|  |  |  |
| --- | --- | --- |
| **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 01.05.2020 |
|  |
| 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 Szenario 1 – Messenger 3

2.2 Szenario 2 – Shared Touchpoint Canvas 4

2.3 Szenario 3 – Bouncing Ball 4

2.4 Szenario 4 – Score Board Notepad 4

2.5 Funktionale Anforderungen 6

2.6 Nichfunktionale Anforderungen 7

2.7 Gegenüberstellung von Nearby Connections und MQTT 7

2.8 (Netwerkarchitektur/ Software Interfaces) **Fehler! Textmarke nicht definiert.**

3 Software-Design 11

3.1 Systemarchitektur 11

3.2 System Features (Hier nicht zu detailliert auf Komponenten eingehen. Später dann in Implementierung) 11

4 Implementierung des Nearfly Services 15

4.1 Erstellen der Bibliothek 15

5 Implementierung der Szenarien 16

5.1 Messenger App 16

5.2 Shared Touchpoint Canvas 16

5.3 Skart 16

6 Evaluation 17

6.1 Verifikation der Anforderungen 17

6.2 Erkenntnisse des Messenger Szenarios 17

6.3 Erkenntnisse des Touchpoint Szenarios 17

6.4 Erkenntnisse des Skart Szenarios 18

Fazit 19

Ausblick 19

Quellenverzeichnis 20

# Einleitung

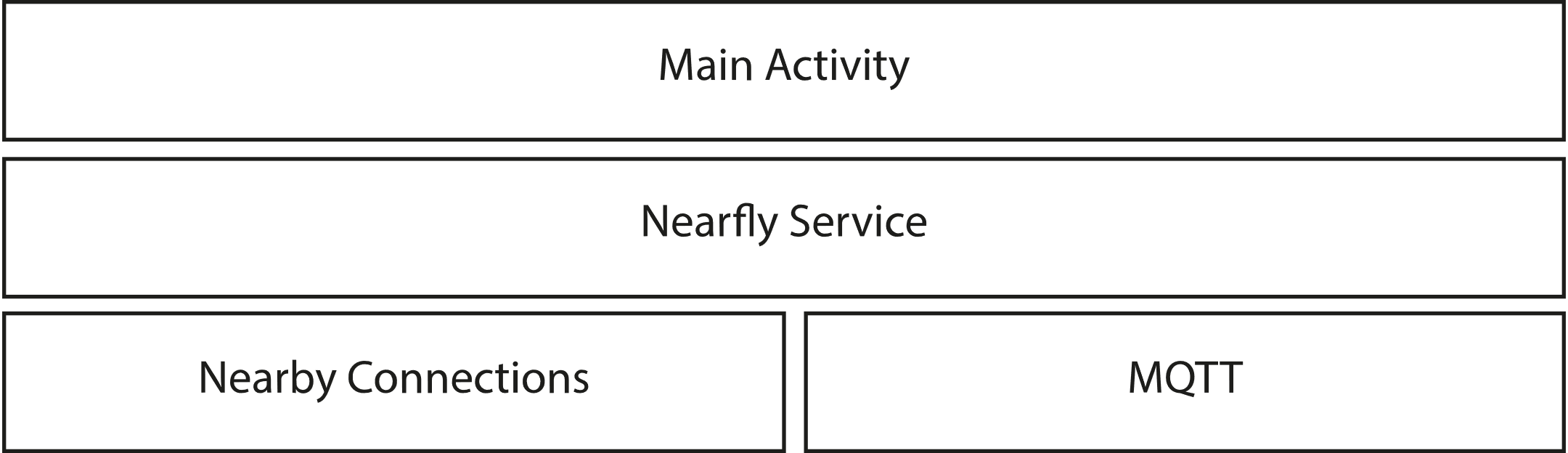
* Obwohl Smartphones ihren Ursprung im Telefon haben, rückt das Telefonieren mit einem Telefon immer weiter in den Hintergrund. Tatsächlich zeigt eine auf Statistica veröffentliche Studie, das bereits im Jahre 2017 in Deutschland die Funktion Kurznachrichten zu versendet mehr als Doppel so häufig, wie das Telefonieren verwendet wird [1]. Solche Studien zeigen den Stellenwert, den die Vernetzung heutzutage in unserm Leben einnimmt.
* Um Daten unter Mobilen Endgeräten zu versendet wird dazu häufig auf bestehende Technologien, wie MQTT zurückgegriffen (link). Dies geschieht zu meist durch einbinden fertiger Bibliotheken, die von den Entwicklern in das Projekt eingebunden werden können.
* Mit dem Ursprung im Internet of Things schafft es MQTT aufgrund seiner wenigen 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 [9].
* 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 bestehende internetverbindung.
* 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 könnnen sich beiden Technologien perfekt ergänzen und zusammen Entwickelrn eine Technologie bietetn, die Sowohl 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, das ein Entwickler bloß eine API angeboten bekommt, welche beide Technologien verwendet)

