

Optimierung des Energieverbrauchs bei mobilen Applikationen

Bachelorarbeit 2015

Student: Pierre-Alain Wyssen

Dozent: Prof. Dr. René Schumann

Studiengang: Wirtschaftsinformatik

Abgegeben am 27. Juli 2015

www.hevs.ch

Zusammenfassung

Entwickler von mobilen Applikationen wissen nach der Lektüre dieser Bachelorarbeit, wie sie die Energieeffizienz einer App maximieren können. Es werden konkrete Handlungsempfehlungen gegeben, wie rechen- oder kommunikationsintensive Applikationen optimiert werden können, um die Akkulaufzeit zu erhöhen.

Der erste Teil dieser Arbeit besteht aus einem Überblick der aktuellen Technologien von Smartphones, welche den Energieverbrauch minimieren sowie der Präsentation einiger Messinstrumente, welche für die Messung des Energieverbrauchs geeignet sind. Anschliessend werden Massnahmen vorgestellt, um den Energieverbrauch zu reduzieren. Diese Massnahmen werden danach mithilfe von Szenario-Analysen in Android auf ihre Effizienz hin geprüft. Anhand dieser Ergebnisse werden letztlich am Ende der Arbeit Empfehlungen zur Maximierung der Energieeffizienz gegeben.

Die relevantesten Ergebnisse dieser Arbeit zeigen, dass die Akkulaufzeit durch das Deaktivieren von Komponenten (z.B. Bluetooth, GPS, usw.) maximiert werden kann; dass das Gruppieren von Datenpaketen sich positiv auf die Leistung des Akkus auswirkt und dass das Auslagern von rechenintensiven Operationen auf Webdienste ebenfalls zur Erhöhung der Akkuleistung beiträgt.

Die Leistung von Smartphones hängt jedoch auch von der installierten Version des Betriebssystems sowie der Hardware ab. Diese zwei Faktoren lassen sich zwar vom Entwickler nicht beeinflussen, jedoch kann er durch die Anwendung der in dieser Arbeit vorgestellten Massnahmen die Akkulaufzeit erhöhen.

Keywords: Energieeffizienz, Smartphone, Android

Vorwort und Dank

Die vorliegende Arbeit entstand zwischen April und Juli 2015. Die Hauptmotivation dieser Arbeit war mein Interesse an der Android-Plattform sowie deren steigende Popularität in den letzten Jahren.

An dieser Stelle möchte ich mich bei den Personen bedanken, welche mir bei der Realisierung dieser Bachelorarbeit geholfen haben. Mein Dank gilt zunächst meinem betreuenden Dozenten, Herrn Prof. Dr. René Schumann, welcher mir stets hilfreiches Feedback und nützliche Inputs gegeben hat. Des Weiteren bedanke ich mich bei Caroline Tamarcaz für ihre Erläuterungen der App zur Simulation von packet coalescing. Letztlich richte ich meinen Dank an Stefan Eggenschwiler für das Korrekturlesen dieser Arbeit sowie für sein Feedback.

Eine der wichtigsten Quellen dieser Arbeit war das Buch "Smartphone Energy Consumption; Modeling and Optimization" von Sasu Tarkoma, Matti Siekkinen, Eemil Lagerspetz und Yu Xiao, welches mir von der HES-SO Valais//Wallis in elektronischer Version zur Verfügung gestellt wurde.

Um diese Arbeit leserfreundlicher zu gestalten, wird auf die Erwähnung der weiblichen Form absichtlich verzichtet, wobei aber Frauen sowie Männer gleichermassen angesprochen werden.

Inhaltsverzeichnis

Tabellenverzeichnis	v
Abbildungsverzeichnis	vi
Abkürzungsverzeichnis	viii
1 Einleitung	1
2 Aktuelle Situation	2
2.1 Motivation	2
2.2 Einsatz von mehreren Prozessoren	3
2.3 Aktuelle Kommunikationstechnologien	4
2.4 Features der mobilen Betriebssysteme	7
3 Messinstrumente / Tools	8
3.1 ARM DS-5 Development Studio 5.21.1	8
3.2 AT&T Application Resource Optimizer	9
3.3 PowerTutor	10
3.4 JouleUnit	12
3.5 Xcode Energy Diagnostics	12
3.6 Fazit	13
4 Metriken	13
5 Massnahmen	14
5.1 Allgemeine Empfehlungen	14
5.2 Packet coalescing	17
5.3 Wakelocks	17
5.4 Job Scheduler	21
5.5 Lokale vs. entfernte Operationen	24
5.6 Native Programmierung	24
5.7 Zwischenfazit	25
6 Szenarien	25
6.1 Testumgebung	25
6.2 Szenario 1: Allgemeine Empfehlungen	26
6.3 Szenario 2: Packet coalescing	31
6.4 Szenario 3: Lokale vs. entfernte Operationen	36

7	Beurteilung.....	46
	Schlussfolgerung.....	49
	Literaturverzeichnis.....	50
	Anhang I: Implementierung der App zur Messung der Akkukapazität	53
	Anhang II: Implementierung der App zur Simulation von Packet coalescing	56
	Anhang III: Implementierung des Webdienstes zur Berechnung der Fibonacci-Folge.....	57
	Anhang IV: Beschreibung der erfassten Daten	62
	Selbstständigkeitserklärung des Verfassers	68

Tabellenverzeichnis

Tabelle 2.1: Smartphone-Trends in 5-Jahres-Intervallen seit 1995 (Berkel, 2009)	3
Tabelle 2.2: Die verschiedenen Zustände des RRC-Protokolls bei HSPA (3G) (nach Tarkoma et al., 2014, S. 119)	5
Tabelle 4.1: Übersicht der messbaren Komponenten der ausgewählten Tools (eigene Darstellung)	13
Tabelle 5.1: Übersicht der Wakelock-Level bei Android (Google Inc., 2015c).....	18
Tabelle 6.1: Einstellungen des Testgeräts für Szenario 1 (eigene Darstellung)	26
Tabelle 6.2: Ergebnisse Szenario 2 (eigene Darstellung)	35
Tabelle 6.3: Ausführungszeiten bei Android 5.0.1 (eigene Darstellung)	38
Tabelle 6.4: Gemessene Werte bei Szenario 3 – Lokal (Android-Version 4.4.2; eigene Darstellung)	39
Tabelle 6.5: Gemessene Werte bei Szenario 3 – Remote (Android-Version 4.4.2; eigene Darstellung)	41
Tabelle 6.6: Gemessene Werte bei Szenario 3 – Lokal (Android-Version 5.0.1; eigene Darstellung)	42

Abbildungsverzeichnis

Abbildung 2.1: Wechsel der Zustände des RRC-Protokolls bei HSPA (3G) (nach Tarkoma et al., 2014, S. 119)	5
Abbildung 2.2: Die verschiedenen Status bei LTE (4G) (AT&T Inc., 2015c).....	6
Abbildung 2.3: Energiesparoptionen bei Android 4.4.2 (eigene Darstellung)	7
Abbildung 3.1: Energieeffizienz-Simulation eines Beispiel-Traces des AT&T ARO-Tools (eigene Darstellung)	10
Abbildung 3.2: Beispiel der graphischen Anzeige des Energieverbrauchs bei PowerTutor (Gordon et al., 2011a).....	11
Abbildung 3.3: Screenshot des Energy Diagnostics Tools bei Xcode (Apple Inc., 2014).....	12
Abbildung 5.1: Anzeige der Komponenten, welche am meisten Akku verbrauchen (eigene Darstellung)	15
Abbildung 5.2: Anzeige der zuletzt verwendeten Apps bei Android (Version 4.4.2) (eigene Darstellung)	16
Abbildung 5.3: Aktive Apps anzeigen bei Android (Version 4.4.2) (eigene Darstellung)	16
Abbildung 5.4: Liste der Wakelocks beim Verwenden der YouTube-App (eigene Darstellung)....	18
Abbildung 5.5: Liste der Wakelocks beim Verwenden der Musik-App (eigene Darstellung)	18
Abbildung 5.6: Beispiel 1 mit suboptimalem Einsatz von Wakelocks (links) und die effizientere Lösung (rechts) (Alam et al., 2013).....	20
Abbildung 5.7: Beispiel 2 mit suboptimalem Einsatz von Wakelocks (links) und die effizientere Lösung (rechts) (Alam et al., 2013).....	20
Abbildung 5.8: Anlegen eines Jobs, welcher die Aktualisierung von Apps via den PlayStore simuliert (Pierick, 2015).....	22
Abbildung 5.9: Registrierung des Dienstes in der Datei AndroidManifest.xml (Pierick, 2015)	23
Abbildung 5.10: MyJobService.java – Dienst, welcher für die Ausführung des Jobs verantwortlich ist (Pierick, 2015)	23
Abbildung 5.11: Festlegung (scheduling) eines Jobs (Pierick, 2015).....	24
Abbildung 6.1: Testumgebung für Szenario 2 und 3 (eigene Darstellung)	26
Abbildung 6.2: Verlauf der Akkukapazität der beiden Konfigurationen bei Android 4.4.2 und 5.0.1 (eigene Darstellung)	28
Abbildung 6.3: Auszug aus dem Log; von PowerTutor generiert (eigene Darstellung)	29
Abbildung 6.4: Gesamtverbrauch Konfiguration A bei PowerTutor (eigene Darstellung).....	29
Abbildung 6.5: Gesamtverbrauch Konfiguration B bei PowerTutor (eigene Darstellung).....	29
Abbildung 6.6: Standardwerte für den Energieverbrauch beim AT&T ARO (eigene Darstellung)	30
Abbildung 6.7: Gesamtverbrauch Konfiguration A beim AT&T ARO (eigene Darstellung).....	31
Abbildung 6.8: Gesamtverbrauch Konfiguration B beim AT&T ARO (eigene Darstellung)	31
Abbildung 6.9: Pseudocode der Simulation von packet coalescing (nach Schumann & Taramarcaz, 2015).....	32
Abbildung 6.10: Pseudocode der eigens entwickelten App (eigene Darstellung)	32
Abbildung 6.11: Benutzeroberfläche beim Start der für Szenario 2 entwickelten App (eigene Darstellung)	33

Abbildung 6.12: Beispiel einer CSV-Datei für Szenario 2 (eigene Darstellung).....	34
Abbildung 6.13: CSV-Dateien zur Simulation von Szenario 2 (eigene Darstellung)	34
Abbildung 6.14: Gemessene Werte für den Gesamtverbrauch des Wi-Fi (eigene Darstellung) ...	35
Abbildung 6.15: Energieeffizienz der beiden Methoden (eigene Darstellung).....	36
Abbildung 6.16: Ausführungszeiten beim Berechnen der Fibonacci-Folge (eigene Darstellung)..	37
Abbildung 6.17: Vergleich der Ausführungszeiten nach Android-Version (eigene Darstellung)...	38
Abbildung 6.18: Durchschnittlicher Verbrauch bei lokaler Berechnung (eigene Darstellung)	40
Abbildung 6.19: Gesamtverbrauch und Ausführungszeit bei lokaler Berechnung (eigene Darstellung)	40
Abbildung 6.20: Energieverbrauch des Prozessors bei entfernter Berechnung (eigene Darstellung)	41
Abbildung 6.21: Gesamtverbrauch und Ausführungszeit bei entfernter Berechnung (eigene Darstellung)	41
Abbildung 6.22: Wasserfallmodell der Verbindungen bei 40 Stellen (eigene Darstellung).....	43
Abbildung 6.23: TCP-Sessions bei 40 Stellen (eigene Darstellung)	43
Abbildung 6.24: Wasserfallmodell der Verbindungen bei 50 Stellen (eigene Darstellung).....	43
Abbildung 6.25: TCP-Sessions bei 50 Stellen (eigene Darstellung)	43
Abbildung 6.26: Vergleich durchschnittlicher Verbrauch (eigene Darstellung).....	44
Abbildung 6.27: Vergleich Gesamtverbrauch (eigene Darstellung).....	44
Abbildung 6.28: Vergleich Ausführungsdauer (eigene Darstellung)	45
Abbildung 6.29: Vergleich Gesamtverbrauch und Ausführungsdauer bei der Berechnung von 40 Stellen (eigene Darstellung)	45
Abbildung 6.30: Vergleich des Akkuverlaufs der Android-Versionen (eigene Darstellung)	46
Abbildung 7.1: Traffic beim Ausführen einer entfernten Berechnung (Szenario 3) in Wireshark (eigene Darstellung)	47
Abbildung I.1: Vereinfachte Funktionsweise der App zur Messung der Akkukapazität (eigene Darstellung)	53
Abbildung I.2: Implementierung des OnClickListener (eigene Darstellung)	54
Abbildung I.3: Implementierung des Tasks, welcher die Akkukapazität ausliest (eigene Darstellung)	55
Abbildung I.4: Benutzeroberfläche der App nach einer Messung (eigene Darstellung).....	55
Abbildung II.1: Implementierung des Hintergrundtasks zum Senden der Daten (eigene Darstellung)	57
Abbildung III.1: Webdienst zur Berechnung der Fibonacci-Folge (eigene Darstellung)	58
Abbildung III.2: Benutzeroberfläche beim Start der App (eigene Darstellung)	59
Abbildung III.3: Die ersten 20 Zahlen der Fibonacci-Folge via Webdienst berechnet (eigene Darstellung)	59
Abbildung III.4: Implementierung der lokalen Berechnung (eigene Darstellung)	60
Abbildung III.5: Implementierung der entfernten Berechnung (eigene Darstellung)	61

Abkürzungsverzeichnis

2G/3G/4G	Zweite / dritte / vierte Generation der Mobilfunkstandards
AP	Access Point
ANP	Air Navigation Platform
ANR	Android Not Responding
API	Application Programming Interface
ARM	Advanced RISC Machines
ARO	Application Resource Optimizer
AT&T	American Telephone & Telegraph
bzw.	beziehungsweise
BB	Basic Block
BLE	Bluetooth Low Energy
BSD	Berkeley Software Distribution
CPU	Central Processing Unit
CR	Continuous Reception
CSV	Comma-separated values
d.h.	das heisst
DHCP	Dynamic Host Configuration Protocol
DRX	Discontinuous Reception
FD	Fast Dormancy
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HSPA	High Speed Packet Access
HTC	High-Tech Computer

HTTP	H ypertext T ransfer P rotocol
Inc.	I ncorporation
J	J oule
Ltd.	L imited company
LTE	L ong- T erm E volution
LTE-A	L ong- T erm E volution A dvanced
mW	M illiwatt
MP	M ulti- P rocessing
NDK	N ative D evelopment K it
PSM	P ower S aving M ode
RFID	R adio- f requency i dentification
RRC	R adio R esource C ontrol
SMS	S hort M essage S ervice
TCP/IP	T ransmission C ontrol P rotocol / I nternet P rotocol
UMTS	U niversal M obile T elecommunications S ystem
u.a.	u nter a nderem
usw.	u nd s o w eiter
vgl.	vergleiche
VM	V irtuelle M aschine
WCDMA	W ideband C ode D ivision M ultiple A ccess
Wi-Fi	W ireless F idelity
WLAN	W ireless L ocal A rea N etwork
WP	W ork P ackage

1 Einleitung

Im Jahr 2014 besaßen 69 Prozent (4,3 Millionen) der Schweizer Bevölkerung ein Smartphone, wie eine aktuelle repräsentative Studie von comparis.ch zeigt (2014). Dabei achten die Käufer zunächst auf die Akkuleistung, gefolgt vom Preis und dem Betriebssystem (CreditPlus Bank AG, 2012). Die Akkuleistung von aktuellen Geräten lässt jedoch zu wünschen übrig; erfahrungsgemäss muss das Smartphone bei ständigem Gebrauch bereits nach einigen Stunden wieder aufgeladen werden. Neben der Hardware ist auch die Software entscheidend für den Energieverbrauch. Entwickler können dazu beitragen, die Akkuleistung eines Smartphones zu verbessern.

Um die Akkulaufzeit zu erhöhen, müssen drei Ebenen betrachtet werden: Hardware, Betriebssystem sowie Programme (Apps). Zunächst wird die Hardware laufend weiter entwickelt, einerseits durch die Forschung und die Entwicklung von leistungsstärkeren Akkumulatoren, andererseits durch den Einsatz von energiesparenden Bauteilen im Smartphone. Auf Betriebssystemebene wird der Energieverbrauch auch laufend optimiert, Verantwortung dafür tragen jedoch die Entwickler bei Google (Android), Apple (iOS), Microsoft (Windows Phone) und weitere Anbieter mobiler Betriebssysteme. Die letzte Ebene zur Erhöhung der Akkulaufzeit ist die Optimierung der Programme (Apps) auf dem Smartphone durch den Entwickler.

Die Hardware und das Betriebssystem haben zwar einen Einfluss auf den Energieverbrauch, jedoch lassen sich diese zwei Faktoren vom App-Entwickler nicht beeinflussen. Der Einfluss des Betriebssystems wird im Verlauf dieser Arbeit analysiert durch den Vergleich der beiden Android-Version 4.4.2 und 5.0.1. Der Fokus dieser Arbeit liegt jedoch auf der Programmebene: was kann ein Entwickler tun, um die Energieeffizienz einer App zu maximieren?

Diese Arbeit ist wie folgt aufgebaut: in Kapitel 2 werden der aktuelle Stand der Technologien bei Smartphones sowie Features der mobile Betriebssysteme vorgestellt. Eine Evaluation von aktuellen Tools zur Messung des Energieverbrauchs ist in Kapitel 3 zu finden, es werden deren fünf vorgestellt. Kapitel 4 beschreibt die Metriken betreffend des

Energieverbrauchs, welche im Verlauf dieser Arbeit verwendet wurden. Anschliessend werden in Kapitel 5 sechs Massnahmen präsentiert, mit welchen die Energieeffizienz einer App maximiert werden kann. Drei dieser Massnahmen werden darauf in Kapitel 6 anhand von Anwendungsszenarien auf ihre Wirksamkeit geprüft. Letztlich werden die Auswertungen und die Ergebnisse dieser Szenarien in Kapitel 7 vorgestellt und aus diesen werden schlussendlich Handlungsempfehlungen abgeleitet.

2 Aktuelle Situation

2.1 Motivation

In den letzten Jahren ist die Entwicklung von Smartphones rasant fortgeschritten. Obwohl die Energieeffizienz von mobilen Geräten bereits seit über einem Jahrzehnt ein aktuelles Thema ist, ist das Hauptziel dieses Forschungsgebiets nach wie vor die Entwicklung von Methoden zur Reduktion des Energieverbrauchs bei gleichbleibender Leistung des Geräts (Tarkoma, Siekkinen, Lagerspetz, & Xiao, 2014, S. 3). Heutige Smartphones sind aus Hardware- sowie Softwaresicht um einiges komplexer als sogenannte "feature phones" oder "dumbphones". Ein Feature Phone bezeichnet hierbei ein einfaches Mobiltelefon mit grundsätzlichen Funktionen wie beispielsweise Telefonie sowie das Senden und Empfangen von SMS (Oxford University Press, 2015). Dem gegenüber stehen aktuelle Smartphones mit Apps von Drittanbietern, neuen Features und Kommunikationstechnologien wie beispielsweise TCP/IP, Webbrowser und E-Mail, Radio, GPS-Empfänger oder Musik- und Videowiedergabe (Tarkoma et al., 2014, S. 7).

In der nachfolgenden Tabelle 2.1 zeigt Berkel (2009) die Entwicklung von Smartphones in den letzten 20 Jahren; in Fünf-Jahres-Intervallen:

	1995	2000	2005	2010	2015
Mobilfunkgeneration	2G	2.5-3G	3.5G	Übergang zu 4G	4G
Mobilfunkstandard	GSM	GPRS UMTS	HSPA	HSPA LTE	LTE LTE-A
Downlink (Mb/s)	0.01	0.1	1	10	100
Anzeigepixel (x1000)	4	16	64	256	1024

	1995	2000	2005	2010	2015
Kommunikationsmodule	-	-	Wi-Fi, Bluetooth	Wi-Fi, Bluetooth	Wi-Fi, Bluetooth Low Energy RFID
Taktfrequenz CPU (MHz)	20	100	200	500	1000
Leistung CPU (W)	0.05	0.05	0.1	0.2	0.3
MHz / W	400	2000	2000	2500	3000
Akkukapazität (Wh)	1	2	3	4	5

Tabelle 2.1: Smartphone-Trends in 5-Jahres-Intervallen seit 1995 (Berkel, 2009)

Diese Tabelle lässt erkennen, dass sich die Leistung alle fünf Jahre um Faktor 4 (bei den Anzeigepixeln) bzw. Faktor 5 (Taktfrequenz des Prozessors) bzw. Faktor 10 (Downlink) vervielfacht hat, sich die Akkukapazität jedoch nur minimal gesteigert hat. Dies liegt unter anderem daran, dass die Akkukapazität durch chemische Eigenschaften bestimmt und somit begrenzt ist. Durch grössere Akkus wird auch die Kapazität gesteigert, dadurch ist jedoch auch das Gerät grösser; dies ist allerdings nicht wünschenswert (Tarkoma et al., 2014, S. 8). Eine Ausnahme bilden hierbei die so genannten *Phablets*, ein Hybrid zwischen Phone und Tablet oder Geräte wie beispielsweise das iPhone 6 Plus; bei diesen Geräten zeigt sich wiederum ein Trend zu grösseren Displays.

In den nachfolgenden Kapiteln werden einige Möglichkeiten vorgestellt, wie die Energieeffizienz eines Smartphones trotz den vorher erwähnten Einschränkungen verbessert werden kann; es sind dies der Einsatz von mehreren Prozessoren, aktuelle Kommunikationstechnologien (Bluetooth Low Energy; BLE, 3G, 4G) sowie Features welche von den Entwicklern bei Google (Android) und Apple (iOS) zur Verfügung gestellt werden.

2.2 Einsatz von mehreren Prozessoren

Als Beispiel nennen Tarkoma et al. (2014, S. 10) das Samsung Galaxy S4. Dieses Gerät benutzt zur Datenverarbeitung die "big.LITTLE"-Technologie von ARM Ltd. (2014). Diese Technologie setzt zwei Prozessoren ein, einerseits einen leistungsfähigen und andererseits einen energieeffizienten. Je nach Leistungsanforderung eines Tasks wird dieser durch die big.LITTLE Multi-Processing-Software an den oder die entsprechenden Prozessorkerne zugeteilt (ARM Ltd., 2014).

2.3 Aktuelle Kommunikationstechnologien

Nachfolgend werden einige Technologien vorgestellt, um den Energieverbrauch bei der Kommunikation zu minimieren. Es sind dies:

- PSM (Power-saving mechanism; Energiesparmechanismus) des 802.11-Standards (WLAN)
- Bluetooth Low Energy
- Mobilfunkstandards (3G, 4G)

2.3.1 Energiesparmechanismus des 802.11-Standards

Beim 802.11-Standard (WLAN) existiert ein Mechanismus, welchen den Energieverbrauch verringern soll: "The 802.11 standard defines a power-saving mechanism (PSM)" (Tarkoma et al., 2014, S. 112). Gemäss dieser Quelle kann der Client (das Smartphone) in den Schlafmodus wechseln, wenn nicht aktiv Daten gesendet oder empfangen werden. Bevor dieser Schlafmodus aktiviert werden kann, muss der Access Point (AP), mit welchem das Smartphone verbunden ist, informiert werden, damit der AP die Pakete für das Smartphone puffert.

2.3.2 Bluetooth Low Energy

Bluetooth Low Energy, auch unter dem Namen *Bluetooth Smart* bekannt, ist die vierte Version (v4.0) dieser Kommunikationstechnologie. Diese ermöglicht die Kommunikation mit Geräten, welche Knopfzellen als Energiequelle verwenden, wie beispielsweise Smartwatches (Bluetooth, 2015). Gemäss dieser Quelle können Hersteller die Reichweite auf über 200 Fuss (ca. 61 Meter) festlegen; der Energieverbrauch ist dabei tiefer als bei den Vorgängerversionen.

