technische universität
dortmund

Masterarbeit

Generierung repräsentativer Android-Benchmarks auf Basis dynamischer Anwendungsanalysen

Fabian Bruckner
13. Dezember 2016

Betreuer:
Prof. Dr.-Ing. Olaf Spinczyk
MSc Inf. Alexander Lochmann

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl 12
Arbeitsgruppe Eingebettete Systemsoftware
`http://ess.cs.tu-dortmund.de`

Zusammenfassung

Es ist wünschenswert über ein Verfahren zur Leistungsbewertung für *Android* zu verfügen, welches auf einfache Art und Weise mit zusätzlichen Daten, die zur Leistungsbewertung abgespielt werden, erweitert werden kann. Die vorliegende Arbeit führt durch die Entwicklung eines solchen Verfahrens, welches in zwei verschiedenen *Android*-Applikationen resultiert. Die eine Applikation zeichnet das Verhalten einer beliebigen Applikation auf und die andere spielt die aufgezeichneten Daten möglichst genau wieder ab. Eine Evaluation des Verfahrens und der gesammelten Daten zeigt, dass in manchen Aspekten noch Optimierungsbedarf besteht. Darüber hinaus zeigt sich, das die aufgezeichneten Daten für die Analyse des Verhaltens der aufgezeichneten Applikation verwendet werden können.

Inhaltsverzeichnis

1 Einleitung

Die Verwendung von Benchmarks zur qualitativen Bewertung diverser Leistungsaspekte ist im Desktop- und Serverbereich weit verbreitet. Dies trifft sowohl auf den Endkundenbereich, als auch auf das wissenschaftliche Umfeld zu. Dabei gibt es zwei verschiedene Szenarien, in denen Benchmarks verwendet werden. Beim ersten Szenario handelt es sich um eine allgemeine Leistungsbeurteilung, welche häufig verwendet wird, um verschiedene Systeme anhand einer einheitlichen Skala zu vergleichen. Das andere Szenario hat das Ziel Änderungen, welche an einem System vorgenommen wurden, im Hinblick auf diverse Aspekte zu bewerten. Dabei kann ein Benchmark eine beliebige Anzahl von Kriterien verwenden, um die Leistung des untersuchten Systems zu beurteilen. Dies können allgemeine Kriterien, wie die Performanz des Prozessors, sein oder spezielle Aspekte, wie die Performanz von parallelen Computern [9, 1].

Nach Huan et al. verbreiten sich Smartphones mit einer enormen Geschwindigkeit [18] und auch Gartner bestätigt diese Aussage [14]. Hierdurch wächst der Bedarf an Verfahren zur Leistungsbewertung, welche die Besonderheiten von mobilen Geräten berücksichtigten. Zwischen Betriebssystemen für den mobilen Einsatz und den klassischen Betriebssystemen für den Einsatz auf Desktop- und Server-Geräten besteht eine gewisse Ähnlichkeit [8]. Dennoch ergeben sich zusätzliche Bewertungskriterien abseits der Schnittmenge mit klassischen Betriebssystemen, wie beispielsweise die Verwendung mobiler Daten, welche in ein solches Verfahren aufgenommen werden müssen. Durch die Aufnahme dieser spezifischen Leistungsmerkmale wird die Aussagekraft für mobile Geräte gewährleistet.

Besonders wünschenswert wäre für ein solches Verfahren, dass es keine synthetischen Daten verwendet um die Leistung zu bewerten. Stattdessen sollten reale Nutzungsdaten zur Leistungsbewertung herangezogen werden. Darüber hinaus sollten diese Daten einfach austauschbar sein. Dies würde es ermöglichen mit Hilfe eines Simulators auf einfache Art und Weise verschiedene Betriebssystemversionen mit verschiedenen Daten zu bewerten. Dieser Prozess könnte in dem Fall sogar automatisiert werden und so auf unkomplizierte Art und Weise viele Daten für eine Evaluation erzeugen.

1.1 Aufbau der Arbeit

Nachfolgend wird der Aufbau der vorliegenden Arbeit vorgestellt. Zunächst werden in Kapitel 2 sämtliche Grundlagen vermittelt, welche zum Verständnis der Arbeit notwendig sind. Darüber hinaus wird in Kapitel 2.1 ein Überblick über bestehende Verfahren zur Leistungsmessung gegeben und dabei sowohl im Endkundenbereich als auch im wissenschaftlichen Umfeld gesucht. Ebenfalls werden potentielle Nutzer solcher Verfahren identifiziert und deren bisherige Lösungen betrachtet, um den Bedarf an dieser Arbeit weiter zu begründen.

Kapitel 3 wird genutzt um alle Anforderungen, welche an das zu entwickelnde Verfahren gestellt werden, zu definieren und einzugrenzen. Dabei werden zunächst in Kapitel 3.1 kritische Aspekte betrachtet und im Anschluss zeigt Kapitel 3.2 einige Aspekte auf, welche nicht kritischer Natur sind, aber trotzdem während der Entwicklung berücksichtigt werden sollten.

Die Modellierung des gesamten Verfahrens wird in Kapitel 4 ausführlich vorgestellt. Dabei wird in Kapitel 4.1 das Verfahren zum Erfassen von Daten, welche anschließend für den eigentlichen Benchmark genutzt werden, im Detail modelliert. Das Verfahren um diese Daten, zum Zwecke der Leistungsbewertung, abzuspielen wird anschließend in Kapitel 4.2 vorgestellt und modelliert.

Diverse Details der Implementierung werden in Kapitel 5 beschrieben. Dabei wird zuerst die Datenerfassung in Kapitel 5.1 betrachtet und im Anschluss die Wiedergabe in Kapitel 5.2. Darüber hinaus werden in Kapitel 5.3 Hilfswerkzeuge vorgestellt, die für die Wiedergabe und Analyse von Daten verwendet werden.

Die Ergebnisse der Arbeit werden in Kapitel 6 betrachtet und untersucht. Dabei wird in Kapitel 6.1 zunächst die Güte des entwickelten Verfahrens bestimmt und im Anschluss, in Kapitel 6.2, wird versucht aus Daten, die mit dem entwickelten Verfahren erfasst wurden, Wissen über die beobachteten Applikationen zu gewinnen.

Abschließend wird in Kapitel 7 ein Fazit gezogen und in Kapitel 7.1 ein Ausblick auf mögliche, zukünftige Arbeiten zu diesem Thema gegeben.

2 Grundlagen

In diesem Kapitel wird zunächst ein Überblick über verwandte Arbeiten und anschließend über verwendete Technologien gegeben. Zum einen wird hierdurch der Bedarf an dieser Arbeit verdeutlicht und zum anderen werden die Kenntnisse vermittelt, die zum Verständnis der Arbeit notwendig sind.

2.1 Verwandte Arbeiten

Es existieren diverse Werkzeuge zur Leistungsmessung und -beurteilung, welche teils exklusiv für *Android* zur Verfügung stehen. Dabei sind zwei verschiedene Gruppen von Ansätzen zu unterscheiden. Beide Gruppen werden in diesem Abschnitt betrachtet. Die erste Gruppe wird im ersten Abschnitt erläutert. Sie umfasst Anwendungen, die im Endkundenbereich ihren größten Erfolg haben und im Regelfall ohne wissenschaftlichen Hintergrund entstanden sind. Der zweite Unterabschnitt widmet sich der anderen Gruppe von Anwendungen. Diese umfassen diverse Benchmarks, die im wissenschaftlichen Umfeld entstanden sind und oftmals für einen speziellen Anwendungsfall entwickelt worden sind. Der letzte Unterabschnitt gibt Beispiele für wissenschaftliche Arbeiten, die einen Bedarf an Benchmarks für Smartphones haben. Die Arbeiten, die in diesem letzten Abschnitt vorgestellt werden, sind potentielle Einsatzgebiete für den Ansatz, der im Rahmen dieser Arbeit entwickelt wird.

2.1.1 Kommerzielle Benchmarks

Die Anwendungen zur Leistungsbewertung im Endkundenbereich können verschiedenste Kriterien oder Kombinationen dieser zur Leistungsbeurteilung heranziehen.

Ein Beispiel liefert die Benchmark-Sammlung *AnTuTu*[1]. Diese versucht, die Performanz eines Smartphones vollständig zu bewerten. Hierfür werden 12 Kriterien verwendet, welche in sechs Kategorien unterteilt sind. Die Kriterien umfassen unter anderem Gleitkomma- und Ganzzahl-Leistung, Speicheroperationen, Ein-/Ausgabe-Operationen, 2D/3D-Graphik und Multitasking. In jeder dieser Kategorien müssen spezielle Aufgaben gelöst werden. Basierend auf der Laufzeit oder

[1] Siehe `http://www.antutu.com/index.shtml`, aufgerufen am 28.05.2016

den absolvierten Iterationen in einem Zeitraum werden in jeder Kategorie Punkte vergeben. Die Summe aller Punkte aus allen Kategorien wird am Ende als Leistungsindex verwendet.

Ein anderes Beispiel ist die Applikation *CF-Bench*[2]. Diese ist auf die Leistungsbewertung von Prozessor und Hauptspeicher ausgelegt. Die Applikation verwendet 12 Kriterien, um die Performanz dieser beiden Komponenten zu berechnen. Dabei werden für manche Kriterien Punkte, wie bei *AnTuTu*, vergeben und für andere wird ein prozentuales erreichen des optimalen Wertes verwendet. Alle diese Werte fließen mit unterschiedlicher Gewichtung in die Gesamtpunktzahl ein.

Ein letztes Beispiel ist der 3D-Benchmark *GFXBench*[3]. Dieser verwendet 22 einzelne Tests, in denen eine Punktzahl erreicht werden kann. Dabei zielen die einzelnen Tests auf Bewertung unterschiedlicher Szenarien ab. Einige betrachten *onscreen* und *offscreen* Performanz bei der Darstellung diverser Szenerien. Ein Teil dieser Szenarien ist noch weiter spezialisiert, beispielsweise auf *Tesselation*[4] oder die exzessive Nutzung der ALUs. Andere Tests bewerten die *General Purpose Computation* Leistung durch die verarbeitete Datenmenge in diversen Szenarien. Diese Szenarien umfassen beispielsweise Gesichtserkennung, die Berechnung von Umgebungsverdeckung oder die Anwendung des Gaußschen Weichzeichner auf Bilder. Jeder dieser Tests verfügt über eine eigene Rangliste und es existiert auch keine (gewichtete) Gesamtbeurteilung.

Zwischen all diesen Beispielen gibt es Gemeinsamkeiten. Zum einen dienen sie alle dem Zweck die Leistung eines Smartphones bzw. einzelner Komponenten zu bewerten. Dabei ist der Grad der Spezialisierung frei wählbar. Es können einzelne Komponenten, Gruppen dieser oder ganze Geräte bewertet werden. Die andere große Gemeinsamkeit ist, dass alle diese Applikationen sogenannte synthetische Benchmarks verwenden. Dies bedeutet, dass die, zur Leistungsbewertung durchgeführten Aufgaben weder auf realen Nutzungsdaten von Anwendern, noch auf Nutzungsprofilen bestimmter Anwendungen basieren. Stattdessen werden sie künstlich erzeugt, um ein gewisses Nutzungsprofil nachzustellen und/oder eine möglichst große Teilmenge der verfügbaren Funktionen abzudecken. Es existiert eine Vielzahl von solch synthetischen Benchmarks, welche teilweise domänenspezifisch sind. Weitere Informationen über synthetische Benchmarks und Beispiele für diese können diversen wissenschaftlichen Arbeiten entnommen werden [9, 37, 34, 13].

Aus diesem Vorgehen ergeben sich verschiedene Probleme für die Aussagekraft von synthetischen Benchmarks. So ist es zum Beispiel möglich die erzielten Ergebnisse beim Benchmark zu verfälschen [36]. Dies kann beispielsweise über eine

[2]Siehe `https://www.chainfire.eu/`, aufgerufen am 28.05.2016

[3]Siehe `https://gfxbench.com/result.jsp`, aufgerufen am 24.10.2016

[4]Siehe `https://software.intel.com/en-us/articles/tessellation-for-opengl-es-31-on-android`, aufgerufen am 24.10.2016

extra hohe *CPU*-Frequenz, die im Normalbetrieb nie - oder nur für sehr kurze Zeiträume - erreicht werden kann, oder ein erhöhtes Temperaturlimit für die *CPU* geschehen. Genau dieses Vorgehen wurde in der Vergangenheit bei großen Herstellern von Smartphones beobachtet[5]. Die andere große Problematik besteht darin, dass die Aktionen, welche zur Leistungsbewertung verwendet werden, sich im besten Fall an realen Nutzungsdaten orientieren, diese aber nie genau wiedergeben. Außerdem ist insbesondere bei kommerziellen Benchmarks der Ursprung der Aufgaben, welche zur Leistungsbewertung durchgeführt werden müssen, selten bekannt. Hierdurch wird die Aussagekraft synthetischer Benchmarks geschmälert, da die Performanz in realen Anwendungsfällen höchstens abgeschätzt werden kann. Auch ist die Aussagekraft für den einzelnen Nutzer eingeschränkt, da sein persönliches Anwendungsprofil nur in den seltensten Fällen exakt durch den Benchmark wiedergegeben wird. Solch synthetische Benchmarks reichen im besten Fall für einen groben Vergleich einzelner Systeme bzw. Systemkonfigurationen. Ebenfalls können diese synthetischen Benchmarks für den Vergleich der maximalen Performanz in den Anwendungsfällen verwendet werden.

