hochschule mannheim

Fakultät für Informationstechnik

Thesis

Im Studiengang:

technische Informatik

im Bereich

Netzunabhängige lokalen Peer-To-Peer-Verbindungen auf Mobilen-Geräten

Vorgelegt von Alexis Danilo Morgado dos Santos

Studiengang Technische Informatik

Betreuer Peter Barth

Zweitbetreuer Mark Hastenteufel

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 18.04.2020

Alexis Danilo Morgado dos Santos

Danksagung

Zusammenfassung

Abstract auf deutsch

Abstract

Inhaltsverzeichnis

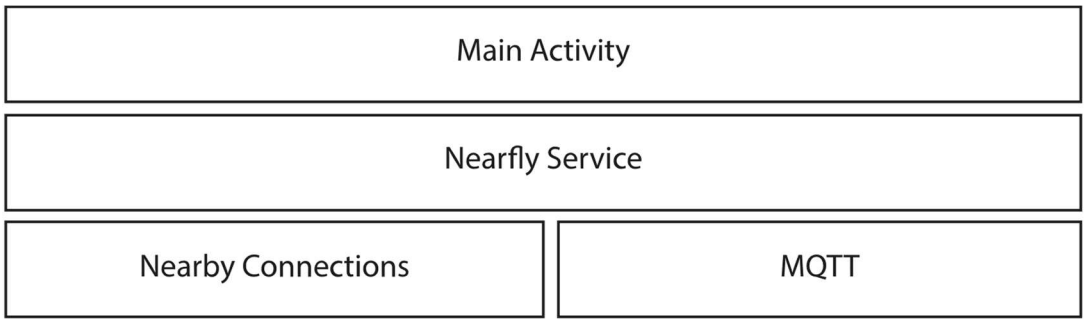
1 Einleitung

- 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 (Alexander Kunst 2017). 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 (Zhang 2011).
- 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 besteht oder aber auch an Schulenn umfragen in klassenzimmer durchgeführt werden, ohne einen Internetzugang zu haben.

1.1 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)

1.2 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

2 Anforderungsanalyse

Zur Ermittlung der konkreten Anforderungen an das zu entwickelnde System sollen die funktionalen Anforderungen von vier Szenarien ermittelt. So soll etwa das erste Szenario (Messenger App) testen, wie sich das System beim Übertragen von größeren binären Daten verhält, während das zweite Szenario (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. Das dritte Szenario (Bouncing Ball) entspricht einer Erweiterung des vorhergehenden Szenarios und verlangt bereits bei 4 Nutzer eine Übertragungsgeschwindigkeit von 4*30FPS/s. Zuletzt soll das letzte Szenario (Score Board Notepad) als Rundenbasiertes Spiel die Ausfallsicherheit und generelle Zuverlässigkeit des Systems bei mittlerer Datenübertragung testen.

2.1 Szenario 1 – Messenger

Marius ist der Scrum-Master eines 5-köpfigen Entwicklungsteams. 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.

Aus dem ersten Szenario lassen sich die funktionalen Anforderungen einer klassischen primitiven Messenger App herauskristallisieren. So soll etwa das System dem Nutzer die Möglichkeit geben, Text- und Bildnachrichten zu versenden. Denkbar wäre jedoch auch die Anforderung auf beliebige Binärdaten zu pauschalisieren und damit den Datentransfer von Audiodaten, wie auch kleineren APKs zu ermöglichen. Weiterhin soll durch das erstellen und betreten von Chatrooms eine Isolation innerhalb derselben App geschaffen werden dürfen, welche die Sichtbarkeit dedizierter Nachrichten auf befugte Nutzer reduziert. Ebenso soll der Absender, wie auch die Absende-Zeit einer Nachricht von anderen Nutzern identifiziert werden können. Als letzte Anforderung ist die Differenzierung unterschiedlicher Nachrichten und deren Priorisierbarkeit erkennbar, welche dem Nutzer die Möglichkeit darüber lässt, zu entscheiden, welche Nachrichttypen dieser beim parallelen Empfange ähnlich-großer Nachrichten unterschiedlicher Typen zuerst geladen haben möchte.