2.3.3 Mobilfunkstandards (3G, 4G)

Gemäss Tarkoma et al. sind HSPA (3G) und LTE (4G) momentan die zwei dominierenden Mobilfunkstandards für die Datenübertragung (2014, S. 118). Im HSPA- sowie im LTE-Mobilfunknetz wird der Energieverbrauch durch das Radio Resource Control (RRC) Protokoll gesteuert. Für diese Arbeit interessant sind die vier Zustände dieses Protokolls. Diese werden in der nachfolgenden Tabelle 2.2 vorgestellt:

Kriterien → Zustand ↓	Datenvolumen	Energieverbrauch	Bemerkungen
CELL_DCH	Gross	Hoch	
CELL_FACH	Klein	Niedrig	
CELL_PCH	-	Niedrig	Keine aktive Verbindung
IDLE	-	Niedrig	Keine RRC-Verbindung

Tabelle 2.2: Die verschiedenen Zustände des RRC-Protokolls bei HSPA (3G) (nach Tarkoma et al., 2014, S. 119)

Der Zustand CELL_DCH wird dabei für die Übertragung von grösseren Datenmengen verwendet, wie beispielsweise bei Streaming-Apps. Für das Senden und Empfangen von kleineren Datenmengen wechselt das RRC-Protokoll in den Zustand CELL_FACH. Der Zustand CELL_PCH wird verwendet, wenn keine aktive Verbindung vorhanden ist (Tarkoma et al., 2014, S. 119). Die Übergänge in die oben genannten Zustände sind auf Abbildung 2.1 ersichtlich:

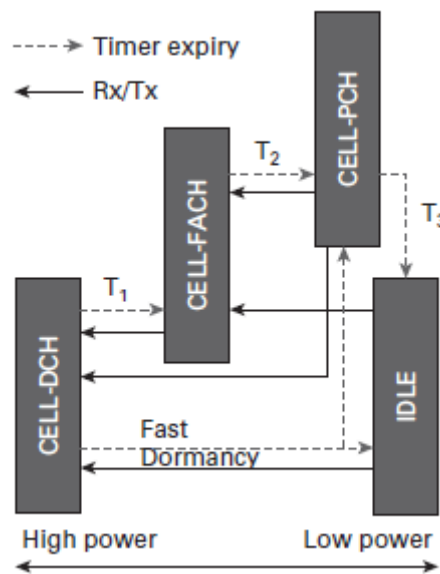


Abbildung 2.1: Wechsel der Zustände des RRC-Protokolls bei HSPA (3G) (nach Tarkoma et al., 2014, S. 119)

Die Werte für T_1 bis T_3 auf der obigen Abbildung werden vom Netzbetreiber festgelegt. Wie auf der obigen Abbildung ebenfalls ersichtlich ist, erlaubt Fast Dormancy (FD) den direkten Wechsel von CELL_DCH zu CELL_PCH oder IDLE; der Energieverbrauch bei CELL_PCH und IDLE ist etwa derselbe (Tarkoma et al., 2014, S. 119).

Bei LTE (4G) gibt es lediglich zwei Zustände: RRC_CONNECTED und RRC_IDLE. Die Leistungsaufnahme bei RRC_CONNECTED ist dabei um einiges höher als bei RRC_IDLE.

2.4 Features der mobilen Betriebssysteme

2.4.1 Android

Android bietet in der Version 5.0 (Lollipop) diverse Tools, um die Akkulaufzeit zu erhöhen. Diese unter dem Namen *Project Volta* vorgestellten Features sind ein Aufgabenplaner (Job Scheduler) sowie ein Befehl zur Generierung von Akkunutzungsstatistiken (Google Inc., unbekannt-a).

Zusätzlich zu den oben erwähnten Features bietet Android 5.0 einen Energiesparmodus (Battery saver), welcher, wenn aktiviert, die Leistung des Geräts drosselt und somit den Energieverbrauch senkt (Dobie, 2014). Der Energiesparmodus bei einem Testgerät (Samsung Galaxy S4 mit Android 4.4.2) lässt sich manuell ein- oder ausschalten und bietet folgende Optionen:

- Die maximale Leistung des CPUs lässt sich einschränken
- Die Helligkeit des Displays wird auf ein Minimum festgelegt
- Haptisches Feedback kann deaktiviert werden (d.h. kurzes Vibrieren beim Betätigen der Tasten des Smartphones)

Die folgende Abbildung 2.3 wurde auf dem oben genannten Testgerät erstellt. Auf dieser Abbildung sind die oben aufgelisteten Optionen ersichtlich:

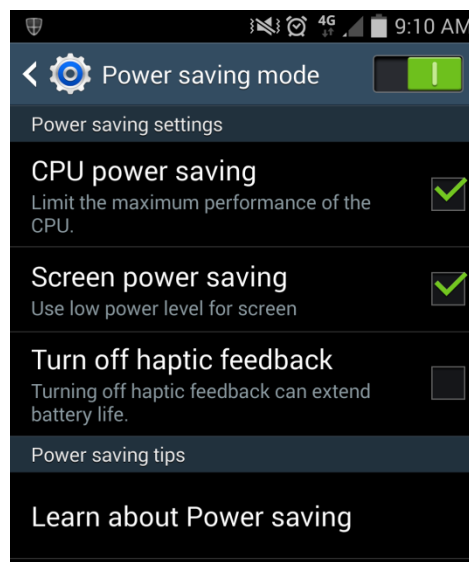


Abbildung 2.3: Energiesparoptionen bei Android 4.4.2 (eigene Darstellung)

Diese Optionen sind via Einstellungen – Mein Gerät – Energiesparmodus erreichbar. Im Verlauf der Arbeit wurde Android 5.0.1 auf dem Testgerät installiert. Die Energiesparoptionen wurden in dieser Version nicht verändert.

Wie sich diese Energiesparoptionen, besonders die Einschränkung der Leistung des Prozessors, auf die Akkukapazität auswirken, wird in Kapitel 6.2 untersucht.

2.4.2 Windows Phone

Windows Phone bietet ähnliche Features wie Android. Auf diesem Betriebssystem ist ebenfalls ein Stromsparmodus (Battery saver) integriert (Microsoft, 2014). Auf einem Testgerät war dieser standardmässig installiert und aktiviert.

2.4.3 iOS

Im Gegensatz zu den anderen beiden mobilen Betriebssystemen scheint es nach gründlicher Recherche im Internet keinen Stromsparmodus für iOS-Geräte (iPhone, iPad und iPod Touch) zu geben. Apple gibt lediglich Tipps zur Erhöhung der Akkulaufzeit (Apple Inc., 2015). Diese Tipps gelten für alle mobilen Betriebssysteme und werden im Kapitel 5 – Massnahmen (Seite 14) aufgelistet.

3 Messinstrumente / Tools

In diesem Kapitel werden einige aktuelle Tools vorgestellt, mit welchen der Energieverbrauch gemessen werden kann.

3.1 ARM DS-5 Development Studio 5.21.1

"DS-5 is a professional software development solution for Linux-based systems and bare-metal embedded systems, covering all stages in development from boot code and kernel porting to application and bare-metal debugging including performance analysis" (ARM Ltd., 2015a). Unter einem *bare-metal embedded system* versteht man hierbei einen Computer (Hardware), bei welchem die Software fehlt (TechTerms.com, 2013). Ein Teil dieses Development Studios heisst *Streamline*, welches Entwicklern helfen soll, eine möglichst energieeffiziente App bei gegebener Hardware zu entwickeln.

DS-5 ist in drei verschiedenen Editionen verfügbar; Ultimate Edition, Professional Edition und Community Edition (ARM Ltd., 2015b). Gemäss dieser Quelle unterstützt die Ultimate Edition die neusten ARM-Prozessoren und –Technologien wie beispielsweise ARMv8. Die Professional Edition unterstützt ARM-Prozessoren bis und mit ARMv7. Die Community Edition, welche kostenlos ist, bietet eine begrenzte Funktionalität der Tools, hier miteinbezogen ist auch das oben erwähnte Streamline.

Dieses Tool basiert auf der "Eclipse"-Entwicklungsumgebung. Dies war einer der Gründe, weshalb DS-5 nicht für diese Arbeit geeignet war; Google empfiehlt die Verwendung von Android Studio für die Entwicklung und das Debuggen von Android-Applikationen¹. Der zweite Grund ist dass dieses Tool wie oben erwähnt für bare-metal-Systeme gedacht ist und somit das Ziel dieser Arbeit verfehlt.

3.2 AT&T Application Resource Optimizer

Im Gegensatz zu DS-5 ist dieses Tool kostenlos. "AT&T Application Resource Optimizer (ARO) is a free, open-source tool for developers and testers that provides recommendations which allow you to optimize the performance of mobile web applications, make battery usage more efficient, and reduce data usage." (AT&T Inc., 2015a). Mithilfe dieses Tools können Geräte mit Android (Smartphone), Windows 8 (Desktop-PC oder Laptop) und iOS auf die Energieeffizienz getestet werden (AT&T Inc., 2015b). Gemäss dieser Quelle unterstützt diese Software momentan keine Windows Phone 8-Geräte oder auf Windows 8 RT-basierende Geräte (wie beispielsweise das Surface).

Auf derselben Seite sind unter dem Punkt *Effective Testing* Empfehlungen zu finden, wie ein Entwickler seine App möglichst effektiv testen kann. AT&T Inc. (2015b) empfiehlt dabei generell zwei Szenarien, ein aktives sowie ein Leerlauf-Szenario. Beim aktiven Szenario soll der Entwickler die App während 20 Minuten so verwenden, wie sie der Endbenutzer verwenden würde. Im Leerlauf-Szenario soll die App gestartet werden und während 30 Minuten nicht benutzt werden. Dadurch wird eine Grundlinie (*baseline*) gemessen bezüglich Netzwerkaktivität sowie Energieverbrauch unabhängig vom Verwenden der App.

¹ Android Studio basiert auf der Entwicklungsplattform IntelliJ von JetBrains (Google Inc., unbekannt-c)

Folgende Metriken bezüglich des Energieverbrauchs werden vom AT&T ARO gemessen (alle Angaben in Joule):

- Total Energieverbrauch des Netzwerkinterfaces
- GPS: Aktiv und Standby
- Verbrauch der Kamera
- Bluetooth: Aktiv und Standby
- Energieverbrauch des Displays

Auf Abbildung 3.1 ist zu sehen, wie die Daten eines Beispiel-Traces aussehen:

Energy Efficiency Simulation

IDLE->Continuous Reception:	2.74 J
Continuous Reception:	20.83 J
Continuous Reception Tail:	15.36 J
Short DRX:	1.51 J
Long DRX:	184.43 J
IDLE:	4.21 J
Total RRC Energy:	213.72 J
Joules per Kilobyte:	0.49
GPS Active:	0.00 J
GPS Standby:	0.00 J
Total GPS Energy:	0.00 J
Total Camera Energy:	0.00 J
Bluetooth Active:	0.00 J
Bluetooth Standby:	0.00 J
Total Bluetooth Energy:	0.00 J
Total Screen Energy:	229.55 J

Abbildung 3.1: Energieeffizienz-Simulation eines Beispiel-Traces des AT&T ARO-Tools (eigene Darstellung)

AT&T ARO bietet eine Auswahl von mehreren Profilen, um den Energieverbrauch je nach Interface (3G, 4G oder Wi-Fi) zu simulieren. Auf Abbildung 3.1 wurde das Profil "4G" gewählt. Der Energieverbrauch der vier Zustände (CR, Short DRX, Long DRX und IDLE; siehe Kapitel 2.3.3) lässt sich mit diesem Tool messen.

3.3 PowerTutor

PowerTutor ist ein Tool, welches im Google Play Store vorhanden ist. "PowerTutor is an application for Google phones that displays the power consumed by major system components such as CPU, network interface, display, and GPS receiver and different

applications." (Gordon, Zhang, & Tiwana, PowerTutor, 2011a). Dieses Tool gibt dem Benutzer Informationen zum Energieverbrauch der momentan laufenden Prozesse. Diese Informationen werden mithilfe von Grafiken verständlich dargestellt, wie folgende Abbildung 3.2 zeigt:

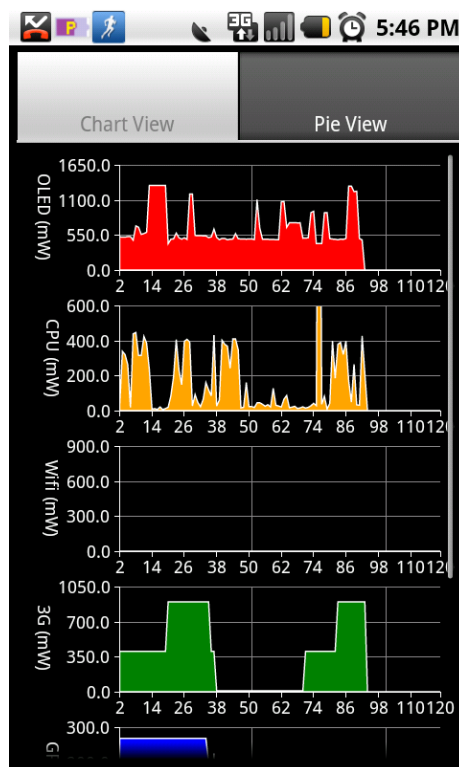


Abbildung 3.2: Beispiel der graphischen Anzeige des Energieverbrauchs bei PowerTutor (Gordon et al., 2011a)

Gemäss Gordon et al. (2011a) funktioniert diese App nur auf dem HTC G1, HTC G2 und dem Nexus One. Auf anderen Geräten, wie beispielsweise auf dem Testgerät des Verfassers (Samsung Galaxy S4 mit Android 4.4.2), sind die Ergebnisse geschätzt.

Ausserdem bietet dieses Tool eine Option, die Ergebnisse zu exportieren. Der Entwickler erhält mit dieser Option die Möglichkeit, alle Details in einer Textdatei anzusehen (Gordon et al., 2011a). In diesen Log-Dateien sind die folgenden Angaben vorhanden:

- Energieverbrauch in Milliwatt (mW)
 - Bildschirm
 - Prozessor
 - Wi-Fi
 - 3G
 - Audio
 - Gesamtverbrauch

Wenn PowerTutor aktiviert ist, wird jede Sekunde der Energieverbrauch der soeben genannten Komponenten gemessen. Bei einer durchgeführten Testmessung wurde der Energieverbrauch des Wi-Fis von diesem Tool nicht gemessen. Das Testgerät war in einem Wi-Fi-Netzwerk und zu Testzwecken wurde ein YouTube-Video mit Ton abgespielt.

3.4 JouleUnit

JouleUnit ist ein Tool, welches einige Features von JUnit nutzt, um den Energieverbrauch bei Android-Geräten zu messen (Wilke, 2013). Jedoch wurde die JouleUnit Workbench für die "Eclipse"-Entwicklungsumgebung entwickelt. Google Inc. empfiehlt jedoch als Entwicklungsumgebung Android Studio² zu verwenden (Google Inc., unbekannt-b). Aus diesem Grund wird dieses Tool für diese Arbeit nicht verwendet.

Nach ausführlicher Recherche existieren für Android Studio keine Tools / Plugins zur Optimierung des Energieverbrauchs. Das einzige vorhandene Tool mit dem Namen *Android Energy Optimization* (AEON) befindet sich noch in der Alpha-Phase und es existiert keine Dokumentation (Gonzalez, 2014).

3.5 Xcode Energy Diagnostics

Xcode, die Entwicklungsumgebung für Apple's iOS, liefert standardmässig Tools zur Diagnose des Stromverbrauchs einer App. Dieses ist auf Abbildung 3.3 ersichtlich:

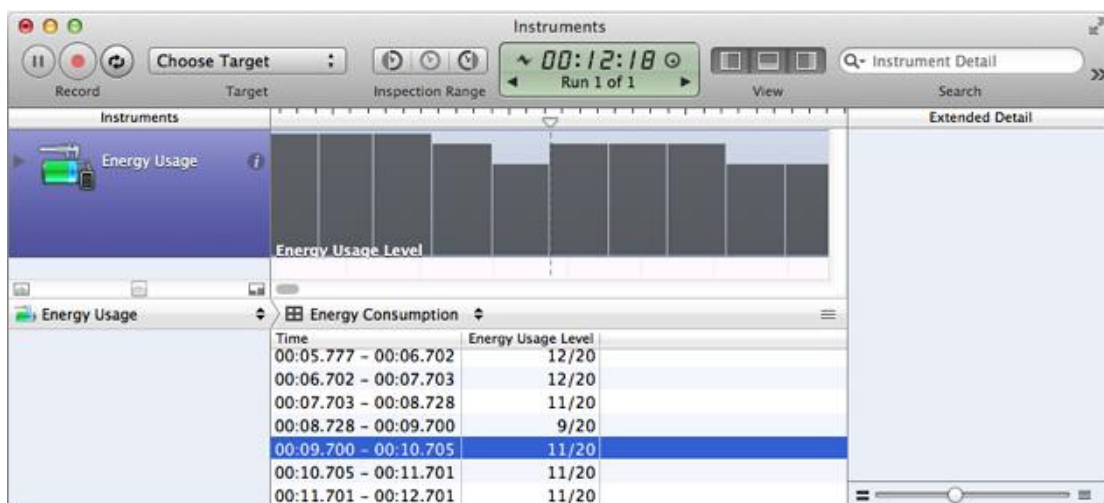


Abbildung 3.3: Screenshot des Energy Diagnostics Tools bei Xcode (Apple Inc., 2014)

² Android Studio basiert auf der Entwicklungsplattform IntelliJ von JetBrains (Google Inc., unbekannt-c)

Dieses Energy Diagnostics Tool verwendet eine Skala von 0 bis 20 um den Energieverbrauch darzustellen (siehe "Energy Usage Level" auf der obigen Abbildung). Dieser wird jede Sekunde gemessen, wie es anhand der Spalte "Time" ersichtlich ist.

3.6 Fazit

In diesem Kapitel wurden einige Messinstrumente vorgestellt, mit welchen der Energieverbrauch gemessen werden kann. Ausser PowerTutor, welches ausschliesslich auf dem Smartphone verwendet wird, handelt es sich bei den anderen vier Tools um Software, welche auf Arbeitsstationen (PC oder Laptop mit Windows) verwendet werden. Eine Ausnahme bildet hierbei das Energy Diagnostics Tool für Xcode, welches nur auf Geräten mit einem Betriebssystem von Apple verwendet werden kann.

Für diese Arbeit interessant sind die Tools "AT&T Application Resource Optimizer" und "PowerTutor". Diese zwei Tools erfüllen die Anforderungen dieser Arbeit; sie sind kostenlos verwendbar und bieten eine ausreichende Anzahl an Metriken.

4 Metriken

In diesem Kapitel werden die Metriken festgelegt, welche für diese Arbeit verwendet werden. In der nachfolgenden Tabelle 4.1 ist ersichtlich, für welche Komponenten der Energieverbrauch gemessen werden kann:

Tool → Komponente ↓	AT&T ARO	PowerTutor
GPS	x	
Kamera	x	
Bluetooth	x	
Bildschirm	x	x
3G / 4G	x	x
Prozessor		x
Wi-Fi³	x	x
Audio		x

Tabelle 4.1: Übersicht der messbaren Komponenten der ausgewählten Tools (eigene Darstellung)

³ Wie in Kapitel 3.3 erwähnt wurde, ist der Energieverbrauch des Wi-Fis nicht gemessen worden

Die fett markierten Komponenten sind mit beiden Tools messbar. Der Energieverbrauch der Komponenten wird beim AT&T ARO in Joule (J) und bei PowerTutor in Milliwatt (mW) gemessen.

Eine weitere Metrik ist der Akkustand. Nach einer gewissen Zeit wird dieser gemessen und daraus kann die Leistungsdauer des Smartphones berechnet werden. Zusätzlich dazu kommt die Energieeffizienz bei der Datenübertragung, welche in Joule pro Kilobyte (J/KB) gemessen wird.

5 Massnahmen

In den nachfolgenden Unterkapiteln werden zunächst allgemeine Tipps für die Reduktion des Energieverbrauchs bei Smartphones gegeben, ab Kapitel 5.2 folgen einige konkrete Massnahmen, wie ein Entwickler den Energieverbrauch gezielt minimieren kann.

5.1 Allgemeine Empfehlungen

Diese Empfehlungen richten sich primär an Benutzer von Smartphones. Um die Laufzeit des Akkus eines Smartphones zu maximieren, sollten folgende Tipps befolgt werden:

- **Netzwerkmodus ändern:** Diese Option steht bei Android sowie bei iOS zur Verfügung. Bei iOS ist diese Einstellung unter Einstellungen – Mobiles Netz – Sprache & Daten zu finden, bei Android unter Einstellungen – Verbindungen – Weitere Einstellungen (im Bereich Netzwerkverbindungen) – Mobile Netzwerke – Netzmodus. Bei iOS kann zwischen LTE, 3G und 2G ausgewählt werden; um dieselbe Einstellung bei Android zu erzielen, muss der entsprechende Punkt ausgewählt werden:
 - LTE (iOS) = LTE/WCDMA/GSM (Android)
 - 3G = Nur WCDMA
 - 2G = Nur GSM
- **Helligkeit anpassen:** Bei Android können die Komponenten eingesehen werden, welche am meisten Akku verbrauchen. In der Version 4.4.2 lassen sich diese Informationen unter Einstellungen – Optionen – Akku anzeigen (siehe Abbildung 5.1):

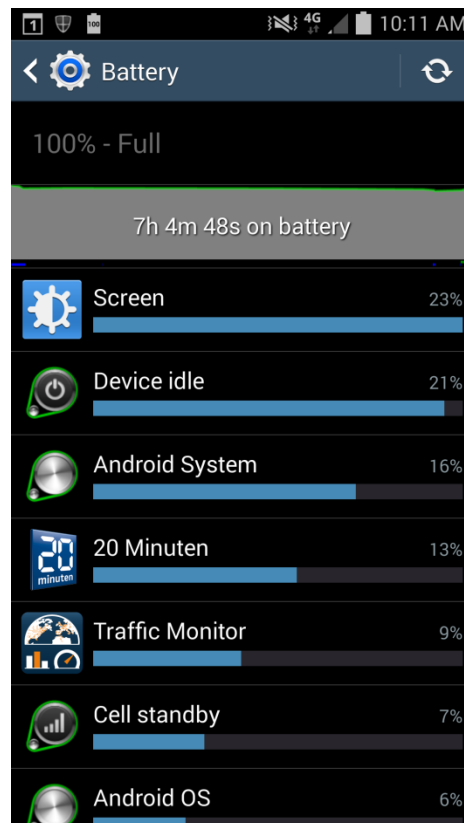


Abbildung 5.1: Anzeige der Komponenten, welche am meisten Akku verbrauchen (eigene Darstellung)

Aus dieser Abbildung ist ersichtlich, dass der Bildschirm am meisten Akku verbraucht (23%). Um die Akkulaufzeit zu erhöhen, sollte die Helligkeit so niedrig wie möglich eingestellt werden. Ebenfalls ersichtlich ist die geschätzte verbleibende Laufzeit des Akkus.

- **Zusatzfunktionen ausschalten:** Gemeint sind hierbei Funktionen wie GPS oder Bluetooth. Sofern diese nicht verwendet werden, sollten sie deaktiviert sein.
- **Statisches Hintergrundbild verwenden:** Android erlaubt die Verwendung von animierten Hintergrundbildern. Aus Sicht der Akkulaufzeit sollten jedoch statische Bilder verwendet werden.
- **Apps im Hintergrund beenden:** Dieser Tipp bezieht sich auf Android sowie iOS. Bei iOS lassen sich Apps im Hintergrund beenden, indem zunächst zweimal die Home-Taste gedrückt wird. Danach lassen sich die laufenden Apps durch Wischen der Vorschau nach oben beenden. Bei Android werden die zuletzt verwendeten Apps durch längeres Drücken der Home-Taste angezeigt (vgl. Abbildung 5.2).

