|  |  |  |
| --- | --- | --- |
| **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 15.05.2020 |
|  |
| 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 Motivation 1

1.2 Zielsetzung 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 4

2.1.4 Szenario 4 – Score Board Notepad 5

2.2 Funktionale Anforderungen 6

2.3 Nichtfunktionale Anforderungen 7

2.4 Gegenüberstellung von Nearby Connections und MQTT 8

3 Software-Design 13

4 Implementierung des Nearfly Services 15

4.1 Obligatorische Berechtigungen **Fehler! Textmarke nicht definiert.**

4.2 Erstellen der Bibliothek **Fehler! Textmarke nicht definiert.**

5 Verbindungsaufbau 16

5.1 Erstellung eines autonomen Peer-to-Peer Netzwerks 16

6 Übertragung von Daten 19

7 Benutzung der API 24

8 Evaluation 25

8.1 Vergleichen der Prozesslaufzeiten zwischen MQTT und Nearby Connections 25

8.2 Verifikation der Anforderungen 25

8.3 Erkenntnisse des Messenger Szenarios 26

8.4 Erkenntnisse des Touchpoint Szenarios 26

8.5 Erkenntnisse des Bouncing Ball Szenarios 28

Fazit 29

Ausblick 29

Quellenverzeichnis Fehler! Textmarke nicht definiert.

# Einleitung

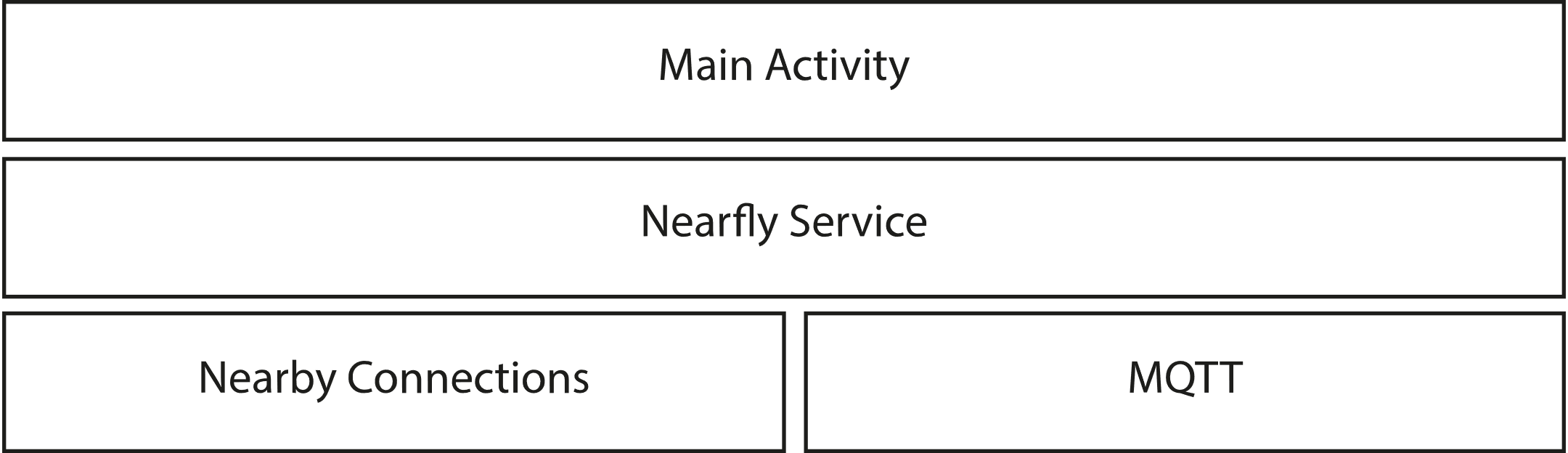
* Obwohl Smartphones ihren Ursprung im Telefon haben, rückt das Telefonieren mit einem Telefon immer weiter in den Hintergrund. Tatsächlich zeigt eine auf Statistica veröffentliche Studie, das bereits im Jahre 2017 in Deutschland die Funktion Kurznachrichten zu versendet mehr als Doppel so häufig, wie das Telefonieren verwendet wird [2]. Solche Studien zeigen den Stellenwert, den die Vernetzung heutzutage in unserm Leben einnimmt.
* Um Daten unter Mobilen Endgeräten zu versendet wird häufig auf bestehende Technologien, wie MQTT zurückgegriffen (link). Diese werden in Form einer Bibliothek, welche komplexe Sachverhalte abstrahieren und durch eine zu meist anwenderfreundlichen API ansprechbar sind, in das Projekt eingebunden.
* Mit dem Ursprung im Internet of Things schafft es MQTT aufgrund des geringen Batteriekonsums auch in den Bereich der Mobilen Endgeräte. So benutzt etwa Facebook dieses Protokoll für seinen Messenger. Begründet durch dessen Auslegung auf eine schnelle Datenübertragung bei relativ niedriger Bandbreite und einem niedrigem Batteriekonsum, trotz persistenter Verbindung [13].
* Doch internet ist nicht überall von vorteil, so kann es etwa bei schulen sein, das
* Auch Peer to Peer technologien, wie die im Juli 2017 veröffentlichte zweite Version des Nearby Conenctions haben einen Vorteil. Sie sorgen für einen Verbenutzung ohne bestehenden Internetzugang.
* So könnnenn etwa lokale Spiele gespielt werden, auch da wo kein internet bestehet oder aber auch an Schulenn umfragen in klassenzimmer durchgeführt werden, ohne einen Internetzugang zu haben.

## Motivation

* Beide Technologien bieten demnach die Möglichkeiten einer Ergänzung. Bisher gibt es einige wenige Anwendungen(link?), die die Nearby Technologie verwenden und einige die MQTT verwenden. Doch können sich beiden Technologien perfekt ergänzen und zusammen Entwicklern eine Technologie bietet, die wowohl online, wie auch offline funktioniert.
* (MQTT als Online Protokoll und Nearby Connections zur offline Datenübertragung. Nun wäre es denkbar, die Vorteile beider Technologien in solcher Form zu kombinieren, dass dem Entwickler eine API angeboten wird, welche beide Technologien vereint)

## Zielsetzung

* Im Rahmen dieser Arbeit sollen beide Technologien alanysiert werden und eine Wrapper Bibliothek konzipiert und implementiert werden, welche beide technologien in solcher Form abstrahiert das dem Entwickler eine möglichst minimalistische API zur Verfügung gestellt wir…
* Wie in der folgenden Abbildung verdeutlicht:



Um das Ziel zu erreichen werden im Rahmen dieser Thesis folgende Meilensteine definiert:

Meilenstein 1: Analyse der Anforderungen an das zu entwickelnde System durch implementierung von Szenarien

Meilenstein 2: Implementierung der Szenarien

Meilenstein 3: Iterationen über die Bibliothek

* Vorallem nachvollziehbarere SW Entwurf z.B. durch Verwendung bekannten Entwurfmuster
* Anwendung in unterschiedlichen Anwendungen strukturieren
* Durch spätere Anforderungsanalyse werden Anforderungen später dazu spezifiziert

# Anforderungsanalyse

Die Anforderungsanalyse beinhaltet die Szenarien, welche als Hilfs-Maß zur Ermittlung der konkreten Anforderungen an das zu entwickelnde System dienen. Dabei soll die *Messenger App* 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 4 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.

## Szenarien

### Szenario 1 – Messenger

*Marius ist der Scrum-Master eines 5-köpfigen Entwicklungsteams 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. Die Nachrichten-Priorisierung stellt Marius auf Text ein, weil dieser weiß das seine Teammitglieder vor dem Senden größerer Daten stehts Text anhängen und er immer zuerst den Problembereich identifizieren will. Er fordert seine Teammitglieder auf seiner Gruppe beizutreten. Danach schreibt er an allen einen Begrüßung Text und frag ob es denn derzeit Probleme geben würde. Sofort antwortet Tommy, dass die Kaffee-Pads leer seien und sendet ein Foto von den Kaffee-Pads 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 wäre jedoch auch die Anforderung auf beliebige Binärdaten zu pauschalisieren 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 reduziert. Auch sollen Nachrichten einem Absender zuordenbar sein. Weiterhin soll das System dem Nutzer die Möglichkeit der Nachrichten-Priorisierung geben und damit dem Nutzer die Entscheidung überlassen, welche Nachrichten dieser beim parallelen Empfange ähnlich-großer Nachrichten unterschiedlicher Typen zuerst geladen haben möchte (z.B. Bild oder Text).