2.2 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 Feld aus und Beni gibt das Start-Signal damit die Berührungspunkte möglichst zeitgleich ausgeblendet 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.

Das zweite Szenario umfasst demnach das Entwickeln einer minimalistischen Anwendung, welche eine Leinwand beinhaltet, die von allen Benutzern gemeinsam bemalt werden kann. Es ergeben sich die Anforderungen, dass das System jeweils bis zu 10 Berührung einzelner Nutzer erfassen und an alle anderen Spieler senden muss. Da alle gemeinsam Zeichnen wollen, muss das System die Daten möglichst zeitnah empfange. Zuletzt muss das System neu verbundene Nutzer allen Spielern mitteilen.

2.3 Szenario 3 – Bouncing Ball

Mary, Robert und Frederik 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.

Das dritte Szenario stellt eine Beispielanwendung dar, welches entsprechend der hohe FPS rate eine genau so hohe Datenraten anfordert, um dem Geschick der einzelnen Spieler nicht entgegen zu wirken. Weiterhin entstehen auch die Anforderungen, dass das System die Neigungsdaten aller Spieler kontinuierlich erfassen und die Kugel entsprechend der Summe aller erfassten Neigungsdaten bewegt. Das System sollte ebenso allen Spielern ein synchrones starten der Runde ermöglichen.

2.4 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 beginnt und trifft die Mülltonne, der Ball berührt zuvor jedoch die Wand, sodass Mark auf die „+1"-Schaltfläche berührt und Rickys Punktestand inkrementiert. Daraufhin Ricky trifft die Mülltonne ohne, dass der Ball etwas berührt, woraufhin Mark die „+2"-Schaltfläche berührt und damit die Punkte von Ricky um zwei erhöht. Nach Ablauf acht weitere versuche, drückt Mark auf die „next player"-Schaltfläche. Nun ist Steffan am Zug und Ricky zuständig für Steffans Punkte. Nach Ablauf von 20 Runden, in denen jeder jeweils 10-mal werfen durfte, berührt Mark die „end game"-Schaltfläche, woraufhin der Gewinner ermittelt und angezeigt wird.

Als rundenbasiertes Spiel hat das vierte Szenario einen mäßigen Datentransfer, welcher nach Ablauf des jeweiligen Spielerrunde auftritt. Dabei muss sowohl die Neuzuordnung des Schreibers wie auch das ermitteln des aktiven Spielers und der Zuschauer erfolgen. Weiterhin ergibt sich die Anforderung einer Smartphone-übergreifende Synchronisation der Punktetabelle.

2.5 Funktionale Anforderungen

Bei Betrachtung der ermittelten Anforderungen in einer Anforderungsmatrix (siehe Tabelle 1) lassen sich korrelierende Anforderungen erkennen. 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 Systemen-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
erkennbare Anforderungen	Bildnachrichten 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-Verhalten bei 30 FPS pro Spieler			x	
extrapo-	Sichtbarkeit der Nachrichten nur innerhalb derselben App	x	x	x	x
	Wechselt der Nutzer von Online- in den Offline-Modus sollen keine Daten keine Daten verloren gehen				

	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.				

2.6 Nichtfunktionale Anforderungen

Als Nichtfunktionale Anforderungen wird für die Nearfly-Bibliothek die Zielplattform deklariert, sowie eine Maßnahme, welche dem Zweck einer laufenden Qualitätssicherung dient:

Zielumgebung: Das System muss eine Kompatibilität zu allen Android Betriebssystemen der API 24 (Android 7.0) oder höher aufweisen.

Leichte Benutzbarkeit: Die API soll möglichst minimalistisch gehalten werden und damit möglichst Einstiegsfreundlich für die Entwickler sein. Dazu sollen besonders alle nach außen Sichtbaren Methoden ausreichend dokumentiert werden, sodass dem späteren Nutzer der API, dies als Leitfaden dienen kann.