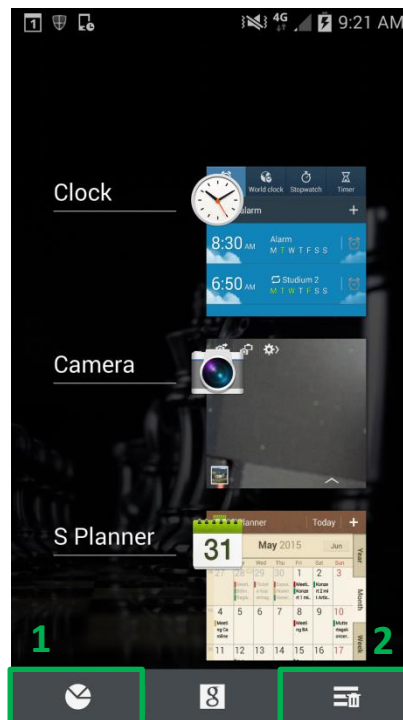


Abbildung 5.2: Anzeige der zuletzt verwendeten Apps bei Android (Version 4.4.2) (eigene Darstellung)

Um die im Hintergrund laufenden Apps anzusehen und zu beenden, muss der Button ganz links (1) gedrückt werden. Dieser öffnet den Task-Manager, mit welchem einzelne Apps (mit dem Button "End") oder alle beendet werden können (Button "End all", vgl. Abbildung 5.3).

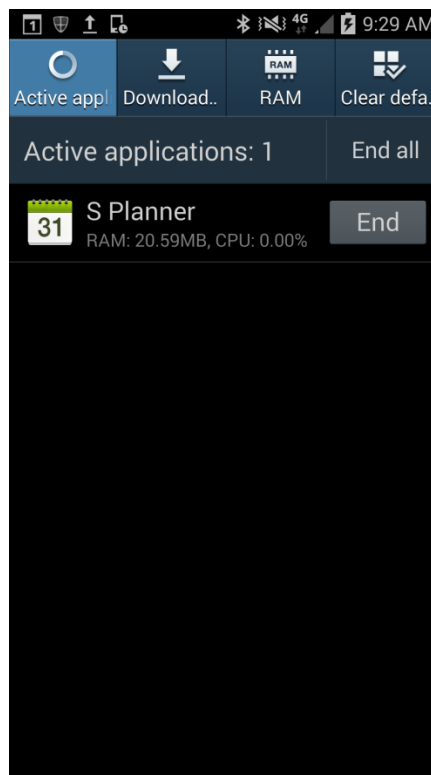


Abbildung 5.3: Aktive Apps anzeigen bei Android (Version 4.4.2) (eigene Darstellung)

Der Button ganz rechts bei Abbildung 5.2 (2) löscht die Liste der zuletzt verwendeten Apps.

- **Energiesparmodus verwenden:** Sofern möglich sollte der Energiesparmodus immer aktiviert sein. Dieser ist bei Android unter Einstellungen – Mein Gerät – Energiesparmodus zu finden.

Messungen des Energieverbrauchs beim Anwenden dieser Empfehlungen im Vergleich zu möglichst suboptimalen Einstellungen erfolgen in Kapitel 6.2 (Szenario 1: Allgemeine Empfehlungen).

5.2 Packet coalescing

Unter dem Begriff *packet coalescing* versteht man die Gruppierung von Datenpaketen. Das Ziel dieser Technologie ist das Vermeiden von vielen kleinen Aktualisierungen über das Netzwerk. Stattdessen werden diese Aktualisierungen lokal gruppiert und dann beispielsweise als Array hochgeladen. Dadurch wird das Kommunikationsinterface (3G, 4G oder Wi-Fi) nur einmal aktiviert statt bei jeder Aktualisierung (Tarkoma et al., 2014, S. 140).

Um diese Technik einzusetzen, müssen zwei Werte eingeführt werden: ein Grenzwert sowie ein Time-Out (Herrería-Alonso, Rodríguez-Pérez, Fernández-Veiga, & López-García, 2012). Wenn einer dieser Werte überschritten wird, werden die Daten übertragen. Der Grenzwert muss dabei an die App angepasst werden; wenn der Wert zu hoch ist, wird es lange Verzögerungen bei der Datenübertragung geben, wenn der Wert jedoch zu tief ist, werden die Daten zu oft übertragen und trägt somit nicht zur Optimierung des Energieverbrauchs bei. Wie der Grenzwert muss auch der Time-Out vom Entwickler auf einen sinnvollen Wert festgelegt werden.

Die Effizienz dieser Technik wird in Kapitel 6.3 (Szenario 2: Packet coalescing) untersucht, gemessen und bewertet.

5.3 Wakelocks

Wakelocks können nicht zur Reduzierung des Energieverbrauchs eingesetzt werden. Ihr Einsatz wird immer einen erhöhten Verbrauch verursachen. Dieses Kapitel gibt einen Einblick in diese Technologie und Empfehlungen wie diese – falls notwendig – korrekt eingesetzt werden können.

Wakelocks bei Android lassen den Entwickler sogenannte *locks* ("Sperrern") einsetzen, welche eine bestimmte Komponente aktiviert lässt (Tarkoma et al., 2014, S. 151). Wakelocks sind nützlich, wenn eine Aufgabe (Job) ohne Unterbrechung ausgeführt werden muss (Alam, Panda, Tripathi, Sharma, & Narayan, 2013). Jedoch ist zu beachten, dass der falsche Einsatz dieser Technologie negative Auswirkungen auf die Akkulaufzeit hat.

Zwei Beispiele für Wakelocks sind die YouTube- sowie die Musik-App (Bird, 2012). Wenn ein Video über die **YouTube-App** abgespielt wird, wird sich der Bildschirm nicht automatisch ausschalten, auch wenn das Time-out in den Systemeinstellungen überschritten wird. Wenn der Benutzer jedoch die Power-Taste betätigt, schaltet der Bildschirm aus und die Wiedergabe wird unterbrochen. Mithilfe des Befehls "adb shell dumpsys power" kann man erkennen, dass YouTube zwei Wakelocks verwendet (siehe Abbildung 5.4):

```
mLocks.size=2:
PARTIAL_WAKE_LOCK      'AudioOut_2' activated <minState=0, uid=1013, pid=1853>activeT=35139
SCREEN_BRIGHT_WAKE_LOCK 'KEEP_SCREEN_ON_FLAG' activated <minState=3, uid=1000, pid=2156>activeT=34688
```

Abbildung 5.4: Liste der Wakelocks beim Verwenden der YouTube-App (eigene Darstellung)

Bei der **Musik-App** wird ein anderer Wakelock verwendet; der Bildschirm wird sich automatisch ausschalten, sobald die angegebene Zeit in den Einstellungen überschritten wird. Die Musik wird jedoch weiterhin abgespielt, auch wenn der Benutzer die Power-Taste betätigt, d.h. der Prozessor läuft weiter. Bei der Eingabe desselben Befehls wie bei der YouTube-App erhält man das auf Abbildung 5.5 zu sehende Resultat:

```
mLocks.size=3:
PARTIAL_WAKE_LOCK      'android.media.MediaPlayer' activated <minState=0, uid=10157, pid=6066>activeT=15867
PARTIAL_WAKE_LOCK      'AudioOut_2' activated <minState=0, uid=1013, pid=1853>activeT=15853
PARTIAL_WAKE_LOCK      'com.sec.android.app.music.CorePlayerService' activated <minState=0, uid=10157, pid=6066>activeT=15794
```

Abbildung 5.5: Liste der Wakelocks beim Verwenden der Musik-App (eigene Darstellung)

In der nachfolgenden Tabelle 5.1 sind alle Wakelock-Level ersichtlich. Diese schliessen sich gegenseitig aus, d.h. es kann nur einer der unten ersichtlichen Werte (Flag value) angegeben werden.

Wert (Flag value)	CPU	Bildschirm	Tastatur	Deprecated?
PARTIAL_WAKE_LOCK	An	Aus	Aus	Nein
SCREEN_DIM_WAKE_LOCK	An	Gedimmt	Aus	Ja (17)
SCREEN_BRIGHT_WAKE_LOCK	An	Hell	Aus	Ja (13)
FULL_WAKE_LOCK	An	Hell	Hell	Ja (17)

Tabelle 5.1: Übersicht der Wakelock-Level bei Android (Google Inc., 2015c)

Um Wakelocks zu verwenden, muss die entsprechende Berechtigung in der Android-Manifest-Datei vergeben werden. Jedoch sind drei der oben erwähnten Werte veraltet (*deprecated*), nur `PARTIAL_WAKE_LOCK` ist noch zulässig (Google Inc., 2015c). Die Werte in den Klammern geben den API-Level an, in welchen der Wert als *deprecated* erklärt wurde. Wenn ein Entwickler das Smartphone daran hindern will, den Bildschirm auszuschalten, sollte dieser die Flag "`FLAG_KEEP_SCREEN_ON`" verwenden.

Um ein Wakelock anzufordern, stehen zwei Methoden zur Verfügung:

- `acquire()`
- `acquire(long timeout)`

Der Unterschied besteht darin, dass bei der zweiten Methode ein Time-out gesetzt werden kann, nach welchem der Wakelock automatisch freigegeben wird. Bei der ersten Methode muss dies manuell mit der Methode `release()` getan werden. Wenn der zugehörige Prozess einer App beendet wird, werden auch nicht freigegebene Wakelocks entfernt (Google Inc., 2007). Diese Quelle enthält den Quellcode der `PowerManagerService`-Klasse⁴.

Alam et al. (2013) geben in ihrer Arbeit zwei Beispiele, wie Wakelocks falsch platziert wurden und anschliessend wie diese korrekt gesetzt werden sollten. In der ersten Abbildung 5.6 ist ersichtlich, wie ein Wakelock ineffizient platziert wurde und dessen effizientere Lösung:

⁴ Der Quellcode ist unter folgendem Link verfügbar:

<https://android.googlesource.com/platform/frameworks/base/+b267554/services/java/com/android/server/power/PowerManagerService.java>

Auf Zeile 628 ist zu sehen, was beim Beenden des Prozesses passiert. Das Entfernen des Wakelocks erfolgt auf Zeile 718.

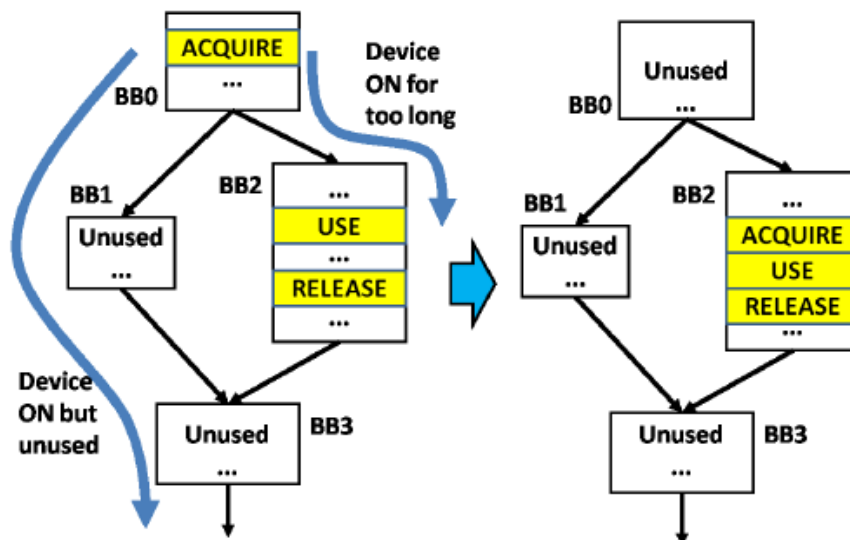


Abbildung 5.6: Beispiel 1 mit suboptimalem Einsatz von Wakelocks (links) und die effizientere Lösung (rechts) (Alam et al., 2013)

Links wird das Wakelock bereits im BB0⁵ akquiriert. Dies führt zu zwei Problemen: Erstens wird beim Pfad BB0 → BB1 → BB3 das Wakelock nicht freigegeben und zweitens vergeht zu viel Zeit beim Pfad BB0 → BB2, wodurch unnötig Akkuleistung verschwendet wird. Die Lösung dieses Problems ist das Wakelock unmittelbar vor dessen Gebrauch zu platzieren und anschliessend direkt freizugeben (siehe BB2 rechts in Abbildung 5.6). Dieser Lösungsansatz funktioniert allerdings nicht in allen Fällen, wie in Beispiel 2 (Abbildung 5.7) ersichtlich ist:

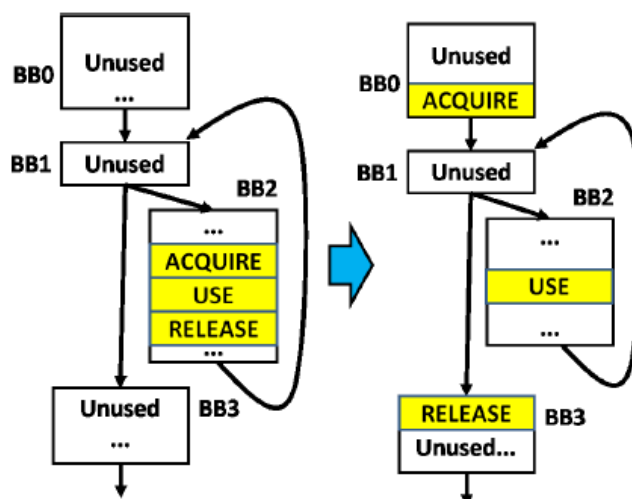


Abbildung 5.7: Beispiel 2 mit suboptimalem Einsatz von Wakelocks (links) und die effizientere Lösung (rechts) (Alam et al., 2013)

Beispiel 2 verdeutlicht, dass das Platzieren von Wakelocks unmittelbar vor und nach deren Verwendung nicht immer die optimale Lösung ist, beispielsweise in einer Schleife

⁵ BB(n) steht für Basic Block. Dieser Begriff beschreibt eine Sequenz von Quellcode mit genau einem Einstiegspunkt und genau einem Ausgangspunkt (Free Software Foundation, Inc., 2015)

(BB1 → BB2). Bei jeder Iteration wird das Wakelock akquiriert und wieder freigegeben, was wiederum unnötig Energie verbraucht. Die Lösung für dieses Problem ist das Verschieben der Akquirierung und der Freigabe des Wakelocks vor bzw. hinter der Schleife (siehe rechte Hälfte in Abbildung 5.7). Die Methode `acquire()` wird in BB0 aufgerufen und die Methode `release()` in BB3.

Falls ein Entwickler sich entscheidet, Wakelocks einzusetzen, muss er sich vergewissern, dass diese so platziert werden, dass die Energieeffizienz maximiert wird. Der Einsatz von Wakelocks per se trägt nicht zur Maximierung der Effizienz bei. Aus diesem Grund wurde zu dieser Massnahme kein Szenario entwickelt. Jedoch werden Wakelocks in den beiden Szenarien 1 und 3 verwendet.

5.4 Job Scheduler

Ziel des Job Schedulers ist das Vermeiden von rechenintensiven Operationen während sich das Smartphone im Akkubetrieb befindet oder das zeitliche Verschieben von Uploads oder Downloads auf einen Zeitpunkt, wenn das Smartphone mit einem WLAN verbunden ist.

Der Job Scheduler, eine weitere Technologie bei Android, lässt den Entwickler den Ausführungszeitpunkt von verschiedenen Arten von Aufgaben (Jobs) festlegen (Google Inc., 2015a). Um diesen Scheduler zu verwenden, muss der Entwickler zunächst Jobs (Klasse "JobInfo") anlegen (Google Inc., 2015b). Beispielsweise können Jobs angelegt werden, welche voraussetzen, dass das Gerät an eine Stromquelle angeschlossen ist (Methode "`setRequireCharging()`") oder dass sich das Gerät im Ruhezustand befindet (Methode "`setRequiresDeviceIdle()`").

Ein Beispiel für den Einsatz dieser JobScheduler API ist das Aktualisieren von Apps über den Google PlayStore (Pierick, 2015). Dieser Job sollte nur dann ausgeführt werden, wenn das Smartphone:

- via Wi-Fi mit einem drahtlosen Netzwerk verbunden ist (Zeile 3 in Abbildung 5.8),
- sich im Ruhezustand befindet (der Bildschirm ist ausgeschaltet; Zeile 4) und
- an eine Stromquelle angeschlossen ist (Zeile 5).

Auf Abbildung 5.8 ist ersichtlich, wie dieser Job angelegt wird:

```
1  ComponentName serviceName = new ComponentName(context, MyJobService.class);
2  JobInfo jobInfo = new JobInfo.Builder(JOB_ID, serviceName)
3      .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
4      .setRequiresDeviceIdle(true)
5      .setRequiresCharging(true)
6      .build();
```

Abbildung 5.8: Anlegen eines Jobs, welcher die Aktualisierung von Apps via den PlayStore simuliert (Pierick, 2015)

Auf Zeile 1 ist eine Java-Klasse namens *MyJobService* ersichtlich. Diese Klasse erweitert (*extends*) die Klasse `android.app.job.JobService` und ist gemäss Pierick (2015) der Dienst, welcher für die Ausführung des Jobs verantwortlich ist. Die Klasse `JobService` enthält zwei abstrakte Methoden, welche in `MyJobService.java` implementiert werden müssen:

- `onStartJob(JobParameters params)`
 - Diese Methode wird aufgerufen wenn die Parameter (Zeile 3 bis 5 auf Abbildung 5.8) des Jobs erfüllt sind
- `onStopJob(JobParameter params)`
 - Diese Methode wird aufgerufen wenn ein oder mehrere Parameter (Zeile 3 bis 5 auf Abbildung 5.8) nicht mehr erfüllt sind, d.h. wenn der Benutzer beispielsweise das Smartphone von der Stromquelle trennt

Gemäss Pierick (2015) müssen drei wichtige Details beim Einsatz von `JobService` beachtet werden:

- Der `JobService` läuft auf dem Hauptthread; es liegt in der Verantwortung des Entwicklers, dass die Arbeit (in diesem Beispiel das Aktualisieren der Apps) in einem separaten Thread ausgeführt wird. Ansonsten wird der Benutzer eine ANR-Fehlermeldung (Android Not Responding) erhalten. Auf Abbildung 5.10 auf Zeile 9 wird ein Task im Hintergrund (`AsyncTask`) gestartet, welcher ab Zeile 25 definiert ist.
- Der Job muss explizit abgeschlossen werden mithilfe der Methode `jobFinished(JobParameters params, boolean needsReschedule)`; siehe Abbildung 5.10, Zeile 39). Der Job Scheduler hält ein Wakelock für den Job, d.h. wenn die soeben genannte Methode nicht aufgerufen wird, bleibt der Wakelock bestehen und verursacht einen erhöhten Energieverbrauch
- Der Job muss im Android Manifest registriert sein. Wenn dies nicht der Fall ist, wird der Dienst vom System nicht erkannt und auch niemals ausgeführt. Auf Abbildung 5.9 (Zeile 5 bis 8) ist ersichtlich, wie der Dienst registriert wird:


```
1 <application
2     .... stuff ....
3 >
4
5 <service
6     android:name=".MyJobService"
7     android:permission="android.permission.BIND_JOB_SERVICE"
8     android:exported="true"/>
9 </application>
```

Abbildung 5.9: Registrierung des Dienstes in der Datei AndroidManifest.xml (Pierick, 2015)

Auf Abbildung 5.10 ist ein Teil dieses Dienstes (MyJobService) ersichtlich:

```
01 public class MyJobService extends JobService {
02
03     private UpdateAppsAsyncTask updateTask = new UpdateAppsAsyncTask();
04
05     @Override
06     public boolean onStartJob(JobParameters params) {
07         // Note: this is preformed on the main thread.
08
09         updateTask.execute(params);
10
11         return true;
12     }
13
14     // Stopping jobs if our job requires change.
15
16     @Override
17     public boolean onStopJob(JobParameters params) {
18         // Note: return true to reschedule this job.
19
20         boolean shouldReschedule = updateTask.stopJob(params);
21
22         return shouldReschedule;
23     }
24
25     private class UpdateAppsAsyncTask extends AsyncTask<JobParameters, Void, JobParameters[]> {
26
27
28         @Override
29         protected JobParameters[] doInBackground(JobParameters... params) {
30
31             // Do updating and stopping logical here.
32             return params;
33         }
34
35         @Override
36         protected void onPostExecute(JobParameters[] result) {
37             for (JobParameters params : result) {
38                 if (!hasJobBeenStopped(params)) {
39                     jobFinished(params, false);
40                 }
41             }
42         }
43
44         ...
45     }
46 }
```

Abbildung 5.10: MyJobService.java – Dienst, welcher für die Ausführung des Jobs verantwortlich ist (Pierick, 2015)

Die beiden abstrakten Methoden `onStartJob(JobParameters params)` und `onStopJob(JobParameter params)` wurden in der obigen Klasse implementiert (Zeile 6 bzw. 17), sowie ein Hintergrundtask (`AsyncTask`), welcher das Aktualisieren der Apps simuliert (Zeile 25).

Der letzte Schritt dieses Beispiels ist das Festlegen (*scheduling*) des Jobs, wie es auf Abbildung 5.11 (Zeile 2) ersichtlich ist:

```
1  JobScheduler scheduler = (JobScheduler) context.getSystemService(Context.JOB_SCHEDULER_SERVICE);
2  int result = scheduler.schedule(jobInfo);
3  if (result == JobScheduler.RESULT_SUCCESS) Log.d(TAG, "Job scheduled successfully!");
```

Abbildung 5.11: Festlegung (*scheduling*) eines Jobs (Pierick, 2015)

Fazit: Der Job Scheduler ist nützlich, wenn rechen- oder kommunikationsintensive Jobs nur unter gewissen Voraussetzungen ausgeführt werden sollen. Dadurch wird verhindert, dass unnötig Energie im Akkubetrieb verschwendet wird oder Kosten durch das Übertragen von Daten via Mobilfunknetz entstehen.

Für diese Technik wird kein Szenario in Kapitel 6 erstellt, da kein geeignetes Beispiel für diese Arbeit gefunden wurde.

5.5 Lokale vs. entfernte Operationen

Das Ziel der entfernten Ausführung ist das Entlasten des Smartphones, da dieses in seiner Leistung (Prozessor, Arbeitsspeicher, Akku) stark begrenzt ist. Eine Möglichkeit ist das Auslagern (*Offloading*) von rechenintensiven Operationen auf Webdienste. Der Server, auf welchem ein Webdienst läuft, ist um ein vielfaches leistungsfähiger als ein aktuelles Smartphone. Es gilt jedoch abzuwägen, ob es sich lohnt, eine Operation auszulagern, wenn das Kommunikationsvolumen betrachtet wird.

Ein Beispiel von Offloading ist das Berechnen der Fibonacci-Reihe. Es handelt sich hierbei um eine unendliche Reihe von Zahlen, beginnend mit zweimal der Zahl eins, bei welcher immer die zwei vorherigen Zahlen addiert werden: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, usw.

In Kapitel 6.4 (Szenario 3: Lokale vs. entfernte Operationen) wird eine App vorgestellt, welche das Berechnen dieser Reihe lokal oder mithilfe eines Webdienstes erlaubt.

5.6 Native Programmierung

Eine weitere Methode ist das Verwenden des Android NDK (Native Development Kit); dieses erlaubt die Implementierung einer App oder Teile davon in nativen Sprachen wie C oder C++ (Google Inc., unbekannt-d). Dieses NDK wird beispielsweise empfohlen für Game-Engines. Generell empfiehlt Google Inc. (unbekannt-d) dieses Kit nur wenn es wirklich nötig

ist, keinesfalls weil der Entwickler C oder C++ bevorzugt. Der Umfang dieses Themas ist für diese Arbeit zu gross und wird deswegen nicht genauer behandelt. Aus diesem Grund wurde auch kein entsprechendes Szenario entwickelt.