2.1.2 Wissenschaftliche Benchmarks

Die Benchmarks, die in diesem Abschnitt vorgestellt werden sind aus dem wissenschaftlichen Umfeld entstanden. Die Verfahren sind meist nicht ohne weiteres für Endkunden zugänglich. Häufig werden einzelne bis einige wenige Kriterien zur Bewertung eines speziellen Leistungsmerkmals verwendet. Es existieren aber auch Benchmark-Sammlungen, welche die Gesamtleistung eines Gerätes bewerten sollen. Das Spektrum der verfügbaren Verfahren ist also ähnlich breit gefächert wie bei den kommerziellen Benchmarks.

Als erstes wird die Arbeit von Carroll und Heiser betrachtet [3]. In ihrer Arbeit entwickeln die Autoren ein Verfahren, um möglichst detaillierte Energiemodelle für einzelne *Hardware*-Komponenten zu erzeugen. Durch die resultierenden Modelle sollen Aussagen über den Energieverbrauch gemacht werden. Dabei werden verschiedene Arten von Benchmarks zur Ermittlung der Modelle verwendet. Hochspezialisierte Mikro-Benchmarks werden verwendet, um den Energieverbrauch unter Last und im Ruhezustand für einzelne Komponenten zu ermitteln. Diese sind rein synthetischer Natur. Im Anschluss werden Makro-Benchmarks verwendet, um die Last in bestimmten Szenarien, beispielsweise dem Abspielen von Musik, nachzustellen. Diese Macro-Benchmarks wurden auf zwei verschiedene Weisen entworfen. Die eine Gruppe basiert auf realen Anwendungsszenarien, aber nicht auf konkre-

[5] Siehe `http://www.anandtech.com/show/7384/state-of-cheating-in-android-benchmarks`, aufgerufen am 27.09.2016

ten Applikationen. Somit werden keine Besonderheiten einzelner Applikationen erfasst. Stattdessen wird versucht das typische Verhalten von Applikationstypen nachzuahmen. Interaktive Applikationen, wie beispielsweise Webbrowser, werden durch *Traces* nachgestellt. Somit werden an dieser Stelle reale Nutzungsdaten für die Leistungsbewertung verwendet. Dabei werden keinerlei Angaben zu Ursprung und Quantität der verwendeten *Traces* gemacht, wodurch keine Möglichkeit besteht die Aussagekraft zu beurteilen.

Ein anderes Ziel verfolgt die Arbeit von Lin et al. [23]. Da es in *Android* möglich ist, Applikationen sowohl in *Java* als auch *C/C++* zu programmieren, haben die Autoren sich das Ziel gesetzt, die Performanz der beiden Programmiersprachen im Kontext von *Android* zu vergleichen. Dabei verwenden die Autoren 12 einzelne Tests, welche sowohl in *Java* als auch *C/C++* implementiert wurden. Jeder Test umfasst einen einfachen Algorithmus, wie die Berechnung der *Fibonacci*-Folge oder *Heapsort*. Durch Vergleiche der Ausführungszeit von nativem Programmcode und der *Java*-Implementierung wird eine Evaluation durchgeführt. Somit ist der Benchmark von synthetischer Natur und erhebt nicht einmal den Anspruch, auf realen Szenarien oder Daten zu basieren. Es werden simple Algorithmen in geringer Anzahl verwendet. Hierdurch wird die Aussagekraft der Ergebnisse beschnitten. Für weitere Informationen über die Dalvik VM, die in Android für die Ausführung von speziellem *Java-Bytecode* (sogenannter *Dex-Bytecode*) zuständig ist, sei auf die offizielle Dokumentation verwiesen[6].

Bei *AndroBench* handelt es sich um einen Benchmark, der darauf ausgelegt ist, die Performanz der Ein- und Ausgabe zu bewerten [20]. Zu diesem Zweck werden zufällige/sequentielle Ein- und Ausgaben und diverse *SQL*-Transaktionen durchgeführt. Die Methoden, um die Ein-/Ausgabe-Leistung zu messen, sind rein synthetischer Natur. Das Datenbankschema, welches für die *SQL*-Benchmarks verwendet wird entspricht dem einer nicht näher spezifizierten Applikation, um Kontakte zu verwalten. Durch die Beschränkung auf ein einzelnes reales Szenario und synthetische Daten ist die Aussagekraft höchstens unwesentlich besser als bei rein synthetischen Benchmarks. Dies gilt insbesondere, da die *SQL*-Operationen, die auf der Datenbank durchgeführt werden, synthetischer Natur sind und nur den Zweck haben diverse Aktionen mit festgelegter Anzahl auszulösen. Der Ursprung des Datenbankschemas wird nicht mit hinreichender Genauigkeit benannt und auch nicht weiter beschrieben. Hierdurch ist es nicht - oder nur schwerlich - möglich, irgendwelche Schlüsse über die allgemeine Leistung der Datenbank zu ziehen.

Als nächstes wird die Arbeit von Huang et al. untersucht [18]. Die Autoren entwickeln in ihrer Arbeit eine Benchmark-Sammlung mit dem Namen *Moby*. Dabei werden 10 Szenarien ausgewählt, welche die Autoren als typische Smartphone-

[6]Siehe `https://source.android.com/devices/tech/dalvik/`, aufgerufen am 29.09.2016

Nutzung einschätzen. Darin enthalten sind unter anderem die Nutzung sozialer Netzwerke, das Empfangen und Lesen von Emails und die Ausführung von Spielen. Für jedes Szenario wird die typische Last und deren Eingabe bestimmt. Anschließend wird für jedes Szenario eine Applikation als Repräsentant gewählt, welche zur Leistungsmessung im jeweiligen Szenario verwendet wird. Durch spezielle Tools zur Aufnahme und Wiedergabe von Nutzereingaben werden die einzelnen Benchmarks erzeugt. Nachteilig an diesem Ansatz ist, dass es für den Benutzer nicht auf einfache Art und Weise möglich ist, neue Applikationen bzw. Szenarien in die Benchmark-Sammlung aufzunehmen. Ebenfalls wird pro Szenario nur eine Aufnahme von Daten für die Bewertung verwendet, welche aber ausgetauscht werden kann. Außerdem muss beachtet werden, dass regelmäßig neue Applikationen für die einzelnen Szenarien ausgewählt werden müssen, da die Verbreitung von Applikationen sich schnell ändert und somit die Gefahr besteht, Applikationen mit geringer Verbreitung als repräsentativ zu verwenden. Da für jede neue Applikation eine eigene Aufnahme der Nutzung erzeugt werden muss, ergibt sich zusammen mit der qualifizierten Auswahl der Applikation ein relativ hoher Aufwand, um neue Applikationen in das Portfolio aufzunehmen.

Als letztes wird *AppScope* betrachtet, welches von Yoon et al. entwickelt wurde [38]. Hierbei handelt es sich um ein Werkzeug welches dynamisch den Energieverbrauch verschiedener *Hardware*-Komponenten misst. Mit diesen Messungen werden Aussagen über die Auswirkungen auf den Akku von verschiedenen Applikationen und Aktionen gemacht. Dabei wurden diverse Anpassungen am *Android* Betriebssystem gemacht, um die benötigten Daten auf Basis von Ereignissen zu erfassen. Nach ausgiebiger Recherche stellte sich heraus, dass der Quellcode von *AppScope* nicht öffentlich zugänglich ist. Dies ist darin begründet, dass das Projekt von der Firma *Samsung Electronics* gefördert wird und aus diesem Grund der Geheimhaltung unterliegt [7]. Wäre dies nicht der Fall, könnte *AppScope* eine gute Basis für ein generisches Benchmark-Verfahren bilden, da bereits viele relevante Stellen zum Zwecke der Energiemessung angepasst wurden. Daher wäre der notwendige Aufwand, um aus dem Verfahren zur Energiemessung eine allgemeine Leistungsmessung zu erhalten überschaubar.

Bei fast allen Ansätzen, die in diesem Abschnitt vorgestellt wurden, handelt es sich um synthetische Benchmarks. In vielen Fällen ist der Ursprung der Daten mehr oder weniger konkret bekannt. Viele der verwendeten Daten basieren auf realen Anwendungen und deren Lastprofilen. Andere Verfahren stützen sich auf rein synthetische Daten. Hierdurch ist die Aussagekraft im Vergleich zu den kommerziellen Benchmarks aus dem vorherigen Abschnitt größer, da zumindest der Ursprung der Daten mit einbezogen werden kann. Somit kann identifiziert werden, welche

[7] Siehe `https://www.youtube.com/watch?v=K1ZuRhAicL8`, aufgerufen am 25.10.2016

Einsatzgebiete die einzelnen Benchmarks repräsentieren und wie sehr sie sich für das Szenario, in dem sie angewandt werden sollen, eignen. Dieser Vorteil ist bei den kommerziellen Benchmarks nicht gegeben. Das Problem, das alle Verfahren gemeinsam haben, besteht darin, dass nur durch größeren Aufwand neue Lastszenarien in die Benchmarks eingefügt werden können. Im Vergleich zu den kommerziellen Benchmarks sind die Bewertungskriterien wesentlich klarer definiert. In der Regel werden direkte Vergleiche der Laufzeit oder des Energieverbrauchs verwendet. Es wird somit auf die Abstrahierung durch eine Punkteskala und ggf. unbekannte Gewichtungen verzichtet.

2.1.3 Wissenschaftliche Anwendungsgebiete für Benchmarks

Abschließend für dieses Unterkapitel soll betrachtet werden, wie Benchmarks aussehen, die in wissenschaftlichen Arbeiten verwendet werden. Dabei werden im Folgenden ausschließlich Arbeiten vorgestellt, die einen eigenen Benchmark entwerfen und verwenden. Diese haben häufig ein sehr spezielles Anwendungsgebiet und werden den wissenschaftlichen Benchmarks, welche im vorherigen Abschnitt vorgestellt wurden, vorgezogen. Dies ist zum Teil auch in der großen Anzahl an *Hardware*-Komponenten und Anwendungsmöglichkeiten, über die Smartphones verfügen, begründet.

Zuerst wird die Arbeit von Qian et al. betrachtet [30]. Die Autoren versuchen im Rahmen der Arbeit die Zustandsmaschine, die vom Mobilfunkstandard *UMTS* verwendet wird, zu optimieren. Dabei werden verschiedene Aspekte betrachtet, wie der Energieverbrauch des Endgeräts und der *Overhead* durch das Netzwerkmanagement. Gleichzeitig ist es Ziel der Arbeit, Limitierungen eben dieser Zustandsmaschine aufzuzeigen und so Ansätze zur Verbesserung zu liefern. Um die durchgeführten Optimierungen zu evaluieren werden Netzwerkdaten abgespielt, die über einen Zeitraum von drei Stunden direkt von einem *UMTS*-Anbieter aufgezeichnet wurden. Somit verwendet der Benchmark reale Daten, was zunächst positiv zu bewerten ist. Problematisch ist, dass die Daten nur über einen kurzen Zeitraum aufgezeichnet wurden und dass nur ein Datensatz verwendet wird. Hierdurch ist es nicht möglich sämtliches Nutzungsverhalten abzubilden. Darüber hinaus ist es möglich, dass durch besondere Ereignisse - beispielsweise eine Sportveranstaltung - das Nutzungsverhalten im Aufnahmezeitraum von der durchschnittlichen Nutzung abweicht und so ein verzerrtes Bild von der realen durchschnittlichen Nutzung erzeugt. Hierdurch wird die Aussagekraft der Daten geschmälert, da die Evaluierung unter Umständen mit Daten durchgeführt wird, welche in der Realität nur selten auftreten. Somit würde auch die Zustandsmaschine für ein unwahrscheinliches Szenario optimiert.

