hochschule mannheim

Fakultät für Informationstechnik

Praktikumsbericht

Im Studiengang:

technische Informatik

im Bereich

Konzeption und Implementierung einer Wrapper Bibliothek zur Erstellung von Internetunabhängigen lokalen Peer-To-Peer-Verbindungen auf Android-Geräten

Vorgelegt von	Alexis Danilo Morgado dos Santos
Matrikelnummer	1631401
Studiengang	Technische Informatik
Fachsemester	7
Betreuer	Peter Barth
Projekt	Nearfly-Bibliothek (Android)

06.04.2020, Alexis dos Santos

Peter Barth

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 06.04.2020

Alexis Danilo Morgado dos Santos

Danksagung

Zusammenfassung

Abstract auf deutsch

Abstract

Inhaltsverzeichnis

Eidesstattliche Erklärung **2**

Danksagung **3**

Zusammenfassung **4**

Abstract **5**

Inhaltsverzeichnis **6**

1 Einleitung **8**

1.1 Motivation 8

1.2 Zielsetzung 8

1.3 Aufgabenstellung 8

2 Anforderungsanalyse **9**

2.1 Szenario 1 – Messenger 9

2.2 Szenario 2 – Shared Touchpoint Canvas 9

2.3 Szenario 3 – Bouncing Ball 10

2.4 Funktionale Anforderungen 10

2.5 Nichfunktionale Anforderungen 11

3 Software-Design **12**

3.1 Gegenüberstellung von Nearby Connections und MQTT 12

3.2 Art der zu verwendenden Bibliothek 13

3.3 (Netwerkarchitektur/ Software Interfaces) 13

3.4 Systemarchitektur 13

3.5 System Features 13

4 Implementierung des Nearfly Services **17**

5 Implementierung der Szenarien **18**

5.1 Messenger App 18

5.2 Shared Touchpoint Canvas 18

5.3 Skart 18

6 Evaluation **19**

6.1 Verifikation der Anforderungen 19

6.2 Erkenntnisse des Messenger Szenarios 19

6.3 Erkenntnisse des Touchpoint Szenarios 19

6.4 Erkenntnisse des Skart Szenarios 20

Fazits 21
Ausblick 21
Quellenverzeichnis ..... Fehler! Textmarke nicht definiert.

1 Einleitung

1.1 Motivation

1.2 Zielsetzung

- Möglichst einfach zu bedienen
- Gut dokumentiert
- Vorallem nachvollziehbarere SW Entwurf z.B. durch Verwendung bekannten Entwurfmuster
- Anwendung in unterschiedlichen Anwendungen strukturieren
- Durch spätere Anforderungsanalyse werden Anforderungen später dazu spezifiziert

1.3 Aufgabenstellung

- Middleware für Android, die MQTT und Nearby verwendet.
- Wie auch MQTT und Nearby soll das zu entwerfende System eine Publish/ Subscribe API anbieten.
- Prioritätenbasierte
- Die Kernfunktionalitäten der zu Neafly-Bibliothek bestehen aus:1, 2, 3

2 Anforderungsanalyse

(Zusammenfassung der folgenden Unterkapitel)

In diesem Kapitel sollen die konkreten Anforderungen an das zu entwickelnde System (Android-Bibliothek) ermittelt werden. Um die Anforderungsanalyse zu erleichtern sollen hierfür zunächst die funktionalen Anforderungen dreier Szenarien (Beispielanwendungen) bestimmt werden. Die ermittelten Anforderungen ermöglichen eine Deduktion, aus welcher sich die konkreten Anforderungen an die Nearfly-Bibliothek ergeben.