### 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. Nach erfolgreicher Bewältigung der Aufgabe bekommen die 3 Freunde eine Benachrichtigung einer Person, die der Partie beigetreten ist und Betina kommt in das Büro.*

Als kollaboratives Spiel, umfasst das zweite Szenario eine Leinwand, welche gemeinsam von allen beteiligten Benutzern durch antippen bemalt werden kann. Aufgrund der natürlichen Limitierung des Menschen können bis zu 10 Berührungspunkte pro Benutzer zeitgleich gesetzt werden. Da alle gemeinsam Zeichnen wollen, muss das System die Daten möglichst zeitnah empfange. Zuletzt muss das System neu verbundene Nutzer allen Spielern mitteilen.

### Szenario 3 – Bouncing Ball

*Mary, Robert und Frederik befinden sich während der großen Pause im Klassenzimmer und versuchen mithilfe einer Smartphone-App gemeinsam eine Kugel zu balancieren. Da das Verhalten der Kugel, die Summe der Aktionen aller Spieler ist, müssen alle jeweils die Bewegungen der Mitspieler kompensieren. Nach einiger Zeit wird die Kugel einem kleinen Stoß ausgesetzt, woraufhin jeder anfängt gegenzulenken. Nach erfolgreichem zentrieren der Kugel, wir die Kugel erneut einem Impuls ausgesetzt. Nach 4 Minuten fällt die Kugel aus der Arena und die 3 Freunde schaffen ihre persönliche Bestleistung. Währenddessen befinden sich im benachbarten Zimmer die Lehrer, welche den bevorstehenden Unterricht vorbereiten und sich dazu gegenseitig Bilder und Artikel über die Nearfly Messenger App senden.*

Als interaktives Spiel, müssen beim dritten Szenario sehr viele Daten innerhalb kürzester Zeit übertragen werden. Dazu muss das System die Neigungsdaten aller Spieler kontinuierlich erfassen und die Kugel entsprechend der Summe aller erfassten Neigungsdaten bewegen. Zeitgleich fordert das Spiel eine möglichst kleine Latenz, um dem Geschick der einzelnen Spieler nicht entgegen zu wirken und dadurch ein synchrones starten der Runde 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. Zum Notieren und kontrollieren der Zwischenergebnisse verwenden alle die Score-Board-Notepad-App. Derjenige der als letzten dran war, ist dafür zuständig die Punkte des Spielers zu verwalten, welcher gerade dran ist. Mark beginnt mit der Rolle des Schreibers und verwaltet die Punkte, während die zwei anderen als Zuschauer kontrollieren, ob Mark die Punkte richtig eingibt. Ricky startet und trifft die Mülltonne. Der Ball berührt zuvor jedoch die Wand, sodass Mark Rickys Punktestand um Eins inkrementiert. Daraufhin trifft Ricky die Mülltonne ohne, dass der Ball etwas berührt und Mark erhöht seine Punkte um zwei. Acht weitere Versuche später, wird die Rickys Runde beendet. Nun ist Steffan dran und sein Vorgänger Ricky zuständig für dessen Punkte. Nach Ablauf von 20 Runden, in denen jeder jeweils 10-mal werfen durfte wird das Spiel beendet und der Gewinner angezeigt.*

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

*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.*

Im 5. Szenario werden

## Funktionale Anforderungen

Die Eignung zur Übertragung von Multimedia-Daten ist unterschiedli

* Hauptsächlich werden bei Smartphones Bilder(PDF, JPEG wie auch GIFs) versendet. Dabei sind Bilder die Datei die im am Unterschiedlichsten ist
* Soll ein Bilder, vergleichbar WhatsApp innerhalb z.B.: von 10 Sekunden übertragen werden, so ist eine Übertragungsgeschwindigkeit von \_\_ nötig.

Tabelle 1: Durchschnittsgröße von Multimedia-Daten

|  |  |  |
| --- | --- | --- |
| Multimedia-Datentyp | Kleines Exemplar | Größeres Exemplar |
| PDF | 90 KB |  |
| JPEG-Bilder |  |  |
| PNG |  |  |
| MP4 |  |  |
| APK |  |  |

* Aus vorhergehenden Szenarien nun folgende Anforderungen
* Die Messenger App verlangt, das senden von beliebigen Multimedia Nachrichten.
* Die
* Während aus der Messenger App die anforderung ensteht, beliebige binäre Nachrichten zu versenden, müssen andere

Aus den vorhergehenden Szenarien kann

Die aufgestellte Anforderungsmatrix (siehe Tabelle 2) zeigt die Korrelation der aus den Szenarien ermittelten Anforderungen. So ist etwa klar festmachbar, dass das Versenden von textbasierten Nachrichten eine fundamentale Anforderung ist, welche in allen der o.g. Szenarien benötigt wird. Weiterhin lässt sich aus der Anforderungsmatrix schließen, dass die Szenarien mit mäßigem Datenaufkommen und deinigen mit höheren Datenaufkommen sehr ähnliche Anforderungen besitzen, demnach grobe 2 Arten von System-Typen entstehen, auf welcher die Eignung des zu implementierenden Services im späteren getestet werden kann.

Zu den Anforderungen, welche sich bereits aus den zuvor genannten Szenarien identifizieren ließen, sind weitere sinnvolle Anforderungen, wie eine App-übergreifende Sichtbarkeitsbeschränkung denkbar, da sich verschiedene Anwendungen nie beeinflussen sollen. Auch wäre es wünschenswert, dass der Nutzer bei vorhandenen WLAN Netz auf MQTT umschalten oder im Umkehrfall zurück auf Nearby Connections schalten kann, ohne das Daten verloren gehen.

Tabelle 2: Anforderungsmatrix der Nearfly-Bibliothek

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **funktionale Anforderungen** | Messenger | Shared Touchpoint Canvas | Bouncing Ball | Score Board Notepad |
| Multimedia-Nachrichten senden | x |  |  |  |
| Text Nachrichten senden | x | x | x | x |
| Priorisierung der Nachrichtentypen | x |  |  |  |
| Trennbarkeit von unterschiedlichen Nachrichten innerhalb derselben App | x |  |  | x |
| Die Nachrichten müssen den Absender enthalten | x | x | x | x |
| Verzögerungsarmes Empfangen der Nachrichten |  | x | x |  |
| Möglichkeit konfigurierbar Nachrichten zeitgleich zu empfangen, auch wenn verzögert |  | x | x |  |
| Alle benachrichtigen sobald sich ein neuer Spieler dazu verbindet |  | 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 |
| Kein Datenverlust beim Modus-Wechsel (Nearby Connections/ MQTT) |  |  |  |  |
| Das System muss dem Nutzer die Möglichkeit geben, zu entscheide ob bei Verbindungsabbruch, alle noch nicht versandten Pakete bis zur wiederhergestellten Verbindung gehalten oder verworfen werden sollen | x | x | x | x |
| Das System soll dem Entwickler die Möglichkeit geben, die höchste auftretenden Round-Trip Time während des Betriebes zu messen. |  |  |  |  |

## Nichtfunktionale Anforderungen

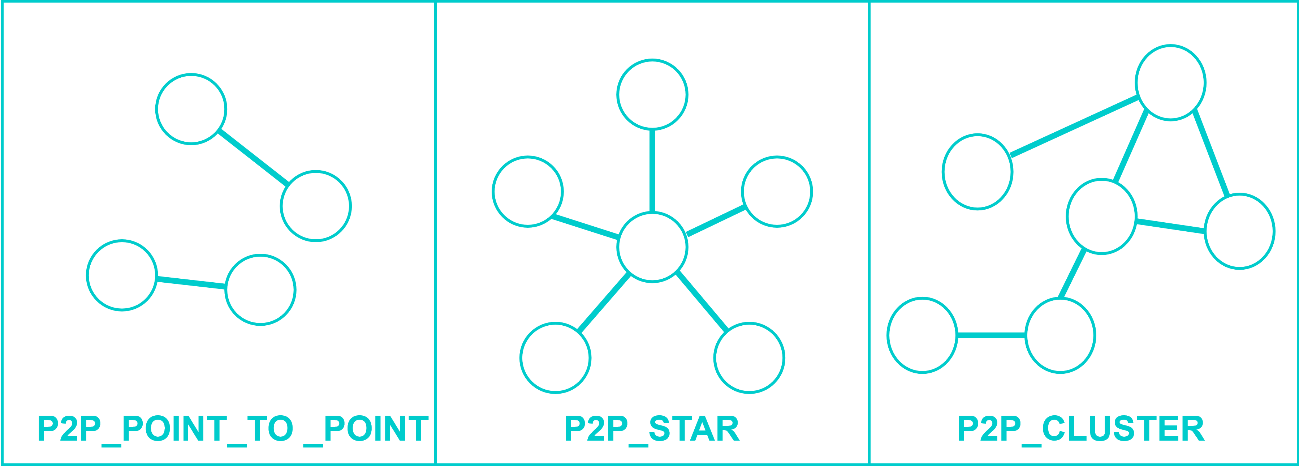
Die zu entwickelnde Bibliothek sollte eine Kompatibilität zu allen Android Versionen ab dem API Level 24 (Android 7.0) aufweisen. Weiterhin sollte das System sowohl im Nearby-Modus wie auch im MQTT-Modus ein ähnliches Verhalten in Bezug auf den Verbindungsaufbau, das Senden und dem Empfangen der Nachrichten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