5.7 Zwischenfazit

In diesem Kapitel wurden einige Methoden vorgestellt, wie ein Entwickler den Energieverbrauch seiner App gezielt minimieren kann. Drei dieser Methoden werden im nachfolgenden Kapitel 6 - Szenarien auf ihre Effizienz hin geprüft.

6 Szenarien

Anhand der in Kapitel 5 beschriebenen Massnahmen werden mithilfe der in Kapitel 4 genannten Metriken Szenarien beschrieben und gemessen. Folgende im Kapitel 5 vorgestellten Massnahmen werden in den folgenden Kapiteln auf ihre Effizienz geprüft:

- Allgemeine Empfehlungen (siehe Kapitel 5.1)
- Packet coalescing (siehe Kapitel 5.2)
- Lokale vs. entfernte Operationen (siehe Kapitel 5.5)

6.1 Testumgebung

Für Szenario 1 wurden lediglich das Smartphone (Samsung Galaxy S4 mit Android 4.4.2) für die Messungen sowie der Laptop für die Auswertung und Dokumentation verwendet. Für die anderen beiden Szenarien (2 und 3) wurde die in Abbildung 6.1 ersichtliche Testumgebung verwendet:

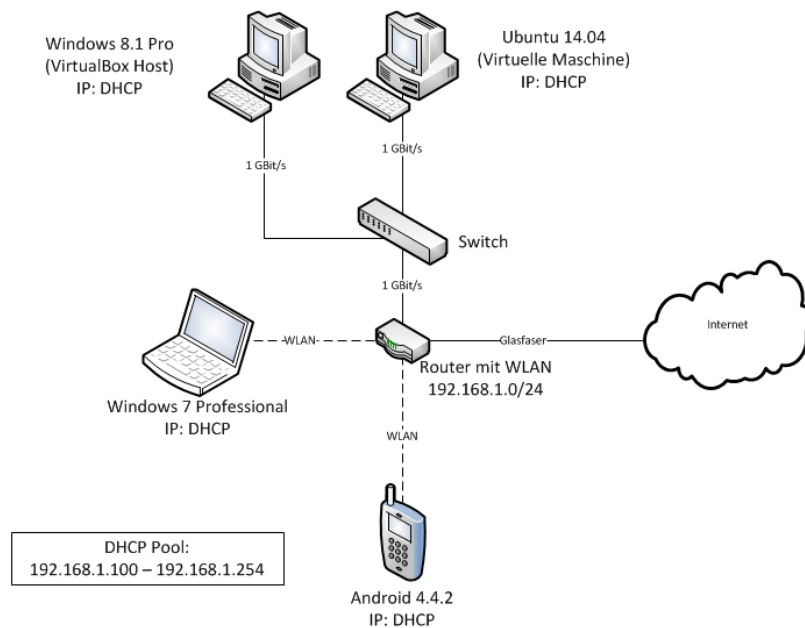


Abbildung 6.1: Testumgebung für Szenario 2 und 3 (eigene Darstellung)

Auf dem PC mit Windows 8.1 Pro ist Android Studio 1.2.2 sowie VirtualBox von Oracle installiert. Auf der virtuellen Maschine ist die "Eclipse"-Entwicklungsumgebung installiert. Der Laptop mit Windows 7 Professional wird zur Dokumentation sowie zur Auswertung der Messungen verwendet. Das Smartphone mit Android 4.2.2 (später 5.0.1) ist das Testgerät, auf welchem die entwickelten Apps installiert wurden. Wie die einzelnen Geräte in den Szenarien verwendet werden, wird jeweils zu Beginn jedes Kapitels beschrieben.

6.2 Szenario 1: Allgemeine Empfehlungen

Dieses Szenario soll den Unterschied zwischen zwei unterschiedlichen Konfigurationen eines Smartphones aufzeigen. Das Ziel von Konfiguration A ist es möglichst viel Energie zu verbrauchen, Konfiguration B soll durch die Anwendung der Empfehlungen in Kapitel 5.1 möglichst wenig Energie verbrauchen. Auf folgender Tabelle 6.1 sind die verwendeten Einstellungen ersichtlich:

Einstellungen	Konfiguration A	Konfiguration B
WLAN	An	Aus
GPS	An	Aus
Helligkeit	100%	0%
Energiesparmodus	Aus	An
Mobile Daten	An	Aus
Bluetooth	An (mit Smartwatch verbunden)	Aus

Tabelle 6.1: Einstellungen des Testgeräts für Szenario 1 (eigene Darstellung)

Das Testgerät ist ein Samsung Galaxy S4 mit Android Version 4.4.2 (später 5.0.1). Zu Beginn der beiden Konfigurationen wurde der Akku des Smartphones jeweils zu 100% geladen. Zur Messung der beiden Konfigurationen wurde eine App entwickelt, welche alle 10 Minuten die verbleibende Akkukapazität misst. Um den Bildschirm während diesen Messungen eingeschaltet zu lassen, wurde ein SCREEN_BRIGHT_WAKE_LOCK verwendet. Details zur Implementierung dieser App sind im Anhang I: Implementierung der App zur Messung der Akkukapazität zu finden.

Im Verlauf der Arbeit wurde eine Aktualisierung des Betriebssystems auf dem Testgerät durchgeführt. Mit der neuen Version 5.0.1 wurden die oben erwähnten Konfigurationen erneut gemessen.

Die gemessenen Werte wurden anschliessend in Excel übernommen und nachfolgend ausgewertet.

6.2.1 Ergebnisse – Android 4.4.2

Für Konfiguration A war nach 210 Minuten (3 Stunden, 30 Minuten) noch 10% Akku vorhanden. Die Messungen haben gezeigt, dass pro Zeiteinheit (10 Minuten) durchschnittlich 4.1% Akku verbraucht wurde. Angenommen, der Energieverbrauch setzt sich danach linear fort, beträgt die totale Laufzeit etwa 250 Minuten (4 Stunden).

Für Konfiguration B war nach 410 Minuten (6 Stunden, 50 Minuten) noch 9% Akku vorhanden. Die Messungen mit der App haben gezeigt, dass alle 10 Minuten durchschnittlich 2.2% Akku verbraucht wurde. Angenommen, der Energieverbrauch setzt sich linear fort, hält der Akku etwa 450 Minuten (7.5 Stunden).

Diese beiden Messungen verdeutlichen, dass der Akku bei Konfiguration B etwa doppelt so lange hält.

6.2.2 Ergebnisse – Android 5.0.1

Für Konfiguration A war nach 260 Minuten (4 Stunden, 20 Minuten) noch 12% Akku vorhanden. Die Messungen mit der App haben gezeigt, dass pro Zeiteinheit (10 Minuten)

durchschnittlich 3.4% Akku verbraucht wurde. Bei einer linearen Fortsetzung dieses Verlaufs hält der Akku etwas mehr als 290 Minuten (fast 5 Stunden).

Bei Konfiguration B war nach 470 Minuten (7 Stunden, 50 Minuten) noch 11% Akku vorhanden. Durchschnittlich wurde 1.9% Akku pro Zeiteinheit (10 Minuten) verbraucht. Angenommen, der Verbrauch setzt sich linear fort, beträgt die totale Laufzeit 530 Minuten (8 Stunden, 50 Minuten). Folgende Abbildung 6.2 verdeutlicht die Resultate dieser Messungen graphisch:

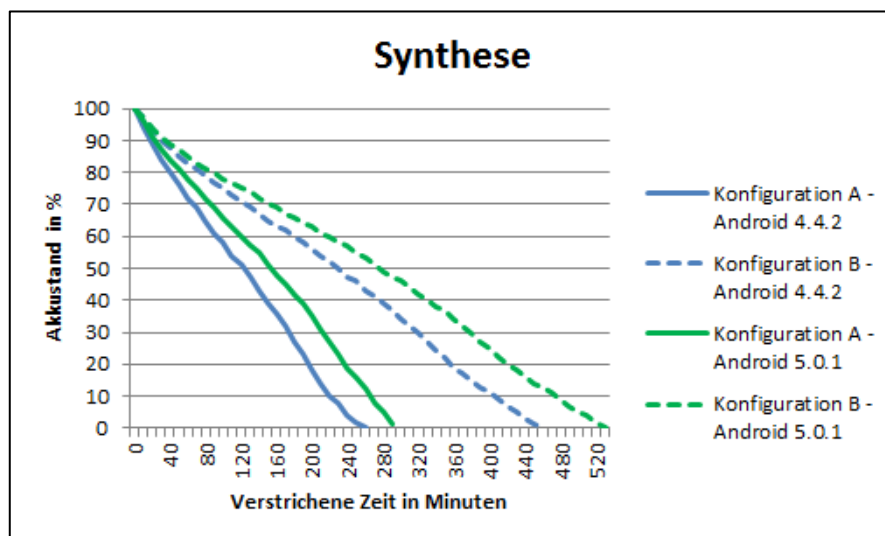


Abbildung 6.2: Verlauf der Akkukapazität der beiden Konfigurationen bei Android 4.4.2 und 5.0.1 (eigene Darstellung)

Aus dieser Abbildung ist zu erkennen, dass Android 5.0.1 bei beiden Konfigurationen energieeffizienter ist als Android 4.4.2. Der Akku hält bei Android 5.0.1 bei Konfiguration A um 16%, bei Konfiguration um 17.8% länger.

6.2.3 Ergebnisse – PowerTutor

Das Tool PowerTutor bietet eine Funktion, mit welcher die Logs auf dem Smartphone gespeichert werden können. Auf Abbildung 6.3 ist ein Auszug aus einer dieser Log-Dateien zu sehen:

```
begin 457
total-power 345
LCD 311
...
CPU 34
CPU-sys 8
CPU-usr 0
CPU-freq 384.0
CPU-0 5
CPU-1000 8
...
CPU-10255 0
Wifi 0
Wifi-on false
3G 0
3G-on false
GPS 0
GPS-state-times 1.0 0.0 0.0
GPS-sattelites 0
Audio 0
Audio-on false
```

Abbildung 6.3: Auszug aus dem Log; von PowerTutor generiert (eigene Darstellung)

Die Zahl nach "begin" gibt die verstrichene Zeit seit Start der Messung in Sekunden an, die Zahl nach "total-power" bezeichnet den Energieverbrauch in mW (Gordon, Zhang, & Tiwana, PowerTutor, 2011b). Die nächsten zwei Zeilen geben den Verbrauch des Bildschirms (LCD) und des Prozessors (CPU) jeweils in mW an. Der Verbrauch des LCDs war bei beiden Konfigurationen konstant; bei Konfiguration A waren es 900 mW (100% Helligkeit), bei Konfiguration B 311 mW (0% Helligkeit). Der durchschnittliche Gesamtverbrauch lag bei Konfiguration A bei 945 mW, bei Konfiguration B bei 356 mW.

Auf Abbildung 6.4 sowie Abbildung 6.5 ist der Gesamtverbrauch graphisch ersichtlich:

Szenario 1 - Konfiguration A

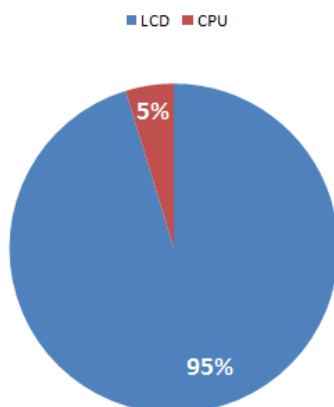


Abbildung 6.4: Gesamtverbrauch Konfiguration A bei PowerTutor (eigene Darstellung)

Szenario 1 - Konfiguration B

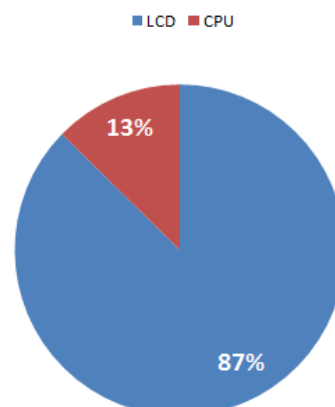


Abbildung 6.5: Gesamtverbrauch Konfiguration B bei PowerTutor (eigene Darstellung)

6.2.4 Ergebnisse – AT&T ARO

Der gesamte Energieverbrauch bei Konfiguration A lag bei 521.02 Joule, derjenige für Konfiguration B bei 369.99 Joule. Um die Werte, welche beim AT&T ARO Tool gemessen worden sind, von Joule (E_{Joule}) in mW ($E_{Milliwatt}$) umzurechnen, wurde folgende Formel verwendet: $E_{Milliwatt} = \frac{E_{Joule} * 1000}{t} \approx E_{Joule} * 1.\bar{6}$. Die Zeit t war dabei nicht genau zehn Minuten (600 Sekunden), also wurde der Faktor $1.\bar{6}$ für die Umrechnung verwendet.

Dies resultierte in 864.05 mW Energieverbrauch für Konfiguration A und 613.58 mW für Konfiguration B. Die Unterschiede bei den Messungen mit AT&T ARO und PowerTutor bei Konfiguration B kommen daher, dass es sich beim AT&T ARO um eine Simulation der Energieeffizienz handelt. Für jede Komponente wird ein fixer Wert für den Energieverbrauch verwendet (siehe Abbildung 6.6). Diese Werte werden dann mit der totalen Laufzeit der Messung multipliziert.

Device Attribute	Profile Value
Device Name	Default WiFi
Average power Wi-Fi connected (w)	0.403
Average power Wi-Fi inactive (w)	0.02
Average power for active GPS (w)	0.28
Average power for standby GPS (w)	0.02
Average power when camera is on (w)	0.95
Average power for active Bluetooth (w)	0.761
Average power for standby Bluetooth (w)	0.02
Average power when screen is on (w)	0.58

Abbildung 6.6: Standardwerte für den Energieverbrauch beim AT&T ARO (eigene Darstellung)

Für die Simulation mit dem AT&T ARO musste das WLAN-Interface auf dem Smartphone eingeschaltet sein, da dieses Tool eine VPN-Verbindung für die Messungen verwendet. Auf Abbildung 6.7 sowie Abbildung 6.8 ist der Gesamtverbrauch graphisch ersichtlich:

Szenario 1: Konfiguration A

■ Total RRC Energy ■ Total GPS Energy ■ Total Screen Energy

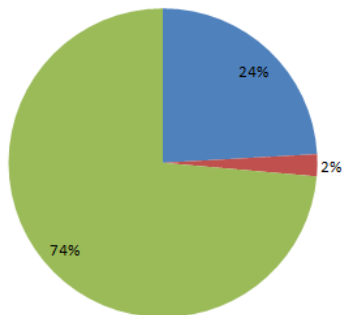


Abbildung 6.7: Gesamtverbrauch Konfiguration A beim AT&T ARO (eigene Darstellung)

Szenario 1: Konfiguration B

■ Total Wi-Fi Energy ■ Total Screen Energy

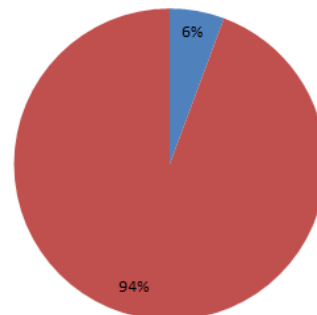


Abbildung 6.8: Gesamtverbrauch Konfiguration B beim AT&T ARO (eigene Darstellung)

Der Verbrauch des Wi-Fis in Abbildung 6.8 lag dabei bei 21 Joule (35 mW).

6.2.5 Zusammenfassung der Ergebnisse

In diesem Szenario wurde der Energieverbrauch eines Testgeräts gemessen und anschliessend die totale Laufzeit geschätzt. Die Ergebnisse haben gezeigt, dass die neuere Android-Version 5.0.1 energieeffizienter war als die Version 4.4.2. Zudem wurde gezeigt, dass der Bildschirm den grössten Teil der Energie verbraucht hat.

6.3 Szenario 2: Packet coalescing

Mithilfe dieses Szenarios wird die Technik des *packet coalescing* simuliert. In der in Kapitel 6.1 ersichtlichen Testumgebung wurde der PC mit Windows 8.1 Pro für die Entwicklung der App und die virtuelle Maschine mit Ubuntu 14.04 für die Simulation des Servers verwendet. Der Laptop wurde für die Messungen und Dokumentation verwendet und das Smartphone für das Testen der App und somit für das Senden der Datenpakete. Zur Simulation des packet coalescing wurde eine ähnliche App wie diejenige im Abschlussbericht der Air Navigation Platform (ANP) – Arbeitspaket 6: Energieverbrauch ("Final Report: ANP: Air Navigation Platform – WP 6 Energy Consumption") entwickelt (Schumann & Tamarcaz, 2015). Auf Abbildung 6.9 ist der Pseudocode der in diesem Abschlussbericht erwähnten App ersichtlich:

```

1  function readFile(filePath)           // first, function to read the CSV file
2  {
3      openFile();                       // open the file
4      for(i < nbrLines)                 // loop through the file
5      {
6          readTheLine();                // read the line of the file
7          putDataInArray();             // put size of data and time to wait in an array
8      }
9  }
10
11 function sendData(arrayOfData)        // function to send data to the server
12 {
13     openSocketConnection();           // opening the connection to the server
14     for(arrayOfData != empty)         // loop through the array of data
15     {
16         sendData(sizeData);           // send random data of the defined size
17         wait(time);                   // wait defined seconds before sending next data
18     }
19     closeSocketConnection();          // close the connection to the server
20 }

```

Abbildung 6.9: Pseudocode der Simulation von packet coalescing (nach Schumann & Tamarcaz, 2015)

Der Quellcode dieser App wurde dem Verfasser zur Verfügung gestellt. Anschliessend wurde eine App mit einer ähnlichen Funktionsweise für Android entwickelt. Der Unterschied besteht darin, dass für jedes Datenpaket eine Socket-Verbindung hergestellt wird und diese danach direkt wieder geschlossen wird, sobald das Datenpaket übertragen wurde. Der Grund hierfür ist dass im obigen Pseudocode nur eine Verbindung hergestellt wird. Anschliessend werden alle Datenpakete gesendet. Dies entspricht nicht den Voraussetzungen dieses Szenarios. Der Pseudocode der entwickelten App ist in Abbildung 6.10 ersichtlich:

```

1  onCreate() {
2      readFile();
3
4      if(action == "sendData") {
5          sendData(); // when the user taps on the button "Send Data"
6      } else {
7          reset(); // when the user taps on the button "Send Again"
8      }
9  }
10
11 readFile() {
12     openFile(); // open the file with the data
13
14     for(i < numberOfLines) { // loop through file
15         splitLine(); // split by comma (,)
16         addDataToLists(); // add data sizes and time-outs to corresponding list
17     }
18 }
19
20 sendData() {
21     convertListsToArrays();
22
23     for(i < arrayLength) { // loop through the array of data
24         openSocketConnection(); // open the connection to the server in the test environment
25         sendToServer(); // send data of defined size
26         closeSocketConnection(); // close the connection to the server
27         wait(time); // wait for number of seconds defined in the scenario file
28     }
29
30 }
31 }

```

Abbildung 6.10: Pseudocode der eigens entwickelten App (eigene Darstellung)

Für jedes Datenpaket wird eine separate Verbindung aufgebaut (Zeile 24). Anschliessend erfolgt das Senden des Datenpakets (Zeile 25) und das Schliessen der Verbindung (Zeile 26). Letztlich erfolgt der Time-Out auf Zeile 27, bevor das nächste Datenpaket gesendet wird.

Auf dem Server läuft das Programm netcat, welches u.a. das Horchen (engl. listen) auf einem bestimmten Port erlaubt. Der Befehl, welches für dieses Szenario verwendet wurde, lautet: `netcat -klp 1234`. Die verwendeten Parameter haben dabei folgende Bedeutung:

- `-k`: Zwingt netcat dazu, auf weitere Verbindungen zu warten, nachdem die aktuelle beendet wurde. Diese Option muss mit `-l` verwendet werden
- `-l`: Diese Option wird verwendet, um zu spezifizieren, dass netcat auf eingehende Verbindungen warten soll
- `-p`: Gibt den Port der Quelle (in diesem Fall 1234) an, welchen netcat verwenden soll (Berkeley Software Distribution (BSD), 2015)

Beim Start der App ist folgende in Abbildung 6.11 ersichtliche Benutzeroberfläche sichtbar:

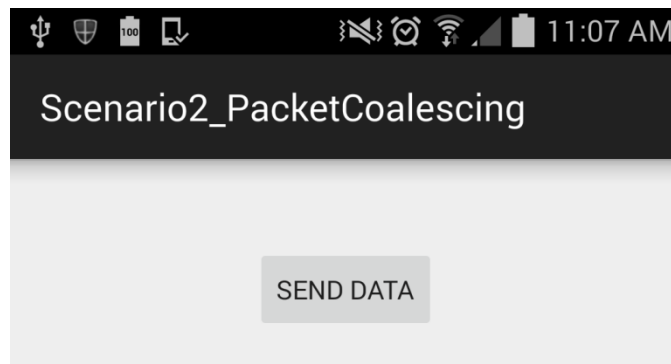


Abbildung 6.11: Benutzeroberfläche beim Start der für Szenario 2 entwickelten App (eigene Darstellung)

Wenn der Benutzer auf den Button "Send Data" drückt, werden verschieden grosse Datenpakete in definierten Zeitabständen an den Server in der Testumgebung gesendet. Details zur Implementierung dieser App sind in Anhang II: Implementierung der App zur Simulation von Packet coalescing zu finden. Zur Definition der Grösse der Datenpakete sowie der zeitlichen Abstände wird eine CSV-Datei verwendet, wie in Abbildung 6.12 ersichtlich:

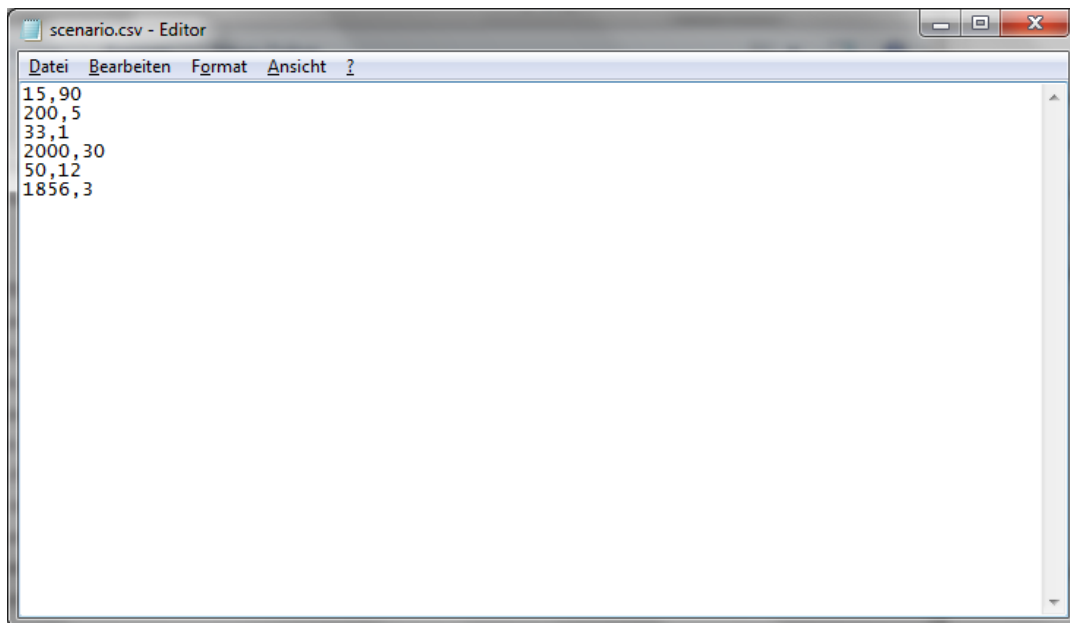


Abbildung 6.12: Beispiel einer CSV-Datei für Szenario 2 (eigene Darstellung)

Diese Datei hat folgendes Format: **x,y**. Dabei steht **x** für die Grösse des Datenpakets in Bytes und **y** für die Anzahl Sekunden bevor das nächste Datenpaket gesendet wird. Um den Unterschied zwischen dem regelmässigen Senden von Daten (kein packet coalescing) und dem Senden von Datenpaketen in gewissen zeitlichen Abständen (mit packet coalescing) zu simulieren, wurden zwei verschiedene CSV-Dateien verwendet, wie in Abbildung 6.13 ersichtlich:

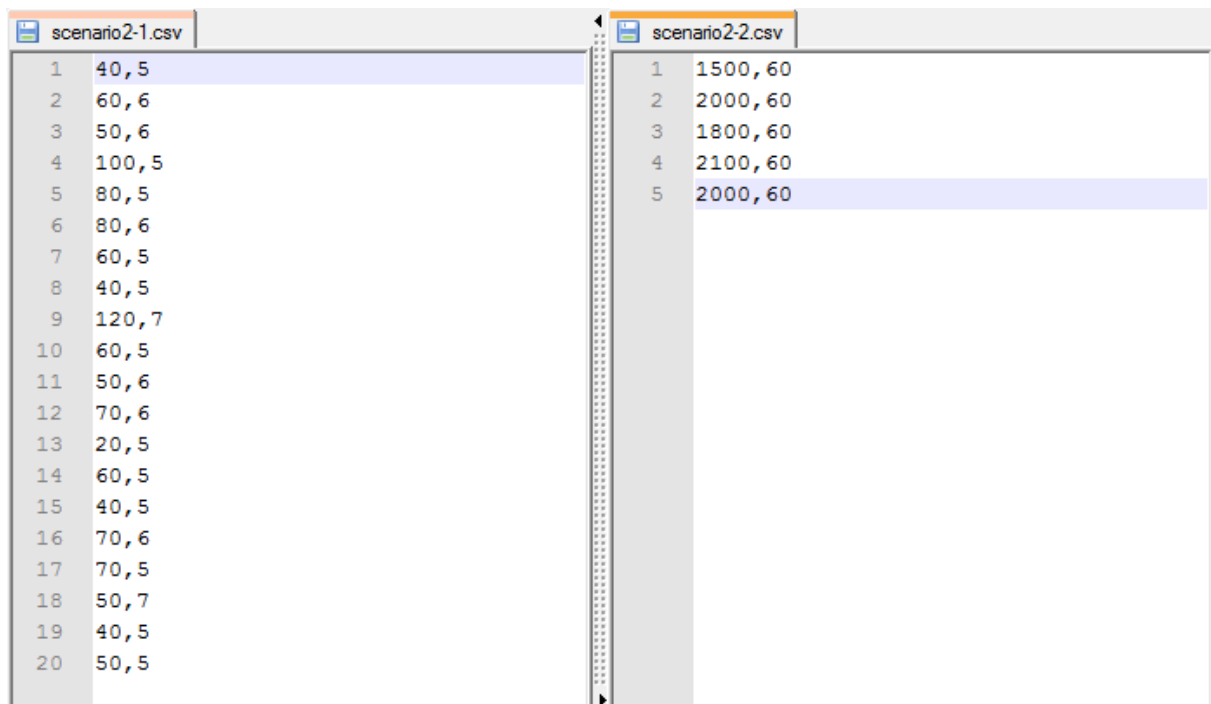


Abbildung 6.13: CSV-Dateien zur Simulation von Szenario 2 (eigene Darstellung)

Die Summe der Bytes der Datei "scenario2-1.csv" (links) beträgt 1'210 Bytes, die Summe der Datei "scenario2-2.csv" (rechts) 9'400 Bytes.

