist dieser von den Kapazitäten des Brokers abhängig. Die Datenerhebung findet auch hier wieder durch einen automatisierten Test statt, dabei wird die Differenzzeit zwischen Verbindungsanfrage und CONNECT-Event der onLogMessage-Methode des NearflyListeners über 160 Verbindungsversuchen festgehalten. Auf **Abbildung 11** ist dabei ersichtlich, dass der LG G2 mini LTE mit durchschnittlich 1100 Millisekunden fast das fünffache der Zeit für die Verbindung benötigt, diese verteilt über alle mit durchschnittlichen 454 Millisekunden jedoch deutlich schneller aufgebaut wird. Dabei beeinflussen sich die Knoten aufgrund der serverbasierten Architektur kaum, sodass das Netzwerk nach einer spätestens aber zwei Sekunden besteht.

**Abbildung 11:** Zeiten für den Verbindungsaufbau mit MQTT

## Zeiten für die Datenübertragung

Um die Datenübertragungen zwischen beiden Connection-Modi zu vergleichen, werden 100 Bytes, welche in etwa dem durchschnittlichen JSON-String-Payload der implementierten Szenarien entspricht, in über 100 Versuchen übertragen. Da auch hier ein automatischer Test benutzt werden soll und sich die Zeiten im Millisekunden-Bereich zwischen den einzelnen Smartphones unterschieden, wird die Zeit gemessen, die eine Nachricht im Loop-Back braucht. Die Loop-Gegenstelle stellt dabei ein zweites Smartphone da, welcher die Nachricht nach Eingang sofort wieder publisht. Die Zeitdifferenz zwischen Sendezeitpunkt und Empfangszeitpunkt des Senders dividiert durch zwei stellt dabei die Latenz eines Versuches da[[13]](#footnote-13). Aus **Tabelle 6**, welche die gemessenen Durchschnittswerte mehrerer Testversuche enthält, ist ersichtlich, dass die durchschnittlichen Latenzen, welche beim publishen von Byte-Nachrichten der Nearfly API entstehen, bei der S9-S7edge-und S9-LG-Kombination durchschnittlich niedriger im im NeCon-ConnectionMode ist. Hierbei war das WiFi-Direct im NeCon-Modus fast[[14]](#footnote-14) durchgehend aktiviert. Bei der G5-G6-Kombination hingegen weißt MQTT, trotz aktiviertem WiFi-Direct die niedrigeren Latenzen auf. Aus den Messungen lässt sich erkennen, dass das Senden von Byte-Nachrichten im Durchschnitt in beiden Connection-Modi annährend gleich ist. Beim Betrachten der Latenzen eines einzelnen Versuchens, wie dies in **Abbildung 12** gezeigt wird, fällt jedoch auf, dass die Übertragung über die Nearby Connections API deutlich mehr Fluktuation (Jitter) ausgesetzt ist und zeitweise Latenzen im Bereich von 40 bis 800 Millisekunden hat.

**Tabelle 6:** Vergleich der Latenzen in beiden Nearfly-Connection-Modis

|  |  |  |  |
| --- | --- | --- | --- |
| Network Participants | MQTT (in ms) | NeCon (in ms) | NeCon Bluetooth only (in ms) |
| S9 and s7edge | 156 | 139 | 66 |
| G9 and LG G2 | 152 | 151 | - |
| G5 and G6 | 101 | 155 | - |

Durch Setzen der Strategie auf P2P\_CLUSTER konnte zudem (wie in **Tabelle 6** zu sehen) die durchschnittliche Latenz im NeCon-Modus ohne Zuhilfenahme des WiFi-Hotspots gemessen werden, die bei einer 100 Bytes Übertragung zwischen S9 und S7Edge mit durchschnittlichen 66 Millisekunden verhältnismäßig niedriger ist, jedoch wie sich im Folgenden zeigen wird, bei größeren Übertragungen deutlich weniger Durchsatz zulässt.