Als nächstes wird der Algorithmus *SALSA* vorgestellt, welcher in der Arbeit

von Ra et al. entwickelt wird [31]. Dieser versucht durch gezielte Verzögerung von Netzwerkverkehr einen möglichst optimalen Kompromiss zwischen Energieverbrauch und Verzögerung zu finden. Der Algorithmus ist im Umfeld des Projekts *Urban Tomography* entstanden [21]. Im Rahmen des Projektes werden von den Teilnehmern in unregelmäßigen Abständen Videos von bestimmten Orten an eine zentrale Stelle geschickt. Dabei sind die Daten, die zur Evaluation von *SALSA* verwendet werden, von den Teilnehmern des Projekts gesammelt worden. Die Daten lassen sich in zwei Gruppen unterteilen. Die eine Gruppe wird aus Daten gebildet, die Ankunftsraten von Videoübertragungen beschreiben, welche im Rahmen des Projektes anfallen. Der andere Teil der Daten wird implizit gesammelt und enthält Informationen darüber, zu welchen Zeitpunkten welche Arten von Netzwerkverbindungen zur Verfügung stehen. Insgesamt wurden auf diese Weise ca. 100 Datensätze gesammelt, welche verwendet werden, um Muster zu erzeugen. Diese werden wiederum im Benchmark verwendet. Auch bei dieser Arbeit ist die Nutzung realer Daten zur Evaluation positiv zu bewerten. Nachteilig auf die Quantität der Daten wirkt sich die geringe Anzahl an Nutzern aus, welche die Daten sammeln. Auch wenn die Daten, die zur Leistungsmessung verwendet werden, den Anwendungsfall vollständig abdecken und somit über eine hohe Qualität verfügen, leidet die Aussagekraft unter der geringen Quantität, da kein allgemeines Nutzerverhalten in den Mustern gefunden werden kann. Im besten Fall ist dies auf die Gruppe der Projektteilnehmer zugeschnitten und erfasst deren Nutzungsverhalten.

Das letzte Beispiel stellt eine allgemeine Studie zur Nutzung von Smartphones dar und ist von Falaki et al. durchgeführt worden [11]. Im Rahmen der Studie werden Nutzungsdaten von 255 Nutzern gesammelt. Bei einer anschließenden Analyse werden verwendete Applikationen und deren Auswirkungen auf den Netzwerkverkehr und den Batteriezustand identifiziert. Darüber hinaus sollen Unterschiede, die in der Nutzung von Smartphones existieren erkannt und aufgezeigt werden. Von den Studienteilnehmern verwenden 222 Smartphones mit einem Betriebssystem der *Windows Phone* Familie. Sie erfassen Daten über einen Zeitraum von 8 bis 28 Wochen. Dabei enthalten die Daten nur Informationen über den Bildschirmzustand (an- bzw. ausgeschaltet) und die verwendeten Applikationen. Die übrigen 33 Teilnehmer der Studie verwenden *Android*-Smartphones und zeichnen Daten für 7 bis 21 Wochen auf. Dabei werden von diesen Nutzern zusätzlich Netzwerkverkehr und der Batteriezustand erfasst. Die Summe der Datensätze kann verwendet werden, um Aussagen über verwendete Applikationen und Nutzungszeiten zu machen. Sollen zusätzlich Schlussfolgerungen über den Netzwerkverkehr und Änderungen des Batteriezustands, welche durch diese Applikationen verursacht werden, gemacht werden, können nur die 33 Datensätze von *Android*-Nutzern verwendet werden. Nur diese Datensätze enthalten die dafür notwendigen Daten. An dieser Stelle muss der lange Zeitraum, in dem Daten erfasst werden, positiv hervorgehoben werden.

Denn dieser ist wesentlich besser geeignet, das allgemeine Verhalten eines Benutzers zu bestimmen, als die kurzen Messzeiträume, welche von anderen Arbeiten verwendet werden. Somit sind die Quantität der Daten über den einzelnen Nutzer und Aussagen über diesen hinreichend gegeben. Problematisch ist die geringe Quantität der relevanten Datensätze, wenn es um allgemeine Schlussfolgerungen bezüglich Netzwerkverkehr und Batteriezustand geht. Dies ist darin begründet, dass viele Faktoren (beispielsweise die Version des Betriebssystems und die *Hardware* des Smartphones) Einfluss auf diese Aspekte haben. Darüber hinaus ist es nicht möglich, alle Unterschiede in der Nutzung von Smartphones zu identifizieren, da die Stichprobe hierfür zu klein ist. Die Wahrscheinlichkeit, dass ein beliebiges Nutzungsverhalten nicht in der Stichprobe enthalten ist, ist groß.

Zusammenfassend lässt sich sagen, dass es einige Ansätze zur Leistungsmessung und -bewertung von Smartphones gibt. Einige dieser Ansätze sind auch im wissenschaftlichen Umfeld entstanden und weisen durch die angewandten wissenschaftlichen Methoden eine höhere Qualität als andere (kommerzielle) Ansätze auf. Viele der existierenden Ansätze haben einen speziellen Anwendungszweck und sind nicht für allgemeine Benchmarks geeignet. Oftmals werden rein synthetische Daten oder reale Nutzerdaten mit geringer Quantität verwendet. Die Kombination dieser Eigenschaften sorgt für die aktuell vorherrschende Situation. Zur Evaluation wissenschaftlicher Arbeiten entwickeln viele Autoren ein eigenes Benchmark-Verfahren, welche den eigenen Ansprüchen genügen. Oftmals weisen diese Verfahren dann identische Probleme zu den existierenden Benchmarks auf. Es werden synthetische Daten verwendet oder die verwendeten realen Daten haben eine zu geringe Quantität, um eine wirkliche Aussagekraft zu besitzen. Aus dieser Situation lässt sich ableiten, dass ein Bedarf an einem Verfahren zur Generierung von Benchmarks existiert. Dieses Verfahren sollte möglichst viele Daten des Smartphones und der gewählten Applikation erfassen. Wichtig ist in diesem Zusammenhang auch die Fähigkeit, die aufgenommenen Daten im Anschluss wieder abspielen zu können. Ein solches Verfahren würde eine große Zahl von Anwendungsgebieten abdecken, da die jeweils relevanten Applikationen aufgezeichnet werden könnten. Schließlich wäre es eine wünschenswerte Eigenschaft für das Verfahren, dass auf einfache Art und Weise neue Daten zum Aufzeichnen hinzugefügt werden können. Hierdurch wäre es möglich Szenarien, die durch den Ansatz nicht abgedeckt werden durch die Erfassung zusätzlicher Daten doch in das Anwendungsspektrum aufzunehmen.

2.2 Android

Bei *Android* handelt es sich um ein mobiles Betriebssystem, welches 2008 von der Firma *Google* veröffentlicht wurde[8]. Es wird seitdem stetig weiterentwickelt und ist aktuell in der Version 7.0 (*Nougat*) verfügbar[9]. *Android* ist das mobile Betriebssystem mit dem größten Marktanteil und erreichte im ersten Quartal 2016 eine Verbreitung von 84% [12]. Ein großer Teil des Betriebssystems ist quelloffen und kann somit von jedem Entwickler angepasst werden. Dies resultiert in einer großen Menge von *Android* Betriebssystemen (sogenannte *Custom ROMs*). Dieser quelloffene Teil von *Android* ist das sogenannte *Android Open Source Project* (*AOSP*) und bildet die Basis für sämtliche *Custom ROMs*[10]. Das *Custom ROM* mit der größten Nutzerbasis ist *CyanogenMod*[11] [27]. Es unterstützt eine Vielzahl verschiedener Smartphones[12]. Darüber hinaus wurde es bereits als offizielles Betriebssystem auf Smartphones ausgeliefert (unter dem Namen *Cyanogen OS*)[13].

Abbildung 2.1 zeigt eine schematische Darstellung des *Android* Betriebssystems. Dabei wird das Betriebssystem von fünf Komponenten-Gruppen gebildet. Diese Gruppen sind auf vier verschiedenen Ebenen angesiedelt. Je weiter oben sich eine Ebene in der Abbildung befindet, desto größer ist der Abstraktionsgrad von der *Hardware*. In der Abbildung sind die einzelnen Ebenen mit Komponenten gefüllt, welche sie enthalten. Dabei werden die Komponenten nur auszugsweise dargestellt. Es wird kein Anspruch auf Vollständigkeit erhoben, die abgebildeten Komponenten dienen nur der Veranschaulichung.

Im weiteren Verlauf dieses Unterkapitels werden die fünf Gruppen einzeln vorgestellt. Dabei werden die einzelnen Gruppen aufsteigend nach ihrem Abstraktionsgrad zur *Hardware* vorgestellt. Somit wird zuerst der *Android Kernel* vorgestellt und zuletzt die Gruppe der *Android Applications*.

2.2.1 Android Kernel

Der Kern des *Android* Betriebssystems wird durch eine modifizierte Version des *Linux Kernels* gebildet[15]. Hierdurch besteht eine gewisse Ähnlichkeit zwischen *An-*

[8] Siehe `http://developer.android.com/guide/basics/what-is-android.html`, aufgerufen am 26.09.2016
[9] Siehe `https://www.android.com/versions/nougat-7-0/`, aufgerufen am 04.10.2016
[10] Siehe `https://source.android.com/`, aufgerufen am 04.10.2016
[11] Siehe `http://www.cyanogenmod.org/`, aufgerufen am 29.05.2016
[12] Siehe `http://wiki.cyanogenmod.org/w/Devices`, aufgerufen am 04.10.2016
[13] Siehe `https://cyngn.com/cyanogen-os`, aufgerufen am 04.10.2016
[14] Siehe `http://source.android.com/security/index.html`, aufgerufen am 13.10.2016
[15] Siehe `https://sourceware.org/systemtap/wiki/SystemTapWithSelfBuiltKernel`, aufgerufen am 04.10.2016

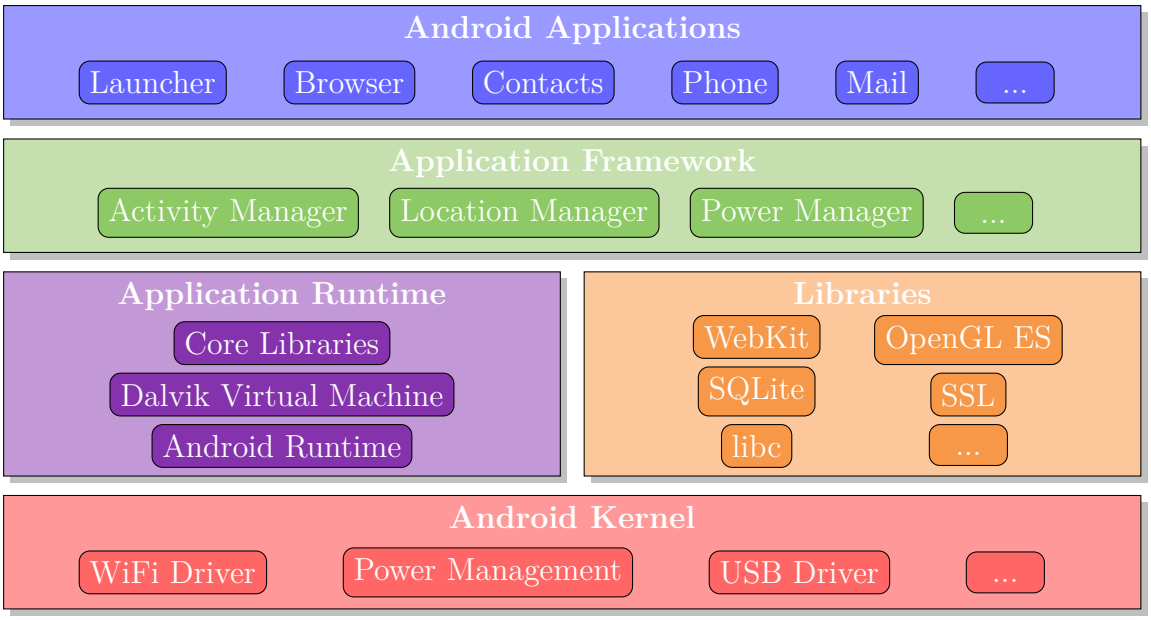

Abbildung 2.1: Schematische Darstellung des Android Betriebssystems, welche alle essentiellen Systemkomponenten zeigt und auf vier Ebenen aufteilt. Angelehnt an eine Graphik der offiziellen *Android*-Dokumentation[14]

droid und klassischen *Linux* Betriebssystemen. Der *Kernel* bietet die Basis für die Kommunikation zwischen *Hardware* und *Software*. Er hat, im Vergleich zu allen anderen Komponenten des Betriebssystems, die geringste Abstraktion von der eigentlichen *Hardware*. Sowohl beim *Kernel* von *Linux*, als auch bei den Versionen, die *Android* verwendet, handelt es sich um monolithische *Kernel* [32]. Jede Version von *Android*, die aus dem *Android Open Source Project* hervorgeht wird gegen eine spezifische Version des *Kernels* gebaut[16]. Der verwendete *Kernel* ist austauschbar, was aber einen gewissen Aufwand mit sich bringt[17]. Durch die Modifikation des *Kernels* ergeben sich weitreichende Möglichkeiten zur Modifikation des allgemeinen Verhaltens des Betriebssystems.