6.3.1 Ergebnisse – AT&T ARO

Für dieses Szenario wurde lediglich das Tool AT&T ARO verwendet, da es mit PowerTutor nicht möglich ist, den Energieverbrauch des Wi-Fis zu messen. Die Werte beim AT&T ARO betreffend den Energieverbrauch des Wi-Fis sind jedoch mit Vorsicht zu geniessen, da es sich dabei lediglich um Schätzungen handelt. Diese geschätzten sowie die weiteren gemessenen Werte wurden anschliessend in Excel erfasst und ausgewertet.

In der nachfolgenden Tabelle 6.2 sind die Ergebnisse dieses Szenarios ersichtlich:

	Packet coalescing	No packet coalescing
Gesamtverbrauch Wi-Fi (in J)	48.03	50.19
Anzahl TCP-Sessions	25	200
Gesamtanzahl Bytes	59'204	88'360
Anzahl IP-Pakete	227	1'405
Energieeffizienz (in J/KB)	0.81	0.57

Tabelle 6.2: Ergebnisse Szenario 2 (eigene Darstellung)

Der Gesamtverbrauch des Wi-Fis wurde hierbei vom AT&T ARO gemessen. Der Ursprung der weiteren Daten sind im Anhang IV: Beschreibung der erfassten Daten zu finden. In den nachfolgenden zwei Diagrammen (Abbildung 6.14 und Abbildung 6.15) sind der Gesamtverbrauch des Wi-Fis sowie die Energieeffizienz graphisch dargestellt:

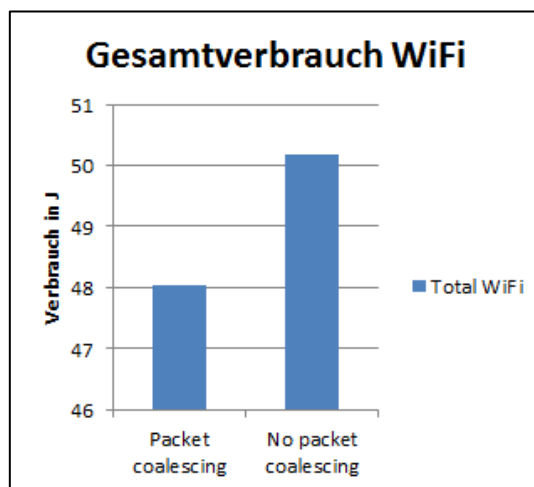


Abbildung 6.14: Gemessene Werte für den Gesamtverbrauch des Wi-Fi (eigene Darstellung)

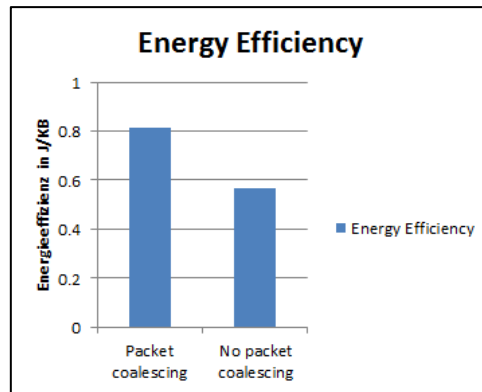


Abbildung 6.15: Energieeffizienz der beiden Methoden (eigene Darstellung)

Der Gesamtanzahl an übertragenen Bytes ist beim direkten Senden der Daten um 50% höher als beim Einsatz von packet coalescing (siehe Tabelle 6.2). Der Gesamtverbrauch des Wi-Fis ist beim direkten Senden der Daten (no packet coalescing) um etwa 4% höher. Bei der Energieeffizienz in Abbildung 6.15 sind grössere Werte als besser zu interpretieren. Zur Berechnung wurde der Energieverbrauch des Wi-Fis durch die Anzahl der übertragenen Kilobytes dividiert. Die Technik des packet coalescings ist dabei um 42% effizienter.

6.4 Szenario 3: Lokale vs. entfernte Operationen

Mit diesem Szenario soll getestet werden, wie sich das Auslagern von rechenintensiven Operationen auf einen Webdienst auf die Akkuleistung auswirkt. In der in Kapitel 6.1 ersichtlichen Testumgebung wurde der PC mit Windows 8.1 Pro für die Entwicklung der App und die virtuelle Maschine mit Ubuntu 14.04 für die Implementierung und das Ausführen des Webdienstes verwendet. Der Laptop wurde für die Messungen und Dokumentation verwendet und das Smartphone für die Konsumation des Webdienstes.

Wie bereits in Kapitel 5.5 erwähnt wurde, wird für dieses Szenario die Berechnung der Fibonacci-Folge gemessen. Dazu wurden ein Webdienst sowie eine App für Android zum Aufrufen dieses Webdienstes entwickelt. Der Webdienst ist unter folgender URL in der Testumgebung aufrufbar: <http://192.168.1.108:8080/Fibonacci/fib/fib/x>. Bei x handelt es sich hierbei um eine positive natürliche Zahl, welche die Anzahl Zahlen der Fibonacci repräsentiert. Ein Aufruf des Webdienstes mit $x = 10$ liefert folgendes Ergebnis zurück: 1 1 2 3 5 8 13 21 34 55; es handelt sich hierbei um die ersten 10 Zahlen der Fibonacci-Reihe. Details zur Implementierung dieses Szenarios (Webdienst und App) sind im Anhang III: Implementierung des Webdienstes zur Berechnung der Fibonacci-Folge zu finden.

6.4.1 Ergebnisse – Ausführungszeit (Android 4.4.2)

Die App erlaubt das Berechnen von 10, 20, 30, 40 oder 50 Zahlen der Fibonacci-Reihe, jeweils entweder lokal oder via Webdienst (remote). Nach der Berechnung wird in der App auch die Zeit angezeigt, die benötigt wurde, um die Berechnung durchzuführen. Dieser Wert ist zwar aus verbrauchstechnischer Sicht irrelevant, gibt jedoch Auskunft über die Performance der beiden Berechnungsmethoden. Für die Sequenzlänge 10 bis 40 wurden lokal sowie entfernt jeweils 15 Messungen durchgeführt und der Durchschnitt berechnet. Für 50 Zahlen der Fibonacci-Reihe wurde lokal eine Messung und entfernt fünf Messungen durchgeführt. Die Ergebnisse der Zeitmessung sind in Abbildung 6.16 ersichtlich:

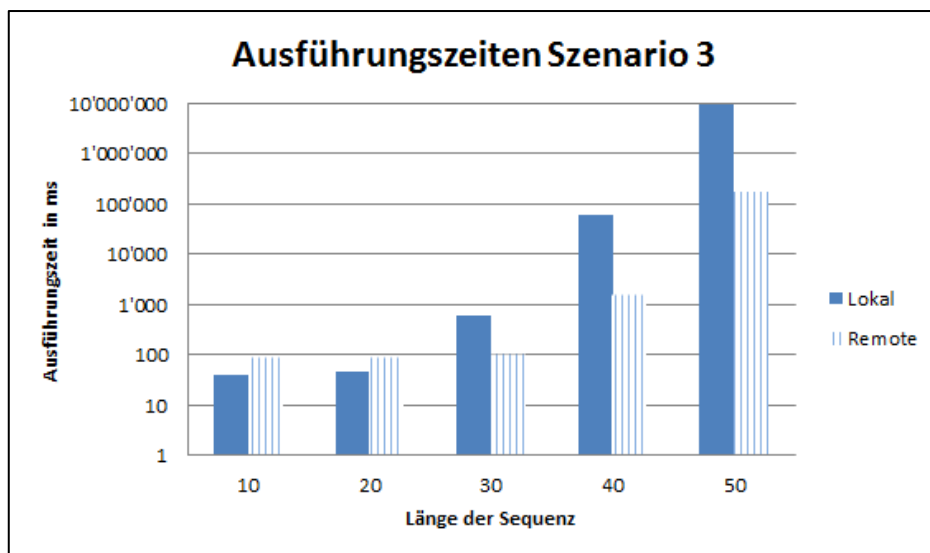


Abbildung 6.16: Ausführungszeiten beim Berechnen der Fibonacci-Folge (eigene Darstellung)

Bei einer Sequenzlänge von 50 dauert die Berechnung lokal circa 2,5 Stunden und via Webdienst etwa 3 Minuten. Ebenfalls erkennbar ist, dass die Berechnungen ab einer gewissen Komplexität (zwischen 20 und 30 Stellen) auf dem Smartphone um ein vielfaches länger dauern als via Webdienst.

6.4.2 Ergebnisse – Ausführungszeit (Android 5.0.1)

Nach der Installation von Android 5.0.1 auf dem Testgerät wurden die Messungen erneut durchgeführt. Erste Testmessungen haben gezeigt, dass die Berechnungen bei Android 5.0.1 um einiges länger dauern als bei der vorherigen Android-Version 4.4.2. Aus diesem Grund wurden zwei Testreihen durchgeführt; bei der ersten war der Energiesparmodus (*Power saving mode*; PSM) aktiviert, bei der zweiten war dieser deaktiviert.

Für die Sequenzlängen 10 bis 30 wurden 15 Messungen, für 40 Stellen wurden 10 Messungen durchgeführt. Folgende durchschnittlichen Ausführungszeiten wurden dabei gemessen (in Millisekunden):

Anzahl Stellen	Mit Power Saving Mode	Ohne Power Saving Mode
10	96.67	93.93
Standardabweichung	11.61	9.79
20	137.13	123.33
Standardabweichung	18	19.29
30	8'167.67	6'060
Standardabweichung	96.49	103.36
40	979'339	883'672.9
Standardabweichung	3'077.77	18'056.42

Tabelle 6.3: Ausführungszeiten bei Android 5.0.1 (eigene Darstellung)

Eine lokale Messung mit 50 Stellen der Fibonacci-Reihe wurde aus Zeitgründen nicht durchgeführt; eine gestartete Berechnung lieferte nach 10 Stunden Laufzeit kein Resultat. Wie in Tabelle 6.3 ersichtlich ist, beträgt der Unterschied bei 30 Stellen etwa zwei Sekunden, bei 40 Stellen anderthalb Minuten.

Beim Vergleich dieser Werte mit den Messungen bei Android 4.4.2 lässt sich ein markanter Unterschied feststellen, wie auf Abbildung 6.17 ersichtlich:

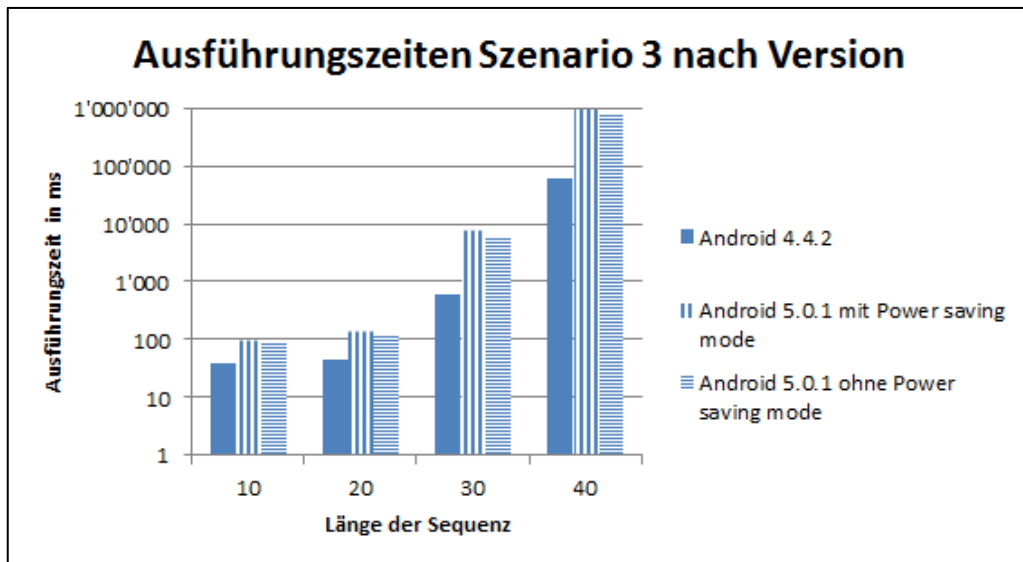


Abbildung 6.17: Vergleich der Ausführungszeiten nach Android-Version (eigene Darstellung)

Die Berechnung der Fibonacci-Reihe dauerte bei Android 5.0.1 ohne PSM bis zu 15-mal länger als bei Android 4.4.2. In der obigen Abbildung ist zu sehen, dass die Berechnung

unabhängig der Sequenzlänge bei Android 4.4.2 schneller durchgeführt wird als bei Android 5.0.1.

6.4.3 Ergebnisse – PowerTutor (Android 4.4.2)

Um den Energieverbrauch zu messen, wurde das Tool PowerTutor verwendet. Es wurde zunächst **lokal** sechs Mal 40 Stellen berechnet, aus den Ergebnissen wurde dann der Mittelwert berechnet. Danach wurde einmal 50 Stellen berechnet. Dieselbe Konfiguration wie bei Szenario 1 – Konfiguration wurde verwendet, d.h.:

- WLAN: aus
- GPS: aus
- Helligkeit: 0%
- Energiesparmodus: an
- Mobile Daten: aus
- Bluetooth: aus

Anschliessend wurden die von PowerTutor erstellten Logdateien in Excel ausgewertet. Die folgenden in Tabelle 6.4 ersichtlichen Resultate wurden dabei beobachtet:

Anzahl Stellen	40	50
Durchschnittlicher Verbrauch CPU (in mW)	585.8	606.5
Standardabweichung	33.6	-
Gesamtverbrauch (in J)	63.6	6'697.4
Standardabweichung	3.1	-
Ausführungszeit (in s)	71	7'299.5
Standardabweichung	3.9	-

Tabelle 6.4: Gemessene Werte bei Szenario 3 – Lokal (Android-Version 4.4.2; eigene Darstellung)

Trotz der Tatsache, dass sechs Mal 40 Stellen der Fibonacci-Reihe berechnet wurden, ist der durchschnittliche Verbrauch nur geringfügig kleiner als bei der einmaligen Berechnung von 50 Stellen (siehe Abbildung 6.18). Der Unterschied beträgt 20.7 mW oder 3.4%. Der Gesamtverbrauch verläuft etwa proportional zur Laufzeit; die Berechnung von 50 Stellen verbraucht circa 105 Mal so viel Energie wie 40 Stellen und dauert etwa 103 Mal länger (siehe Abbildung 6.19).

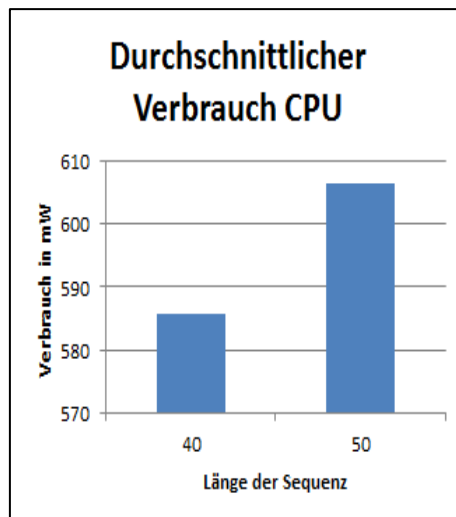


Abbildung 6.18: Durchschnittlicher Verbrauch bei lokaler Berechnung (eigene Darstellung)

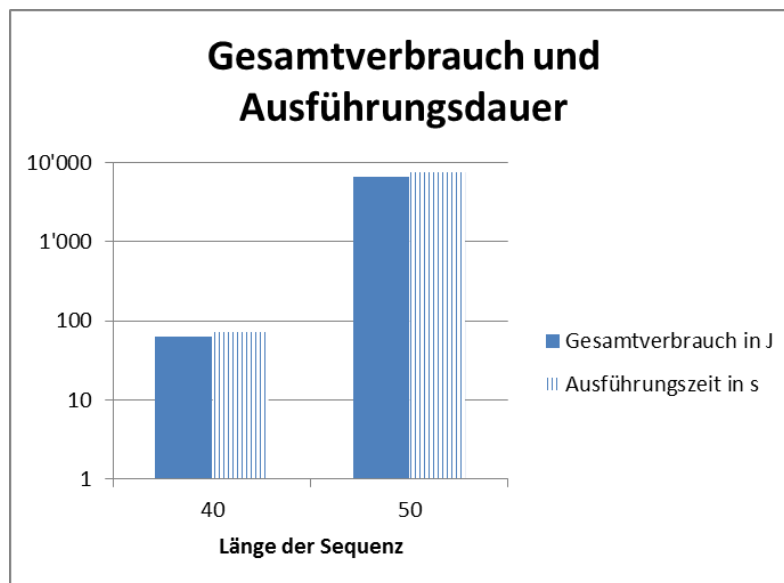


Abbildung 6.19: Gesamtverbrauch und Ausführungsdauer bei lokaler Berechnung (eigene Darstellung)

Bei der **entfernten** Berechnung wurde dieselbe Konfiguration wie bei der lokalen verwendet, mit einem Unterschied: das WLAN-Interface war eingeschaltet, da das Smartphone ansonsten nicht auf den Webdienst in der Testumgebung zugreifen kann. Zur Messung wurden zwei Blöcke mit je sechs Messungen für 40 bzw. 50 Stellen durchgeführt. Aus diesen zwei Blöcken wurde dann mithilfe der erstellten Logdateien sowie Excel jeweils das arithmetische Mittel berechnet. Die folgenden in Tabelle 6.5 ersichtlichen Resultate wurden dabei berechnet:

Anzahl Stellen	40	50
Durchschnittlicher Verbrauch CPU (in mW)	239.1	151.1
Standardabweichung	68.1	45.7

Anzahl Stellen	40	50
Gesamtverbrauch (in J)	5.92	565.5
Standardabweichung	0.9	71
Ausführungszeit (in s)	10.73	1'220.3
Standardabweichung	0.27	32.8

Tabelle 6.5: Gemessene Werte bei Szenario 3 – Remote (Android-Version 4.4.2; eigene Darstellung)

Diese Resultate sind auf Abbildung 6.20 sowie Abbildung 6.21 graphisch dargestellt:

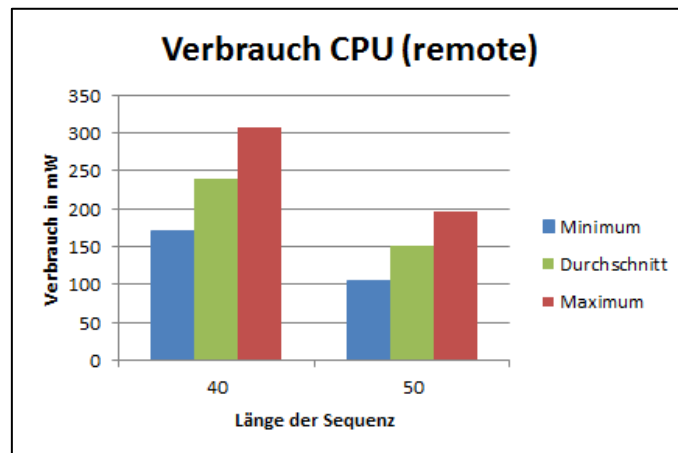


Abbildung 6.20: Energieverbrauch des Prozessors bei entfernter Berechnung (eigene Darstellung)

Der durchschnittliche Verbrauch des Prozessors war bei 40 Stellen um 58% höher als bei 50 Stellen.

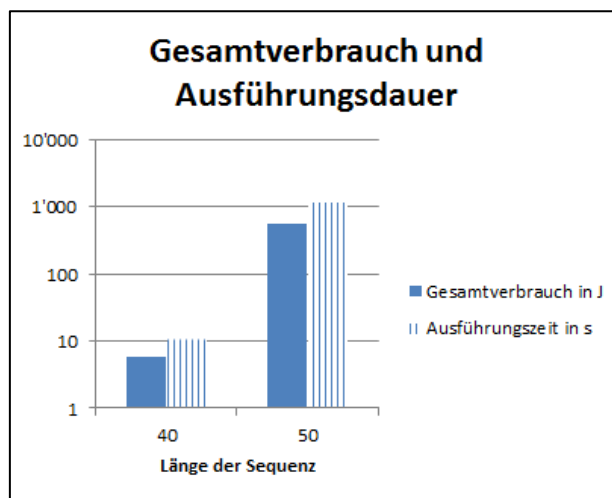


Abbildung 6.21: Gesamtverbrauch und Ausführungsdauer bei entfernter Berechnung (eigene Darstellung)

Auch hier lässt sich wieder ein linearer Zusammenhang zwischen dem Gesamtverbrauch und der Ausführungszeit feststellen. Der Gesamtverbrauch war bei 50 Stellen etwa 96 Mal höher als bei 40 Stellen und die Berechnung von 50 Stellen dauerte etwa 114 Mal länger als 40 Stellen. Zur Berechnung der Ausführungszeit wurde in der App ein Zeitmesser implementiert (siehe Anhang III: Implementierung des Webdienstes zur Berechnung der Fibonacci-Folge). Die gemessenen Werte wurden darauffolgend in Excel übernommen, danach wurden die in Abbildung 6.21 ersichtlichen Mittelwerte berechnet.