**Abbildung 12:** Latenzvergleich beiden Connection-Modi mit der S9-S7edge-Kombination

Des Weiteren sollen nun die Zeiten für die Übertragung von Multimedia-Daten in beiden Connection-Modis erfasst und verglichen werden. Analog zum vorherigen Test, entsprechen auch hierbei die Zeiten der Umlaufzeit (Round-Trip-Time) der Loopback-Nachricht dividiert durch zwei. Beim NeCon-Modus war dieses Mal der Wifi-Hotspot durchgehend aktiviert. Um ein Kontrast zwischen deaktiviertem und aktiviertem Wifi-Direct zu schaffen wurde auch hier durch das temporäre Nutzen der P2P\_CLUSTER-Strategie die Nearby Connections API auf die Nutzung von Bluetooth und BLE beschränkt. **Tabelle 7** zeigt die Stärke der Nearby Connections API bei größeren Datenübertragungen. Im Verhältnis zu MQTT schafft es die Nearby Connections API, dieselben Multimedia-Daten fast, um Faktor drei schneller zu übertragen. Anders sieht dies vor dem aufbauen des WiFi-Hotspot aus, wo die Datenübertragung durchschnittlich um Faktor 27 verlangsamt mit steigender Tendenz bei Größeren Datenmengen[[15]](#footnote-15) wird. Schlussfolgernd sollte vor dem Übertragen größerer Dateien sichergestellt werden, dass der Wifi-Hotspot aktiviert ist.

**Tabelle 7:** Vergleich der Zeiten der pubFile-Methode in beiden Connection-Modi mit der S9-S7edge-Kombination

|  |  |  |  |
| --- | --- | --- | --- |
| Data-Size | MQTT (in ms) | NeCon (in ms) | Bluetooth only (in ms) |
| 67,19 KBytes | 286 | 103 | 1953 |
| 1,01 MByte | 1614 | 556 | 13560 |
| 5,26 MByte | 6121 | 1787 | 71308 |
| 55,57 MByte | 43357 | 14077 | - |

## Verifikation der Anforderungen

Das Überprüfen der Anforderungen mithilfe der umgesetzten Szenarien.

Der Nachrichtenaustausch von Text-und Multimedia-Daten konnten erfolgreich implementiert werden und wurde mit der Messenger Activity ausprobiert, welche durch Nutzung der Nearfly-Kanäle den Nutzern die Möglichkeit zum Betreten verschiedener Chaträume bietet.

Auch ist die Nearfly-Bibliothek ensprechend der in **Tabelle 9** genannten Anforderungen in der Lage priorisierbare Nachrichten im Falle eines Verbindungsabbruchs durch Setzen des Retain-Flags zu puffern, die Verbindung daraufhin automatisch wiederaufzubauen und bei bestehender Verbindung zu versenden.

Durch Vergleichen beider ConnectionModes in der Bouncing Ball App kann die App auf ihre Echtzeitfähigkeit hin getestet werden. Es zeigt sich, wie bereits das die Bouncen Ball App die Nearfly API ausreizt. Im MQTT-Netzwerk mit zwei Knoten steigen die Latenzen immer weiter an, womit nach einiger Zeit ein Spielen nichtmehr möglich ist[[16]](#footnote-16). Im NeCon-Netzwerk mit derselben Anzahl an Knoten hingegen treten zwar zeitweise spürbare Latenzen auf, welche einen kurzen Nachrichtenburst zu Folge haben, doch lässt sich das Spiel flüssig spielen. Ab drei Netzwerkteilnehmer ist jedoch auch das NeCon-Netzwerk überlastet und kommt mit dem Datentransfer nicht hinterher[[17]](#footnote-17). Womit die Anforderung für Echtzeitfähigkeit im Sinne von 30FPS nur sehr beschränkt für ein NeCon-Netzwerk mit zwei Knoten erfüllt wird und damit als nicht erfüllt gesehen werden muss.