## 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). Da Marius viel Wert auf Datenschutz legt und am besten keine Daten über das Internet senden möchte, hat dieser eine Messenger App gefunden, die auch offline funktioniert und bereits eine Gruppe für sein Team erstellt. 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 enthält einer klassischen Messenger App, welche es dem Entwickelteam erlaubt offline zu kommunizieren. So soll etwa das System dem Nutzer die Möglichkeit geben, 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 durch beinhalten von Absenderinformationen (Name, Absende-Zeit) 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 1) 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. **// Muss der obere Text auf alle Tabellenzeilen eingehen?**

Tabelle 1: 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. |  |  |  |  |

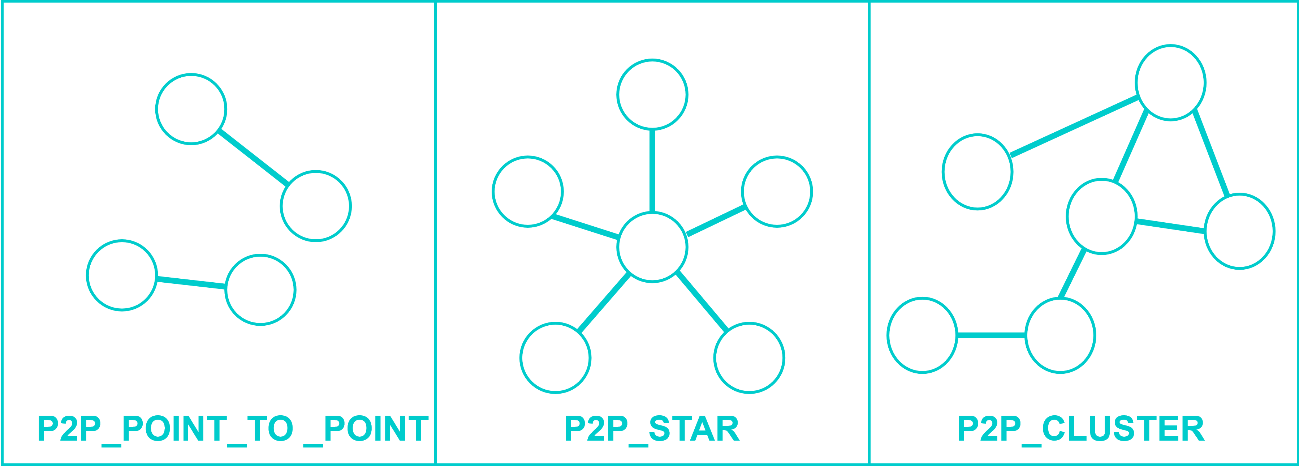
## Nichtfunktionale Anforderungen

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.