* AAR mit Doku

## Gegenüberstellung von Nearby Connections und MQTT

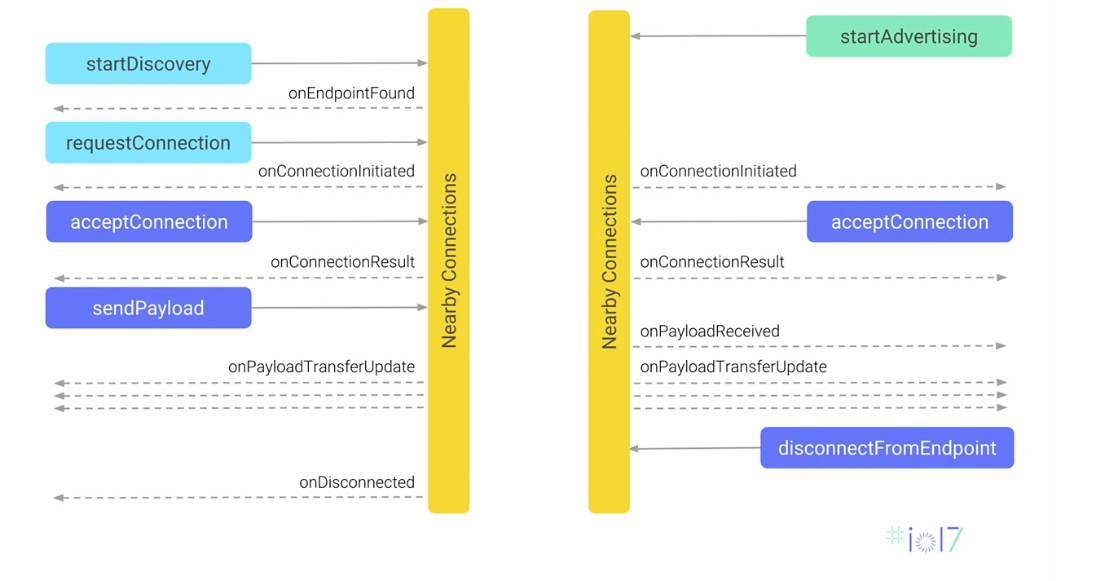
Nearby Connections wird vom Nearby Connection Team als eine *Peer-To-Peer-API* mit hoher Bandbreite und kleiner Latenz beschrieben, welche durch eine optionale Verschlüsselung einen sicheren Datentransfer zwischen den verbundenen Knoten ermöglicht [9]. Verbindungen werden dabei durch Nutzung von Bluetooth, Bluetooth-Low-Energy (BLE) und automatisch angelegten Wifi Hotspot hergestellt [9] 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 [9].

Vor dem eigentlichen Verbindungsaufbau muss eine Netzwerktopologien (**Abbildung 1**) gewählt werden, welche sich ebenfalls auf die maximale Übertragungsrate im Netzwerk auswirkt. Nearby Connections bietet drei verschiedene Topologien (genannt Strategien) an. Die Erste nennt sich P2P\_CLUSTER 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 [5]. Durch die P2P\_START-Strategie 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 [7]. Das Netzwerk kann dann bis zu 7 Knoten aufnehmen [5]. Zuletzt existiert die P2P\_POINT\_TO\_POINT-Strategie, welche im Verhalten der P2P\_STAR-Strategie entspricht, mit dem Unterschied das jedem Knoten nur noch eine Verbindungsknoten gestattet wird.



**Abbildung 1.** 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 [9], was jedoch zu *trashing* führen kann[6] Sendet ein Gerät im *Discovering*-Zustand eine Verbindungsanfrage, welche vom *Advertiser* bestätigt wird, wird eine symmetrische Verbindung (siehe **Abbildung 2**) zwischen beiden Knoten initialisiert. Optional kann eine Authentifizierungsanfrage implementiert werden, welche Verbindungen nur nach einer erfolgreichen Bestätigung des angefragten Gerätes initialisiert. Diese Anfrage aktiviert die bereits erwähnte Verschlüsselte Übertragung zwischen den Endknoten [9]. Sind mindestens 2 Geräte verbunden können entsprechend **Abbildung 2**, durch Angabe eines Empfängers in Form einer *EndpointName*, Daten ausgetauscht werden. Dabei unterscheidet die API zwischen 3 Nachrichtentypen. Diese sind einerseits Byte-Nachrichten, welche auf 32KByte limitiert sind. Andererseits die zwei unlimitierten Datentypen File und Stream [9]. Die Frage nach der Limitation wird auch intern anders gehandhabt. Während bei den zwei unlimitierten Datenübertragungen *onPayloadReceived* als Header zum Starten einer kontinuierlichen Übertragung empfangen wird und *onPayloadTransferUpdate* (**Abbildung 2**) die erfolgreich übertragenden *Chunks* indiziert, werden bei der limitierten Byteübertragung die zu übertragenden Bytes vollständig als *Single-Chunk* im Header mitübertragen, sodass diese bereits bei *onPayloadReceived* zur Verfügung stehen [5] .



**Abbildung 2:** 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. [8]

Eine Verbindung wird mithilfe eines Verbindungsbefehles initiiert und unterstützt UTF‑8 enkodierte Parameter, wie die *ClientId*, *Username*, *Password* sowie Feinjustierungen für einen *LastWill*-Nachricht, die Aufgerufen wird, wenn ein Client unangekündigt, die Verbindung beendet. [8]. 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.

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 [11] haben, sodass der Broker die Nachrichten an Subskribierte Clients weiterleiten kann. [8]. Nachrichten werden dabei stets als single-Chunk versandt.

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].

Der Übersicht sollend die ermittelten relevanten Unterschiede beider Technologien sollen in **Tabelle 3** zusammengefasst werden.

**Tabelle 3:** Überblick über Unterschiede zwischen Nearby Connections und MQTT [9]

|  |  |  |
| --- | --- | --- |
| 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 Hotspot |
| 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 | Wird aktiviert, wenn Entwickler Authentifikation implementiert |
| 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 |
| Chunked-Encoding: | Single Chunk | Bytes-Payload: Single Chunk  File&Stream-Payload: Chunked |

* Wifi-Übertragungsgeschwindigkeit beim s5mini minimal 15 beim s9 max. 60 MBit/s. Durschnittlich 30 Mbit/s.
* Der Durchscnittswert der von Evan Fallis und Petros Spachos ermittelten Durchsätze liegt in etwa bei 30 Mbit/s. Mit Extrema von 19.9 und 42.4 [1].
* Nach Spezifikation hat sowohl Bluetooth 3.0, wie auch Bluetooth 4.0 eine maximale Übertragungsgeschwindigkeit von 25 Mbit/s. Die Tatsächliche Übertragungsgeschwidndigkeit liegt jedoch auch hier, wie Evan Fallis und Petros Spachos zeigen, bei etwas über 1 Mbit/s [1].