2.2.2 Application Runtime

Um Applikationen des Systems und des Nutzers ausführen zu können wurde *Android* ursprünglich mit der sogenannten *Dalvik Virtual Machine* ausgeliefert. Eine solche virtuelle Maschine ist notwendig, da der größte Teil der *Android*-Applikationen in *Java* geschrieben ist. Normalerweise wird *Javacode* durch den *Compiler* in *Bytecode* übersetzt, welcher durch die *Java Virtual Machine* ausgeführt wird [15]. Bei der *Java Virtual Machine* handelt es sich um einen Kellerautomaten [24]. Da *Android* für jede auszuführende Applikation eine eigene virtuelle Maschine startet ist ein Verfahren notwendig, das ressourcenschonender und schneller ist. Aus diesem Grund wurde die *Dalvik Virtual Machine* entwickelt, welche eine Registermaschine ist[18]. Dies ist der größte Unterschied zwischen der *Java Virtual Machine* und der *Dalvik Virtual Machine*. Der *Compiler* erzeugt aus diesem Grund speziellen *Dex-Bytecode*, welcher durch die *Dalvik Virtual Machine* ausgeführt werden kann. Dabei findet die Ausführung, genau wie bei *Java*, *Just-In-Time* statt.

Mit *Android* Version 5.0 (*Lollipop*) wurde die *Android Runtime* eingeführt, welche schrittweise die *Dalvik Virutal Machine* ersetzten soll. Diese verwendet zugunsten der Laufzeit *Ahead-Of-Time* Übersetzung der Applikationen. Dabei werden während der Installation von Applikationen aus dem *Dex-Bytecode* die ausführbaren Dateien für die *Android Runtime* erzeugt. Diese sind auf die jeweilige *Hardware* zugeschnitten sind.

Darüber hinaus sind diverse Bibliotheken Teil der *Application Runtime*. Diese Bibliotheken sind notwendig, um die Funktionalität der *Dalvik Virtual Machine*

[16] Siehe `https://en.wikipedia.org/wiki/Android_version_history`, aufgerufen am 05.10.2016

[17] Siehe `https://source.android.com/source/building-kernels.html`, aufgerufen am 05.10.2016

[18] Siehe `https://sites.google.com/site/io/dalvik-vm-internals`, aufgerufen am 27.10.2016

und der *Android Runtime* zu gewährleisten.

2.2.3 Libraries

Auf derselben Abstraktionsebene wie die *Application Runtime* ist eine große Menge von Bibliotheken angesiedelt. Diese umfassen viele Standards und Funktionen, welche von Applikationen häufig genutzt werden. Beispiele für diese Bibliotheken sind *OpenGL ES* zur Darstellung von 3D Inhalten[19], die *Web Browser Engine WebKit*[20], *SSL*-Funktionalität und eine Datenbank in Form von *SQLite*, die an vielen Stellen im Betriebssystem verwendet wird[21].

2.2.4 Android Application Framework

Das *Android Application Framework* enthält eine große Menge von *Application Programming Interfaces* (*APIs*), welche zur Entwicklung von Applikationen verwendet werden können[22]. Es gibt diverse Schnittstellen zur graphischen Gestaltung von Applikationen. Darüber hinaus existieren viele *APIs*, welche die zuvor genannten Bibliotheken nochmals kapseln und dem Entwickler möglichst komfortabel Zugriff auf die Funktionen gewähren. Außerdem sind Schnittstellen zu *Hardware*-Komponenten, wie dem *GPS*-Modul und dem Netzwerk-Modul enthalten. Diese abstrahieren von der *Hardware*, damit auf verschiedenen *Hardware*-Konfigurationen eine konsistente Schnittstelle für die jeweilige Komponenten und somit keine Notwendigkeit für Anpassungen besteht. Zuletzt seien die Schnittstellen zur Interaktion mit Betriebssystemkomponenten genannt, wie beispielsweise dem *PowerManager*. Nachfolgend werden einzelne *APIs* und andere Komponenten des *Android Application Frameworks* vorgestellt.

Der zuvor erwähnte *PowerManager* erlaubt den Zugriff auf *Wakelocks* und bietet Informationen über den aktuellen Grad der Aktivität des Smartphones[23]. Ebenfalls ist es möglich, durch diese Schnittstelle einen Neustart des Geräts auszulösen.

Zum abspielen von Musik- und Videodaten kann der *MediaPlayer* verwendet werden[24]. Dabei spielt es keine Rolle, ob die Daten auf dem Speicher vorliegen oder aus dem Internet bezogen werden. Sobald der Schnittstelle der Fundort der

[19] Siehe `https://www.khronos.org/opengles/`, aufgerufen am 08.10.2016
[20] Siehe `https://www.webkit.org/`, aufgerufen am 28.10.2016
[21] Siehe `https://www.sqlite.org/`, aufgerufen am 28.10.2016
[22] Siehe `https://developer.android.com/about/index.html`, aufgerufen am 05.10.2016
[23] Siehe `https://developer.android.com/reference/android/os/PowerManager.html`, aufgerufen am 13.10.2016
[24] Siehe `https://developer.android.com/reference/android/media/MediaPlayer.html`, aufgerufen am 13.10.2016

abzuspielenden Daten mitgeteilt wurde, kann der Vorgang des Abspielens komfortabel gesteuert werden. Das eigentliche Beziehen und Verarbeiten der Daten erledigt der *MediaPlayer* im Hintergrund, ohne Notwendigkeit für weitere Eingriffe. Weitere Informationen über den *MediaPlayer* und seine Funktionsweise können in Kapitel 32 des Buchs von Murphy gefunden werden [26].

Eine ähnlich komfortable Abstraktion wird durch den *DownloadManager* geboten[25]. Unter Angabe, welche Inhalte aus dem Internet heruntergeladen werden sollen, übernimmt der *DownloadManager* den Verbindungsaufbau sowie Empfang und Speicherung der Daten. All dies geschieht im Hintergrund. Heruntergeladene Dateien, auch solche, die bereits fertig heruntergeladen wurden, können zentral in einer speziellen Applikation verwaltet werden.

Ein weiterer wichtiger Bestandteil des *Android Application Framworks* ist das Berechtigungssystem[26]. Dieses dient dazu, auf Applikationsebene diverse Rechte, um bestimmte Aktionen auszuführen, zu vergeben. Die Berechtigungen sind dabei in Gruppen zusammengefasst und können von einer Applikation feingranular angefordert werden. Beispielsweise sind Berechtigungen notwendig, wenn auf sensible Nutzerdaten, wie den Standort oder Kontaktdaten, zugegriffen werden soll. Ebenfalls müssen Berechtigungen angefordert werden, wenn durch die ausgeführten Aktionen Kosten für den Nutzer entstehen könnten. Dies betrifft beispielsweise Telefonie-Funktionen und den Zugriff auf mobile Internetverbindungen. Dabei werden die Berechtigungen zum Installationszeitpunkt angefordert und müssen durch den Nutzer gewährt werden. Seit *Android* Version 6.0 (*Marshmallow*) werden Berechtigungen zur Laufzeit angefordert, wenn die jeweilige Aktion ausgelöst werden soll[27]. Somit können die Rechte einzeln und situationsbedingt gewährt oder verweigert werden.

Das zentrale Element der meisten Applikationen ist die *Activity*. Hierbei handelt es sich um eine Komponente, die Inhalte auf dem Bildschirm anzeigen kann und Interaktionen mit dem Benutzer erlaubt[28]. Dabei kann eine Applikation aus mehreren *Activities* und diversen anderen Komponenten bestehen. So kann eine *Activity* aus beliebig vielen Fragmenten bestehen, welche einen Teil der Anzeige kapseln und nach Bedarf ausgetauscht werden können[29]. Aufgaben, die lange Laufzeiten

[25] Siehe `https://developer.android.com/reference/android/app/DownloadManager.html`, aufgerufen am 13.10.2016

[26] Siehe `https://developer.android.com/guide/topics/security/permissions.html`, aufgerufen am 05.10.2016

[27] Siehe `https://developer.android.com/training/permissions/requesting.html`, aufgerufen am 29.10.2016

[28] Siehe `https://developer.android.com/training/basics/activity-lifecycle/index.html`, aufgerufen am 13.10.2016

[29] Siehe `https://developer.android.com/guide/components/fragments.html`, aufgerufen am 29.10.2016

ohne Nutzerinteraktion aufweisen, werden am besten im Hintergrund erledigt, um nicht die Nutzung des Smartphones zu blockieren. Hierfür bieten sich *Services* und verschiedene Abwandlungen dieser an[30].

Von besonderer Wichtigkeit ist die Schnittstelle, welche die sogenannten *Intents* bereitstellt[31]. Hierbei handelt es sich um einen asynchronen Nachrichtendienst. Dieser erlaubt die Kommunikation zwischen verschiedenen Applikationen bzw. Applikationen und dem Betriebssystem [26, Seite 4]. Ebenfalls ermöglicht es die Kommunikation zwischen einzelnen Komponenten einer Applikation, beispielsweise zwischen zwei *Activities*. Es existieren auch Möglichkeiten, um *Broadcasts* und persistente Nachrichten abzusetzen.

Zuletzt wird das *Native Development Kit* vorgestellt[32]. Dieses stellt eine Alternative zum *Android Applikation Framework* dar und ist funktional äquivalent zu diesem. Der Unterschied besteht darin, dass es vollständig in *C/C++* programmiert wurde. Hierdurch sind Entwickler nicht auf die Verwendung von *Java* beschränkt. Das *Native Development Kit* ist besonders für Applikationen geeignet, in denen die Performanz kritisch ist, da eine Leistungssteigerung gegenüber dem *Android Application Framework* existiert [23].

2.2.5 Android Applications

Die größte Abstraktion von der *Hardware* bieten die eigentlichen Applikationen. Sie sind die einzige Komponente, mit denen der Nutzer direkt interagieren kann. Es gibt verschiedene Kategorien von Applikationen.

Zum einen gibt es die System-Applikationen. Diese werden gemeinsam mit dem Betriebssystem ausgeliefert und können vom Nutzer nicht entfernt werden. Für System-Applikationen besteht die Möglichkeit, Berechtigungen zu besitzen, welche für normale Applikationen nicht zugänglich sind. Diese Berechtigungen umfassen Aktionen die globalen Zugriff auf Daten, Funktionen des *Kernels* und Veränderungen an sicherheitskritischen Einstellungen erlauben. Diese Funktionen sind notwendig, da von vielen System-Applikationen grundlegende Funktionalitäten bereitgestellt werden. Ein Beispiel ist die *Settings*-Applikation, welche die zentrale Stelle für sämtliche Konfigurationsmöglichkeiten des Betriebssystems darstellt.

Zum anderen gibt es die Nutzer-Applikationen. Diese können vom Benutzer nach Bedarf installiert und deinstalliert werden. Dabei geschieht die Installation in der

[30] Siehe `https://developer.android.com/guide/components/services.html`, aufgerufen am 29.10.2016

[31] Siehe `https://developer.android.com/guide/components/intents-filters.html`, aufgerufen am 05.10.2016

[32] Siehe `https://developer.android.com/ndk/index.html`, aufgerufen am 05.10.2016

Regel über einen *App-Store*, beispielsweise den *Google Play Store*[33]. Dies bietet diverse Vorteile, beispielsweise eine zentrale Bezugsquelle von Applikationen und Verwaltung für Aktualisierungen. Darüber hinaus ist es möglich Applikationen direkt aus *Android Package*-Dateien (*apk*) zu installieren.

Jeder Applikation wird von Seiten des Betriebssystems eine *SQLite* Datenbank zur Verfügung gestellt. Diese kann zur Speicherung von Daten verwendet werden. Ebenfalls ist es jeder Applikation möglich den vollen Umfang des *Android Application Frameworks* zu verwenden[34].