Um die Grenze für das Verhältnis zwischen Geschwindigkeit und Skalierbarkeit von Byte-Nachrichten besser einschätzen zu können, kann jedoch ein Systemtest mithilfe der Share Touchpoint Canvas durchgeführt werden. Die Ergebnisse dieses Systemtest sind in **Tabelle 8** festgehalten und zeigen wie viele Berührungspunkte gleichzeitig ohne größere Verzögerungen seitens des Empfängers (etwa unter einer Sekunde) im Durchschnitt versandt werden konnten. Dabei wurden die Berührungspunkte von einem Nicht-Root Knoten versandt und der Versuch etwa jede dritte Sekunde wiederholt. Es zeigt sich, dass die Übertragung im NeCon-ConnectionMode wie bereits aus der Bouncing Ball App bekannt, im zwei Knoten Netzwerk schneller als in MQTT verläuft. Kommt ein dritter Knoten hinzu, entspricht das Verhalten etwa dem von MQTT. Betritt ein vierter Knoten das Netzwerk können im Durschnitt noch zwei Berührungspunkte relativ schnell versandt werden. Letzten Endes entstehen beim fünf und sechs-Knoten-Netzwerk beim Senden von mehr als zwei gleichzeitigen Berührungspunk zeitweise größere Latenzen. Im MQTT-ConnectionMode hingegen macht die Größe des Netzwerkes zumindest keinen spürbaren Unterschied.

**Tabelle 8:** Übertragbare Berührungspunkte mit relativ kleiner Latenz

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nodes | 2 | 3 | 4 | 5 | 6 |
| NeCon | 7 | 4 | 2 | 1-2 | 1 |
| MQTT | 4 | 4 | 4 | 4 | 4 |

Des Weiteren konnten die Anforderung eins dynamischen Wechselns zwischen den ConnectionModes implementiert werden. Die Verbindung zum bestehenden Netzwerk wird dabei erst aufgebaut, wenn eine zum Netzwerk der jeweils anderen Technology besteht. Durch die Einführung des Room-Strings konnte zudem eine Sichtbarkeitsbeschränkung für Apps implementiert werden, welche dieselbe Nearfly API verwenden sich jedoch nicht sehen sollen.

Die Score Board Notepad App ist zudem als Anwendung mit wenigem Datenaufkommen gut durch beide ConnectionModes der Nearfly API bedient. Dabei kann es im NeCon-ConnectionMode besonders bei älteren Geräten vorkommen[[18]](#footnote-18), dass die Verbindung während einer Spielsession beendet wird. Um dem Entgegenzuwirken, werden nur Nachrichten mit absoluten Werten und einem gesetztem Retain-Flag gesendet. Diese Wahrscheinlichkeit sollte auch bei anderen Anwendungen, welche die Nearfly API im NeCon-ConnectionMode nutzen nicht außer Acht gelassen werden.

**Tabelle 9:** Anforderungsmatrix

|  |  |
| --- | --- |
| **Funktionale Anforderungen** | Status |
| textbasierte Nachrichten senden | ✓ |
| Multimedia-Nachrichten senden | ✓ |
| Priorisierung der Nachrichtentypen | ✓ |
| Trennbarkeit unterschiedlicher Nachrichten innerhalb derselben App | ✓ |
| Möglichkeit, dass Nachrichten bei getrennter Verbindung gehalten werden | ✓ |
| automatischer Verbindungsaufbau | ✓ |
| Echtzeitfähigkeit für ein flüssiges Multiplayer-Spiel-Verfalten bei 30 FPS pro Spieler | X |
| Sichtbarkeit der Nachrichten nur innerhalb derselben App | ✓ |
| Wechselt der Nutzer von Online- in den Offline-Modus sollen keine Daten keine Daten verloren gehen | ✓ |