## Gegenüberstellung von Nearby Connections und MQTT

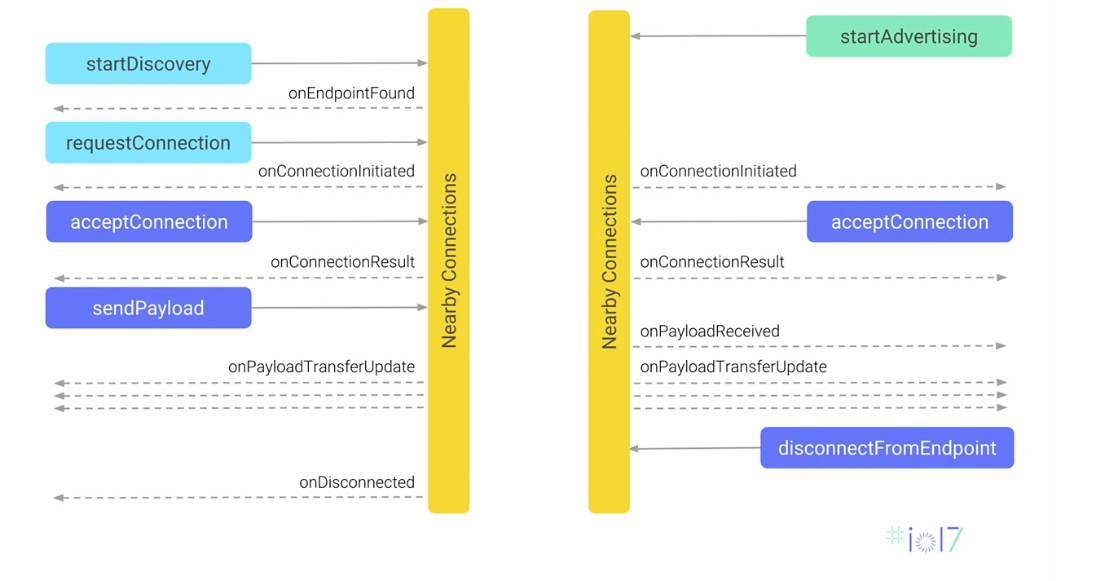
Nearby Connections wird vom Nearby Connection Team als eine *Peer-To-Peer-API* mit hoher Bandbreite und kleiner Latenz beschrieben, welche durch eine optionale Verschlüsselung einen sicheren Datentransfer zwischen den verbundenen Knoten ermöglicht [7]. Verbindungen werden dabei durch Nutzung von Bluetooth, Bluetooth-Low-Energy (BLE) und automatisch angelegten Wifi Hotspot hergestellt [7] 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 [7].

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 [3]. 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 [5]. Das Netzwerk kann dann bis zu 7 Knoten aufnehmen [3]. 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 [7], was jedoch zu *trashing* führen kann[4] Sendet ein Gerät im *Discovering*-Zustand eine Verbindungsanfrage, welche vom *Advertiser* bestätigt wird, wird eine symmetrische Verbindung (siehe **Abbildung 1**) 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 [7]. Sind mindestens 2 Geräte verbunden können entsprechend **Abbildung 1**, durch Angabe eines Empfängers in Form einer *EndpointID*, 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 [7]. 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 1**) 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 [3] .



**Abbildung 1:** 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 [2]. 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. [6]

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. [6]. Die MQTT Spezifikation sieht dabei keine Verschlüsselung der Nachrichten vor, um das Protokoll möglichst simple zu halten [2], 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 [8] haben, sodass der Broker die Nachrichten an Subskribierte Clients weiterleiten kann. [6]. 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 [2].

**Tabelle 2:** Überblick über Unterschiede zwischen Nearby Connections und MQTT [7]

|  |  |  |
| --- | --- | --- |
| Aktion | MQTT | Nearby Connections |
| Netwerktopologie | 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 |
| 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 | EndpointID |
| 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 |

### Architektur der API

* Wahl zwischen API als Service und API ehrbare Aktivity.
* Vor und Nachteile + Wahl der Entscheidung