# Software-Design

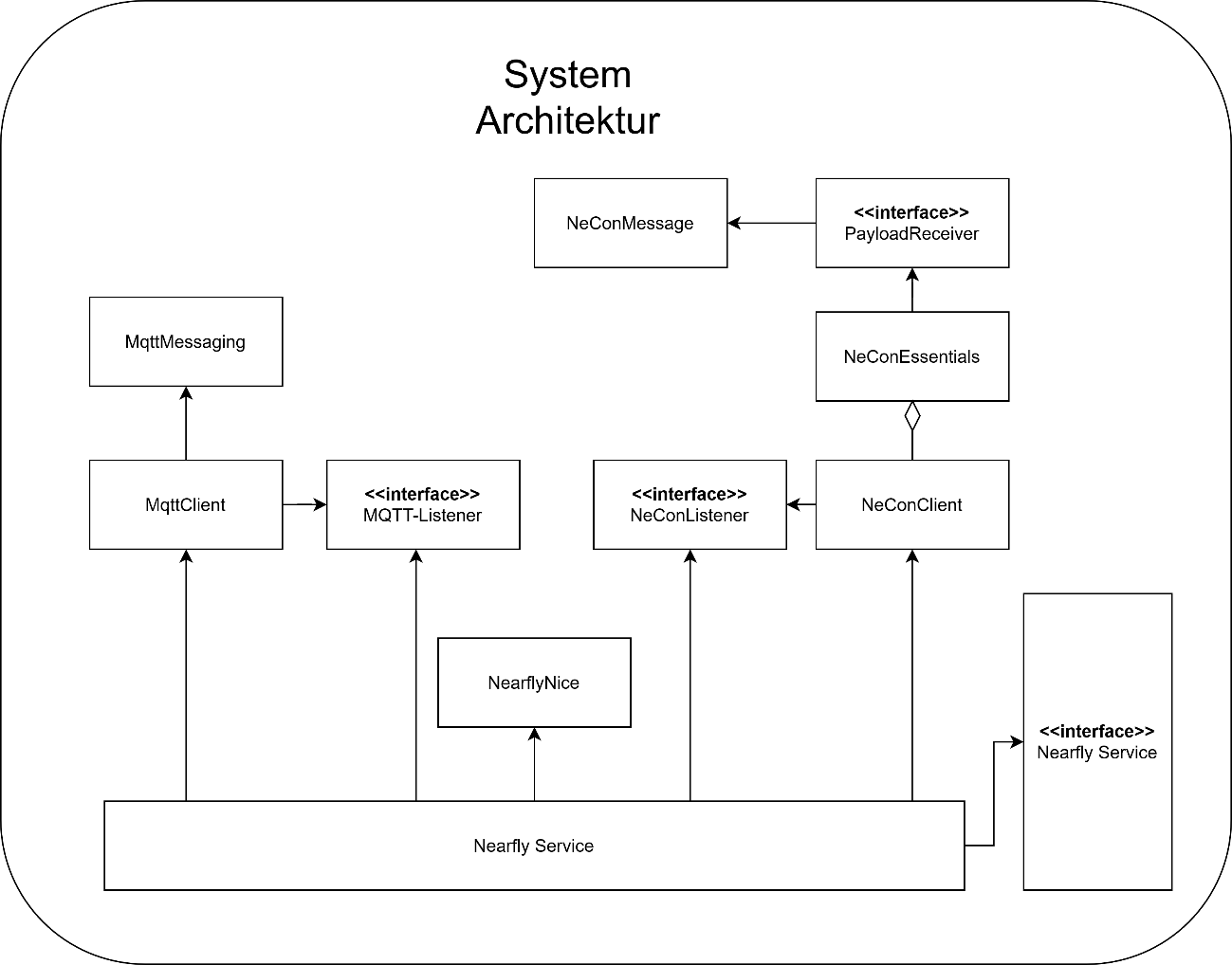
Da die zu entwickelnde Bibliothek ein Wrapper der Nearby Connections und MQTT API ist, soll die API der unterliegenden Technologien für den Benutzer verborgen sein, stattdessen werden diese über den NearflyService-Komponente (siehe **Abbildung 3**) angesprochen. Die NearflyService-API soll dabei ein Android Bound Service sein. Dies hat den Vorteil, dass der Service nach dem einmaligen starten solange wie die eigentliche App im Hintergrund laufen kann. Wird innerhalb derselben Anwendung die *Activity* gewechselt, bleibt die Verbindung zum Netzwerk bestehen. Damit die *Activities* mit dem Service interagieren können, genügt das senden einer Verbindungsanfrage (binding). Nach dem erfolgreichen *binden* bekommt der Client ein Objekt, durch welchen die Methoden wie üblich aufgerufen werden können. Da sich der NearflyService im Hintergrund befindet und demnach mehrere Activities auf diesen zugreifen können, soll die Kopplung zwischen den Activities und dem NearflyService durch Einsatz des Beobachter-Entwurf Musters verringert. Dazu soll der NearflyService beliebigen Activities die Möglichkeit geben, NearflyListener an- und abzumelden.

Die Nearby Connections API soll in zwei verschiedenen Komponenten ausgelagert werden (siehe **Abbildung 3**). 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 NeConClient-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 NeConClient- und der NeConEssentials-Komponente soll dabei die Abhängigkeit einer Basis-/Kind-Klasse bestehen.

Um auf eintreffende Nachrichten zu reagieren bietet die Nearby Connections API einen pirmitiven PayloadCallback-Listener an. Dieser in erweitert Form als PayloadReceiver-Komponente vertreten sein.

Auch die MQTT API soll in zwei Komponenten unterteilt werden. Während auch hier die MqttMessaging-Komponente jene ist, die das Funktionieren von MQTT sicherstellt, werden in der MqttClient-Komponente die für die Nearfly-Bibliothek benötigten Funktionalitäten, wie etwa das Senden von Multimedia umgesetzt.

Die beiden Client-Komponenten der unterliegenden Technologien, sollen ebenfalls über deren Interface-Komponenten zum NearflyService kommunizieren.



**Abbildung 3:** Software-Architektur

# Obligatorische Berechtigungen



Da innerhalb des Android Betriebssystems viel Wert auf Datenschutz gelegt wird brauchen Anwendungen Berechtigung für alle Operationen, welche andere Apps, das Betriebssystem oder den Benutzer nachteilig beeinflussen können [4]. 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 [4]. Diese müssen jeweils explizit von der Anwendung durch implementieren einer Abfrage, zur Laufzeit[[3]](#footnote-3) abgefragt werden. Während MQTT eine *Normal*-Berechtigung zum Öffnen von Internet Sockets braucht, erfordert Nearby Connections inkl. der Möglichkeit zum Senden von Daten[[4]](#footnote-4) insgesamt acht verschiedene Berechtigungen. Zu den *Normal*-Berechtigungen zählen zwei Berechtigungen zum discovern, sowie Koppeln von Geräten via Bluetooth und zwei Berechtigungen zum Auslesen der Wifi-Informationen, sowie Ändern des Wi-Fi-Konnektivitätsstatus, welche für den Hotspot verwendet werden. Die restlichen vier obligatorischen Berechtigungen gehören zum *protection level* dangerous und umfassen zwei Berechtigungen zum Abfragen vom ungefähren (FINE\_LOCATION), wie auch dem genauen Standort(COARSE\_LOCATION) 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.

# Verbindungsaufbau



Hat der Nutzer erfolgreich ein *Binding* zum Nearfly-Service erstellt, kann dieser, nach der Auswahl der zu benutzenden Technologie, welche im folglich als ConnectionMode bezeichnet wird, den Verbindungsaufbau initialisieren. Entsprechend der Anforderung, einer Sichtbarkeitsbeschränkung der Nachrichten außerhalb derselben Anwendung, soll dabei vor dem Verbinden ein 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 einen niedrigeren Datentransfer zulässt. 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 overheadärmerren, stärkere Datenübertragung, die besonders bei niedriger Teilnehmen Anzahl (3-4) 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 7 Teilnehmer beschränkt wird.