Die Auswahl der Szenarien beschränkt sich jedoch nicht nur auf die Findung konkreter Anforderungen, sondern dient außerdem zur späteren Verifikation der Nearfly-Bibliothek. So soll etwa die Messenger App(link) testen, wie sich das System beim Übertragen von binären Daten verhält, während die Shared Touchpoint Canvas App(link) das System aufgrund der Menge an Touchpoint ausreizen kann und demnach Rückschlüsse auf die Eignung der Nearbfly-Bibliothek für Echtzeit-Anwendungen gibt. Zuletzt soll die Skart Anwendung als Rundenbasiertes Spiel die Ausfallsicherheit einer Runde testen.

2.1 Szenario 1 – Messenger

Das 1. Szenario ist die Entwicklung einer Messenger App, in welche konventionelle Chatrooms erstellt werden und durch mehrere Benutzer betreten werden können. Basierend auf klassische Messenger Apps entstehen hierfür folgende funktionale Anforderungen.

Bild- und Text senden: Das System muss dem Nutzer die Möglichkeit geben, Text- und Bildnachrichten zu versenden, welche von allen anderen Benutzern empfangen werden.

Chatroom erstellen: Ein Nutzer kann einen eigenen Chatroom erstellen.

Chatroom beitreten: Ein Nutzer kann einen erstellten Chatroom beitreten.

Echte Parallelität: Werden Bild- und Textnachrichten beinah parallel versendet, muss die kleineren Nachrichten, unabhängig davon welche der beiden Nachrichten zuerst versandt wurde, schneller ankommen.

Differenzierbarkeit: Um die Nutzer zu unterscheiden, soll jedem Nutzer ein Name und eine Farbe zugewiesen werden.

2.2 Szenario 2 – Shared Touchpoint Canvas

Das 2. Szenario umfasst das Entwickeln einer minimalistischen Anwendung, welche eine Leinwand beinhaltet, die von allen Benutzern gemeinsam bemalt werden kann. Jedem Benutzer wird zu Beginn eine Farbe zugeteilt. Berührt ein Benutzer die Leinwand, ent-

steht ein kolorierter Abdruck (Touchpoint), welcher nach Ablauf einer gewissen Zeit verschwindet. Die Umsetzung der fundamentalen Funktionen inkludiert folgende Anforderungen:

Touchpoints Broadcasting: Falls die Leinwand berührt wird, muss das System die Berührung erfassen und diese den anderen Benutzern mitteilen.

Shared Canvas: Falls ein Berührungspunkt empfangen wird, muss das System diesen mit der dem Benutzer zugewiesenen Farben an die entsprechende Position zeichnen.

Differenzierbarkeit: Falls die Anwendung gestartet wird, muss das System dem Benutzer eine Farbe zuweisen.

Auto-Clean: Nach einer festgelegten Zeit, muss das System die kolorierten Touchpoints aus der Leinwand entfernen.

Join-Benachrichtigung: Falls ein Benutzer die Anwendung startet, muss das System alle aktiven Benutzer derselben Anwendung benachrichtigen.

2.3 Szenario 3 – Bouncing Ball

Das 3. Szenario umfasst das Entwickeln einer App, welche eine Kugel beinhaltet, die durch schwenken des Android Gerätes balanciert wird. Jeder Benutzer balanciert dieselbe Kugel zeitgleich, sodass das Verhalten der Kugel, die Summe der Aktionen aller Spieler ist.

- A1: Das System muss die kontinuierlich die Neigungsdaten aller Spieler erfassen und an alle Spieler senden.
- A2: Nach dem Erhalt der Neigungsdaten aller Spieler, muss das System ein resultierendes Verhalten für die Kugel berechnen.
- A3: Das System muss jedem Spieler die Möglichkeit geben, das Spiel synchron (als Team) zu starten.
- A4: Das System muss jedem Spieler die Möglichkeit geben, das Spiel zum Anfang einer Runde beizutreten.

2.4 Funktionale Anforderungen

Bei Betrachtung der ermittelten Anforderungen lassen sich korrelierende Anforderungen erkennen. So ist etwa das Betreten eines Raumes, d.h. die konkrete Trennung von Datenströmen innerhalb derselben Anwendung in allen drei Szenarien enthalten. Weiterhin konnten folgende funktionale Anforderungen identifiziert werden:

Channelbased Broadcasting: Zu sendende Nachrichten müssen einem Empfangskanal zugewiesen werden.