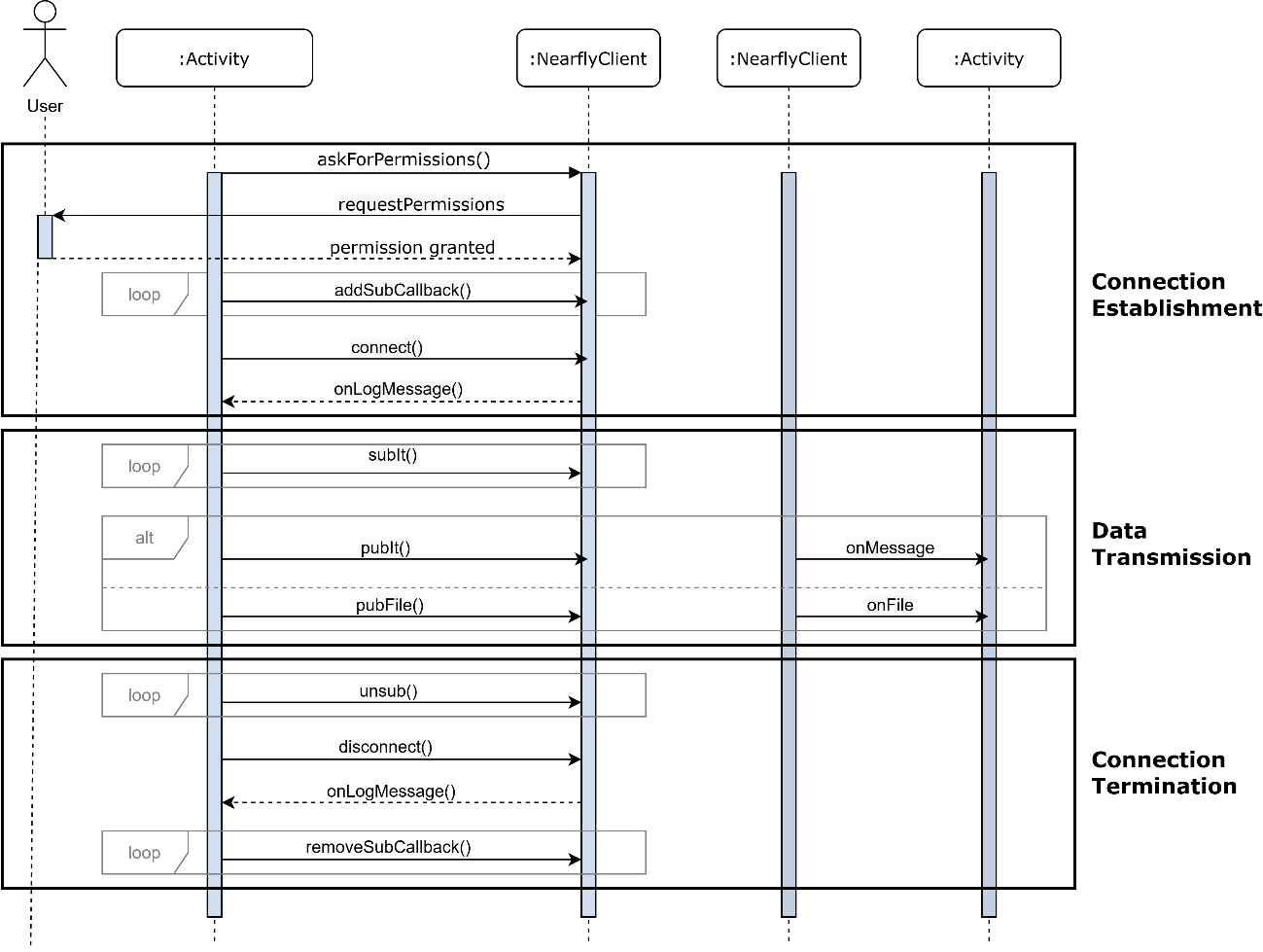
## Nutzung der Nearfly API

Nachdem die Bibliothek erfolgreich eingebunden und die obligatorischen Berechtigungen im Manifest eingetragen wurden, kann die Activity gestartet werden. Vor dem Verbinden zum NearflyClient müssen hierfür die mandatorischen Dangerous-Berechtigungen z.B. durch Aufrufen der askForPermissions-Methode[[19]](#footnote-19) abgefragt und vom Anwendungsnutzer (wie in **Abbildung 13**) gewährt werden.