## Erstellung eines autonomen Peer-to-Peer Netzwerks

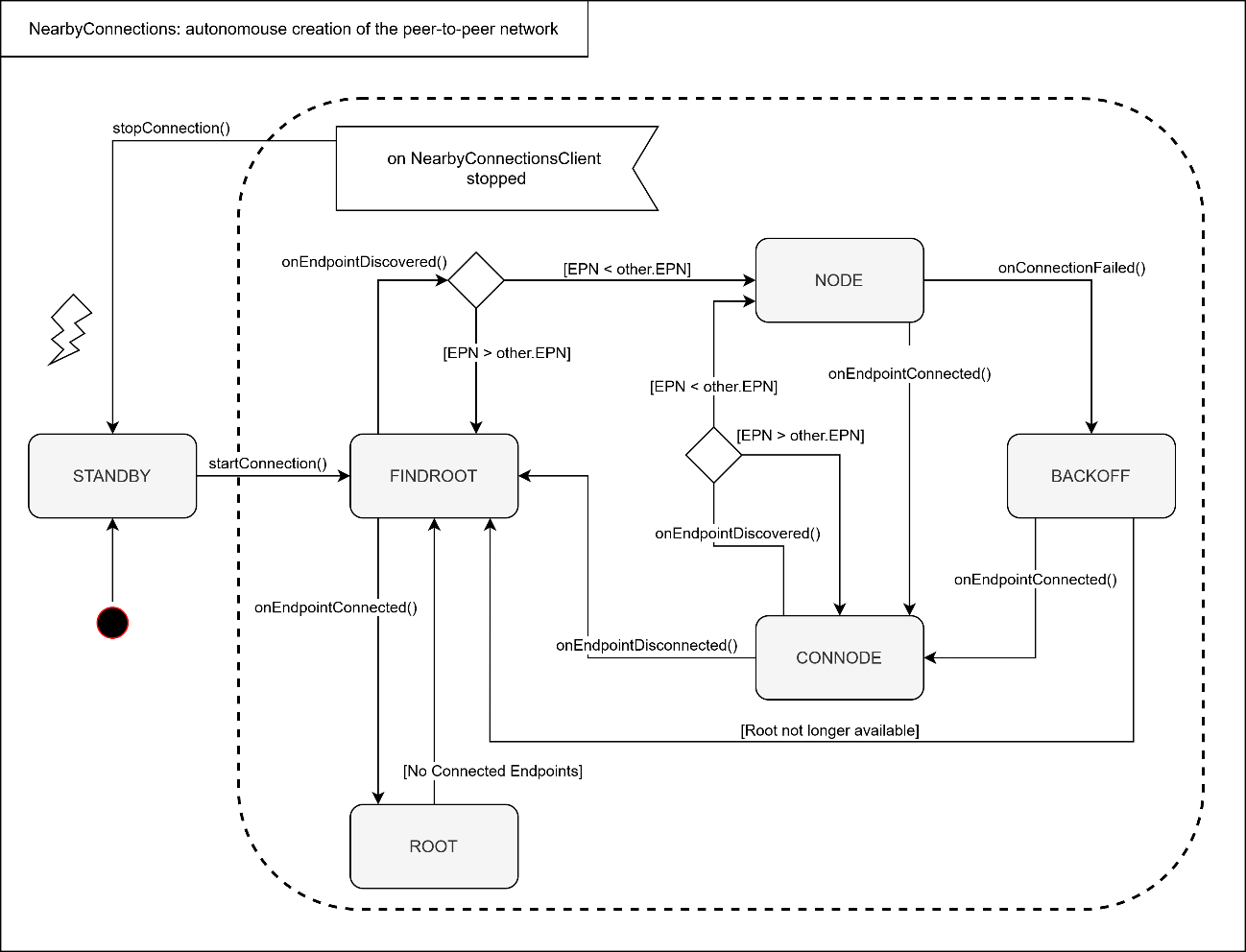
Um Gebrauch vom der Wifi-Direct-Technologie zu machen, wird das sternenförmige Netzwerk implementiert. Da jedem Gerät während des *Discovering*s, der Endpoint-Name aller entdeckter *Advertiser* angezeigt wird, muss dieser über den Endpoint-Namen entscheiden, welches Gerät als Root-Knoten dienen kann. Dazu soll der Endpoint-Name entsprechend Formel 1 aus der maximalen CPU Frequenz in Megahertz () des Gerätes und der Differenz zwischen der absoluten () und aktuellen Tageszeit in Millisekunden zusammengestellt werden. 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.

Formel 1: Berechnung des *Entpoint*-Namens

|  |
| --- |
|  |

Der in **Abbildung 3** gezeigte NearbyConnectionsClient soll 6 Zustände (siehe **Abbildung 4**) benutzen. Während der NearbyConnectionsClient nicht vom Nearfly-Service 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 eine eingehende Verbindunganfrage (onConnectionInstantiated) und befindet sich dieser nicht im Verbindungsaufbau myit einem anderen Knoten, akzeptiert dieser diese. Nachdem die Verbindung erfolgreich aufgebaut wurde (onEndpointConnection), wechselt dieser in den ROOT-Zustand und deaktiviert das *Discovering*, da der Root-Knoten nunmehr gefunden werden will, während der Knoten im NODE-Zustand in den CONNODE-Zustand wechselt. Befindet sich dieser gerade hingegen im Verbindungsaufbau, wird die Verbindungsanfrage abgelehnt (onConnectionFailed). Der anfragende NODE verfällt daraufhin in den BACKOFF-Zustand und wartet innerhalb eines gegebenen Zeitfensters eine zufällige Zeit (1-2 Sekunden aus der Empirie gewonnen), bis dieser daraufhin eine erneute Verbindungsanfrage lossendet. Da der Verbindungsaufbau auch dann fehlschlagen kann, wenn der Root-Knoten in diesem Moment aus dem Netzwerk austritt (*peer churn*), wird in diesem Fall der BACKOFF-Zustand verlassen werden. Der Knoten wechselt daraufhin 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, da dieser der einzige ist, welcher das *Advertising* aktiviert hat. Hat der neue Netzwerkteilnehmer aufgrund der eingestellten einen größeren Endpoint-Namen als der aktuelle Root, verbinden sich die 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 Nearfly-Service die Verbindung zum NearbyConnectionsClient, kehrt der Knoten zum STANDBY-Zustand. Dabei wird das *Advertising* und *Discovering* deaktiviert, sowie bestehende Verbindungen getrennt und aufgeräumt.



**Abbildung 4:** Zustandsdiagramm vom Verbindungsaufbau des NearbyConnectionClients

# Übertragung von Daten



Um die Trennung von Nachrichten innerhalb derselben App zu gewährleisten, 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 3 Nachrichtentypen. Zum einen können Status-Events (onLogMessage), wie die Zustandsänderungen seitens der NearbyConnections API oder den erfolgreichen Verbindungsaufbau wahrgenommen werden. Zum anderen können auf Multimedia-Nachrichten (onFile) und Binary-Nachrichten (onMessage) reagiert werden.

Besteht eine Verbindung zum Netzwerk, können Daten mithilfe eines der zwei vom NearflyService angebotenen Methoden (pubIt und pubFile) übertragen werden. Während kleinere Byte-Daten bis zu 30KBytes[[5]](#footnote-5) (MAX\_PUBIT\_MESSAGE\_SIZE) durch pubIt versandt werden können. Müssen größeren multimediale binäre Daten mithilfe der pubFile Methode übertragen werden. Dabei können zusätzlich zur zwingen erforderlichen Angabe des Empfangskanals, die Methoden mit einem Prioritätslevel von -20 (niedrigste Priorität) bis 19 (höchstmögliche Priorisierung), sowie ein Retain-Flag parametrisiert werden.

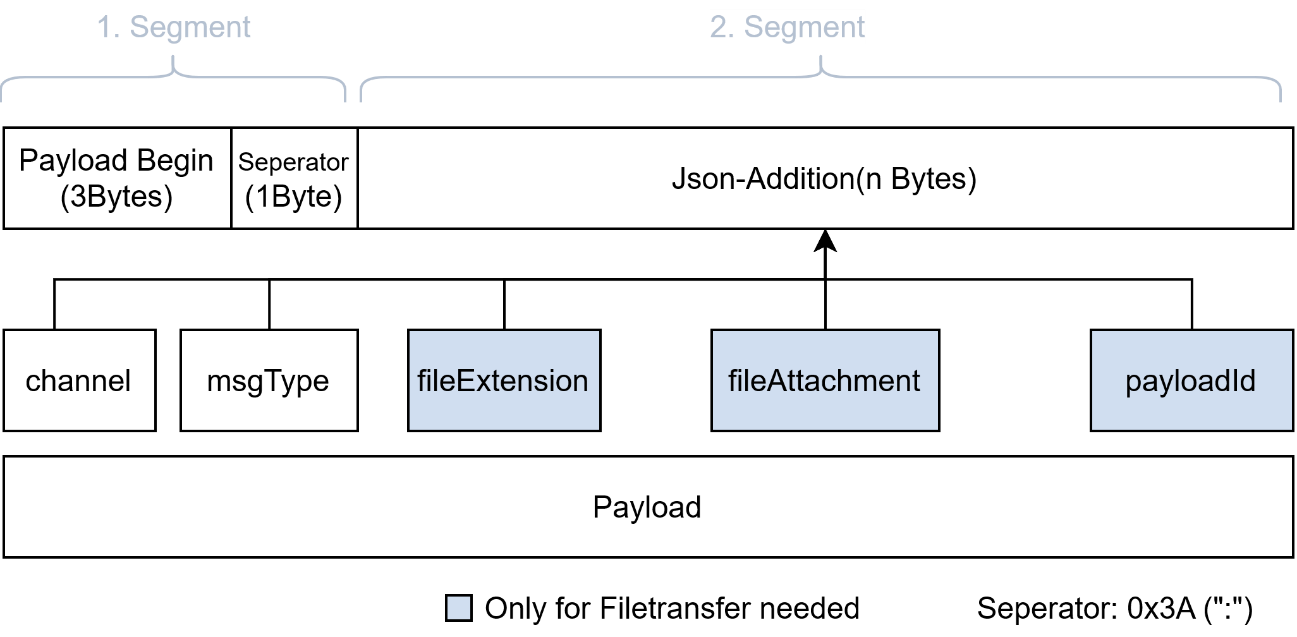
Ein gesetztes Retain-Flag meint dabei, das Nachrichten selbst im Verbindungslosen Zustand „gesendet“ werden können. Dazu gelangen diese zunächst in einen prioritätenbasierten Sendepuffer und werden erst dann gesendet, wenn eine Verbindung zur unterliegenden Technologie besteht und der TechSwitcher (dazu später mehr) 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 [9], reicht die Realisierung einer Sendeseitigen Priorisierung.