Kontextualität: Nachrichten sind nur im Kontext derselben Anwendung sichtbar

channelbased listening: Das System kann auf eingehende Nachrichten unterschiedlicher Kanäle verschieden reagieren.

Traceability: Der Sender einer Nachricht muss identifiziert werden können

Priorisierte Textnachricht: Werden 2 Nachrichten unterschiedlicher Größe versandt, kommt die kleinere beider Nachrichten erster an.

Zudem lassen sich aus den bestehenden Anforderungen folgende weitere Bibliotheksnähere Anforderungen extrapoliere, welche

Flawless Switch: Wechselt das System die unterliegende Technologie während des Betriebes, soll dies möglichst reibungslos funktionieren.

Autoverbinndungsaufbau: Scheitert die Verbindung muss das System in einen kontrollierten Zustand wechseln, in welchem die Verbindung wiederaufgebaut wird, dabei soll eingestellt werden können ob nicht versandte (pending) Nachrichten zwischengespeichert oder verworfen werden sollen.

Einbindbarkeit

Debug-Tool: Das System soll dem Entwickler die Möglichkeit geben, die höchste auftretenden Round-Trip Time während des Betriebes zu messen.

2.5 Nichfunktionale Anforderungen

Parallelität: Das Empfangen, wie auch das Senden muss parallel geschehen.

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.

Konsistenz: Unabhängig der verwendeten Technologie (Nearby/ MQTT), soll sich das System möglichst ähnlich verhalten.

3 Software-Design

3.1 Gegenüberstellung von Nearby Connections und MQTT

- Um Konkrete Lösungsansätze für das Wappen beider zu verwendeten Technologien zu gewinne müssen die beiden zu verwendeten Technologien miteinander verglichen werden.
- Ähnlichkeiten können dabei für die API verwendet werden.
- Evtl. Gegenüberstellung am Ende in Form einer Tabelle (als kleines Fazit) z.B.:

Tabelle 1 Überblick über Unterschiede zwischen Nearby Connections und MQTT (Nearby Team 2018)

Aktion	MQTT	Nearby Connections
Patern für Nachrichtenaustausch	Publish/ Subscribe	Vgl. One-Way Message Exchange
Patern für Verbindung	Server mit MQTT Broker (zentral)	Peer-to-Peer
Datentypen zur Übertragung	UTF-8 String: Topic, Client-ID, Benutzername, Passwort Byte: Payload	3 Arten Payload: Bytes
Max Packet-Größe	Maximal 256MB (Seitens des Brokers einstellbar) (Roger Light 2020)	Bytes: 32KB Files: unbegrenzt Stream: unbegrenzt
Übertragungsart der	Single Chunk (link)	Bytes als Single Chunk, sonst Chunked
Verwendete Protokolle	TCP/ TLS	Bluetooth, Bluetooth Low Energy (BlE), Wifi Hotspot

3.2 Art der zu verwendenden 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)

3.3 (Netwerkarchitektur/ Software Interfaces)

- Welche Schnittstellen existieren außerhalb des Systemkontextes und sind demnach als Abhängigkeiten zu sehen? Z.B. Server mit MQTT Broker und Android Geräte in der Nähe für Nearby…

3.4 Systemarchitektur

- 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

3.5 System Features

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.