Verfügt die App über alle nötigen Berechtigungen, kann ein oder mehr NearflyListener registriert und durch connect unter Angabe vom ConnectionMode die Verbindung zur unterliegenden Technologie hergestellt. Wurde die Verbindung erfolgreich hergestellt wird das onLogMessage-Event vom NearflyListener mit dem CONNECT-Schlüsselwort abgerufen. Dies ist besonders bei Nutzung der Nearby Connections API nützlich, da der Verbindungsaufbau über mehrere Sekunden dauern kann und zuvor kein Datenaustausch möglich ist. Zu beachten ist, dass der NeConAdapter eine Verbindung als aufgebauten Netzwerk mit mindestens zwei Knoten definiert. Werden mehr Knoten erwartet, muss dies durch den API Nutzer implementiert werden.

Nun können beliebige Kanäle durch die subIt-Methode abonnieren und im Anschluss beliebige Nachrichten durch pubIt und pubFile gesendet werden. Der Empfang dieser Nachrichten löst dabei bei den anderen Netzwerkteilnehmern das onMessage bzw. onFile-Event aus.

Soll die Verbindung abgebaut werden, müssen die zum Verbindungsaufbau entsprechenden Gegenoperationen (wie in **Abbildung 13**) in umgekehrter Reihenfolge verwendet werden. Demnach müssen die abonnierten Kanäle durch unsub deabonniert und ein disconnect aufgerufen werden. Durch das Abmelden der registrierten NearflyListener wird schlussendlich der NearflyClient in den Ausgangszustand versetzt.



**Abbildung 13:** Ablauf der Nutzung der Nearfly API

# Fazit

Die Ergebnisse der Evaluation haben gezeigt, dass der Netzwerkaufbau im NeCon-ConnectionMode noch verbessert werden kann

Die wesentlichen Erkenntnisse dieser Arbeit

* + Besteht die Möglichkeit eines Internetzugangs sollte beim Erstellen von Netzwerken mit mehr als drei Netzwerkteilnehmern MQTT als ConnectionMode bevorzugt werden.
  + Netzwerke mit zwei bis drei Knoten sind mit dem NeCon-ConnectionMode, jedoch besser bedient und können bei aktiviertem Wifi-Hotspot größere Datenmengen deutlich schnellen Austauschen. Besteht ein Netzwerk hingegen aus zwei Knoten, ist sogar eine Echtzeit mit 30 FPS möglich.
  + Ist das Internet nicht zugänglich, kann der NeCon-ConnectionMode bei Anwendungen mit geringem Datenaufkommen, wie etwa lokalen Umfrage Apps oder rundenbasierten Spielen für bis zu sieben Teilnehmern verwendet werden.

## Ausblick

…

1. Ausführlichere Diagramme zum Verbindungsaufbau im NeCon-ConnectionMode

**Abbildung 14:** Zeiten für den Verbindungsaufbau zwischen dem LG G2 und dem S9

**Abbildung 15:** Zeiten für den Verbindungsaufbau zwischen dem G5 und dem G6

Quellenverzeichnis

Gedruckte Quellen

[1] 2018 Global Information Infrastructure and Networking Symposium (GIIS). Thessaloniki, Greece: IEEE; 2018.

Online-Quellen

[2] *Alexander Kunst*. Statista-Umfrage Telekommunikation 2017. https://de.statista.com/statistik/daten/studie/722248/umfrage/umfrage-zur-nutzung-von-smartphone-funktionen-nach-haeufigkeit-in-deutschland/.