Eine Charakteristik, die sich alle Applikationen teilen, ist der gemeinsame Lebenszyklus. Eine Besonderheit des Lebenszyklus von *Android*-Applikationen ist, dass die Lebensdauer nicht durch die Applikation selber bestimmt wird. Stattdessen entscheidet das Betriebssystem auf Basis der Lebenszyklen aller Komponenten, welche die Applikation verwendet, welche Transitionen im Lebenszyklus der Applikation notwendig sind[35]. In jedem Fall is die Applikation Aktiv, solange mindestens eine ihrer Komponenten aktiv ist. Darüber hinaus gibt es die Unterscheidung zwischen sichtbaren, im Vordergrund bzw. Hintergrund laufenden und gesicherten (pausierten) Applikationen.

Der Fokus des Lebenszyklus auf die verwendeten Komponenten ist damit zu begründen, dass Applikationen eine große Anzahl von Komponenten besitzen können, welche ggf. im Hintergrund Arbeit verrichten, ohne das überhaupt eine Nutzerinteraktion notwendig ist. Die am häufigsten verwendete Komponente, die im Hintergrund Aufgaben ausführt, ist der *Service*.

Bei *Activities* handelt es sich um das zentrale Element der sichtbaren Komponenten, mit denen der Nutzer interagieren kann. Am einfachsten lässt sich eine *Activity* als Bildschirmseite beschreiben. Sie kann beliebige Inhalte und Komponenten enthalten und auf vielfältige Art mit dem Benutzer interagieren (oftmals durch die enthaltenen Komponenten). Aus diesem Grund wird der Lebenszyklus dieser Komponente im Folgenden etwas genauer betrachtet. Abbildung 2.2 zeigt den Lebenszyklus einer *Activity* in minimal vereinfachter Form, dargestellt durch einen Zustandsautomaten. Die grün gefärbten Knoten markieren die aktiven Zustände, wogegen gelb und rot für Inaktivität, bzw. eine beendete *Activity* stehen. Für zerstörte *Activities* wird die Farbe grau verwendet. Zunächst wird eine neue

[33] Siehe `https://support.google.com/googleplay/answer/190860?hl=en`, aufgerufen am 28.10.2016

[34] Siehe `https://sourceware.org/systemtap/wiki/SystemTapOnAndroidARM`, aufgerufen am 29.10.2016

[35] Siehe `https://developer.android.com/guide/topics/processes/process-lifecycle.html`, aufgerufen am 29.10.2016

[36] Siehe `https://developer.android.com/training/basics/activity-lifecycle/starting.html`, aufgerufen am 20.11.2016

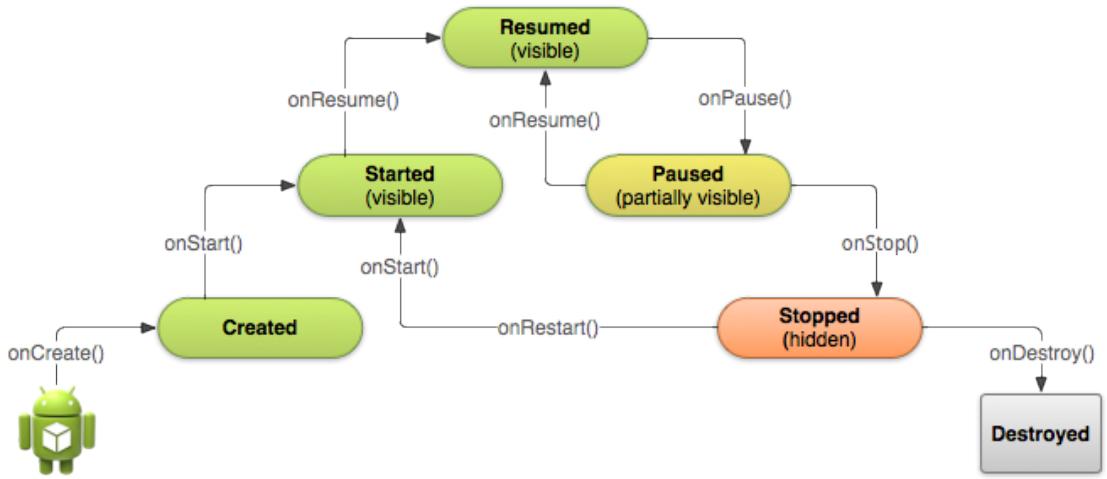

Abbildung 2.2: Darstellung des *Activity*-Lebenszyklus. Der offiziellen *Android*-Dokumentation entnommen[36].

Activity erzeugt, im Anschluss gestartet und die Ausführung fortgesetzt. Dabei entspringt die Bezeichnung des Fortsetzens dem Lebenszyklus, da die Ausführung an dieser Stelle formal gestartet und nicht fortgesetzt wird. Die Übergänge zwischen den Zuständen *Created* & *Started*, sowie *Started* & *Resumed* werden automatisch ausgelöst, sobald die Ausführung der Funktionen, welche die Transitionen auslösen, abgeschlossen ist. Im Rahmen dieser Funktionen werden Vorbereitungen gemacht, um eine ordnungsgemäße Ausführung zu gewährleisten. In allen drei diesen Zuständen ist die *Activity* aktiv und in den Zuständen *Started* und *Resumed* auch sichtbar. Der Zustand *Resumed* ist auch der, in dem die eigentliche Ausführung von Aktionen und Entgegennahme von Nutzerinteraktionen stattfindet. Zwischen den Zuständen *Paused* & *Resumed* kann beliebig hin und her gewechselt werden. Die laufende Ausführung wird in *Paused* unterbrochen und die *Activity* ist zumindest in Teilen sichtbar. Diese partielle Sichtbarkeit kommt beispielsweise dann Zustande, wenn eine durchsichtige *Activity* eine andere verdeckt. Ebenfalls kann eine pausierte Applikation vollständig gestoppt werden. In diesem Zustand wird die Ausführung abgebrochen und kann nicht mehr fortgeführt werden, ohne erneut den Zustand *Started* zu erreichen. Dies liegt daran, dass im Zustand *Stopped* bereits einige Ressourcen freigegeben werden, aber die *Activity* noch existiert. Somit ist es möglich aus diesem Zustand einen Neustart einzuleiten, oder die *Activity* vollständig zu zerstören und sämtliche verwendeten Ressourcen freizugeben.

```
1  probe begin {
2      printf("Hello World\n")
3  }
4
5  probe syscall.open {
6      printf("Opened %s", $filename)
7  }
8
9  probe end {
10     printf("Bye World\n")
11 }
```

Abbildung 2.3: Beispiel für ein *SystemTap*-Skript

2.3 SystemTap

Bei *SystemTap* handelt es sich um ein quelloffenes Werkzeug zur Instrumentierung des Betriebssystems [10, 29]. *SystemTap* kann auf verschiedenen Prozessorarchitekturen und mit diversen Versionen des *Linux Kernels* verwendet werden. Da der *Android Kernel* eine modifizierte Version des *Linux Kernels* ist, kann SystemTap auch hier genutzt werden. Zur Instrumentierung werden spezielle Skripte verwendet. Diese Skripte sind in einer *C*-ähnlichen Sprache geschrieben und werden für den Einsatz in *Kernel*-Module übersetzt[37]. Dabei muss für jeden *Kernel*, der instrumentiert werden soll, ein eigenes Modul erzeugt werden. Von *SystemTap* gehen nur wenige Anforderungen an den *Kernel* aus. Es ist nur notwendig, über die Konfiguration des *Kernels* die Daten verfügbar zu machen und dynamisches Laden und Entladen von Modulen zu erlauben [19].

Nachfolgend soll durch ein minimales Beispiel der Aufbau eines *SystemTap*-Skriptes erläutert werden. Dafür zeigt Abbildung 2.3 ein Beispiel für ein einfaches Skript. Es besteht aus drei verschiedenen *Probes*. Jede *Probe* wird durch das Schlüsselwort `probe` eingeleitet und besteht aus einem *Probe Point* und einem *Handler*. Der *Probe Point* definiert welche Stellen durch die jeweilige *Probe* instrumentiert werden sollen. In diesem Beispiel werden die zwei speziellen *Probe Points*, `begin` und `end` verwendet. Diese werden beim Laden und Entladen als erstes, bzw. als letztes ausgeführt und dienen der Initialisierung und dem anschließenden Aufräumen von Daten. Darüber hinaus wird der Systemaufruf zum Öffnen von Dateien durch den *Probe Point* `syscall.open` instrumentiert. Es existieren zahlreiche Möglichkeiten, um *Probe Points* zu definieren. Der *Handler* ist die Folge von Aktionen, die ausgeführt wird, sobald ein *Probe Point* ausgelöst wird. In diesem Beispiel werden

[37] Siehe `http://www.tldp.org/LDP/lkmpg/2.6/html/x40.html`, aufgerufen am 05.10.2016

nur Ausgaben produziert. Dabei wird beim instrumentierten Systemaufruf auf den Übergabeparameter `filename` zugegriffen. Es ist ebenfalls möglich auf eventuelle Rückgabewerte zuzugreifen. Für einen Überblick über den vollständigen Funktionsumfang von *SystemTap* sei auf die offizielle *Tapset*-Referenz und die Spezifikation der Sprache verwiesen [33, 6].

3 Anforderungen

Dieses Kapitel erläutert die Anforderungen, die an das Benchmark-Verfahren gestellt werden, welches im Rahmen dieser Arbeit entwickelt wird. Dabei werden zunächst die harten Anforderungen aufgezeigt und im Anschluss die weichen. Die harten Anforderungen sind für die Güte des Ansatzes kritisch. Diese Anforderungen werden im Rahmen der Evaluation verwendet, um die Güte des Verfahrens zu bewerten. Aus diesem Grund sollen die harten Anforderungen vollständig erfüllt werden. Bei den weichen Anforderungen handelt es sich nicht um primäre Bewertungskriterien, welche als Merkmal der Güte betrachtet werden. Sie besitzen aber so viel Relevanz, dass sie nicht vollends außer Acht gelassen werden können.

3.1 Harte Anforderungen

Zunächst werden die Anforderungen definiert, die zwingend in die Entwicklung des Verfahrens integriert werden müssen. Die Auflistung der Anforderungen wird dabei unterteilt in Anforderungen, die das Aufzeichnen von Daten betreffen und solche, die sich auf das anschließende Abspielen beziehen. Bevor die Unterteilung beginnt, wird eine Anforderung betrachtet, die für das Abspielen und das Aufzeichnen gleichermaßen von Relevanz ist.

Android Zielplattform

Für die vorliegende Arbeit wurde *Android Lollipop* in der Version 5.1 als Zielplattform ausgewählt[1]. Die Notwendigkeit, eine Version von *Android* für die Entwicklung des Verfahrens zu wählen ist darin begründet, dass eine große Anzahl von Versionen auf dem Markt vertreten ist[2]. Dabei bringen die meisten Versionen neue Funktionalitäten mit oder markieren existierende Funktionen als veraltet. Aus diesem Grund sollte eine möglichst aktuelle Version gewählt werden, um eine hohe

[1] Siehe `https://developer.android.com/about/versions/lollipop.html`, aufgerufen am 07.10.2016 & `https://developer.android.com/about/versions/android-5.1.html`, aufgerufen am 07.10.2016

[2] Siehe `https://developer.android.com/about/dashboards/index.html`, aufgerufen am 07.10.2016

Abdeckung der verfügbaren Geräte zu erhalten. Knapp über 40% aller Geräte mit *Android*-Betriebssystem laufen mit der gewählten oder einer aktuelleren Version. Alle diese Geräte werden von dem Verfahren, das im Rahmen dieser Arbeit entwickelt wird, unterstützt. Dabei muss berücksichtigt werden, dass Funktionalitäten, welche ab *Android Marshmallow*[3] (Version 6.0) hinzugefügt worden sind, nicht durch das Verfahren abgedeckt werden. Vorgängerversionen zur gewählten Version werden nicht unterstützt, da beim Abspielen die Gefahr besteht, Aktionen ausführen zu wollen, die in der jeweiligen Version nicht existent sind. Die Aufnahme von Daten aus älteren Versionen ist theoretisch möglich, aber praktisch nicht sinnvoll, da die erfassten Daten nicht das volle Spektrum an möglichen Informationen abdecken, weil gewisse Funktionen nicht im Betriebssystem vorhanden sind.