6.4.4 Ergebnisse – PowerTutor (Android 5.0.1)

Um die Leistung zwischen Android 4.4.2 und 5.0.1 zu vergleichen, wurden dieselben Messungen wie in Kapitel 6.4.3 durchgeführt. In der App wurden lokal sechs Mal 40 Stellen der Fibonacci-Reihe berechnet und aus den gemessenen Werten wurde der Mittelwert ermittelt.

Anschliessend wurden die von PowerTutor erstellten Logdateien in Excel ausgewertet. Die folgenden in Tabelle 6.6 ersichtlichen Resultate wurden dabei beobachtet:

Anzahl Stellen	40 (mit PSM)	40 (ohne PSM)
Durchschnittlicher Verbrauch CPU (in mW)	655.16	727.52
Standardabweichung	6.17	29.54
Gesamtverbrauch (in J)	954.63	890.5
Standardabweichung	6.03	23.31
Ausführungszeit (in s)	988.08	858.67
Standardabweichung	6.24	43.33

Tabelle 6.6: Gemessene Werte bei Szenario 3 – Lokal (Android-Version 5.0.1; eigene Darstellung)

Der durchschnittliche Verbrauch ist zwar mit PSM niedriger als ohne, jedoch ist der Gesamtverbrauch bei aktiviertem PSM höher. Dies ist auf die erhöhte Laufzeit zurückzuführen.

6.4.5 Ergebnisse – AT&T ARO

Dieses Tool wurde verwendet, um das Kommunikationsverhalten der App zu untersuchen. Es bietet detaillierte Angaben zu TCP-Sessions, Paketgrössen, übertragene

Bytes usw. Für diese Messungen wurde zunächst 40, danach 50 Stellen sechs Mal entfernt berechnet. Nachfolgend die Ergebnisse für 40 bzw. 50 Stellen (Abbildung 6.22, Abbildung 6.23, Abbildung 6.24 und Abbildung 6.25):

40 Stellen

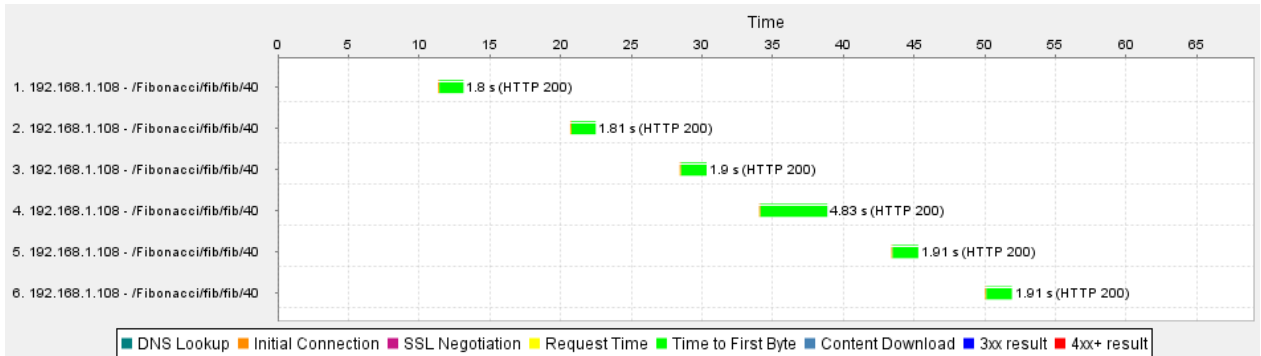


Abbildung 6.22: Wasserfallmodell der Verbindungen bei 40 Stellen (eigene Darstellung)

Domain TCP Sessions						
Time	Remote IP Address	Local Port	Sessio...	Bytes Transmitted	Session Cl...	Closed By
11.335	192.168.1.108	48775	21.9	980	20.003	Server
20.682	192.168.1.108	48779	21.9	1'084	20.084	Server
28.431	192.168.1.108	48781	21.9	980	19.885	Server
34.043	192.168.1.108	48783	4.9	992		
43.391	192.168.1.108	48785	2.0	888		
50.020	192.168.1.108	48788	1.9	992		

Abbildung 6.23: TCP-Sessions bei 40 Stellen (eigene Darstellung)

50 Stellen

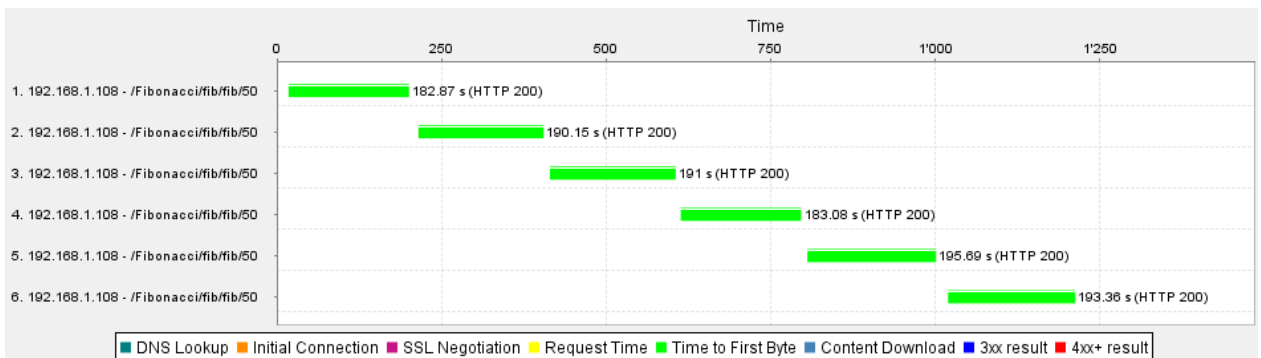


Abbildung 6.24: Wasserfallmodell der Verbindungen bei 50 Stellen (eigene Darstellung)

Domain TCP Sessions						
Time	Remote IP Address	Local Port	Sessio...	Bytes Transmitted	Session Cl...	Closed By
17.881	192.168.1.108	52985	202.9	1'088	19.883	Server
215.001	192.168.1.108	52992	210.3	1'088	20.033	Server
414.770	192.168.1.108	52994	211.1	1'088	20.143	Server
613.391	192.168.1.108	52998	203.1	1'088	19.873	Server
805.624	192.168.1.108	53000	215.8	1'192	19.967	Server
1019.373	192.168.1.108	53004	213.4	1'088	19.956	Server

Abbildung 6.25: TCP-Sessions bei 50 Stellen (eigene Darstellung)

Aus den Wasserfallmodellen (Abbildung 6.22 und Abbildung 6.24) ist ersichtlich, dass die Session jedes Mal solange dauert, bis das Ergebnis auf dem Smartphone angezeigt wird und dann beendet wird. Aus den TCP-Session-Tabellen (Abbildung 6.23 und Abbildung 6.25) ist erkennbar, dass die übertragenen Daten bei 40 Stellen durchschnittlich 986 Bytes und bei 50 Stellen 1'105.33 Bytes betragen. Die unterschiedlichen Werte kommen daher, dass pro Session eine unterschiedliche Anzahl an Datenpaketen gesendet wurde.

6.4.6 Zusammenfassung der Ergebnisse

Nachfolgend eine Zusammenstellung der Werte der lokalen sowie der entfernten Berechnung bei Android 4.4.2 graphisch dargestellt (Abbildung 6.26, Abbildung 6.27 sowie Abbildung 6.28):

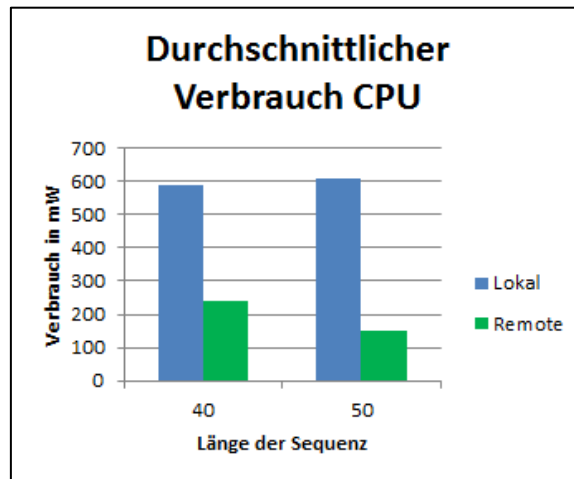


Abbildung 6.26: Vergleich durchschnittlicher Verbrauch (eigene Darstellung)

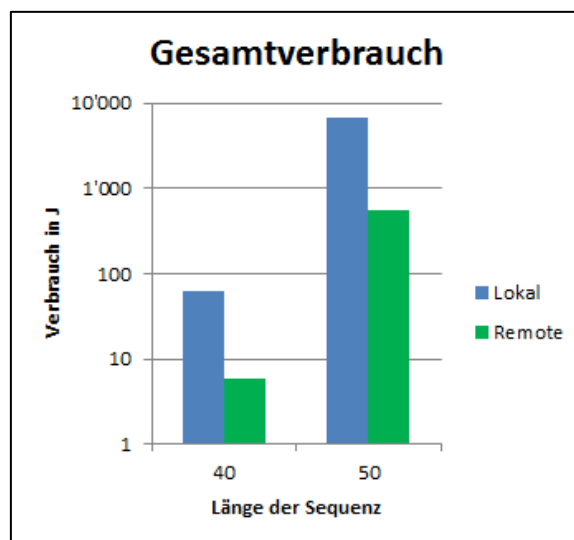


Abbildung 6.27: Vergleich Gesamtverbrauch (eigene Darstellung)

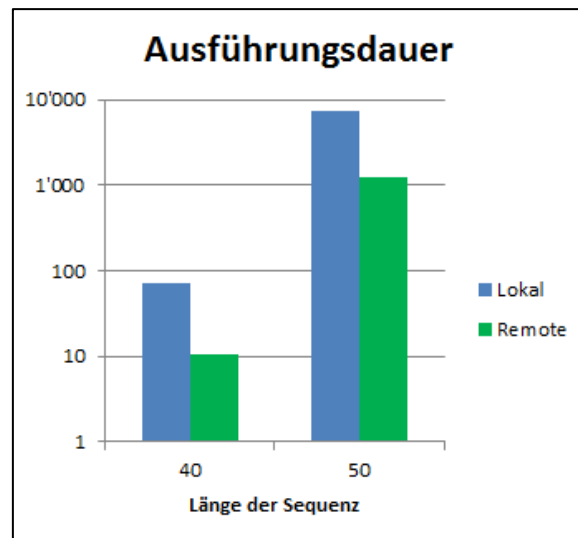


Abbildung 6.28: Vergleich Ausführungsdauer (eigene Darstellung)

Aus diesen Vergleichen ist ersichtlich, dass rechenintensive Operationen wenn möglich auf Webdienste ausgelagert werden sollten. Der durchschnittliche Verbrauch war lokal bei 40 Stellen 2,5 Mal, bei 50 Stellen 4 Mal so hoch wie bei der entfernten Berechnung. Beim Gesamtverbrauch sind die Unterschiede um einiges grösser; bei 40 Stellen war der Verbrauch lokal 11 Mal, bei 50 Stellen 12 Mal so hoch im Gegensatz zur Berechnung via Webdienst. Letztlich war die durchschnittliche Ausführungsdauer bei der lokalen Berechnung mit 40 Stellen 6,5 Mal, mit 50 Stellen 6 Mal länger als bei der entfernten Berechnung.

Beim Vergleich der beiden Android-Versionen (4.4.2 und 5.0.1) wurden folgende Unterschiede beobachtet:

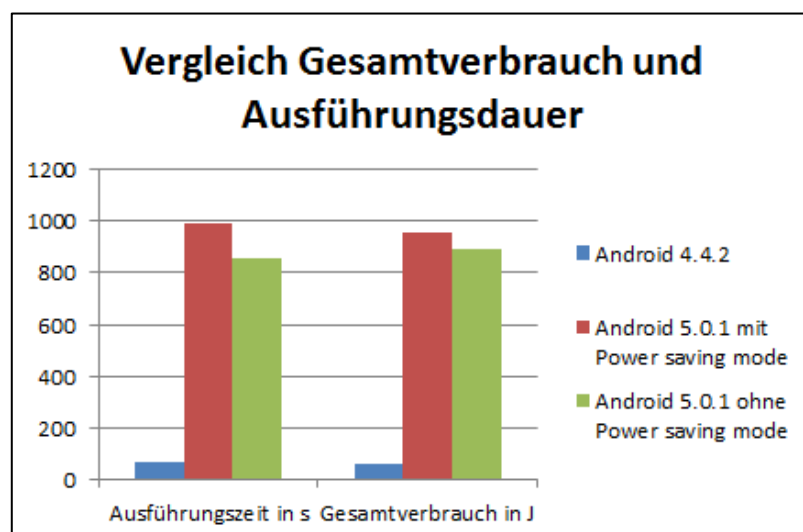


Abbildung 6.29: Vergleich Gesamtverbrauch und Ausführungsdauer bei der Berechnung von 40 Stellen (eigene Darstellung)

Auf Abbildung 6.29 ist ersichtlich, dass die Leistung von Android 5.0.1 auch ohne PSM deutlich unter derjenigen von Android 4.4.2 liegt. Die durchschnittliche Ausführungszeit war bei Android 5.0.1 ohne PSM 12-mal höher im Vergleich mit Android 4.4.2, der Gesamtverbrauch war 14-mal höher.

Beim Akkuverlauf wurden ebenfalls Unterschiede beobachtet, wie auf Abbildung 6.30 ersichtlich:

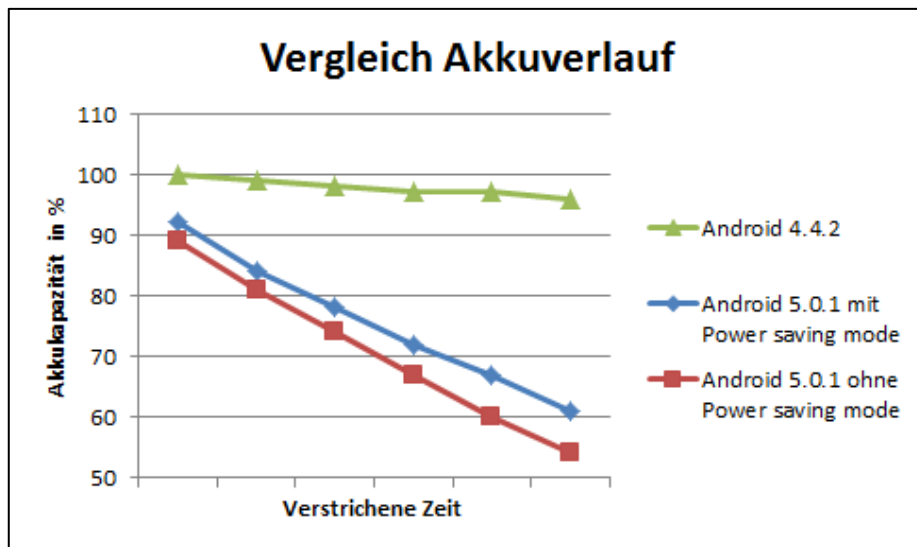


Abbildung 6.30: Vergleich des Akkuverlaufs der Android-Versionen (eigene Darstellung)

Bei Android 4.4.2 war die erste Berechnung nach 74 Sekunden abgeschlossen und die Akkukapazität lag bei 100%. Im Gegensatz dazu war die erste Berechnung bei Android 5.0.1 mit PSM nach 998, ohne PSM nach 763 Sekunden abgeschlossen. Die verbleibende Akkukapazität nach dieser ersten Berechnung lag danach bei 92% (mit PSM) bzw. 89% (ohne PSM).

7 Beurteilung

In diesem Kapitel werden die Ergebnisse der drei Szenarien-Analysen zusammengefasst und entsprechende Empfehlungen vorgelegt. Zunächst werden die beiden verwendeten Tools bewertet:

PowerTutor eignet sich zur Messung des Energieverbrauchs des Prozessors und des Bildschirms, sowie weitere Komponenten wie GPS und Lautsprecher. Detaillierte Logs, welche den Verbrauch für jede Sekunde beinhalten, waren für diese Arbeit sehr nützlich und

zählen zu den Vorteilen dieses Tools. Der grösste Nachteil ist, dass der Energieverbrauch des Wi-Fis sowie des 3G/4G-Interfaces nicht gemessen wurde. Somit eignet sich dieses Tool für Apps, welche nicht sehr kommunikationsintensiv sind.

Das zweite Tool, **AT&T ARO**, war bezüglich des Energieverbrauchs nur bedingt nützlich, da es sich lediglich um Schätzungen handelt. Der positive Aspekt dieses Tools sind die detaillierten Angaben bezüglich des Netzwerkes. Jeder Trace enthält eine CAP-Datei, welche beispielsweise mit Wireshark⁶ geöffnet werden kann, wie auf Abbildung 7.1 ersichtlich:

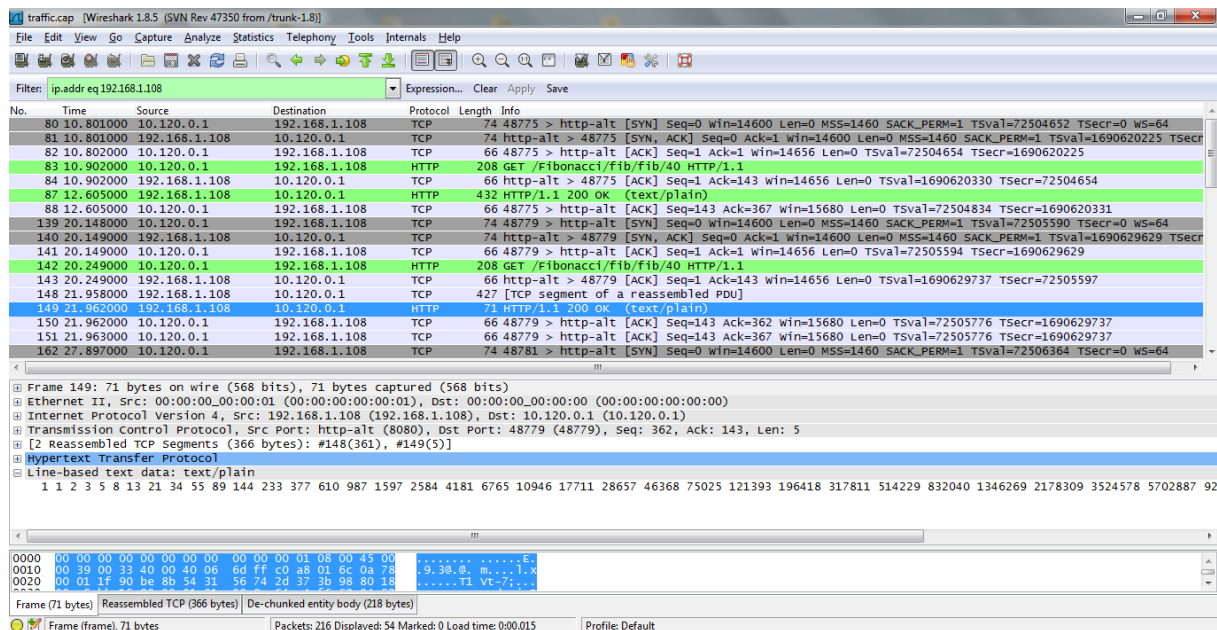


Abbildung 7.1: Traffic beim Ausführen einer entfernten Berechnung (Szenario 3) in Wireshark (eigene Darstellung)

Auf dieser Abbildung sind die Datenpakete ersichtlich, welche zwischen der virtuellen Maschine und dem Smartphone ausgetauscht werden. Mit dem Filter "ip.addr == 192.168.1.108" lassen sich alle Pakete anzeigen, welche von dieser IP-Adresse stammen (Source) oder an diese IP-Adresse gerichtet sind (Destination). Der Eintrag mit der Nummer 142 ist der HTTP-Request an den Server (408 Stellen), die Antwort (HTTP-Response) erfolgt im Eintrag mit der Nummer 149 (blau markiert). Der Nachteil des AT&T ARO ist dass es sich beim Energieverbrauch der restlichen Komponenten lediglich um Schätzungen handelt. Somit eignet sich dieses Tool für detaillierte Analysen des Traffics.

⁶ Bei Wireshark handelt es sich um ein Netzwerk-Protokoll-Analyse-Tool (Combs, unbekannt)

Mithilfe der durchgeführten Szenarien konnten folgende Erkenntnisse gewonnen werden:

Szenario 1 hat gezeigt, dass nicht verwendete Komponenten wie beispielsweise Bluetooth oder GPS deaktiviert sein sollten und die Helligkeit des Bildschirms so niedrig wie möglich eingestellt sein sollte, um die Akkuleistung des Smartphones zu maximieren. Der Entwickler sollte in seiner App Komponenten nur dann aktivieren, wenn diese nötig sind und danach wieder deaktivieren. Die Messungen haben gezeigt, dass der Akku in diesem Szenario länger hält, wenn Android 5.0.1 auf dem Smartphone installiert ist.

Die Ergebnisse von Szenario 2 zeigen, dass die lokale Gruppierung von Datenpaketen nur einen geringen Einfluss auf den Energieverbrauch hat, jedoch die Anzahl der übertragenen Bytes beim direkten Senden um 50% grösser ist. Die Anzahl der IP-Pakete ist beim direkten Senden 6.2 Mal grösser als bei der lokalen Gruppierung. Bei Apps mit einer hohen Kommunikationsintensität ist es durchaus sinnvoll, die Technik des packet coalescing einzusetzen, um das Kommunikationsvolumen möglichst gering zu halten. Jedoch ist zu beachten, dass die zwei Parameter (Grenzwert und Time-Out) sinnvoll gewählt werden, d.h. der Entwickler muss eine genaue Vorstellung der Grösse der Datenpakete haben. Bei einem zu kleinen Grenzwert werden die Pakete zu oft gesendet und verfehlt somit das Ziel des packet coalescing. Wenn der Grenzwert zu hoch angesetzt wird, erfolgt das Senden der Daten in zu grossen zeitlichen Abständen. Auch der zweite Parameter, der Time-Out, muss einen passenden Wert erhalten. Ein zu niedriger Wert verursacht wiederum ein zu häufiges Senden der Datenpakete, ein zu hoher Wert führt zu Verzögerung in der App. Es ist dem Entwickler überlassen, auch nur einen dieser Parameter zu verwenden.

Letztlich wurde beim Szenario 3 untersucht, wie sich das Auslagern von rechenintensiven Operationen auf einen Webdienst auf den Energieverbrauch auswirkt. Die durchgeführten Messungen haben gezeigt, dass der Gesamtverbrauch sowie die Ausführungsdauer ab einer gewissen Komplexität lokal um einiges höher waren als bei der entfernten Berechnung. Ein Beispiel für den Einsatz des Offloadings sind mathematische Apps, mit welchen sich komplexe Aufgaben (Gleichungssysteme als Beispiel) lösen lassen.

Schlussfolgerung

In dieser Arbeit wurde zunächst ein Überblick über den aktuellen Stand der Technologie bei Smartphones und den damit verbundenen Technologien wie beispielsweise Bluetooth oder Mobilfunkstandards gegeben. Anschliessend wurden fünf aktuelle Tools vorgestellt, um den Energieverbrauch von mobilen Applikationen zu messen. Zwei dieser Tools wurden folglich für den weiteren Verlauf dieser Arbeit verwendet. Anhand dieser Tools wurden nachfolgend Metriken zur Messung des Energieverbrauchs definiert. Der Fokus dieser Arbeit lag zwar auf Android-Applikation, jedoch lassen sich die in Kapitel 5 erwähnten Massnahmen auch bei anderen mobilen Betriebssystemen wie beispielsweise iOS von Apple oder Windows Phone von Microsoft anwenden. Drei dieser Massnahmen wurden folglich in Szenarien auf ihre Effizienz geprüft. Die Ergebnisse dieser Szenarien sowie Handlungsempfehlungen sind in Kapitel 7 zu finden. Konkret werden dort Empfehlungen im Umgang mit kommunikations- und rechenintensiven Apps gegeben.