[3] *Banks* *A., Gupta* *R.* MQTT Version 3.1.1 - OASIS Standard. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[4] *Boshell* *B.* Average App File Size: Data for Android and iOS Mobile Apps. https://sweetpricing.com/blog/2017/02/average-app-file-size/.

[5] *Google*. App permissions. https://developer.android.com/guide/topics/permissions/overview.

[6] *Harmon* *W.* Google Nearby Connections 2.0 capabilities. https://stackoverflow.com/questions/51976470/google-nearby-connections-2-0-capabilities.

[7] *Harmon* *W.* How can I speed up Nearby Connections API discovery? https://stackoverflow.com/questions/52825617/how-can-i-speed-up-nearby-connections-api-discovery/52882054#52882054.

[8] *Harmon* *W.* How performant is Nearby Connections? https://stackoverflow.com/questions/54434616/how-performant-is-nearby-connections/54470958#54470958.

[9] *HiveMQ Team*. Client, Broker / Server and Connection Establishment - MQTT Essentials: Part 3. https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/.

[10] *Ihlenfeld* *J.* Bluetooth 3.0 HS mit WLAN-Beschleunigung - Golem.de. Golem.de. 22.4.2009.

[11] *Nearby Connections Team*. Nearby Connections API Leitfaden. https://developers.google.com/nearby/connections.

[12] *nguyen* *d.* Sending and Receiving Pictures From a Raspberry Pi via MQTT - IBM Developer Recipes. https://developer.ibm.com/recipes/tutorials/sending-and-receiving-pictures-from-a-raspberry-pi-via-mqtt/.

[13] *Perez* *S.* Google opens its Nearby Connections tech to Android developers to enable smarter offline apps. TechCrunch. 31.7.2017.

[14] *Roger Light*. mosquitto.conf man page. https://mosquitto.org/man/mosquitto-8.html.

[15] *Schnabel* *P.* Bluetooth Low Energy (4.0 / 4.1 / 4.2). https://www.elektronik-kompendium.de/sites/kom/1805171.htm.

[16] *Schnabel* *P.* Bluetooth 5. https://www.elektronik-kompendium.de/sites/kom/2107121.htm.

[17] *Siahaan* *A. P. U.* Base64 Character Encoding and Decoding Modeling; 2017.

[18] *Zhang* *L.* Building Facebook Messenger. https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920.