Im Folgenden werden die Anforderungen aufgezeigt, welche sich auf das Aufzeichnen von Daten beziehen. Auch wenn sich die Anforderungen nur auf die Aufzeichnung beziehen, werden Aspekte des Abspielens mit in die Betrachtung einbezogen, was beispielsweise bei den zu erfassenden Daten deutlich wird.

Erfassung verschiedenster Datenquellen

Um das Verhalten einer Applikation in möglichst vollem Umfang erfassen zu können ist es notwendig, dass viele verschiedene Daten der entsprechenden Applikation gesammelt werden. Nachfolgend werden alle Bezugsquellen für Daten, die berücksichtigt werden müssen, aufgelistet und die Aspekte dieser Quellen aufgezeigt. Vorweg sei darauf hingewiesen, dass nicht das Verhalten des Nutzers erfasst werden soll, sondern das Verhalten der verwendeten Applikation. Dies bedeutet, dass es beispielsweise uninteressant ist, wenn ein Benutzer durch eine Berührung des Bildschirms einen Knopf drückt, welcher das Herunterladen einer Datei auslöst. Stattdessen soll der anfallende Netzwerkverkehr und die entstehende Prozessorlast erfasst werden. Hierdurch ist das Verfahren losgelöst von der aufgezeichneten Applikation. So ist es möglich, beim Abspielen von Daten das Verhalten einer Applikation akkurat zu imitieren, ohne Kenntnisse über diese Applikation zu haben und ohne das sie auf dem Gerät installiert ist.

CPU-Last Jede Applikation erzeugt eine gewisse Prozessorlast. Das ist die Gemeinsamkeit, die alle Applikationen miteinander teilen. Darüber hinaus ist das Maß, in dem diese Last erzeugt wird, wichtig, da der Prozessor eines der Bauteile ist, an dem häufig Ressourcen-Engpässe entstehen. Aus diesem Grund soll die anfallende Last erfasst werden. Dabei reicht eine prozentuale Auslastung des Pro-

[3] Siehe `https://developer.android.com/about/versions/marshmallow/android-6.0.html#fingerprint-authentication`, aufgerufen am 08.10.2016

zessors alleine nicht aus, um die Daten akkurat auf verschiedenen Geräten wieder abzuspielen. So ist eine Auslastung von $x\%$ auf einem Zweikern-Prozessor nicht gleichbedeutend mit $x\%$ Last auf einem Vierkern-Prozessor. Aus diesem Grund sollen neben der prozentualen Auslastung weitere Daten erfasst werden. Diese umfassen die Anzahl an Prozessorkernen, die aktuelle Frequenz der Kerne sowie die maximale Frequenz der Kerne. Durch diese zusätzlichen Daten ist es beim Abspielen möglich, die Prozessorlast in Relation zur tatsächlichen Hardware zu setzen.

Netzwerkverkehr Neben den klassischen Internetprotokollen *TCP* und *UDP* unterstützt das *Android* Betriebssystem eine große Anzahl an Protokollen für die drahtlose Kommunikation[4]. In dieser Menge sind beispielsweise direktes *Peer-to-Peer* zwischen Geräten und Protokolle zur Serviceentdeckung enthalten. Ein Großteil der Applikationen verwendet drahtlose Kommunikation. Dabei ist die Datenübertragung bzw. der Datenempfang über das Internet die häufigste Anwendung. Da das Verhalten einer Applikation aber die Nutzung aller verfügbaren Kommunikationsmittel enthält, sind alle verfügbaren Möglichkeiten zur drahtlosen Kommunikation von Interesse. Dennoch ist es möglich, eine Wertung der einzelnen Kommunikationsprotokolle vorzunehmen, da häufiger benutzte Möglichkeiten eine höhere Relevanz und Signifikanz haben als selten genutzte Protokolle.

Ein-/Ausgabe-Operationen Ähnlich zum bekannten *Memory-gap* existiert eine Differenz in der Geschwindigkeit des Prozessors und des persistenten Speichers [4]. Diese Lücke ist erheblich größer als der *Memory-gap*. Dies gilt auch für den, typischerweise in Smartphones verbauten, schnellen *Flash*-Speicher [22]. Deswegen sind Ein-/Ausgabe-Operationen ein dauerhafter Flaschenhals. Aus diesem Grund wird die Anforderung definiert, dass sämtliche Dateioperationen protokolliert werden. Außerdem sollen alle gesammelten Daten auf den persistenten Speicher geschrieben werden. Dies ist notwendig, da die Daten für die Zukunft nutzbar sein sollen und es keine andere Möglichkeit für Applikationen gibt, Daten zu persistieren.

Nutzung des GPS-Moduls Bei Smartphones handelt es sich um mobile Geräte. Durch das in der Regel verbaute *GPS*-Modul haben Smartphones die Möglichkeit, ihre eigene Position zu ermitteln. Auf Basis der Positionsdaten können bestimmte Aktionen ausgelöst oder Profile verwendet werden. Wie beim Netzwerkverkehr wird durch die Nutzung der entsprechenden *Hardware*-Komponente Energie verbraucht. Außerdem handelt es sich bei den Positionsdaten um höchst

[4]Siehe `https://developer.android.com/training/building-connectivity.html`, aufgerufen am 08.10.2016

sensible Informationen über den Nutzer. Aus diesen Gründen sind die Informationen darüber, in welchen Situationen und in welcher Häufigkeit eine Applikation Positionsdaten verwendet, eine relevante Datenquelle. Sie sind nicht nur für ein möglichst genaues Replizieren des Verhaltens einer Applikation notwendig, sondern erlauben unter Umständen diverse Schlussfolgerungen auf Basis der gesammelten Daten. Dabei können keine konkreten Problematiken im Bezug auf den Datenschutz aufgedeckt werden, aber es können Indizien für solche geliefert werden. Neben der direkten Positionsabfrage sollen auch verfügbare Komfortfunktionen protokolliert werden. Diese umfassen beispielsweise Benachrichtigungen, wenn sich der Benutzer einer bestimmten Position auf eine gewisse Distanz nähert oder einen definierten Bereich verlässt.

Bildschirmzustand Der Bildschirm ist, neben dem Mobilfunk-Chip, das Bauteil, welches die meiste Energie im Betrieb verbraucht [3, 25]. Daher ist es notwendig, den Bildschirm in die Betrachtung mit einzubeziehen. Es gibt Bildschirmarten, bei denen der angezeigte Inhalt direkten Einfluss auf den Stromverbrauch hat[5]. Da eine Erfassung des Bildschirminhalts in massiven Datenmengen resultieren würde und darüber hinaus datenschutzrechtliche Problematiken entstehen würden, wird der angezeigte Bildschirminhalt für die weitere Betrachtung ausgeschlossen. Neben dem Inhalt des Bildschirms hat die Hintergrundbeleuchtung und die Intensität selbiger einen großen Einfluss auf den Energieverbrauch. Daher sollen Daten über die Hintergrundbeleuchtung ebenso protokolliert werden, wie Informationen darüber, zu welchen Zeitpunkten der Bildschirm an- bzw. ausgeschaltet wird.

Applikation-Lebenszyklus Die zentralen Ereignisse im Lebenszyklus einer Applikation umfassen das Starten, Stoppen, Pausieren, Fortführen sowie das Verschieben einer Applikation in den Vorder-/Hintergrund. Diese Ereignisse können im Betriebssystem zu Zustandsänderungen führen, beispielsweise kann das Betreten eines Energiesparmodus verhindert werden, wenn kurz zuvor eine Applikation gestartet wurde. Außerdem wird es durch die Daten möglich, in einer anschließenden Analyse Sitzungen zu identifizieren, in welchen der Nutzer eine bestimmte Applikation verwendet hat. Aus diesem Grund soll der vollständige Lebenszyklus der beobachteten Applikation erfasst werden.

Power Management Das *Android*-Betriebssystem bietet eine Vielzahl an Funktionen, um Applikationen optimal im Hinblick auf den Batterieverbrauch zu gestalten, beispielsweise indem periodische Aufgaben vom Betriebssystem ausgelöst

[5] Siehe `https://senk9.wordpress.com/2015/05/24/thoughts-amoled-power-consumption-tested-and-explored/`, aufgerufen am 08.10.2016

werden[6]. Hierdurch muss nicht jede Applikation eine eigene Verwaltung für Hintergrundprozesse implementieren und es wird sichergestellt, dass eine fehlerhafte Implementierung sich nicht negativ auf den Energieverbrauch und die Laufzeit auswirkt. Wenn durch das Betriebssystem eine periodische bzw. verzögerte Aktion ausgelöst wird, werden die Auswirkungen der ausgelösten Aktion implizit bereits festgehalten. Dies wird durch die Erfassung der anderen Datenquellen, wie z.B. Prozessorlast und Netzwerkverkehr, sichergestellt. Aus diesem Grund ist es nicht notwendig die Nutzung der Mechanismen zur periodischen bzw. verzögerten Ausführung zu protokollieren. In jedem Fall erfasst werden sollte das Belegen und Freigeben von *Wakelocks*, da es sich hierbei um ein invasives Verfahren handelt, welches direkte Auswirkungen auf das allgemeine Verhalten des Betriebssystems, den Energieverbrauch und Laufzeiten hat. Denn durch die Verwendung von *Wakelocks* wird verhindert, dass das Smartphone in energiesparende Zustände übergeht und in diesem Kontext ggf. Prozesse beendet oder suspendiert.

Batteriezustand Es ist nicht das Ziel der Arbeit ein Verfahren zu entwickeln, welches genaue Aussagen und Schlussfolgerungen über den Energieverbrauch erlaubt. Trotzdem ist der Batterieverbrauch ein Indiz dafür, wie genau das Verfahren das Verhalten aufgezeichneter Applikationen wiedergibt. Dies kann ermittelt werden, indem aufgezeichnete Daten während des Abspielens nicht nur repliziert sondern auch analysiert werden. Außerdem können diese Daten verwendet werden, um grobe Vergleiche zwischen verschiedenen Smartphone-Modellen vorzunehmen, indem dieselben Daten auf unterschiedlichen Geräten abgespielt werden. Deswegen ist es eine Anforderung, periodisch zu erfassen, zu welchem Grad die Batterie geladen ist. Für detailliertere Vergleiche und genauere Aussagen über den Energieverbrauch ist ein wesentlicher größerer Aufwand notwendig. Ein Beispiel für eine Analyse des Energieverbrauchs von Smartphones kann in der Arbeit von Carroll et al. eingesehen werden [3].

Audiowiedergabe und -aufnahme Für die Wiedergabe beliebiger Töne wird ein gewisses Maß an Energie benötigt [35]. Ebenfalls entsteht auf der Ebene des Betriebssystems eine Last, da *Android* die Verwaltung von Mediendateien selbst vornimmt. Für eine vollständige Betrachtung des Verhaltens einer Applikation ist es somit notwendig, das Abspielen und die Aufnahme von Audiodaten zu erfassen. Gleichzeitig spielen diese Daten eine untergeordnete Rolle, da die Nutzung dieser Funktionen nur bei wenigen Applikationen eine wichtige Rolle einnimmt. Hierunter fallen beispielsweise Applikationen zum Abspielen von Musik. Außerdem ergibt

[6]Siehe `https://developer.android.com/training/scheduling/alarms.html`, aufgerufen am 08.10.2016

sich die Problematik, dass für ein möglichst genaues Abspielen die abgespielten und aufgenommenen Audiodaten mit hoher Genauigkeit erfasst werden müssen. Dies ist im Bezug auf den Datenschutz nicht unbedenklich. Aus diesen Gründen werden sämtliche Audiodaten von der Erfassung und dem anschließenden Abspielen ausgeschlossen.

GPU-Nutzung Die *GPU* ist in Smartphones für verschiedene Aufgaben zuständig. Neben der normalen Anzeige des Bildschirminhalts werden hier Videos für die Anzeige decodiert (sofern die verbaute *GPU* dies unterstützt, was in seltenen Fällen zutrifft). Außerdem verwenden 3D-Anwendungen die Bibliothek *OpenGL ES* zur Darstellung von Inhalten. Hinreichend detaillierte Daten für ein anschließendes Abspielen von *OpenGL ES* zu sammeln und gleichzeitig die Performanz nicht so weit zu reduzieren, dass die Applikation unbenutzbar wird, ist keine triviale Aufgabe für eine so umfangreiche Bibliothek. Ähnliches gilt für den Bildschirminhalt von normalen Applikationen, die nicht *OpenGL ES* verwenden. Es existiert eine große Anzahl (über 120) von Anzeigekomponenten, sogenannte *Views*, welche von Applikationen verwendet werden können, um den Bildschirm mit Inhalt zu füllen[7]. In Kombination mit der Möglichkeit beliebige, neue *Views* zu definieren, wächst die Menge an Daten, die für ein akkurates Abspielen notwendig sind, beliebig an[8]. Noch größer wird die Datenmenge durch spezielle Komponenten, die den Inhalt von Webseiten darstellen können[9]. Aufgrund des enormen Aufwands, der notwendig wäre, um diese Datenmenge aufzuzeichnen und nutzbar zu machen, wird auf die Instrumentierung der *GPU*-Nutzung für diese Arbeit verzichtet.