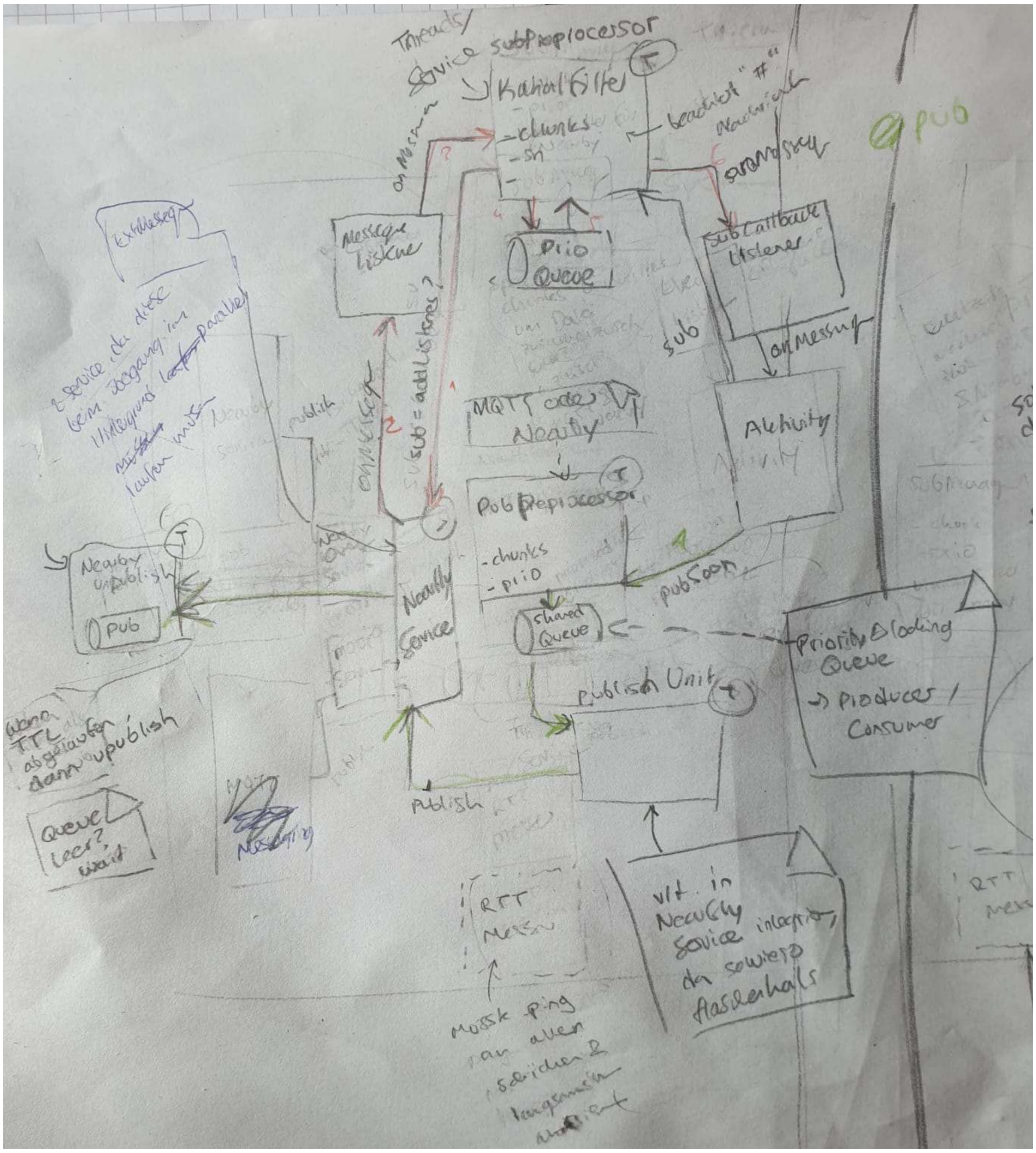
SubCallBack Listener: Benachrichtigt Aktivity bei eintreten neuer Nachrichten

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.