2.7 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 (Nearby Connections Team 2018). Verbindungen werden dabei durch Nutzung von Bluetooth, Bluetooth-Low-Energy (BLE) und automatisch angelegten Wifi Hotspot hergestellt (Nearby Connections Team 2018) 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 (Nearby Connections Team 2018).

Vor dem eigentlichen Verbindungsaufbau muss eine der drei folgenden Topologien (Strategie) gewählt werden, welche das Netzwerk annimmt:

P2P_CLUSTER: Ermöglichst eine N-M Topologie. Benutzt jedoch nur Bluetooth und ist damit in der Bandbreite eingeschränkt.

Zudem ist die realistische maximale Anzahl der gleichzeitig verbundenen Geräte[1] auf 3-4 limitiert (Harmon 2018a).

P2P_STAR: Ermöglichst eine 1-N Topologie, 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 (Harmon 2019). Das Netzwerk kann dann bis zu 7 Knoten aufnehmen (Harmon 2018a).

P2P_POINT_TO_POINT: Entspricht dem Verhalten des P2P_START Topologie, jedoch wird werden die Knoten auf maximal eine Verbindungsknoten beschränkt.

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. Ebenso sei erwähnt das ein Gerät beide Zustände zeitgleich einnehmen kann (Nearby Connections Team 2018), was jedoch *trashing* bewirken kann (Harmon 2018b) 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 (Nearby Connections Team 2018). 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 (Nearby Connections Team 2018). 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

[1] Inkludiert sind Smart-Watches, Wireless-Kopfhörer wie auch andere über Bluetooth gekoppelte Geräte.

mitübertragen, sodass diese bereits bei *onPayloadReceived* zur Verfügung stehen (Harmon 2018a) .

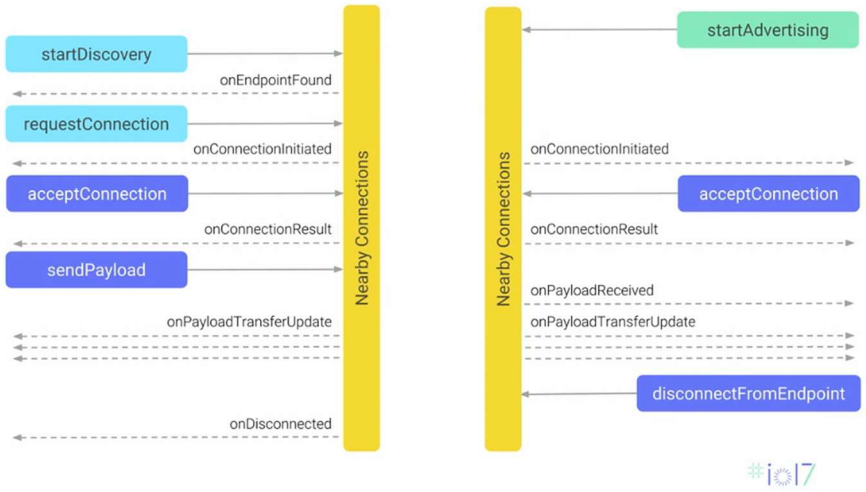


Abbildung 1: Nachrichtenverlauf der Nearby Connections API

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 (Banks und Gupta 2014). Im Gegensatz zu Nearby Connections setzt MQTT eine TCP[2] Verbindung zum Server (Broker) voraus, welcher die Verbindungen und Nachrichtenübermittlung verwalteten und die Clients voneinander entkoppelt, sodass diese nie direkt miteinander verbunden sind. (HiveMQ Team 2019)

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. (HiveMQ Team 2019). Die MQTT Spezifikation sieht dabei keine Verschlüsselung der Nachrichten vor, um das Protokoll möglichst simple zu halten (Banks und Gupta 2014), 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*[3] (z.B. „Test/topic“) durchführen. Dabei

[2] Die MQTT Spezifikation erlaubt auch andere Protokolle, welche eine geordnete, verlustfreie, bi-direktionale Verbindung zulassen (Banks und Gupta 2014.).