Diverse Sensoren Moderne *Android* Smartphones bieten eine Vielzahl von Sensoren, welche direkt oder indirekt durch den Benutzer ausgelöst werden. Beispielsweise den Näherungssensor, der den Bildschirm abschaltet, wenn das Smartphone beim telefonieren an das Ohr gehalten wird, einen Fingerabdrucksensor zur Authentifizierung und den *Touchscreen*, über den der Nutzer das Gerät bedient[10]. Darüber hinaus hat jeder Smartphone-Hersteller die Möglichkeit beliebige zusätzliche Sensoren zu verbauen. Alle diese Sensoren haben mehr oder weniger direkte

[7] Siehe `https://developer.android.com/reference/android/view/View.html`, aufgerufen am 01.11.2016

[8] Siehe `https://developer.android.com/training/custom-views/index.html`, aufgerufen am 01.11.2016

[9] Siehe `https://developer.android.com/reference/android/webkit/WebView.html`, aufgerufen am 01.11.2016

[10] Siehe `https://developer.android.com/guide/topics/sensors/sensors_overview.html`, aufgerufen am 08.10.2016 & `https://source.android.com/devices/input/touch-devices.html`, aufgerufen am 08.10.2016

Auswirkungen auf das Verhalten des Smartphones. Dabei beziehen sich die Auswirkungen in der Regel auf das Betriebssystem und seltener auf einzelne Applikationen. Die mittelbaren Auswirkungen von Sensordaten auf Applikationen werden erfasst, sofern sie relevant für die aufgezeichnete Applikation sind. Beispielsweise wird das Abschalten des Bildschirms durch Interaktion mit dem Näherungssensor erfasst. Da die Menge an verbauten Sensoren und deren Auswirkungen auf Applikationen nicht klar definiert ist und die relevanten Ereignisse, welche durch Sensoren verursacht werden, erfasst werden, sind diese Sensoren von der weiteren Betrachtung ausgeschlossen.

Hauptspeichernutzung In welcher Menge eine Applikation Hauptspeicher über welchen Zeitraum belegt und wie oft darauf lesend bzw. schreibend zugegriffen wird, ist eine relevante Charakteristik einer Applikation. Informationen über diese Nutzung zu erhalten ist in *Android* problematisch, da jede Applikation in einer eigenen *Sandbox* läuft[11]. Durch die Nutzung der *Android Debug Bridge* ist es möglich, diese Daten über die *USB*-Schnittstelle auf einem externen Gerät zu betrachten[12]. Um diese Daten *online* auf einem Gerät zu sammeln, wären massive Eingriffe in das Betriebssystem notwendig. Diese würden das Aushebeln der *Sandbox* enthalten, welche ein wichtiges Sicherheitsmerkmal darstellt. Außerdem würde dies eine Hürde für die Datenaufzeichnung auf verschiedenen Geräten, in beliebigen Umfeldern bedeuten. Mit dieser Begründung werden die Daten, welche die Hauptspeichernutzung betreffen von der Aufzeichnung ausgeschlossen.

NFC & Bluetooth *Android* unterstützt, neben der Kommunikation über das Mobilfunk-Modul, *NFC*[13] und *Bluetooth*[14]. Beide diese Verfahren werden weit weniger genutzt als der Mobilfunk-Chip. Dennoch sind sie, genau wie die weniger verbreiteten Netzwerkkommunikationsmöglichkeiten, Teil eines vollständigen Applikationsprofils und sollten ebenfalls protokolliert werden. Dennoch ist die Relevanz durch die geringe Verbreitung nicht so hoch, wie bei den zentralen Datenquellen, welche bei allen bzw. der Mehrheit der Applikationen verwendet werden.

[11] Siehe `https://developer.android.com/training/articles/security-tips.html`, aufgerufen am 08.10.2016

[12] Siehe `https://developer.android.com/studio/command-line/adb.html`, aufgerufen am 01.11.2016 & `https://developer.android.com/topic/performance/memory.html`, aufgerufen am 08.10.2016

[13] Siehe `https://developer.android.com/guide/topics/connectivity/nfc/index.html`, aufgerufen am 08.10.2016

[14] Siehe `https://developer.android.com/guide/topics/connectivity/bluetooth.html`, aufgerufen am 08.10.2016

Aufzeichnung beliebiger Applikationen

Da es das Ziel ist, einen generischen Ansatz für die Leistungsmessung zu entwickeln, ist es von höchster Wichtigkeit, dass Daten von beliebigen Applikationen aufgezeichnet werden können. Nur so ist gewährleistet, dass für jeden Anwendungsfall die passenden Applikationen beobachtet wird und damit relevante Daten gesammelt werden können. Dabei sollte es möglich sein, die aufgezeichnete Applikation ohne großen Aufwand auszuwählen und zu wechseln. Es findet bewusst keine Vorauswahl bzw. Eingrenzung in Bezug auf die Applikationen, die aufgezeichnet werden können, statt.

Einfach zu verarbeitendes Datenformat

Die aufgezeichneten Daten dienen zum einen dem Zweck, wieder abgespielt zu werden. Zum anderen können sie für Analysen zur Ressourcennutzung von Applikationen verwendet werden. Außerdem existieren weitere Anwendungsszenarien für die gesammelten Daten. Beispielsweise wäre es möglich, eine große Menge gesammelter Daten von einer Applikation zu einem allgemeinen Profil zusammenzuführen, welches das durchschnittliche Verhalten einer Applikation widerspiegelt. Da es somit diverse Anwendungsfälle für die Daten gibt, ist ein einfaches Datenformat, welches leicht weiterverarbeitet werden kann und ggf. von Menschen lesbar ist, vorzuziehen. Hier muss ein Kompromiss gefunden werden zwischen der einfachen Verarbeitung von Daten und der Ausgabegröße, da durch das Schreiben der Ausgabe kein Flaschenhals entstehen soll, welcher die Ein- und Ausgabe anderer Applikationen behindert.

An dieser Stelle wurden alle Anforderungen, welche sich auf die Aufzeichnung von Daten beziehen, betrachtet und es werden im Folgenden die Anforderungen für das Abspielen von Daten betrachtet.

Geräteanforderungen

Ziel ist es, die potentielle Nutzermenge, welche die aufgezeichneten Daten abspielen kann, zu maximieren. Der notwendige Aufwand zur Vorbereitung des Abspielens von Daten soll hierzu minimiert werden. Zum Aufzeichnen von Daten ist eine modifizierte Version des *Android* Betriebssystems notwendig. Dies ist erforderlich, da verschiedene Stellen des Betriebssystems angepasst werden, um die notwendigen Daten zu erfassen und an eine zentrale Stelle zu liefern. Hierdurch entsteht eine Hürde für den Nutzer, wenn es um die Aufzeichnung von Daten geht, da das modifizierte Betriebssystem, unter Umständen noch für das verwendete Smartphone angepasst werden muss. Da die Daten zum Abspielen benötigt werden, entsteht in-

direkt auch an dieser Stelle eine Hürde. Um diese zu umgehen wäre es beispielsweise möglich, in der Zukunft die gesammelten Daten an einer zentralen Stelle frei zugänglich für alle Nutzer bereitzustellen. Zum Abspielen sind solche Modifikationen am Betriebssystem nicht notwendig und darüber hinaus nicht wünschenswert, da das Abspielen von Daten das Verhalten einer Applikation widerspiegeln soll und nicht in das Betriebssystem eingreift. Aus diesen Gründen soll die Anwendung, welche die Daten abspielt eine normale Applikation sein, welche auf einem nicht modifiziertem *Android* Betriebssystem läuft.

Wiedergabe

Da es Ziel des Abspielens ist, das Verhalten einer Applikation möglichst genau zu simulieren, ist es von großer Wichtigkeit, dass die aufgezeichneten Daten mit hoher Genauigkeit wiedergegeben werden. Das bedeutet, dass beim Abspielen die Zeit zwischen Ereignissen und die Art und Reihenfolge dieser möglichst genau den Daten entsprechen sollen, welche zur Wiedergabe verwendet werden. Die Genauigkeit der Wiedergabe ist das kritischste Kriterium für die spätere Bewertung der Güte des Abspielens. Die Genauigkeit entscheidet darüber, wie genau das ursprüngliche Verhalten der aufgezeichneten Applikation wiedergegeben wird. Die Genauigkeit ist auch aus dem Grund wichtig, da sie sicherstellt, dass mehrere Abspielvorgänge eines Datensatzes zu einem identischen Verhalten des abspielenden Gerätes führen. Dies ist für die Reproduzierbarkeit und Aussagekraft von Messungen, die mit dem Verfahren durchgeführt werden, wichtig. Somit ist es das Ziel, Abweichungen zwischen aufgezeichneten und wiedergegebenen Daten zu vermeiden, oder zumindest so minimal wie möglich zu halten.

3.2 Weiche Anforderungen

Neben den vorgestellten harten Anforderungen, welche zwingend umgesetzt werden müssen und im Rahmen der Evaluation zur Bewertung der Güte verwendet werden, existieren weiche Anforderungen. Diese Anforderungen werden nicht gesondert während der Entwicklung berücksichtigt, um die Erfüllung zu gewährleisten und auch nicht zur Bewertung der Güte verwendet. Dennoch sollten diese Anforderungen nicht vollständig ignoriert werden und an Stellen, an denen es sich anbietet in die Entwicklung mit einbezogen werden. Nachfolgend wird ein Überblick über diese weichen Anforderungen gegeben.

Durch die zusätzliche Last während der Aufnahme und dem *Overhead* beim Abspielen ergibt sich ein gesteigerter Energieverbrauch. Diese Steigerung sollte so gering wie möglich ausfallen. Denn es beeinträchtigt direkt den Nutzungskomfort

des Nutzers, wenn der Akku des Geräts häufiger geladen werden muss.

Der zuvor angesprochene *Overhead*, welcher beim Abspielen anfällt, lässt sich nicht vollständig vermeiden. Dies liegt daran, dass irgendeine Art von Logik notwendig ist, um die abzuspielenden Daten zu interpretieren und die entsprechenden Aktionen zur richtigen Zeit auszulösen. Je geringer dieser *Overhead* ausfällt, desto geringer ist die Gefahr, dass durch die Abspiellogik Engpässe an Ressourcen entstehen, welche die Ausführung verfälschen und so die Güte des Abspielens beeinträchtigt.

Zugunsten des Nutzungskomforts sollte auch die zusätzliche Last, welche durch die Aufnahme entsteht, möglichst gering gehalten werden. Denn wenn die Ausführung des zusätzlichen Codes, welcher zur Instrumentierung verwendet wird, zu lange dauert, leidet die Interaktivität mit dem Nutzer durch entstehende Wartezeiten. Darüber hinaus besteht die Gefahr von Ressourcenengpässen, beispielsweise bei der Ein-/Ausgabe, wenn die zusätzliche Last zu groß ausfällt.

Generell sollte von funktionalen Änderungen am Betriebssystem abgesehen werden, sofern dies möglich ist. Die Differenz im Verhalten eines unmodifizierten Betriebssystems und dem für die Aufnahme verwendeten sollte so minimal wie möglich sein. So wird sichergestellt, dass die aufgezeichneten Daten auch auf das Verhalten einer Applikation, welche auf einem nicht modifizierten Betriebssystem läuft, zutreffen.

4 Modellierung

Dieses Kapitel führt durch die Modellierung des generischen Verfahrens zur Leistungsmessung, dessen Entwicklung das Ziel der Arbeit ist. Dabei werden die Anforderungen aus Kapitel 3 in den Prozess der Modellierung mit einbezogen. Für einen groben Überblick wird in Abbildung 4.1 nochmals das *Android* Betriebssystem schematisch dargestellt. Dabei wurden die Elementgruppen, welche für das Verfahren von Relevanz sind, hervorgehoben. Im Rahmen der Aufzeichnung werden Daten aus dem *Linux Kernel* gesammelt und das *Application Framework* für die Datensammlung instrumentiert. Auf Ebene der *Android Applications* wird eine Applikation zum Aufzeichnen von Daten und eine weitere zum Abspielen dieser eingefügt.