# Software-Design

* Aus den Anforderungen ist ersichtlich, das
* Die Nearfly-Komponente wird als Bound Service implementiert.
  + Dadurch ensteht der Vorteil das dieser Service, nach dem einmaligen starten solange, wie die eigentliche App im Hintergrund läuft. Wird innerhlab derselben Anwendung die App gewechelt muss der Service nicht erneut gestartet werden. Es genügt ein bind-Anfrage, um die Aktivity mit dem Service zu verbinden.
  + Service bietet dabei ein client-server interface
  + Wird der Service nicht länger benötigt, kann sich der client durch aufrufen eines unbindService() vom Service löchen.
  + Ein Service kann dabei entwerder durch bindService() oder startService() gestartet werden. Wird ein Service durch ersteres gestartet, wird dessen Lebenszeit durch die verbundenen Clients bestimmt. Sodass dieser automatisch vom Android Betriebssytem beendet wird, sobald der letzte Client ein unbindService aufruft.
* Entsprechend den MQTT und Nearby
* Die Schnittstellen, welche System anbietet sollen ähnlich der Schnittstellen des unterliegenden Systemen sein
* Während NearbyConnectionsAbstract die Grund-Funktionalitäten von Nearby Connection, wie etwa:
  + Starten und Stoppen von Advertising und Discovering
  + Die Speicherung von gefundenen, wie verbundenne und pendin Entpunkten
  + Das Akzeptieren Verbindungen
* Verwaltet, erbt der MyNearflyConnecitonsClient diese Funktionalitäten und benutzt diese. Durch Benutzung einer zweiten Komponente soll möglicchst wenig an der NearbyConnectionsAbstract verändert werden.
* In MyNearflyConnectionsClient werden die Funktionen implementiert, welche benötigt werden, damit das System im Nearfly-Modus autonom ein Netzwerk aus den Nahliegeneden Knoten formen kann. Sowie das Nearbyseitige senden und empfangen von Nachrichten, filtern…
* Receive Payload Callback enspricht einer Erweiterung des von Nearby angeboteten Interfaces PayloadCallback, zum handeln.
* Ähnlich wie die beziehung zwischen NearbyConnectionsclient und derer Basisklasse, soll auch MQTTMessaging nicht direkt bearbeitet werden, sondern die Funktionalitätetn im My MQTT Client implementiert werden.
* Wie üblicherweise im Android-Framework soll auch die Nearfly-Serivce Bibliothek mit der eigenltichen Aktivity über ein Nearby Interface kommunizieren. Dabei soll diese die Methode onFile, onMessage und onLog anbieten.
* Wie auch bereits in den Anforderungen sollen Reintext-Nachrichten von File-Nachrichten unterschieden werden. Dies liegt daran das Nearby selbst beide Nachrichtentypen anders handhabt und die Files nach dem Empfangen direkt speichert.
* MsgForwarder Komponente wird im späteren benötigt, weil die Nachrichten später weitergeleitet werden müssen, da Nearby

Im Folgenden wird Design veranschaulicht:

**NearflyMessage:** Erweiterte Nachricht, welche hilft Nearby-Messsages an MQTT-Messages assimilieren. Beinhaltet bisher: Topic, Payload… zukünftig wahrscheinlich noch sequence number, Gesamtanzahl der Chunks pro Paket, Priorität, sendLatest + evtl. timestamp

* Evlt.

**Nearby Unpublisher:** Wird gebraucht, da Nearby im gegensatz zum MQTT Protokoll Nachrichten „persistent“ publisht. Damit werden gepublishte Nachrichten, ähnlich MQTT publishes mit aktiver retain flag, bis zu ihrem unpublish von allen subscribern ein mal empfagen.

* Nach publish, wartet dieser durch sleep bis zum aublauf der timetolive der gepublishten Nearby Nachricht und unpublisht diese.
* Sonst legt dieser sich schlafen, ein erneutes publishen wacht diesen durch ein notify auf