## Übertragung seitens NearbyConnections

Während Bytes-Nachichten 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[[6]](#footnote-6) 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, welche die Dateiendung und den File-Identifikator (zur späteren Zuordnung der Datei) beinhaltet. Diese gesonderte Byte-Nachricht soll im Folgenden als NeConFileInformation bezeichnet werden und zusätzlich dem späteren Entwickler die Möglichkeit gegeben, der file-Nachricht ein String-Beitext (textAttachment) anzuhängen.

Ruft der Nearfly-Service nun ein pubFile auf, sendet der NeCon-Client Client die Datei und zusätzlich die NeConFileinformation. Während beim normalen publish die zu versendende Nachricht in eine NeConBytesMessage verpackt wird. Dabei wird für beide NeCon-Nachrichten das in **Abbildung 1** gezeigte NeCon-Nachrichtenpotokoll verwendet. Um eine dynamische Headerlänge zu ermöglichen wird der Header in 2 Segmenten unterteilt. Das erste Segment hat eine fixe Bytelänge und wird aus einem UFT-8 encodierten String[[7]](#footnote-7) 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.



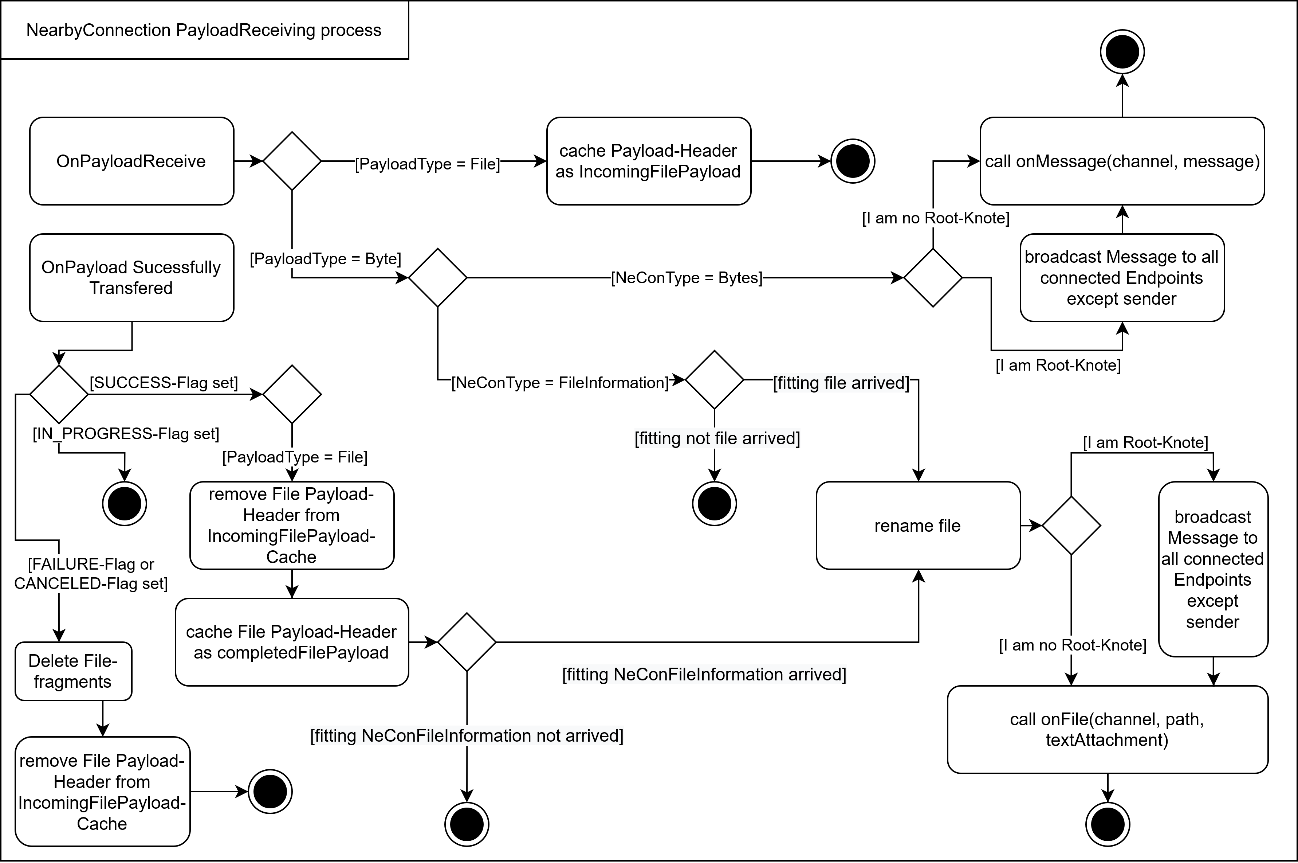
**Abbildung 5:** NeCon-Nachrichtenprotokoll

Empfängt die NearbyConnection API einen Payload-Header (siehe **Abbildung 6**), prüft der PayloadReceiver, ob diese vom Payload-Typ File oder Bytes ist. Handelt es sich hierbei um eine Bytes[[8]](#footnote-8)-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 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 (beinhaltet den Identifikator) 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 welchem 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 werden.

(Dabei ist zu beachten das Google ab Android 10 den scoped storage eingeführt hat, welchem dem Entwickler den Daten-Zugriff auf Daten außerhalb des Anwendungsspezifischen Verzeichnisses beschränkt. So wird dem Nutzer etwa nur der Zugriff auf spezielle Filetypen, bei ausschließlicher Benutzung der Uri gestattet.)

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 werden.



**Abbildung 6:** Prozessablauf beim Empfangen einer Nachricht seitens des NeCon-Clients

Als Einschränkung sei angemerkt, dass aufgrund des Beschränkten Einflusses, welcher dem Nearby Connection API Nutzer auf den Prozess der Dateiübertragung gegeben wird, der Kanal zwar mitübertragen wird, jedoch nicht dazu benutzt werden kann, den Datenübertragungsprozess, im Falle eines nicht abonnierten Kanals abzubrechen.

## Übertragung seitens MQTT

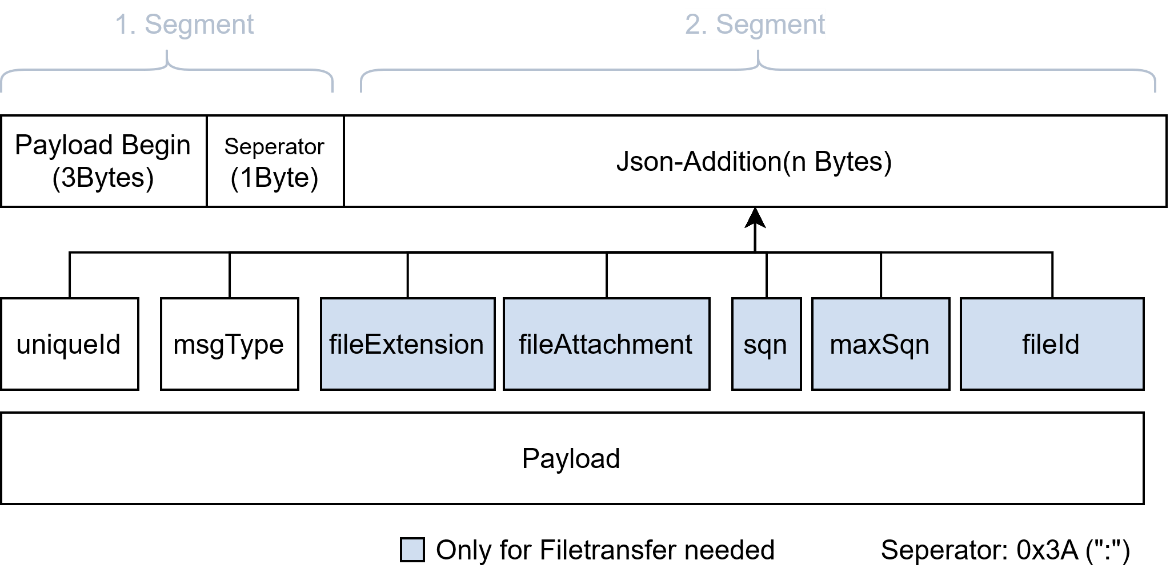
Da der Nearfly-Kanal auf das MQTT-Topic basiert, kann dieser als Topic (room/kanal) dem Room-String angehängt werden. Um das Übertragung von Multimedia-Daten Mithilfe der MQTT API möglichst der Nearby Connections API gleichzustellen, sollen versandte Daten nach Ankunft ebenfalls im Standarddownload-Verzeichnis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