[3] 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“ (Banks und Gupta 2014).

müssen Nachrichten stehts ein *Topic*, wie auch ein Byte Payload unter 256Mbyte (Roger Light 2020) haben, sodass der Broker die Nachrichten an Subskribierte Clients weiterleiten kann. (HiveMQ Team 2019). 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 (Banks und Gupta 2014).

Tabelle 2: Überblick über Unterschiede zwischen Nearby Connections und MQTT (Nearby Connections Team 2018)

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, Bluetooth Low Energy (BlE), Wifi Hotspot
Nachrichtenaustausch	Publish	Bidirektional One-Way
Authentifikation	Möglich durch 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	EntpointID
Max Packet-Größe	Maximal 256MB (Seitens des Brokers einstellbar)	Bytes: 32KB Files: unbegrenzt Stream: unbegrenzt
Chunked-Encoding:	Single Chunk (link)	Bytes-Payload: Single Chunk File&Stream-Payload: Chunked

3 Software-Design

3.1 Systemarchitektur

1. Service für MQTT
2. Nearfly: Autoconnect algorithmus
3. Ext-msg
4. Erwähnen für welche Strategie entschieden
5. Wie wird man Advertiser? Zufällige Zahl + RAM-NR
6. + Aufzeigen der Peer als Broker dient + State Diagramm
7. Permission explizit fragen…

- 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

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

Im Folgenden wird Design veranschaulicht:

ExtMessage: 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

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

SubPreprocessor:

- 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.

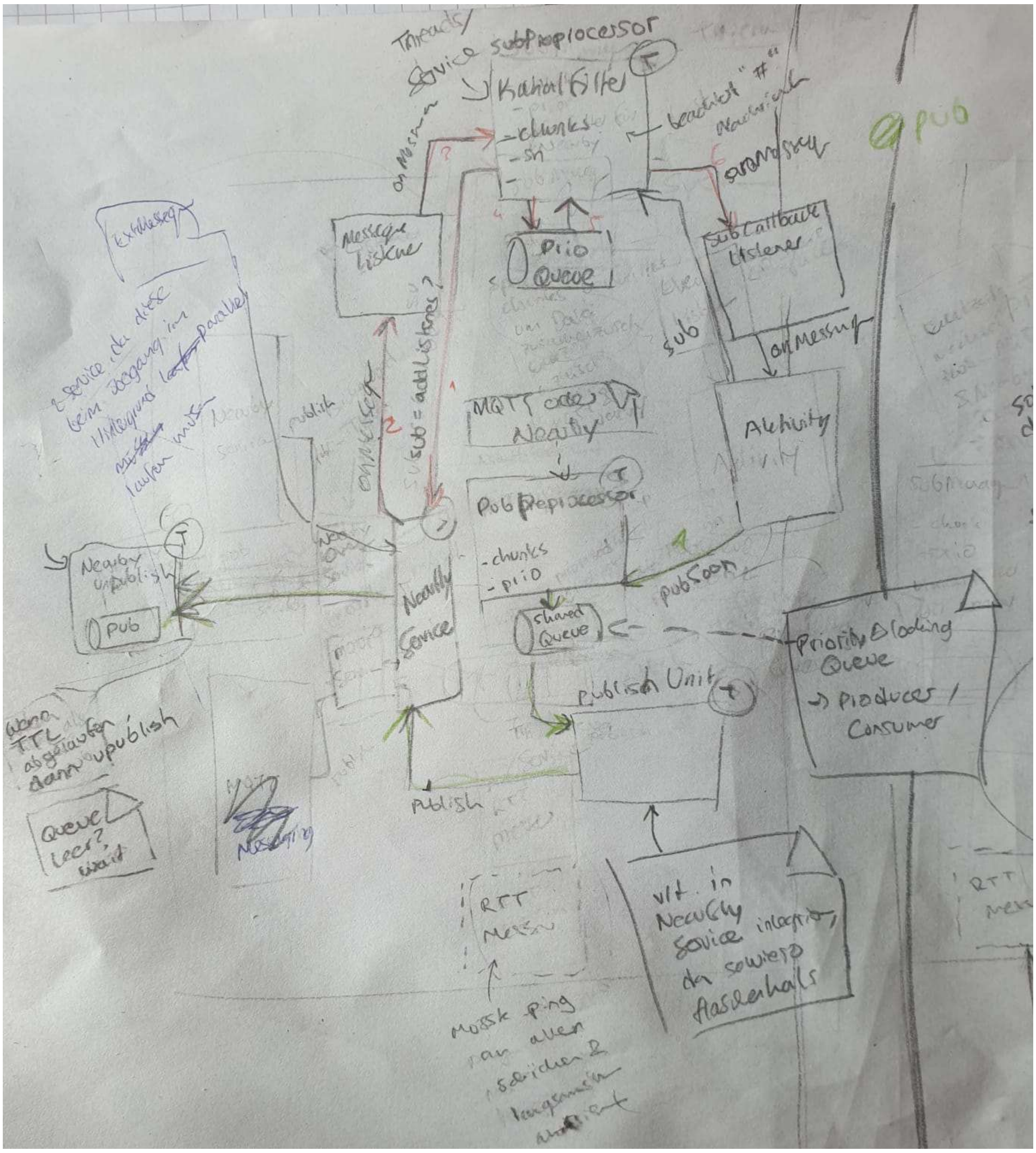
SubCallBack Listener: Benachrichtigt Aktivity bei eintreten neuer Nachrichten