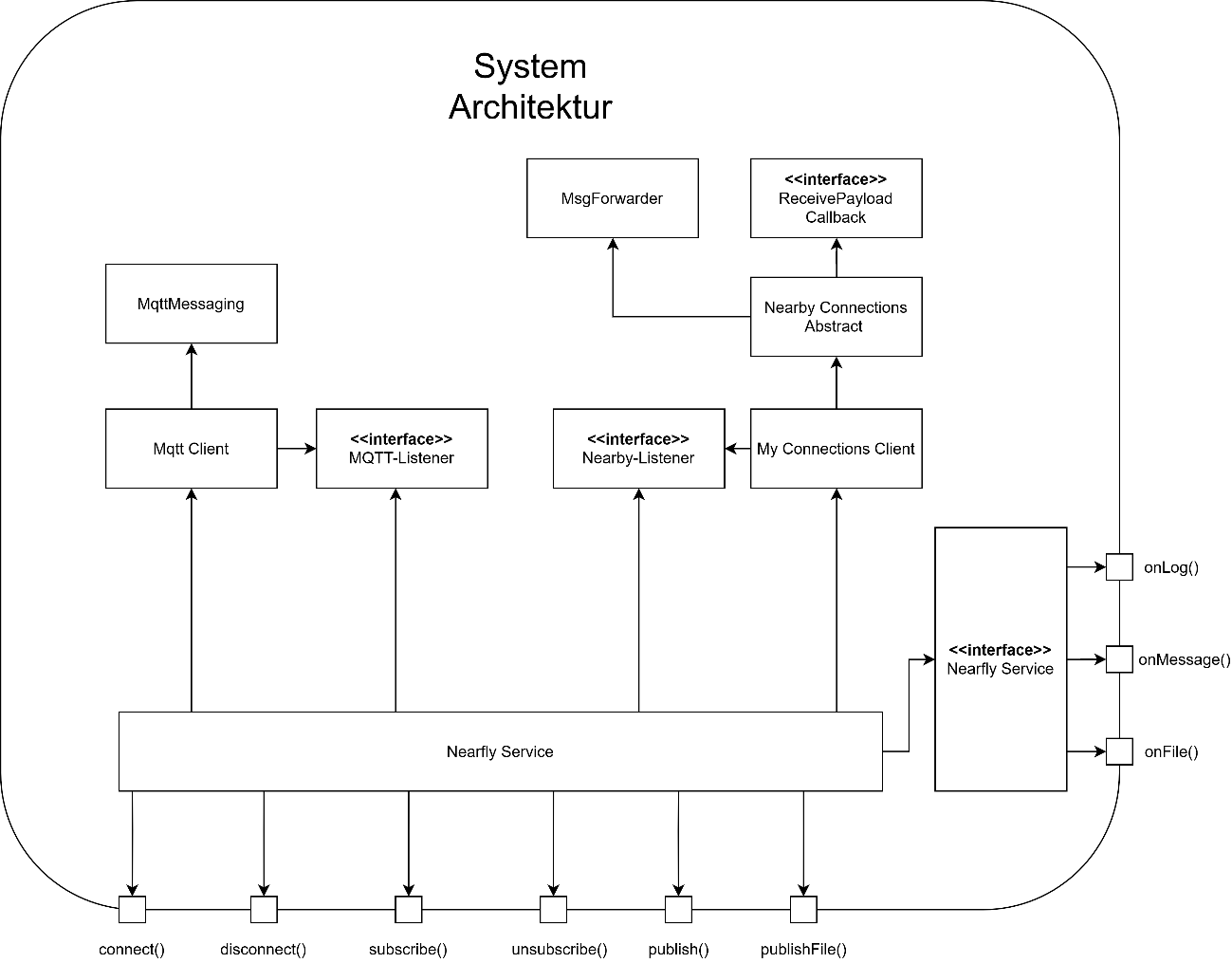
**Neafly Service:** Service Klasse, welche sowohl den selbst implementierten Neafby Service, wie auch den MQTT Service beinhaltet.

* Integrierter Nearby Service beinhaltet: Kanalfilter,

Message Listener(Interface): Dient zur dependency inversion zwischen subManager und Nearfly Service durch z.B. onMessage/ onStatus-Methoden

**MessageManager:**

* Speichert erhaltene Datenpakete nach Eintreffen in PriorityBlockingQueue
* Hat einen Counter (Sequence Number), welcher
* Zuständig erhaltene Chunks zusammenzufügen
* Trifft eine not chunked oder die letzte einer chunked Nachricht ein, werden wenn nötig deren Fragmente aus einer PriorityQueue geholt und der SubCallBack Listener getriggert.



**Abbildung 2:** Software-Architektur

Nearfly API hat folgende Funktionen:

* Connect(MqttOptions, NearbyOptions)
* Subscribe(channel, QoS)
* Publish(channel, payload, priority)

# Implementierung des Nearfly Services



* Weclhe Permission msüsen gewährt werden …+ Fragen zur Runtime und warum das gemacht werden muss
* Wahl der Kanäle:
  + Nearby Connections bietet eine Sichtbarkeitsbeschränkung für nahstehende Geräte an, welche durch die Angabe einer UTF-8 encodierter SERVICE\_ID erfolgt. Sinnvoll ist die Benutzung dieses Features, um unerwünschte Verbindungsanfragen zwischen unterschiedlichen Applikationen und den damit verbundenen Overhead zu vermeiden. Da MQTT wird das top-level Topic als solches gebraucht. Sodass beim Verbindungsaufbau eun NEARFLYCONTEXT angeben muss.
* Dynamisches wechseln zwischen Online- und Offline-Modus
* Realisierung der Callbacks
* 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