Threads/ Service
subPreprocessor
Kanal Filter
- chunks
- sn
beachtet "#" Nachricht
onMessage
Prio Queue
Message Listener
SubCallback Listener
sub
onMessage
Activity
MQTT oder Nearby
PubPreprocessor
- chunks
- prio
pubSoon
shared Queue
Priority Blocking Queue
-> Producer / Consumer
Publish Unit
Nearby Service
sub = addListener?
onMessage
publish
Nearby unpublish
Pub
wenn TTL abgelaufen dann unpublish
Queue leer? wait
RTT Messen
vlt. in Nearby Service integrieren, da sowieso Flaschenhals
Pub

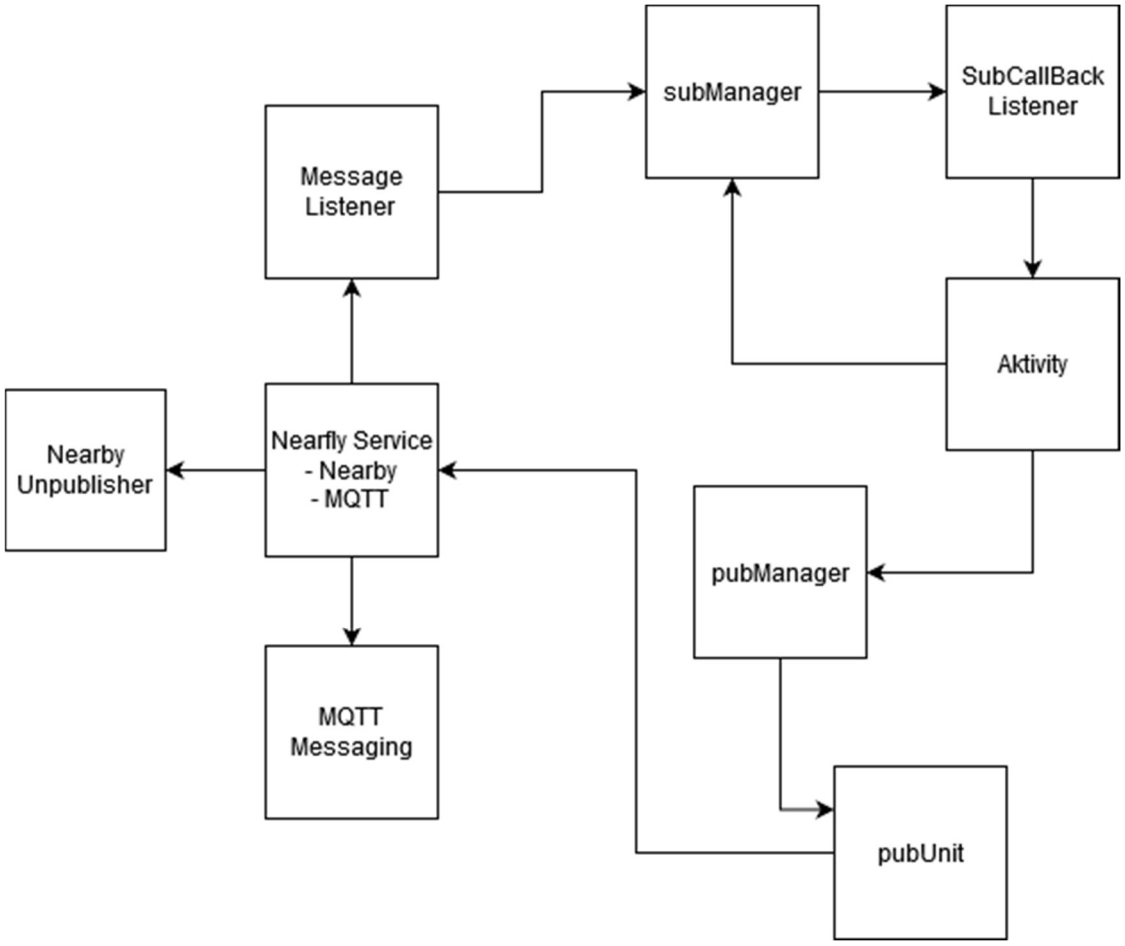


Abbildung 1: Software-Architektur

Nearfly API hat folgende Funktionen:

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

4 Implementierung des Nearfly Services

- 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

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. User Case Diagramme
- 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

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