pubPreprocessor:

- Zerlegt Nachrichten in Chunks und legt diese in unbounded Priorityqueue rein
- Kann evtl. Threadpool für das publishen von Nachrichten verwenden

pubUnit:

- Worker, welcher bei gefüllter PriorityQueue für das publishen der chunked Nachrichten zuständig ist.


subPreprocessor
Kanal Filter
chunks
Message Listener
Prio Queue
SubCallback Listener
onMessage
Activity
MQTT oder Nearby
PubPreprocessor
- chunks
- prio
pubSoon
shared Queue
Priority Blocking Queue
-> Producer / Consumer
Publish Unit
Nearby Service
sub = addListener?
Nearby unpublish
Pub
publish
Queue leer? wait
RTT Messen
@pub

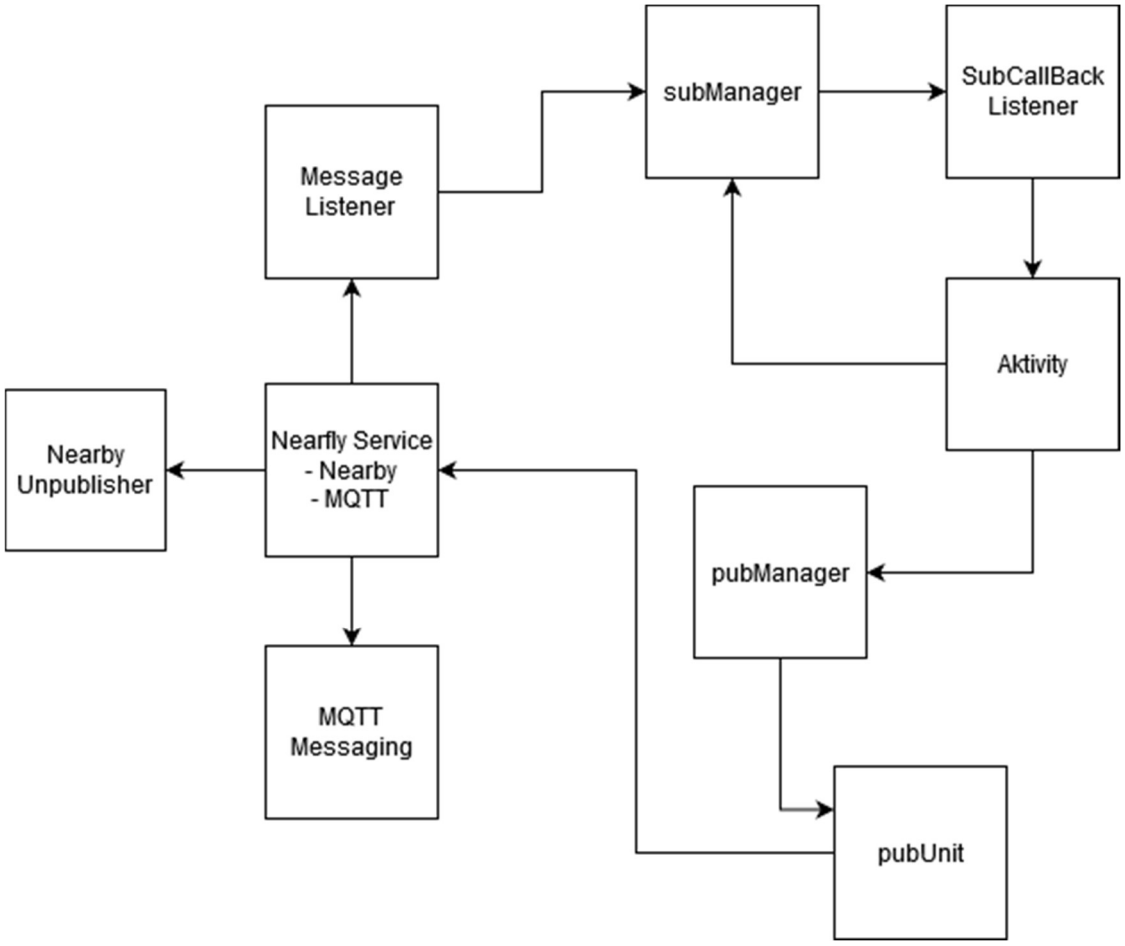


Abbildung 2: Software-Architektur

Nearfly API hat folgende Funktionen:

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

4 Implementierung des Nearfly Services

4.1 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 (Harmon 2018b).

Große Dateien sollte gechunkt werden:

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

5 Implementierung der Szenarien

Werden Komponenten zu MessageApp und anderen Szenarien genannt + Prozesse erläutert? Oder soll dies nur für die Nearfly-Bib der Fall sein?

5.1 Messenger App

- Evtl. Use Case Diagramme (kommmt in Szenario)
- Aktivitätsdiagramme
- Komponenten

5.2 Shared Touchpoint Canvas

- Sequenz-Diagramm

5.3 Skart

6 Evaluation

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

6.1 Verifikation der Anforderungen

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