1. Die MQTT Spezifikation erlaubt auch andere Protokolle, welche eine geordnete, verlustfreie, bi-direktionale Verbindung zulassen ( [3].). [↑](#footnote-ref-1)
2. Ein Topic ist ein UTF-8 Encodierter String der durch einen Schrägstrich (‚/‘ U+002F) hierarchisch, gleichend einer Baumstruktur, untergliedert werden kann z.B. „Baum/Zweig“ inkludiert „Baum/Zweig/Blatt“ ( [3]). [↑](#footnote-ref-2)
3. Hat sich der Entwickler für die Nutzung des NearflyServices entschlossen, darf die Methode zum initialisieren des Verbindungsaufbaus erst nach dem erfolgreichen Binding des Services aufgerufen werden, da die Service-Instanz erst mit dem *Binding* übergeben wird. [↑](#footnote-ref-3)
4. Die 24 Stunden in Millisekunden entsprechen dabei . [↑](#footnote-ref-4)
5. Dies erklärt die für die File-Übertragung notwendige Dangerous-Berechtigung für das Lesen und Schreiben von Daten außerhalb des Anwendungsspezifischen Verzeichnisses [↑](#footnote-ref-5)
6. Jedes String-Element braucht dabei genau ein Byte, da dies für UTF-8 enkodierte Zeichen, welche in der ASCII-Tabelle zu finden sind, garantiert wird. Andere Zeichen hingegen können bis zu 4Bytes beanspruchen. [↑](#footnote-ref-6)
7. Aus eigenen empirischen Untersuchungen zeigt sich das ein kleiner Schwellwert (etwa 30KByte), MQTT stark verlangsamen (auch, wenn dies im IBM Entwicklerforum so demonstriert wird [12]). Ein Schwellwert von 30KByte etwa, würde bei einem 5MByte großen Bild, den Broker zunächst mit 167 Chunks fluten. [↑](#footnote-ref-7)
8. Zu den in **Tabelle 5** genannten Smartphones, musste ein Samsung Galaxy s5 mini, welcher ebenfalls zur Verfügung stand und über Bluetooth 4.0 verfügte, als Testobjekt ausgeschlossen werden, weil dieser des Öfteren Verbindungen ablehnte und dies zu sehr langen Verbindungszeiten führte. [↑](#footnote-ref-8)
9. Durch das Versetzten in den STANDBY-Zustand ist sichergestellt, dass dieser die initiale Root-Findungsphase und die damit verbundene Wartezeit (INITIAL\_DISCOVERY\_TIME) erneut ausdauern muss. [↑](#footnote-ref-9)
10. Dies liegt vermutlich daran, dass das S9 einen Bluetooth-Version 5.0 besitzt. Tatsächlich zeigten weitere kleinere Tests, welche nicht festgehalten wurden, dass die Kombination aus S9-G5 oder S9-S6 deutlich geringere Zeiten im Bereich von 6 Sekunden ergeben, auch wenn dies bei der S7Edge-S9-Kombination wieder etwa s anders erscheint. [↑](#footnote-ref-10)
11. Zwei ausführlichere Diagramme zum Verbindungsaufbau finden sich zudem im Anhang A [↑](#footnote-ref-11)
12. Beim Samsung S5 Mini, welcher nicht in die o.g. Testfälle aufgenommen wurde, konnte zudem festgestellt werden, das im Root-Zustand kein Netzwerk größer als zwei Knoten aufbauen kann, da dieser die weitere Knoten nicht akzeptiert. [↑](#footnote-ref-12)
13. Hierbei sei angemerkt, dass dieser Vergleich in erster Linie dazu dient, die getesteten Szenarien in Werten auszudrücken. Ein umfassend korrekter Vergleich beider Technologien wird hierbei keineswegs angestrebt. [↑](#footnote-ref-13)
14. Die ersten Übertragungen verliefen mit Bluetooth, da Wifi-Direct zu Beginn noch nicht von der Nearby Connections API aktiviert wurde. [↑](#footnote-ref-14)
15. Leider konnte die Datenübertragung der 55,57 MByte großen APK in der Cluster-Topologie nicht erfolgreich abgeschlossen werden, da nach mehreren Versuchen, welche mehrere Minuten dauerten, immer wieder die Verbindung nach einigen übertragenen MBytes getrennt und die Übertragung abgebrochen wurde. [↑](#footnote-ref-15)
16. Tatsächlich ist eine derartige Echtzeitfähigkeit durch verringern der Quality of Service (QoS) auf null möglich. Dies wird derzeitig jedoch nicht von der Nearfly API angeboten, da im NeCon-Adapter bisher kein Pendant zur QoS implementiert wurde. [↑](#footnote-ref-16)
17. Hierbei sollte beachtet werden, dass bei zu schnellen Übertragungen das Smartphone selbst zum Flaschenhals werden kann und dies in diesem Fall (90 Nachrichten/s) nicht vollständig ausgeschlossen werden kann. [↑](#footnote-ref-17)
18. Konkret kommt dies beim LG G2 seltener vor. Auch beim S7 und G5 ist dies bereits vorgekommen. Ganz besonders existiert dieses Problem jedoch beim S5 Mini, welcher deswegen kaum verwendet wurde. [↑](#footnote-ref-18)
19. Bei Nutzung der Nearfly API über den NearflyService, ist hierbei zu beachten, das Methodenaufrufe des NearflyService erst nach dem erfolgreichen Binding des Services möglich sind. [↑](#footnote-ref-19)