## Erstellen der Bibliothek

* Da zu Entwickelnde System eine Bibliothek sein soll, mus zunächst entschieden werden, welche Art benutzt wird
* Android-Studio bietet 2 Arten von Bibliotheken. Java und Android Biblitoheken
* (Kurze Erläuterung der Vorteile der AAR-Bib und weswegen diese zu verwenden ist)
* Um eine Bibliothek zu erstellen, muss das Projekt zunächst in 2 Modulen eingeteilt werden. Dabei muss beinhaltet eines der Module die Nearfly-Bibliothek welche später in anderen Projekten eingebunden werden kann, während das zweite Modul die Beispielanwendungen enthält, welche zunächst erstellt werden, um die Bibliothek auf Ihre Grundfunktionalitäten hin zu testen. Das zweite Modul soll später aber auch um die Szenarien ergänzt werden.
* Die Bibliothek wird als Service implementiert (Design-Entscheidung oder Implementierung?)
* Sequenz- und Aktivitätsdiagramme verdeutlichen die Prozessabläufe und welche Komponenten durchlaufen werden: Wie beim publish- und subscrbe-prozess
* Das Advertisen und Discovern im FINDROOT-State hat den Nachteil, das es für für größere Latenzzeiten bei der Verbindung sorgt [4].

Große Dateien sollte gechunkt werden:

<https://stackoverflow.com/questions/50548446/google-nearby-connections-not-able-to-transfer-large-bytes-between-2-devices>

# Verbindungsaufbau



* 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.

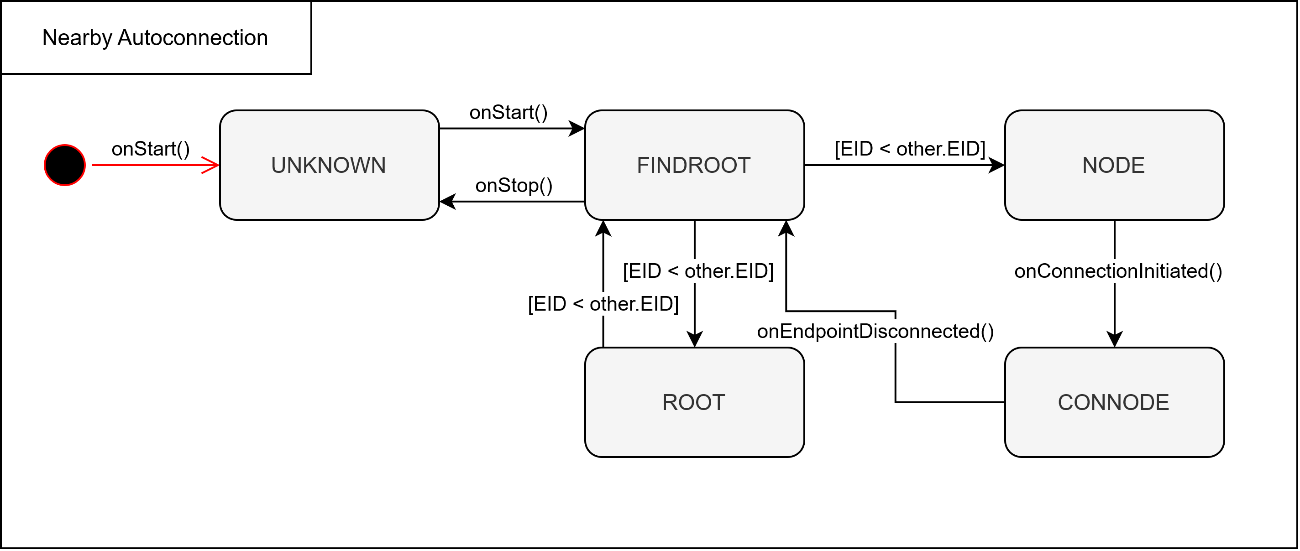
### Wahl der Strategie für das automatischen Verbindungsaufbau im Offline-Modus

Durch die Server/Client Architektur genügt bei der MQTT API eine Aktion (Befehl), um eine Verbindung zum Server und ebenso zu allen verbundenen Clients herzustellen. Anders sieht es bei der einer Nearby Connections API aus, welche als Peer-to-Peer API dem Entwickler die Entscheidung der Netzwerktopologie überlässt. Da die Nearfly-Bibliothek die Anforderung eines ähnlichen Verhaltens beiden gewrappten APIs hat, soll das Verbindungsmanagement des Nearby Connection API dem der MQTT Bibliothek gleichgemacht werden.