Die Ergebnisse der Messungen bei Android 4.4.2 und 5.0.1 deuten darauf hin, dass die Leistung des Prozessors in Version 5.0.1 gedrosselt wurde, damit der Akku im Leerlauf länger hält (siehe Szenario 1). Dies hat jedoch Auswirkungen auf die Akkukapazität, wenn das Smartphone aktiv genutzt wird. Rechenintensive Anwendungen, wie beispielsweise jene in Szenario 3, benötigen mehr Leistung, wodurch der Gesamtverbrauch schlussendlich höher ist als bei Android 4.4.2.

Aufgrund der verwendeten Testumgebung konnte nicht geprüft werden, wie sich der Energieverbrauch bei der Verwendung von Mobilfunknetzen (3G/4G) im Gegensatz zu WLAN verhält.

Letztlich wurden nur Massnahmen für Android beschrieben und angewendet, in zukünftigen Arbeiten können diese auch bei anderen mobilen Betriebssystemen auf ihre Effizienz geprüft werden. Auch die Energieeffizienz beim Einsatz der nativen Programmierung wurde im Rahmen dieser Arbeit nicht untersucht. Der Einfluss des Job Schedulers auf den Energieverbrauch konnte in dieser Arbeit aus Zeitgründen ebenfalls nicht untersucht werden.

Literaturverzeichnis

Alam, F., Panda, P. R., Tripathi, N., Sharma, N., & Narayan, S. (17. Dezember 2013). *Energy Optimization in Android Applications through Wakelock Placement*. Neu Delhi, Indien.

Apple Inc. (10. März 2014). *iOS Developer Library*. Abgerufen am 23. Mai 2015 von Energy Usage Instrument:

https://developer.apple.com/library/ios/documentation/AnalysisTools/Reference/Instruments_User_Reference/EnergyUsageInstrument/EnergyUsageInstrument.html

Apple Inc. (2015). *Apple*. Von Maximizing Battery Life and Lifespan:

<https://www.apple.com/batteries/maximizing-performance/#ios> abgerufen

ARM Ltd. (2014). *ARM*. Abgerufen am 6. Mai 2015 von big.LITTLE Technology:

<http://www.arm.com/products/processors/technologies/biglittleprocessing.php>

ARM Ltd. (2015a). *ARM DS Development Tools*. Abgerufen am 7. Mai 2015 von About DS-5:

<http://ds.arm.com/developer-resources/ds-5-documentation/arm-ds-5-getting-started-guide/about-ds-5>

ARM Ltd. (2015b). *ARM DS-5 Development Tools*. Abgerufen am 7. Mai 2015 von ARM DS-5 Development Studio: <http://ds.arm.com/ds-5/>

AT&T Inc. (2015a). *AT&T Developer*. Abgerufen am 7. Mai 2015 von AT&T Application Resource Optimizer (ARO): <https://developer.att.com/application-resource-optimizer/docs>

AT&T Inc. (2015b). *AT&T Application Resource Optimizer*. Abgerufen am 7. Mai 2015 von Collecting Data: <https://developer.att.com/application-resource-optimizer/docs#data-collector>

AT&T Inc. (2015c). *AT&T Developer*. Abgerufen am 14. Juli 2015 von Comparing LTE and 3G Energy Consumption: <http://developer.att.com/application-resource-optimizer/docs/best-practices/comparing-lte-and-3g-energy-consumption>

Berkel, K. v. (2009). *Multi-Core for Mobile Phones*. Eindhoven, Niederlande.

Berkeley Software Distribution (BSD). (6. Juli 2015). *BSD General Commands Manual. Manual page netcat*.

Bird, C. (20. September 2012). *Intel Developer Zone*. Abgerufen am 3. Juni 2015 von Wakelocks for Android: <https://software.intel.com/en-us/android/articles/wakelocks-for-android>

Bluetooth. (2015). *Bluetooth SIG*. Von The Low Energy Technology Behind Bluetooth Smart: <http://www.bluetooth.com/Pages/low-energy-tech-info.aspx> abgerufen

Combs, G. (unbekannt). *Wireshark*. Abgerufen am 13. Juli 2015 von About Wireshark: <https://www.wireshark.org/about.html>

comparis.ch. (19. Februar 2014). *comparis.ch*. Abgerufen am 20. April 2015 von Aktuelle Studie: iOS weiter deutlich vor Android:

<https://www.comparis.ch/telecom/mobile/news/2014/02/smartphone-markt-2014-iphone-android.aspx>

CreditPlus Bank AG. (12. November 2012). *CreditPlus Bank AG*. Abgerufen am 20. April 2015 von Umfrage zum Smartphone-Kauf: Akku schlägt Marke:

https://www.creditplus.de/fileadmin/03_Ueber_Creditplus/Newsroom_und_Pressebereich/Newsroom/CP_20121005_Quick_Survey_Elektrograete.pdf

Crunchify.com. (18. Februar 2015). *Crunchify.com*. Abgerufen am 8. Juni 2015 von How to build RESTful Service with Java using JAX-RS and Jersey (Example): <http://crunchify.com/how-to-build-restful-service-with-java-using-jax-rs-and-jersey/>

Dobie, A. (4. Juli 2014). *Android Central*. Von Android L preview: Battery and power management: <http://www.androidcentral.com/android-l-preview-battery-and-power-management> abgerufen

Free Software Foundation, Inc. (2015). *GNU Compiler Collection (GCC) Internals*. Abgerufen am 3. Juni 2015 von Basic Blocks: <https://gcc.gnu.org/onlinedocs/gccint/Basic-Blocks.html>

Gonzalez, D. (11. Mai 2014). *JetBrains Plugin Repository*. Abgerufen am 11. Mai 2015 von AEON (Android Energy Optimization): <https://plugins.jetbrains.com/plugin/7444?pr=>

Google Inc. (2007). *Google Git*. Abgerufen am 3. Juni 2015 von PowerManagerService.java: <https://android.googlesource.com/platform/frameworks/base+/b267554/services/java/com/android/server/power/PowerManagerService.java>

Google Inc. (21. Mai 2015a). *Android Developers*. Abgerufen am 23. Mai 2015 von JobScheduler: <https://developer.android.com/reference/android/app/job/JobScheduler.html>

Google Inc. (4. Juni 2015b). *Android Developers*. Abgerufen am 8. Juni 2015 von JobInfo.Builder: <https://developer.android.com/reference/android/app/job/JobInfo.Builder.html>

Google Inc. (2. Juni 2015c). *Android Developers*. Abgerufen am 3. Juni 2015 von PowerManager: <http://developer.android.com/reference/android/os/PowerManager.html>

Google Inc. (30. Juni 2015d). *Android Developers*. Abgerufen am 1. Juli 2015 von Timer: <http://developer.android.com/reference/java/util/Timer.html>

Google Inc. (30. Juni 2015e). *Android Developers*. Abgerufen am 1. Juli 2015 von BatteryManager: <http://developer.android.com/reference/android/os/BatteryManager.html>

Google Inc. (unbekannt-a). *Android Developers*. Von Android 5.0 APIs: <https://developer.android.com/about/versions/android-5.0.html> abgerufen

Pierre-Alain Wyssen

Google Inc. (unbekannt-b). *Android Developers*. Abgerufen am 11. Mai 2015 von Android Developer Tools: <http://developer.android.com/tools/help/adt.html>

Google Inc. (unbekannt-c). *Android Developers*. Abgerufen am 20. Mai 2015 von Android Studio Overview: <http://developer.android.com/tools/studio/index.html>

Google Inc. (unbekannt-d). *Android Developers*. Abgerufen am 16. Juni 2015 von Android NDK: <https://developer.android.com/tools/sdk/ndk/index.html>

Gordon, M., Zhang, L., & Tiwana, B. (9. Oktober 2011a). *PowerTutor*. Abgerufen am 11. Mai 2015 von Overview: <http://ziyang.eecs.umich.edu/projects/powertutor/index.html>

Gordon, M., Zhang, L., & Tiwana, B. (9. Oktober 2011b). *PowerTutor*. Von Using PowerTutor: <http://ziyang.eecs.umich.edu/projects/powertutor/instructions.html> abgerufen

Herrería-Alonso, S., Rodríguez-Pérez, M., Fernández-Veiga, M., & López-García, C. (22. Juni 2012). Bounded Energy Consumption with Dynamic Packet Coalescing. Vigo, Spanien.

Microsoft. (2. Dezember 2014). *Windows Phone*. Von Stromsparmmodus: <https://www.windowsphone.com/de-de/store/app/stromsparmmodus/c551f76f-3368-42bb-92df-7bfbb9265636> abgerufen

Oxford University Press. (2015). *Oxford Dictionaries*. Von <http://www.oxforddictionaries.com/definition/english/dumbphone> abgerufen

Pierick, C. (21. Februar 2015). *TOASTDROID*. Abgerufen am 8. Juni 2015 von How to use Android's Job Scheduler: <http://toastdroid.com/2015/02/21/how-to-use-androids-job-scheduler/>

Schumann, R., & Tamarcaz, C. (2015). *Final Report: ANP: Air Navigation Platform - WP 6 Energy Consumption*. HES-SO Valais-Wallis, Institute of Information Systems. Sion: HES-SO Valais-Wallis.

Tarkoma, S., Siekkinen, M., Lagerspetz, E., & Xiao, Y. (2014). *Smartphone Energy Consumption: Modeling and Optimization*. Cambridge: Cambridge University Press.

TechTerms.com. (8. Januar 2013). *TechTerms.com*. Von Bare Metal: http://techterms.com/definition/bare_metal abgerufen

Wilke, C. (2013). *code.google.com*. Abgerufen am 11. Mai 2015 von JouleUnit - Project Information: <https://code.google.com/p/jouleunit/>

Anhang I: Implementierung der App zur Messung der Akkukapazität

Dieses Kapitel beschreibt die Funktionsweise der App, welche in Szenario 1 (Kapitel 6.2) verwendet wurde. Diese App misst, sobald der Benutzer auf den Button "Start" drückt, alle zehn Minuten die aktuelle Akkukapazität.

Auf Abbildung I.1 ist der Pseudocode dieser App dargestellt:

```
1 onCreate() {
2     getLog(); // if the file exists, display the log
3     openFile(); // open the file output stream
4
5     if(action == "start") { // when the user taps on the button "Start"
6         wakeLock.acquire(); // SCREEN_BRIGHT_WAKE_LOCK (deprecated) is acquired
7
8         clearFile(); // if the file exists, delete and re-create it
9
10        timer = new Timer(); // Timers schedule one-shot or recurring tasks for execution
11        task = new myTimerTask(); // this class represents a task to run at a specified time
12
13        timer.scheduleTimer(task); // schedule the task to be executed every 10 minutes
14    }
15
16    if(action == "stop") { // when the user taps on the button "Stop"
17        wakeLock.release(); // release the wake lock acquired in line 6
18
19        timer.cancel(); // cancels the Timer and all scheduled tasks
20
21        getLog(); // display the new acquired results
22    }
23 }
```

Abbildung I.1: Vereinfachte Funktionsweise der App zur Messung der Akkukapazität (eigene Darstellung)

Die Zeilen 5 bis 22 auf Abbildung I.1 wurden im OnClickListener des Buttons implementiert, wie auf Abbildung I.2 ersichtlich:

```

52     btnStart.setOnClickListener(new View.OnClickListener() {
53         PowerManager powerManager = (PowerManager) getSystemService(PowerManager.class);
54         PowerManager.WakeLock wakeLock = powerManager.newWakeLock(PowerManager.SCREEN_BRIGHT_WAKE_LOCK, "myWakeLock");
55         @Override
56     public void onClick(View v) {
57         final int status = (Integer) v.getTag();
58
59         if(status == 1) {
60             btnStart.setText("Stop");
61
62             wakeLock.acquire();
63
64             //Start timer
65             timer = new Timer();
66             task = new MyTimerTask();
67             timer.scheduleAtFixedRate(task, 0, 10*60*1000); //0 = initial delay, 10*60*1000 = period in milliseconds (10 minutes)
68
69             v.setTag(0);
70         } else {
71             btnStart.setText("Start");
72
73             wakeLock.release();
74
75             //Stop timer
76             timer.cancel();
77
78             //Update the ListView
79             getLog();
80
81             v.setTag(1);
82         }
83     }
84 });

```

Abbildung I.2: Implementierung des OnClickListener (eigene Darstellung)

Auf Zeile 77 in Abbildung I.2 ist ersichtlich, wie der Task zur Messung der Akkukapazität zeitlich festgelegt wird. Es gibt zwei Möglichkeiten wie diese Methode "scheduleAtFixedRate" aufgerufen werden kann (Google Inc., 2015d):

- scheduleAtFixedRate(TimerTask task, long delay, long period)
- scheduleAtFixedRate(TimerTask task, Date when, long period)

Gemäss Google Inc. (2015d) kann entweder eine Wartedauer (long delay) oder ein genauer Zeitpunkt (Date when) für die Ausführung des Tasks angegeben werden. Für diese App wurde die erste Möglichkeit gewählt; der Task wird ohne Wartedauer zum ersten Mal ausgeführt (delay = 0). Der dritte Parameter dieser Methode gibt den Zeitpunkt der nächsten Ausführung des Tasks (in Millisekunden) an. Dieser Wert wurde für diese App auf zehn Minuten (10*60*1000 Millisekunden) festgelegt.

Der Task, welcher auf Zeile 76 bei jedem Start der Messung neu angelegt wird, ist auf Abbildung I.3 ersichtlich:


```
108 private class MyTimerTask extends TimerTask {
109     @Override
110     public void run() {
111         //Get battery information
112         Intent batteryIntent = registerReceiver(null, new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
113         int level = batteryIntent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
114         int scale = batteryIntent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
115
116         //Get current date and time
117         c = Calendar.getInstance();
118
119         String now = c.get(Calendar.HOUR_OF_DAY) + ":" + c.get(Calendar.MINUTE) + ":" + c.get(Calendar.SECOND);
120         String toWrite = now + " - " + level + "/" + scale + "\n";
121
122         //Write information to internal file
123         try {
124             outputStream.write(toWrite.getBytes());
125         } catch (Exception e) {
126             e.printStackTrace();
127         }
128     }
129 }
```

Abbildung I.3: Implementierung des Tasks, welcher die Akkukapazität ausliest (eigene Darstellung)

Jede Messung wird mit der aktuellen Zeit gespeichert. Sobald der Benutzer die Messung durch Drücken auf den Button "Stop" anhält, ist folgende Benutzeroberfläche sichtbar (siehe Abbildung I.4):

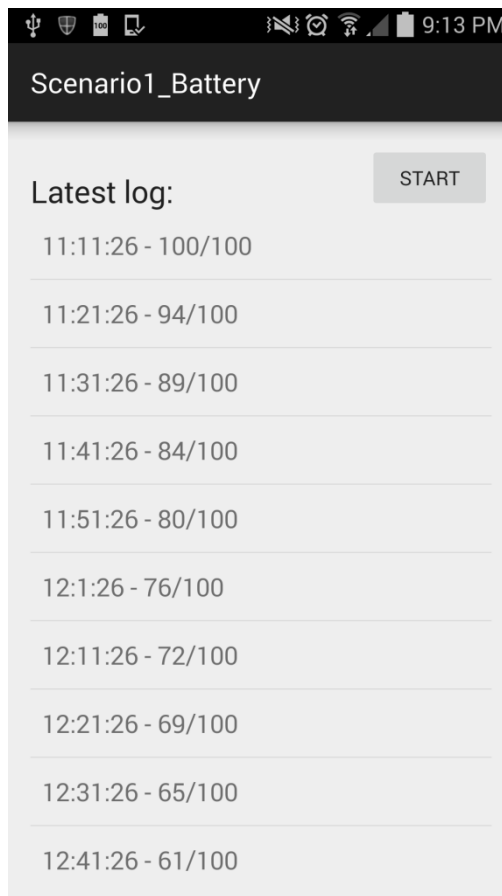


Abbildung I.4: Benutzeroberfläche der App nach einer Messung (eigene Darstellung)

Neben `EXTRA_LEVEL` (aktuelle Akkukapazität, von 0 bis `EXTRA_SCALE`) und `EXTRA_SCALE` (maximale Akkukapazität) gibt es noch acht weitere Konstanten betreffend des Akkus, welche ausgelesen werden können (Google Inc., 2015e). Beispiele hierfür sind:

- `EXTRA_PRESENT`: Gibt an, ob ein Akku vorhanden ist
- `EXTRA_PLUGGED`: Gibt an, ob das Smartphone an eine Stromquelle angeschlossen ist
- `EXTRA_TEMPERATURE`: Die aktuelle Temperatur des Akkus
- `EXTRA_VOLTAGE`: Die aktuelle elektrische Spannung des Akkus

Der komplette Quellcode dieser App ist auf der beigelegten CD im Verzeichnis "3_Szenario1" zu finden.

Anhang II: Implementierung der App zur Simulation von Packet coalescing

Dieses Kapitel beinhaltet Details zur Implementierung der App, welche in Szenario 2 (Kapitel 6.3) verwendet wurde. Diese App bietet eine simple Benutzeroberfläche, welche lediglich aus einem Button besteht. Beim Drücken auf diesen Button "Send Data" werden zuvor definierte Datenpakete in definierten Abständen an einen Server in der Testumgebung gesendet.

Die App wurde mit Android Studio 1.2.2 realisiert und bietet die Möglichkeit, zuvor definierte Datenpakete in zeitlichen Abständen zu senden. Die Datei "scenario.csv" muss dabei im Ordner `/storage/sdcard0` auf dem Smartphone vorhanden sein.

Sobald die Datei gelesen wurde und der Benutzer auf "Send Data" drückt, wird der in Abbildung II.1 ersichtliche Hintergrundtask gestartet:

```

92     private class SendDataTask extends AsyncTask<Void, Void, Void> {
93
94         @Override
95     protected Void doInBackground(Void... params) {
96         try {
97             synchronized (this) {
98                 //convert both lists to arrays
99                 Integer packetSizesArray[] = packetSizes.toArray(new Integer[packetSizes.size()]);
100                Integer timeOutsArray[] = timeOuts.toArray(new Integer[timeOuts.size()]);
101
102                byte[] dataToSend;
103
104                //iterate through array and send data to the server
105                for (int i = 0; i < packetSizesArray.length; i++) {
106                    dataToSend = new byte[packetSizesArray[i]];
107
108                    Socket socket = new Socket("192.168.1.108", 1234);
109                    OutputStream outputStream = socket.getOutputStream();
110
111                    Log.d("#" + (i+1), "Sending " + dataToSend.length +
112                        " bytes of data (time-out is " + timeOutsArray[i] + " seconds)");
113
114                    outputStream.write(dataToSend);
115
116                    socket.close();
117
118                    wait(timeOutsArray[i] * 1000);
119                }
120            }
121        } catch (Exception e) {
122            e.printStackTrace();
123        }
124        return null;
125    }
126
127
128
129
130     @Override
131     protected void onPostExecute(Void aVoid) {
132         btnSend.setText(R.string.btnDone);
133         textView.setText(R.string.tvDone);
134         Log.d("done", "Data sent!");
135     }

```

Abbildung II.1: Implementierung des Hintergrundtasks zum Senden der Daten (eigene Darstellung)

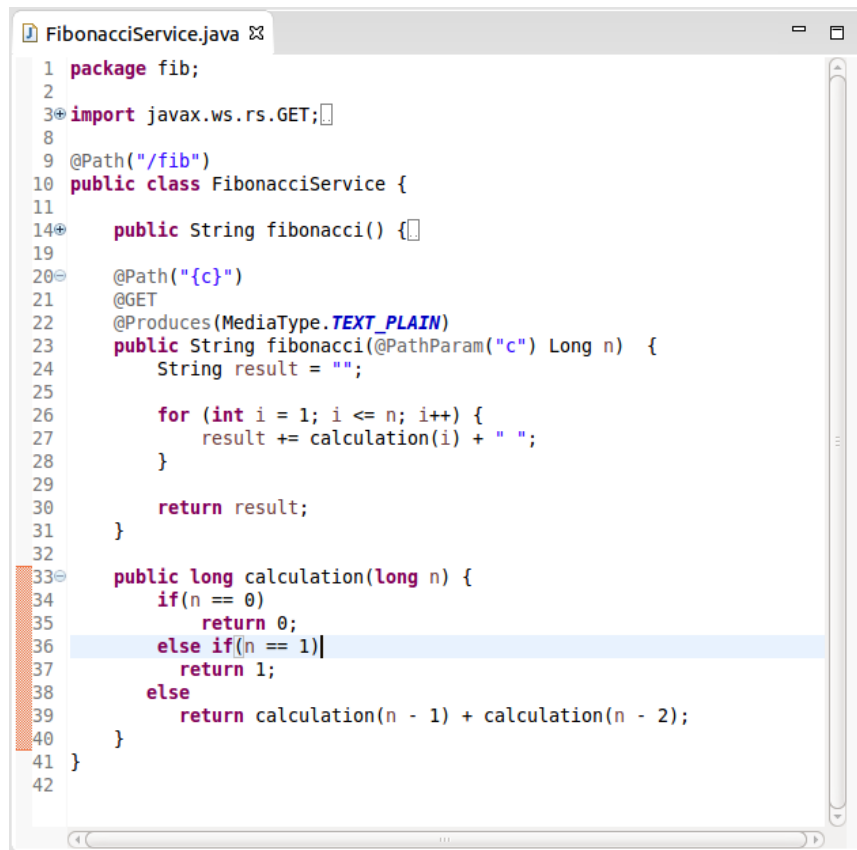
Die Verbindung mit dem Server wird auf Zeile 108 hergestellt, wobei 192.168.1.108 die IP-Adresse dieses Servers ist, mit dem entsprechenden Port (1234) als zweiten Parameter. Das Senden der Daten geschieht auf Zeile 114, anschliessend erfolgt das definierte Time-out auf Zeile 118.

Der komplette Quellcode dieser App ist auf der beigelegten CD im Verzeichnis "4_Szenario2" zu finden.

Anhang III: Implementierung des Webdienstes zur Berechnung der Fibonacci-Folge

In diesem Kapitel wird das Vorgehen erläutert, wie der in Szenario 3 (Kapitel 6.4) verwendete Webdienst und die konsumierende App implementiert wurden. Die App lässt den Benutzer auswählen, wie viele Stellen der Fibonacci-Reihe er berechnen möchte und ob diese Berechnung lokal oder via Webdienst geschehen soll.

Der **Webdienst** wurde mit Eclipse Luna Service Release 2 (4.4.2) realisiert und läuft auf einem lokalen Webserver (Apache Tomcat v7.0). Mithilfe eines Tutorials von Crunchify.com (2015) wurde der in Abbildung III.1 ersichtliche Webdienst realisiert:



```
1 package fib;
2
3 import javax.ws.rs.GET;
4
5
6
7
8
9 @Path("/fib")
10 public class FibonacciService {
11
12
13
14 public String fibonacci() {
15
16
17
18
19
20 @Path("/{c}")
21 @GET
22 @Produces(MediaType.TEXT_PLAIN)
23 public String fibonacci(@PathParam("c") Long n) {
24     String result = "";
25
26     for (int i = 1; i <= n; i++) {
27         result += calculation(i) + " ";
28     }
29
30     return result;
31 }
32
33 public long calculation(long n) {
34     if(n == 0)
35         return 0;
36     else if(n == 1)
37         return 1;
38     else
39         return calculation(n - 1) + calculation(n - 2);
40 }
41 }
42
```

Abbildung III.1: Webdienst zur Berechnung der Fibonacci-Folge (eigene Darstellung)

Sobald der Webdienst auf dem Server läuft, kann dieser unter folgender URL in der Testumgebung aufgerufen werden: <http://192.168.1.108:8080/Fibonacci/fib/fib/x>. Dabei steht x für eine positive natürliche Zahl, welche die Anzahl Zahlen der Fibonacci-Folge repräsentiert. Das Resultat für x = 10 lautet: 1 1 2 3 5 8 13 21 34 55; die ersten 10 Zahlen dieser Folge. Die Berechnung erfolgt rekursiv.