6.2 Erkenntnisse des Messenger Szenarios

- Wie schnell können Größere Nachrichten übertragen werden?
- Konkrete Zahlen. Abhängigkeit zwischen Größe, Geschwindigkeit & Endpunkte in MQTT & Nearby
-

6.3 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 ?
- Was wird

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.

6.4 Erkenntnisse des Skart Szenarios

Fazit

….

Ausblick

…

Quellenverzeichnis

Alexander Kunst (2017): Statista-Umfrage Telekommunikation 2017. Deutschland.

Banks, Andrew; Gupta, Rahul (2014): MQTT Version 3.1.1 - OASIS Standard, zuletzt aktualisiert am 29.10.2014, zuletzt geprüft am 14.04.2020.

Harmon, Will (2018a): Google Nearby Connections 2.0 capabilities, zuletzt aktualisiert am 24.08.2018, zuletzt geprüft am 13.04.2020.

Harmon, Will (2018b): How can I speed up Nearby Connections API discovery?, zuletzt aktualisiert am 18.10.2018, zuletzt geprüft am 13.04.2020.

Harmon, Will (2019): How performant is Nearby Connections?, zuletzt aktualisiert am 31.01.2019, zuletzt geprüft am 13.04.2020.

HiveMQ Team (2019): Client, Broker / Server and Connection Establishment - MQTT Essentials: Part 3, zuletzt aktualisiert am 17.07.2019, zuletzt geprüft am 14.04.2020.

Nearby Connections Team (2018): Nearby Connections API Leitfaden, zuletzt geprüft am 04.04.2020.

Roger Light (2020): mosquitto.conf man page, zuletzt aktualisiert am 27.02.2020, zuletzt geprüft am 04.04.2020.

Zhang, Lucy (2011): Building Facebook Messenger, zuletzt aktualisiert am 12.08.2011, zuletzt geprüft am 13.04.2020.