Prinzipiell stehen 2 Möglichkeiten zur Verfügung, welche benutzt werden können

1. Erstellen eines overlay-Netzwerkes mithilfe einer Shortcut-losen zirkularen distributed Hashtabelle.
2. Vorteil: Dadurch das jeder Knoten zwei Verbindungen hatlen muss, ist das das OverlayNetwerk mit einer Komplexitätsklasse von O (1) sehr gut skalierbar.
3. 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 Overheader sowohl beim Verbindungsaufbau wie auch bei einem Peer Churn,
4. Um die Möglichkeit des WIFI-Direct nutzen zu können, muss also ein sternenförmigen Netwerkes angestrebt werden. Dies hat den Vorteil einer overheadarmen, stärkere Datenübertragung, die besonders bei niedrigerem Teilnehmen (3-4) gut funktioniert. Jedoch wird damit der Root-Gerät im System stärker belastet und die Last nichtmehr verteilt. Weiterhin ist das Netzwerk dadurch auf 7 Teilnehmer beschränkt. Weiterhin ist zu beachten, dass das Netzwerk dadurch eine unmittelbare Abhängigkeit zum Root gewinnt. Fällt dieser weg, muss sich das Netzwerk dann wiederaufbauen.
   1. Entschlossen für das Sternen-Netzwerk.
   2. Der Root dient dann als Server. Durch forwarding aller Nachrichten kann an alle Verbundenen dieser die Nachrichten gewären
   3. Da der Root als zentrale Komponente relativ viel last ausgesetzt wird, sollte aus den lokalen Geräten dasjenige gewählt werden, welche die bessere CPU hat. Dabei soll angenommen werden, das dies ebenfalls der bestmögliche Verteiler ist. Wie wird man Advertiser? Zufällige Zahl + RAM-NR

* Unteres Diagramm erklären Verlauf.
  + Einmal den Kompletten ablauf, bis zur rekonnection durchgehen
    - 1. UNKNOWN Zustand leert Entpoints
      2. FINDROOT soll Discovery und Advertising aktiviert werden.



**Abbildung 2:** Zustandsdiagramm vom Verbindungsaufbau im Offline-Modus

* Mit hilfe der Modelierungssprache UML, sollen sowohl die Komponenten, wie auch die Gesamtarchitektur
* Modelieren des Systems durch zuhilfenahme der Modeliersprache UML
* Dabei soll Gesamtüberblick über das zu Entwickelnde System und dessen Komponenten gewonnen werden

# Übertragung von Daten



Nearby:

Mqtt:

# Benutzung der API

Durch Code kann hier die Benutzung aufgezeigt werden. Ähnlich dem Google Get-Started z.B.:

* Bevor beginn:
  + Installiere Google Play-Service SDK
  + Erteile dem System die nötigen Berechtigungen (Beachte dazu) oder erbe von der ConnectionsActivityWithPermissions, welche im Beispiel zur Verfügung gestellt wird.
* Nachrichten verbinden
* Nachrichten senden
* Nachrichten Nachrichten Empfangen

# 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 klappt durch Discovery(D) und Advertising(A). Möglich ist hierbei, das 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?
* 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?

## Erkenntnisse des Bouncing Ball Szenarios

* Die Anforderugen ,welche die Bouncing Ball stellt konnten im Offline-Modus leider nicht erfüllt werden.

# Fazit

….

## Ausblick

…

Quellenverzeichnis

[1] *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/.

[2] *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.

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

[4] *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.

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

[6] *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/.

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

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

[9] *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 ( [2].). [↑](#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“ ( [2]). [↑](#footnote-ref-2)