Die **App** wurde mit Android Studio 1.2.2 realisiert und bietet die Möglichkeit, eine vordefinierte Anzahl Zahlen (10, 20, 30, 40 oder 50) der Fibonacci-Folge entweder lokal (auf dem Smartphone) oder entfernt (remote; mithilfe des Webdienstes) zu berechnen. Abbildung III.2 zeigt die Benutzeroberfläche der App beim Start:

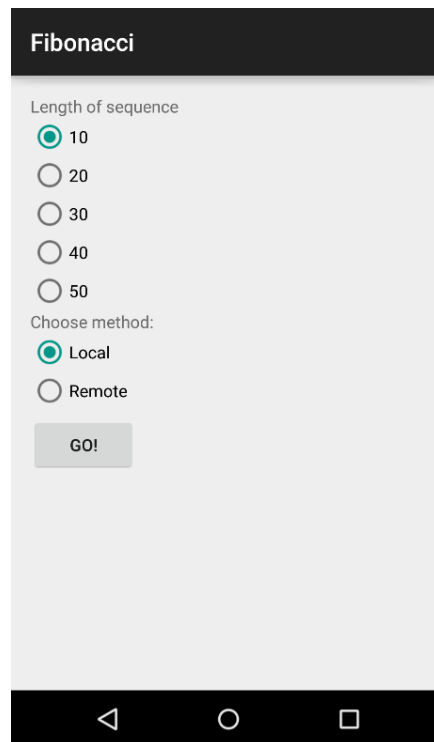


Abbildung III.2: Benutzeroberfläche beim Start der App (eigene Darstellung)

Sobald der Benutzer die Länge der Sequenz und die Methode ausgewählt hat, kann er die Berechnung mit dem "GO!"-Button starten. Auf Abbildung III.3 ist das Ergebnis mit den Werten "20" und "Remote" ersichtlich:

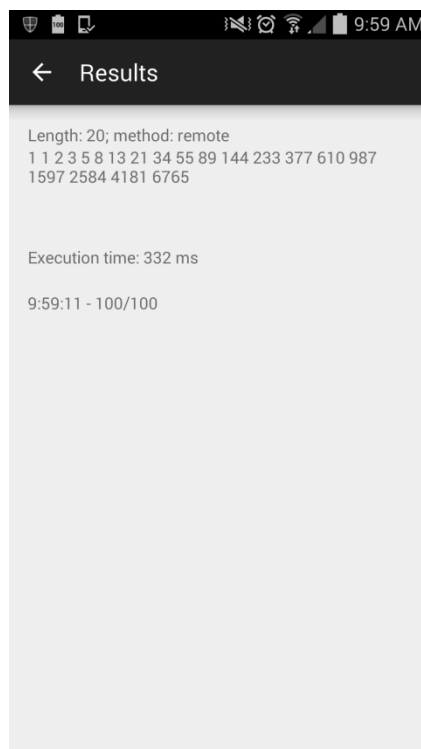


Abbildung III.3: Die ersten 20 Zahlen der Fibonacci-Folge via Webdienst berechnet (eigene Darstellung)

Die lokale sowie die entfernte Berechnung geschehen dabei jeweils in einem Hintergrundtask. Bei der lokalen Berechnung ist dies nötig, weil ansonsten kein Ergebnis angezeigt wird. Beim Versuch die entfernte Berechnung im Hauptthread auszuführen, wird die Ausführung wegen einer Ausnahme abgebrochen, deshalb wird der Zugriff auf den Webdienst ebenfalls in einem Hintergrundtask ausgeführt. Ungeachtet der Berechnungsmethode werden nach der Berechnung die Ausführungszeit, die aktuelle Akkukapazität sowie der Zeitpunkt, zu welchem die Berechnung abgeschlossen war, angezeigt. Auf Abbildung III.4 sowie Abbildung III.5 ist die Implementierung der lokalen sowie der entfernten Berechnung ersichtlich:

```

66     private class FibonacciLocal extends AsyncTask<Long,Void,String> {
67
68         String result;
69         PowerManager powerManager =
70             (PowerManager) getSystemService(POWER_SERVICE);
71         PowerManager.WakeLock wakeLock =
72             powerManager.newWakeLock(PowerManager.SCREEN_BRIGHT_WAKE_LOCK, "myWakeLock");
73
74         @Override
75     protected String doInBackground(Long... params) {
76         startTime = System.nanoTime();
77         wakeLock.acquire();
78         Long n = params[0];
79         result = "";
80
81         for(int i = 1; i <= n; i++) {
82             result += calculateLocal(i) + " ";
83             Log.d("status",i + ": " + this.getStatus().toString());
84         }
85
86         return result;
87     }
88
89     private long calculateLocal(long n) {
90         if(n == 0)
91             return 0;
92         else if (n == 1)
93             return 1;
94         else
95             return calculateLocal(n - 1) + calculateLocal(n - 2);
96     }
97
98     @Override
99     protected void onPostExecute(String results) {
100         if(results != null) {
101             tv2.setText(results);
102             stopTime = System.nanoTime();
103             wakeLock.release();
104             tv3.setText("Execution time: " + (stopTime - startTime)/1000000 + " ms");
105
106             displayTimeAndBattery();
107         }
108     }
109 }

```

Abbildung III.4: Implementierung der lokalen Berechnung (eigene Darstellung)

Wie auch beim Webdienst auf dem Server geschieht die Berechnung auf dem Smartphone rekursiv (Methode `calculateLocal(long n)`). Da die Berechnung von 50

Stellen lokal mindestens zwei Stunden dauert, wurde hier ein WakeLock eingesetzt, welches den Bildschirm des Smartphones eingeschaltet lässt, bis die Berechnung abgeschlossen ist.

```

111     private class FibonacciRemote extends AsyncTask<Long,Void,String> {
112
113         @Override
114         protected String doInBackground(Long... params) {
115             startTime = System.nanoTime();
116             String n = params[0].toString();
117             HttpClient httpClient = new DefaultHttpClient();
118             HttpContext localContext = new BasicHttpContext();
119             HttpGet httpGet = new HttpGet("http://192.168.1.108:8080/Fibonacci/fib/fib/"+n);
120             String result = "";
121
122             try {
123                 HttpResponse response = httpClient.execute(httpGet, localContext);
124
125                 result = inputStreamToString(response.getEntity().getContent());
126             } catch (Exception e) {
127                 e.printStackTrace();
128             }
129
130             return result;
131         }
132
133         //Taken from http://avilyne.com/?p=105
134         private String inputStreamToString(InputStream is) {
135
136             String line = "";
137             StringBuilder total = new StringBuilder();
138
139             // Wrap a BufferedReader around the InputStream
140             BufferedReader rd = new BufferedReader(new InputStreamReader(is));
141
142             try {
143                 // Read response until the end
144                 while ((line = rd.readLine()) != null) {
145                     total.append(line);
146                 }
147             } catch (IOException e) {
148                 Log.e("Error", e.getLocalizedMessage(), e);
149             }
150
151             // Return full string
152             return total.toString();
153         }
154
155         @Override
156         protected void onPostExecute(String results) {
157             if(results != null) {
158                 tv2.setText(results);
159                 stopTime = System.nanoTime();
160                 tv3.setText("Execution time: " + (stopTime - startTime)/1000000 + " ms");
161
162                 displayTimeAndBattery();
163             }
164         }
165     }
166 }

```

Abbildung III.5: Implementierung der entfernten Berechnung (eigene Darstellung)

Der komplette Quellcode dieser App sowie des Webdienstes sind auf der beigelegten CD im Verzeichnis "5_Szenario3" zu finden.

Anhang IV: Beschreibung der erfassten Daten

Auf der beigelegten CD sind im Ordner "2_Messergebnisse" drei Excel-Dateien ("Ergebnisse_Szenario1.xlsx", "Ergebnisse_Szenario2.xlsx" und "Ergebnisse_Szenario3.xlsx") mit den Ergebnissen der Messungen der drei Szenarien zu finden. Die Originaldaten (raw data) dieser Szenarien befinden sich im Ordner "6_Originaldaten". Die oben erwähnten Excel-Dateien beinhalten folgende Daten:

Ergebnisse_Szenario1.xlsx

Szenario1KonfigA_PT

Enthält Daten für Szenario 1 - Konfiguration A, welche mit dem Tool PowerTutor (PT) gemessen wurden. Neben den drei Spalten "total-power" (Gesamtverbrauch in mW), "LCD" (Verbrauch Bildschirm in mW) und "CPU" (Verbrauch Prozessor in mW) enthält dieses Tabellenblatt zwei Prozentwerte (E2 und E3), welche für das Erstellen des Kuchendiagramms verwendet wurden. Letztlich wurden anhand der erfassten Daten das Maximum, das Minimum, der Durchschnitt sowie die Standardabweichung des Gesamtverbrauchs sowie des Verbrauchs des Prozessors berechnet.

Szenario1KonfigB_PT

Enthält Daten für Szenario 1 - Konfiguration B, welche mit dem Tool PowerTutor (PT) gemessen wurden. Der Aufbau dieses Tabellenblattes ist derselbe wie bei Konfiguration A (siehe oben).

Szenario1KonfigA_ARO

Enthält Daten für Szenario 1 - Konfiguration A, welche mit dem Tool AT&T ARO geschätzt wurden. Die drei Zeilen "Total RRC Energy" (Gesamtverbrauch Mobilfunkinterface), "Total GPS Energy" (Gesamtverbrauch GPS) und "Total Screen Energy" (Gesamtverbrauch Bildschirm) enthalten jeweils die geschätzten Werte des Tools sowie Umrechnungen (von Joule zu Joule/s sowie von Joule/s zu mW). Zelle B6 enthält die Laufzeit der Messung, diese war nötig, um die Werte in Joule in Milliwatt umzurechnen. Letztlich enthält dieses Tabellenblatt ein Kuchendiagramm, welches die Werte in den Zellen B1 bis B3 darstellt.

Szenario1KonfigB_ARO

Enthält Daten für Szenario 1 - Konfiguration B, welche mit dem Tool AT&T ARO geschätzt wurden. Die drei Zeilen "Total Wi-Fi Energy" (Gesamtverbrauch Wi-Fi-Interface), "Total GPS Energy" (Gesamtverbrauch GPS) und "Total Screen Energy" (Gesamtverbrauch Bildschirm) enthalten jeweils die geschätzten Werte des Tools sowie Umrechnungen. Zelle B6 enthält die Laufzeit der Messung, diese war nötig, um die Werte in Joule in Milliwatt umzurechnen. Letztlich enthält dieses Tabellenblatt ein Kuchendiagramm, welches die Werte in den Zellen B1 und B3 darstellt.

Szenario1KonfigA_Akku

Enthält Daten für Szenario 1 - Konfiguration A, welche mit der eigens entwickelten App gemessen wurden. Die Spalten A (Android 4.4.2) und F (Android 5.0.1) geben die verstrichene Zeit in Sekunden an, die Spalten B und G die gemessene verbleibende Akkukapazität und die Spalten C und H die jeweiligen Unterschiede bei der Akkukapazität. Letztlich enthalten die Zellen D23 und I28 den durchschnittlichen Verbrauch pro Zeiteinheit (10 Minuten). Bei den Werten ab Zeile 23 bzw. 28 (mit einer Rahmenlinie markiert) handelt es sich um Schätzungen. Das Liniendiagramm wurde mit den Werten in den Spalten B und G erstellt.

Szenario1KonfigB_Akku

Enthält Daten für Szenario 1 - Konfiguration B, welche mit der eigens entwickelten App gemessen wurden. Die Spalten A und F geben die verstrichene Zeit in Sekunden an, die Spalten B und G die gemessene verbleibende Akkukapazität und die Spalten C und H die jeweiligen Unterschiede bei der Akkukapazität. Letztlich enthalten die Zellen D43 und I49 den durchschnittlichen Verbrauch pro Zeiteinheit (10 Minuten). Bei den Werten ab Zeile 43 und 49 (mit einer Rahmenlinie markiert) handelt es sich um Schätzungen. Das erste Liniendiagramm wurde mit den Werten in den Spalten B und G erstellt. Das zweite Liniendiagramm ist ein Vergleich der Ergebnisse von Konfiguration A und B in Android Version 4.4.2 sowie 5.0.1.

Ergebnisse_Szenario2.xlsx

Szenario2_20P_NoPC

Enthält Daten für Szenario 2, welche mit dem Tool AT&T ARO gemessen wurden. Es handelt sich hierbei um Daten und Ergebnisse der Simulation **ohne** packet coalescing. Folgende Daten sind in den Zeilen 2 bis 6 ersichtlich:

- Zeile 2: Gesamtverbrauch Wi-Fi (in Joule): Schätzung des Tools AT&T ARO
- Zeile 3: Anzahl TCP-Sessions während der Simulation; hierfür wurden die Anzahl Zellen zwischen B11 und K30 gezählt
- Zeile 4: Energieeffizienz (in Joule pro Kilobyte): berechnet aus dem Gesamtverbrauch des Wi-Fis dividiert durch die Gesamtzahl an Bytes * 1000 (= Kilobytes)
- Zeile 5: Gesamtanzahl Bytes: berechnet aus der Summe aller Zellen zwischen B11 und K30
- Zeile 6: Gesamtanzahl IP-Pakete: berechnet aus der Summe aller Zellen zwischen B33 und K52

Ab Zeile 10 sind die Originaldaten (raw data), welche mit dem Tool AT&T ARO gemessen wurden, zu finden. Aus diesen Daten wurden jeweils der Durchschnitt sowie die Standardabweichung berechnet.

Szenario2_5P_PC

Enthält Daten für Szenario 2, welche mit dem Tool AT&T ARO gemessen wurden. Es handelt sich hierbei um Daten und Ergebnisse der Simulation **mit** packet coalescing. Folgende Daten sind in den Zeilen 2 bis 6 ersichtlich:

- Zeile 2: Gesamtverbrauch Wi-Fi (in Joule): Schätzung des Tools AT&T ARO
- Zeile 3: Anzahl TCP-Sessions während der Simulation; hierfür wurden die Anzahl Zellen zwischen B11 und F15 gezählt
- Zeile 4: Energieeffizienz (in Joule pro Kilobyte): berechnet aus dem Gesamtverbrauch des Wi-Fis dividiert durch die Gesamtzahl an Bytes * 1000 (= Kilobytes)
- Zeile 5: Gesamtanzahl Bytes: berechnet aus der Summe aller Zellen zwischen B11 und F15
- Zeile 6: Gesamtanzahl IP-Pakete: berechnet aus der Summe aller Zellen zwischen B18 und F22

Ab Zeile 10 sind die Originaldaten (raw data), welche mit dem Tool AT&T ARO gemessen wurden, zu finden. Aus diesen Daten wurden jeweils der Durchschnitt sowie die

Standardabweichung berechnet. Des Weiteren enthält dieses Tabellenblatt ab Spalte I eine Zusammenfassung der Ergebnisse der beiden Simulationen, welche für die beiden Säulendiagramme verwendet wurden.

Ergebnisse_Szenario3.xlsx

Szenario3_lokal_Zeit_4.4.2

Enthält Daten für Szenario 3, welche mit der eigens entwickelten App in Android 4.4.2 gemessen wurden. Es handelt sich hierbei um Daten und Ergebnisse der lokalen Berechnung. Folgende Daten sind in den Zeilen 1 bis 17 dargestellt:

- Zeile 1: Anzahl Stellen: Anzahl der zu berechnenden Stellen der Fibonacci-Folge
- Zeile 2 bis 16: Mit der App gemessene Werte betreffend der Ausführungszeit in Millisekunden
- Zeile 17: Durchschnittliche Ausführungsdauer pro Anzahl Stellen
- Zeile 18: Standardabweichung der gemessenen Werte pro Anzahl Stellen

Letztlich enthält dieses Tabellenblatt zwei Säulendiagramme. Das obere enthält eine Zusammenfassung der Daten der lokalen sowie der entfernten Berechnung in Android 4.4.2. Das untere ist eine Zusammenfassung der Ausführungszeiten in Android 4.4.2, Android 5.0.1 mit PSM und Android 5.0.1 ohne PSM.

Szenario3_lokal_Zeit_5.0.1_PS

Enthält Daten für Szenario 3, welche mit der eigens entwickelten App in Android 5.0.1 mit aktiviertem Energiesparmodus gemessen wurden. Es handelt sich hierbei um Daten und Ergebnisse der lokalen Berechnung. Folgende Daten sind in den Zeilen 1 bis 17 dargestellt:

- Zeile 1: Anzahl Stellen: Anzahl der zu berechnenden Stellen der Fibonacci-Folge
- Zeile 2 bis 16: Mit der App gemessene Werte betreffend der Ausführungszeit in Millisekunden
- Zeile 17: Durchschnittliche Ausführungsdauer pro Anzahl Stellen
- Zeile 18: Standardabweichung der gemessenen Werte pro Anzahl Stellen

Szenario3_lokal_Zeit_5.0.1_NoPS

Enthält Daten für Szenario 3, welche mit der eigens entwickelten App in Android 5.0.1 mit deaktiviertem Energiesparmodus gemessen wurden. Es handelt sich hierbei um Daten und

Ergebnisse der lokalen Berechnung. Der Aufbau dieses Tabellenblattes ist derselbe wie bei der lokalen Berechnung. Letztlich enthält dieses Tabellenblatt ein Säulendiagramm, auf welchem ein Vergleich zwischen aktiviertem und deaktiviertem PSM zu sehen ist.

Szenario3_remote_Zeit_4.4.2

Enthält Daten für Szenario 3, welche mit der eigens entwickelten App in Android 4.4.2 gemessen wurden. Es handelt sich hierbei um Daten und Ergebnisse der entfernten Berechnung. Der Aufbau dieses Tabellenblattes ist derselbe wie bei der lokalen Berechnung (siehe Tabellenblatt "Szenario3_lokal_Zeit_4.4.2").

Szenario3_lokal_Akku

Enthält Daten für Szenario 3, welche mit der eigens entwickelten App und dem Tool PowerTutor gemessen wurden. Es handelt sich hierbei um Daten und Ergebnisse der lokalen Berechnung in Android 4.4.2 und Android 5.0.1. Die Spalten B bis G, M bis R und V bis AA sind die Ergebnisse bei der Berechnung von 40 Stellen, die Spalte K diejenigen von 50 Stellen. Die Spalten H, S und AB enthalten die Durchschnittswerte, die Spalten I, T und AC die Standardabweichung der gemessenen Werte für 40 Stellen. Folgende Daten sind in den Zeilen 2 bis 9 zu finden:

- Zeile 2: Mit der eigens entwickelten App gemessene Werte betreffend der Ausführungszeit in Sekunden
- Zeile 3: Mit der eigens entwickelten App gemessene Werte betreffend der verbleibenden Akkukapazität
- Zeile 4: Mit den Originaldaten von PowerTutor berechneter durchschnittlicher Energieverbrauch des Prozessors; in Milliwatt angegeben
- Zeile 5: Division der Werte in Zeile 4 durch 1000 (= Joule pro Sekunde); Zwischenschritt zur Umrechnung von Milliwatt nach Joule
- Zeile 6: Multiplikation der Werte in Zeile 5 mit der gemessenen Zeit in Zeile 2
- Zeile 7: Mit PowerTutor gemessener Energieverbrauch des Bildschirms; in Milliwatt angegeben
- Zeile 8: Energieverbrauch des Bildschirms in Joule umgerechnet
- Zeile 9: Summe der Zeilen 6 und 8

Ab Zeile 11 sind die mit dem Tool PowerTutor gemessenen Originaldaten (raw data) zu finden. Ausserdem enthält dieses Tabellenblatt drei Säulendiagramme, welche die blau markierten Daten visualisieren sowie ein Liniendiagramm, welches den Akkuverlauf darstellt.

Szenario3_remote_Akku

Enthält Daten für Szenario 3, welche mit der eigens entwickelten App und dem Tool PowerTutor in Android 4.4.2 gemessen wurden. Es handelt sich hierbei um Daten und Ergebnisse der entfernten Berechnung. Die Zeilen 1 bis 9 sind die Ergebnisse bei der Messung von 40 Stellen, die Zeilen 11 bis 19 diejenigen bei der Messung von 50 Stellen. Für diese beiden Sequenzlängen sind die gemessenen Werte in den Spalten B bis G sowie J bis O zu finden. Die Spalten H und P sind berechnete oder gemessene Gesamtwerte und Spalte Q beinhaltet die Mittelwerte dieser beiden Spalten. Letztlich beinhaltet Spalte R die Standardabweichung der gemessenen Werte.

Die Zeilen 2 und 3 wurden mit der eigens entwickelten App gemessen, die Zeilen 4 und 7 wurden mit dem Tool PowerTutor gemessen und bei den Zeilen 5, 6, 8 und 9 handelt es sich um abgeleitete Werte.

Ab Zeile 21 sind die mit dem Tool PowerTutor gemessenen Originaldaten (raw data) zu finden. Des Weiteren befinden sich in diesem Tabellenblatt sechs Säulendiagramme. Die ersten drei sind Visualisierungen der via Webdienst gemessenen Werte. Bei den drei restlichen Diagrammen (mit den blauen und grünen Säulen) handelt es sich um eine Zusammenfassung der Ergebnisse der lokalen sowie der entfernten Berechnung.

Szenario3_remote_ARO

Enthält Daten für Szenario 3, welche mit dem Tool AT&T ARO gemessen wurden. Spalte A gibt die Anzahl Stellen an, die Spalten B bis G die gemessenen Werte und Spalte H der daraus resultierende Durchschnitt.

Originaldaten

Der Ordner "6_Originaldaten" auf der CD enthält die gesammelten Rohdaten (raw data) der drei Szenarien. Im Ordner "Szenario 1" sind zwei Log-Dateien zu finden, welche mit PowerTutor erstellt wurden. Der Ordner "Szenario 2" enthält zwei Unterordner ("Szenario2_5Pakete" und "Szenario2_20Pakete"), welche die Daten vom AT&T ARO enthalten. Ausserdem sind in diesem Ordner die zwei verwendeten CSV-Dateien für Szenario 2 zu finden. Letztlich beinhaltet der Ordner "Szenario 3" vier Unterordner mit jeweils sechs

Pierre-Alain Wyssen

Log-Dateien, welche vom PowerTutor generiert wurden. Diese Log-Dateien sind die Messwerte bei der entfernten Berechnung. Letztlich enthält dieser Ordner weitere vier Log-Dateien: die Dateien "Szenario3_lokal_40_RAW" und "Szenario3_lokal_50_RAW" enthalten die Messwerte der lokalen Berechnung in Android 4.4.2. Die Dateien "Szenario3_lokal_40_5.0.1_PS" und "Szenario3_lokal_40_5.0.1_NoPS" enthalten die Messwerte der lokale Berechnung in Android 5.0.1 mit ("PS") und ohne ("NoPS") Energiesparmodus.

Selbstständigkeitserklärung des Verfassers

"Ich bestätige hiermit, dass ich die vorliegende Bachelorarbeit alleine und nur mit den angegebenen Hilfsmitteln realisiert habe und ausschliesslich die erwähnten Quellen benutzt habe. Ohne Einverständnis des Studiengangsleiters und des für die Bachelorarbeit verantwortlichen Dozierenden sowie des Forschungspartners, mit dem ich zusammengearbeitet habe, werde ich diesen Bericht an niemanden verteilen, ausser an die Personen, die mir die wichtigsten Informationen für die Verfassung dieses Berichts geliefert haben und die ich nachstehend aufzähle: Prof. Dr. René Schumann"

Pierre-Alain Wyssen