Dies kann in einfacher Weise durch das Senden der gesamten Nachricht als String im JSON-Format realisiert werden. Die binäre Multimedia-Datei muss dazu durch das Base64-Kodierungsverfahren in einen String kodiert und beim empfangen erneut dekodiert werden. 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 [12] der ursprünglichen Datengröße benötigt wird. Eine weitere Möglichkeit ergibt sich durch Einsatz des NeCon-Nachrichtenprotokolls (**Abbildung 5**), das durch den dynamischen String Header, welcher durch das JSON-Segment leicht modifiziert werden kann, String von Payload-Bytes trennt und dadurch dem Overhead der ersten Möglichkeit entgegenwirkt.

Da die Nearby Connections API keine theoretische Datengröße angibt und MQTT eine theoretische Dateigröße von 256Mbyte [11] angibt, wird für das MQTT des Weiteren ein Chunking-Mechanismus implementiert, das Pakete ab 2Mbyte[[9]](#footnote-9) (MAC\_CHUNK\_SIZE) zerlegt. Um dies zu tun wird das JSON-Segment, um Sequenznummer (sqn), die maximal zu sendenden Fragmenten (maxSqn) und einen Identifikator (fileId) ergänzt. Womit sich das in **Abbildung 7** gezeigte JSON-Segment ergibt. Wird nun eine Filenachricht größer der MAC\_CHUNK\_SIZE gesendet, wird dieser zunächst mithilfe der vom Sender zufällig generierten fünfstelligen fileid, zwischengespeichert. Kommt ein Chunk aufgrund der eingestellten QoS zweimal an, wird das Duplikat ignoriert. Ist das letzte Paket angekommen. Werden die Zwischengespeicherten Bytes ausgelesen und in eine Datei geschrieben. Daraufhin werden die zwischengespeicherten Bytes gelöscht und die onFile-Methode des NearflyServices getriggert.

(Da die Nachrichten, welche im Peer-to-Peer Netzwerk des NeConClients nicht von den Sendern empfangen werden. Soll das JSON-Segment weiterhin eine uniqueID beinhalten, die dazu benutzt werden kann, um selbst versandte Nachrichten nicht zu empfangen.)

Die Bytes-Nachrichten, sollen ebenfalls mit dem Header versandt werden. Im Gegensatz zum NeConBytes-Pendant werden jedoch keine Headereinträge benötigt, womit zunächst ein 6Byte Overhead vom JSON-Segment entsteht. Der für eine leichtere Erweiterbarkeit in Kauf genommen wird.



**Abbildung 7:** Modifizierter NeCon-Nachrichtenprotokoll zur Nachrichtenübertragung seitens MQTT

# Nutzung der NearflyService API

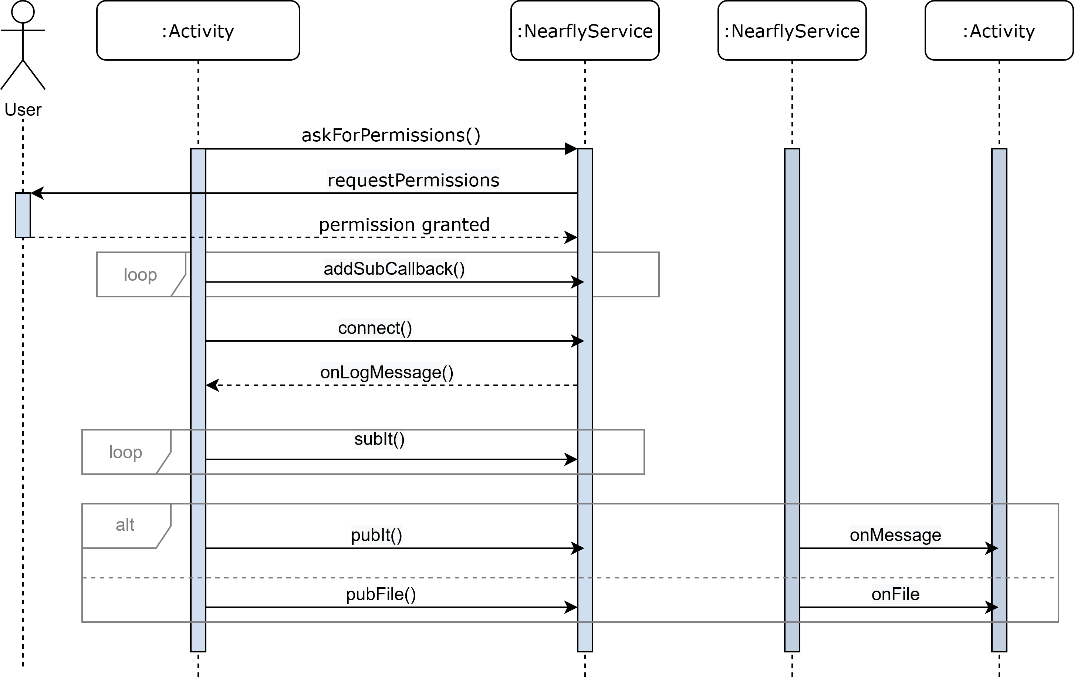
(War mit dem Artikel, dass hier gemeint?)

Nachdem die Bibliothek erfolgreich eingebunden und die obligatorischen Berechtigungen im Manifest eingetragen wurden, kann die Activity gestartet werden. Vor dem Verbinden zum NearflyService müssen die mandatorischen Dangerous-Berechtigungen z.B. durch Aufrufen der askForPermissions-Methode von Anwendungsnutzer gewährt werden.

Verfügt die App über alle nötigen Berechtigungen, kann ein NearflyListener angemeldet werden, um so auf spätere NearflyService-Evenets reagieren zu können. Durch connect kann unter Angabe des 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 10 Sekunden dauern kann und zuvor kein Datenaustausch möglich ist. Zu Beachten ist, das der NeConClient eine Verbindung als aufgebauten Netzwerk mit mindestens zwei Knoten definiert. Werden mehr Knoten erwartet, muss dies durch den API Nutzer implementiert werden.

Der Nutzer kann beliebige Kanäle durch die subIt-Methode abonnieren. pubIt und pubFile lösen bei den anderen Netzwerkteilnehmern das onMessage bzw. onFile-Event aus.



**Abbildung 7:** Ablauf der Nutzung der NearflyService API

# Evaluation

Die durch Implementieren der einzelnen Szenarien gewonnenen Erkenntnisse sollen nun durch vergleichen mit den in der Anforderungsanalyse (link) identifizierten Anforderungen verifiziert werden.

## Vergleichen der Prozesslaufzeiten zwischen MQTT und Nearby Connections

* Verbindungsaufbau mit 2, 3, 4 Peers. Wie gut Skalierbar? Mit Werte Belegen.
  + Z.B. vom Buch: Verbindung 10s 1. Peer, 30.s
  + Aktiviert Nearby Connections jedoch WIFI-Direct ist dies 3mal schneller (siehe buch)
* Senden von Daten:
  + MQTT vs Nearby: Textbasierte Nachrichten
  + Senden von Files, wer ist schneller. Wieviel?

ich schreibe Ihnen aufgrund der Nearby Connection API. Soweit habe ich eine minimalistische Beispiel-App entwickelt, welche praktisch nur zum connecten und publishen der Devices zuständig ist. Das Verbinden an sich funktioniert durch Discovery(D) und Advertising(A). Möglich ist hierbei, dass 1 Device D und der andere A spielt oder beide A&D zeitgleich. Das Verbinden von 2 nahliegenden Devices geht Flott (wenige Sekunden), da bloß 1 Verbindung benötigt wird. Der Nachrichtenaustausch danach deutlich schneller. Versucht man jetzt ein 3. Device hinzuzufügen, fängt die Nearby-Technology an zu humpeln. Während die erste Verbindung vergleichbar dem 2 Devices Anwendungsfall recht zügig geht, kann es bei den folgenden Verbindungen je nach Glück von einer halben Minute bis zu mehreren wenigen Minuten dauern. Extrapoliert man das Verhalten, wird das z.B. bei 5 Devices (bei 10 benötigten Verbindungen) schon deutlich schwieriger. Das obere Verhalten gilt für ein Clustering-Netzwerk. Setzt man jedoch ein Device als localen Server(Stern-Netzwerk), braucht man nurnoch einen der Discovert und alle anderen müssen nurnoch publishen, geht das deutlich besser. Ich experementiere soweit noch mit Versionen, Einstellungen ... Mein anliegen ist nun folgender: Sie haben im Exposé erwähnt gehabt, das Sie keinen Smartphone als Server haben wollten, derzeitig scheint das jedoch tatsächlich dahin zu führen.

## Verifikation der Anforderungen

* Wurden die Anforderungen erfüllt?
* Welche wurden warum nicht erfüllt?

## Erkenntnisse des Messenger Szenarios

* Wie schnell können Größere Nachrichten übertragen werden? JPGs? PNGs?
* Konkrete Zahlen. Abhängigkeit zwischen Größe, Geschwindigkeit & Endpunkte in MQTT & Nearby
* Senden von Files, Wie schnell funktioniert dies im Online-Modus, wie schnell im Offline-Modus

## Erkenntnisse des Touchpoint Szenarios

Nearby Connections:

* Wie im Szenario 2 (link) bereits geschrieben, ist der Hauptzweck der Shared Touchpoint Canvas Anwendung das Testen einer gewissen Echtzeitfähigkeit des Systems in Abhängigkeit zu den verbundenen Geräten.
* Bsp.: Zahlen zur Geschwindigkeit & Skalierbarkeit der Anwendung
* Wie verhielt sich Anwendung bei 1 Device, wie bei 4 ?
* Wie verhielt sich das System bei so und sovielen Werten pro Sekunde?
* Im Log, wird für das Sasmsung S9 desöfteren die Fehlermeldung:
* EndpointManager failed to find EndpointChannel over which to write) angezeigt. Woran das liegt konnte leider nicht herausgefunden werde und die einzig gefundene Frage identische Frage blieb auf Stackoverflow unkommentiert.
* Phänomen das Auftauscht: Besonders wenn versucht wird viele Geräte zu verbinden und e, kann es manchmal passieren, das Geräte keine symmetrische Verbindung aufbauen, somit also nur eines der beiden Peers publishen und der andere nur empfagen kann, jedoch andersherum nicht.
* Phänomen: 4 Smartphones verbunden, eines davon Samsung S5 Mini, Root konnte keine Nachrichten senden, als S5 vom Netzwerk getrennt. Wurden alle Nachrichten, die bis dahin nicht gesendet wurden im Netzwerk verteilt. War der Root überlasstet?
* Erkenntnis: Ältere Smartphones, wie dies beim S5 Mini der Fall ist, scheinen sich, wenn diese als Root gelten, nicht zum 2 Knoten zu verbinden. Haben diese eine Verbindung hergestellt, lehnen diese die zweite Verbindungabfrage trotz bedingungslosem acceptConnectio-aufruf ab.
* Erkenntis: Die Verbindung von Älteren Geräten stürtz zudem des öffteren ab.

Die Übertragung über das MQTT-Protokoll ist relativ leicht, da MQTT als Payload Bytes verw

* Ist der ConnectionMode
* Da der Payload von MQTT eine Binary ist, können mit MQTT beliebige Daten Übertragenwerden.
* Daten sollen nun auf zwei Arten ü
* Erfahrungsgemäß kann es bei Nearby besonders bei etwas älteren Geräte (Bsp. S5mini) vorkommen, das die Verbindung während einer Spielsession beendet wird. Dem muss entgegengewirkt werden, indem die API in solch einem Fall, die zu versendenden Pakete innerhalb einer gewissen Zeit hält und diese dem betroffenen Gerät beim erneuten Verbindungsaufbau zusendet.
* Übertragung von TextNachrichten
  + Nearby: Wahl zwischen Bineary und Stream + konkrete Werte, was für ein unterschied hat was gemacht?
  + Bei MQTT hingegen reicht es die Datei im Stream einzulesen und diese in ein Binary-Array zu konvertieren. Die Binary-Datei kann dann als Single Chunk gesendet werden
* Übertragung von Files
  + Wenn die Aktivity auf Files zugreift, muss darauf geachtet werden, das diese die rechte READ\_EXTERNAL\_STORAGE und WRITE\_EXTERNAL\_STORAGE besitzt.
  + Sequenzdiagram wie im Buch(11.04.20) für Nearby aufzeichnen
* Implementierung von Stream üvertragung. Aufpassen das RAM nicht vollläuft, daher Datei lokal abspeichern.
* Wahl des Threads für die Forwarding komponente: Thread-Pool Executor
* Aktiviert NearbyConnection WIFI-Direct wird das auf älteren Geräten, wie etwa bei mri s5Mnii und Lg wird eine Benachrichtugng angezeigt. Bei neueren Geräte wie etwa dem S7 und S9 ist das nichtmehr der Fall.
* Android hat starke Restriktionen im Bezug auf das Zugreifen von Files
* Angefangen bei READ\_EXTERNAL\_STORAGE berechtiung die ab API level 19 eingeführt wurde
* Die Erfahrung zeigt, das oftmals viele Funktione, wie etwa das kriegen einer Uri aus dem Path nicht ohne weiteres Funktionierien. Beispielsweise kann dort ein FileProvider helfen, welcher jedoch nur im Kontext einer Applikation fuktioniert und damit für eine Bibliothek gänzlich ungeeignet ist.
* Ein Android Syten

## Erkenntnisse des Bouncing Ball Szenarios

* Die Anforderugen ,welche die Bouncing Ball stellt konnten im Offline-Modus leider nicht erfüllt werden.
* Genauso wäre die Möglichkeit des Download-Verzeichnis der Nearby Conenctions API auf den Anwendungsspezifischen Verzeichnis zu ändern erwünscht. Da somit die Dangerous-Berechtigung für den externen Daten-Zugriff entfallen würde.

# Fazit

….

## Ausblick

…

Quellenverzeichnis

[1] 2018 Global Information Infrastructure and Networking Symposium (GIIS). [Place of publication not identified]: IEEE; 2018.

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

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

[6] *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.

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

[8] *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/.

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

[10] *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/.

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

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

[13] *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. Das Anfragen zur Laufzeit wurde Android 6.0(API level 23) eingefügt und betrifft demnach nicht die Geräte, welche Android 5.1(API level 22) haben [4]. [↑](#footnote-ref-3)
4. Sollen keine Daten transferiert werden, fallen zwei Berechtigungen Weg. [↑](#footnote-ref-4)
5. Diese Grenze entsteht aus der Limitierung für Byte-Nachrichten der Nearby Connections API, welche aus dem Kapitel 2.4 bekannt und einer großzügigen Toleranz um größere zukünftige Erweiterungen des NeCon-Protokolls( siehe 6.1) zu kompensieren. [↑](#footnote-ref-5)
6. Dies erklärt die für die File-Übertragung Notwendige Dangerous-Berechtigung für das Lesen und Schreiben von Daten außerhalb des Anwendungsspezifischen Verzeichnisses [↑](#footnote-ref-6)
7. Jedes String-Element braucht dabei genau ein Byte, da dies für Zeichen, welche in der ASCII-Tabelle zu finden sind, garantiert wird. Andere Zeichen hingegen können bis zu 4Bytes beanspruchen. [↑](#footnote-ref-7)
8. Nicht zu verwechseln mit der gleichbenannten NeConByte-Nachricht [↑](#footnote-ref-8)
9. Aus eigenen empirischen Untersuchungen zeigt sich das ein kleiner Schwellwert (etwa 30KByte), MQTT stark verlangsamen (auch, wenn dies im IBM Entwicklerforum so demonstriert wird [10]). Ein Schwellwert von 30KByte etwa, würde bei einem 5Mbyte großen Bild, den Broker zunächst mit 167 Chunks fluten. [↑](#footnote-ref-9)