Im Folgenden wird zunächst die Modellierung des Aufzeichnens von Daten vorgestellt und im Anschluss die Methodik zum Abspielen der aufgezeichneten Daten.

4.1 Aufzeichnen

Bei der nachfolgenden Modellierung der Datenaufzeichnung werden die Ebenen des *Android* Betriebssystems, welche in Abbildung 4.1 markiert wurden, einzeln betrachtet. Hierdurch ergibt sich ein besserer Überblick, welche Daten an welchen Stellen ihren Ursprung haben und wo diese zusammengeführt werden. Dabei werden die relevanten Ebenen von unten nach oben betrachtet, d.h. mit jeder betrachteten Ebene wächst der Abstraktionsgrad zur *Hardware*.

Als erstes wird der *Kernel* von *Android* betrachtet. Der *Kernel* soll im Rahmen dieser Arbeit nicht modifiziert werden. Stattdessen wird *SystemTap* verwendet, um Daten auf dieser Ebene zu sammeln. Hier wird auf *SystemTap* zurückgegriffen, da es keine Anpassungen am *Kernel* erfordert, wenn dieser ausgetauscht wird und dynamisch im laufenden Betrieb aktiviert bzw. deaktiviert werden kann. Es ist somit nur sicherzustellen, dass die Konfiguration des jeweiligen *Kernels* so gewählt ist, dass die notwendigen Anforderungen für *SystemTap* erfüllt sind. Es sollen nur solche Daten auf der Ebene des *Kernels* erfasst werden, die nicht - oder nur unter großem Aufwand - aus dem *Application Framework* erhalten werden können. Dies kann am einfachsten durch eine erneute Betrachtung von Abbildung 4.1 begründet werden. Die beobachteten Applikationen befinden sich in der Abbildung ganz oben

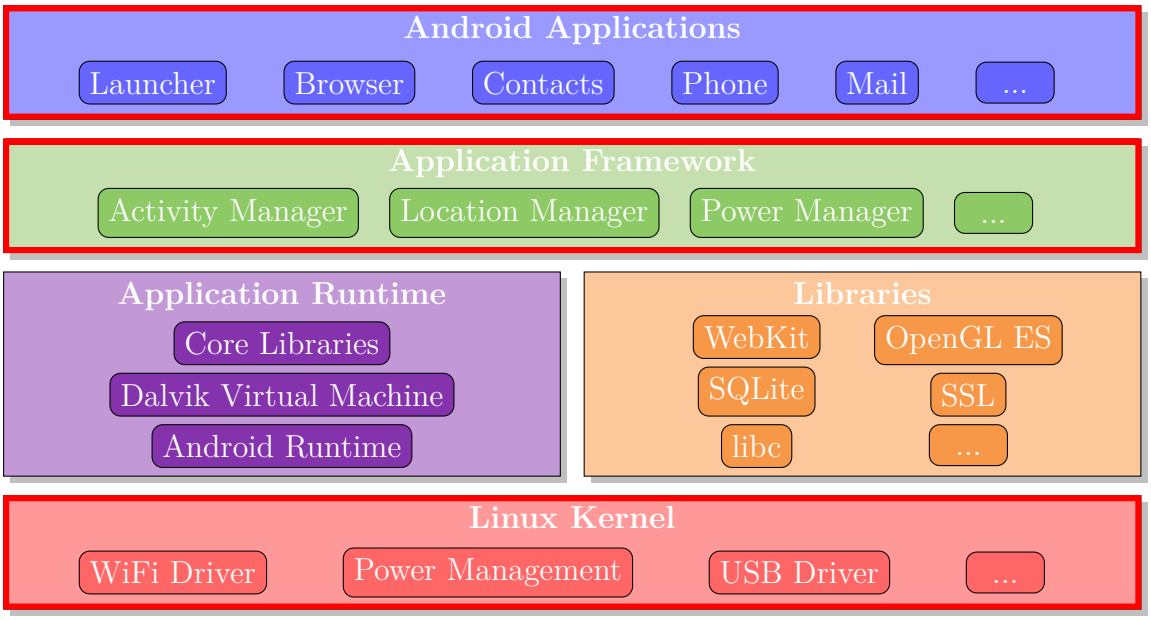

Abbildung 4.1: Schematische Darstellung des Android Betriebssystems. Von der Modellierung betroffene Elemente wurden hervorgehoben. Angelehnt an eine Graphik der offiziellen *Android*-Dokumentation[1]

und weisen den größten Abstraktionsgrad zur *Hardware* auf. Die Funktionalitäten, welche sie verwenden, befinden sich in den allermeisten Fällen im *Application Framework*. In der Regel verwenden die Funktionen, welche im *Application Framework* aufgerufen werden, andere Funktionen, welche ebenfalls im *Application Framework*, einer (vom Betriebssystem mitgelieferten) Bibliothek oder direkt im *Kernel* zu finden sind. Somit kann eine Funktion, welche von einer Applikation verwendet wird, auf unterschiedlich direkten Wegen zu einer Aktion im *Kernel* führen. Dabei werden einige oder sogar alle Abstraktionsebenen durchlaufen. Somit wird durch direkte Datenaufzeichnung im *Kernel* nicht das direkte Verhalten der beobachteten Applikation erfasst. Stattdessen werden die Auswirkungen des Verhaltens im *Kernel* beobachtet und die darüber liegenden Schichten ignoriert. Um das Verhalten von Applikationen möglichst genau zu erfassen, sollte diese Abstraktion vom tatsächlichen Verhalten, wenn möglich, vermieden werden. Denn auch wenn verschiedene Funktionen des *Application Frameworks* identische Aktionen im *Kernel* auslösen, ist es möglich, dass die verschiedenen Funktionen unterschiedliche Änderungen am Zustand des Betriebssystems verursachen, wodurch sich das Verhalten von Applikationen und/oder *Android* verändern kann. Bei den Daten, die im Folgenden aufgelistet und im *Kernel* erfasst werden sollen, wurde ein Kompromiss zwischen Aufwand und Abstraktion zu Gunsten des Aufwands getroffen. Bei der genaue-

ren Betrachtung dieser Datenquellen zeigt sich, dass durch eine hohe Diversität an verfügbaren Funktionen zur Ressourcennutzung ein erheblicher Aufwand notwendig wäre, um alle Daten im *Application Framework* zu erfassen. Da dies den Umfang der vorliegenden Arbeit übersteigen würde wird die Erfassung im *Kernel* durchgeführt. Darüber hinaus steigt die Wahrscheinlichkeit, dass in künftigen Versionen von *Android* Änderungen an beobachteten Funktionen im *Application Framework* vorgenommen werden an, wenn die Menge dieser Funktionen größer wird. Hierdurch entsteht bei jedem Wechsel der Version ein zusätzlicher Aufwand, der durch die Erfassung im *Kernel* massiv reduziert wird. Denn zum einen ist die zu prüfende Menge deutlich kleiner und zum anderen muss diese Prüfung nur bei einer Änderung der Version des *Kernels* vorgenommen werden. Außerdem werden die Änderungen am Zustand und Verhalten des Betriebssystems, welche durch die Nutzung von Funktionen im *Application Framework* hervorgerufen werden und das Verhalten der beobachteten Applikation beeinflussen, implizit durch die Erfassung der anderen Datenquellen abgebildet. Beispielsweise finden sich in der Ausgabe Daten zur anfallenden Prozessorlast, Dateioperationen und Netzwerkverkehr, wenn eine Funktion im *Application Framework* aufgerufen wird, welche automatisiert eine Datei im Hintergrund herunterlädt und abspeichert. Es lässt sich aus diesen Daten nicht ohne Weiteres auf die aufgerufene Funktion schließen, aber das von ihr ausgelöste Verhalten wird erfasst und kann wiedergegeben werden.

Netzwerkdaten

Bei genauer Begutachtung des *Android Developer Guides* zum Thema Netzwerkverkehr zeigt sich, dass das *Application Framework* verschiedene Schnittstellen bietet, um Netzwerkkommunikation mit den Protokollen *TCP* und *UDP* durchzuführen[2]. Neben der normalen Netzwerk-Schnittstelle von *Java* existiert im *Application Framework* eine zusätzliche Netzwerk-Schnittstelle, welche einen stark erweiterten Funktionsumfang bietet[3]. Darüber hinaus existieren Komponenten, die weitere zusätzliche Komfortfunktionen bieten. Als Beispiele seien der `MediaPlayer` und der `DownloadManager` genannt.

Beide bieten Funktionalitäten an, die es erlauben mit minimaler Konfiguration Aktionen im Hintergrund durchzuführen. So wird dem `DownloadManager` eine *URL* übergeben, welche auf die Daten verweist, die heruntergeladen werden sollen. Im Anschluss führt der `DownloadManager` im Hintergrund das Herunterladen mit

[2] `https://developer.android.com/training/basics/network-ops/index.html`, auferufen am 13.10.2016

[3] Siehe `https://developer.android.com/reference/java/net/package-summary.html`, auferufen am 13.10.2016 & `https://developer.android.com/reference/android/net/package-summary.html`, auferufen am 13.10.2016

anschließender Speicherung der Daten durch. Der Nutzer kann alle heruntergeladenen Dateien über eine dedizierte Applikation verwalten und den Status aktueller Downloads einsehen. Der `MediaPlayer` bietet ähnlichen Komfort für Mediendateien, welche Video- bzw. Audiodaten enthalten. Eine Angabe der Quelle auf dem Speicher, bzw. eine *URL* unter der Mediendaten gefunden werden können, reicht aus. Das Lesen der Datei, notwendige Dekodierungen und die Wiedergabe werden automatisch im Hintergrund durch den `MediaPlayer` durchgeführt. Somit existiert eine Vielzahl von Funktionen, welche für eine möglichst genaue Erfassung des Verhaltens einer Applikation alle instrumentiert werden müssen. Gleichzeitig wird durch diese Komfortfunktionen eine gewisse Redundanz in die erfassten Daten eingefügt. Denn neben dem Aufruf einer Komfortfunktion werden auch die Aufrufe darunter liegender Funktionen erfasst, die durch Komfortfunktion verwendet werden. Hierdurch entsteht ein Bedarf an einer Datenvorverarbeitung, welche diese Redundanz wieder beseitigt, bevor die aufgezeichneten Daten korrekt abgespielt werden können. Während der Bereinigung sollen möglichst von der *Hardware* abstrahierende Aktionen erhalten bleiben, da diese das Verhalten der Applikation genauer wiedergeben. Eine Alternative, um die redundanten Daten zu vermeiden, ist es die Datenerfassung auf die Netzwerk-Schnittstellen zu beschränken. Durch das Ignorieren der Komfortfunktionen wird vom tatsächlichen Verhalten der beobachteten Applikation in gewissem Maße abstrahiert, da keinerlei Informationen über die Komfortfunktionen vorliegen bzw. rekonstruiert werden können. Trotzdem wäre die Menge der Funktionen, welche instrumentiert werden müssen, enorm, da die verfügbaren Netzwerkschnittstellen einen großen Umfang haben. Dabei verwenden die Funktionen der Schnittstellen nur einige wenige Systemaufrufe, um die eigentliche Kommunikation abzuwickeln. Aus diesem Grund ist es möglich, zu Gunsten des Aufwands, vom eigentlichen Verhalten der Applikation ein weiteres Stück zu abstrahieren und direkt die Systemaufrufe zu instrumentieren. Es würde somit beispielsweise nicht mehr erfasst, ob eine Internetseite abgerufen wird oder Daten zwischen Smartphones ausgetauscht werden und dabei eine der speziellen Funktionen verwendet wird, welche *Android* dem Entwickler bietet. Stattdessen beschränken sich die erfassten Daten auf das verwendete Transportprotokoll (*TCP* bzw. *UDP*) und die Menge der übertragenen/empfangenen Daten. Deshalb wird zu Gunsten des Aufwands die Erfassung von Netzwerkdaten, für die vorliegende Arbeit, durch die Instrumentierung der verwendeten Systemaufrufe durchgeführt. Um aber nicht zu massiv von dem Verhalten der beobachteten Applikation zu abstrahieren, werden zusätzlich die zuvor angesprochenen Komfortfunktionen instrumentiert, auch wenn dies zusätzlichen Aufwand in der Datenvorverarbeitung bedeutet. Die Erfassung der Komfortfunktionen wird genauer im Abschnitt zu *Application Framework* beschrieben.
An dieser Stelle wurde die Betrachtung bisher auf Netzwerkkommunikation über

die Protokolle *TCP* und *UDP* beschränkt. Darüber hinaus werden *NFC* und *Bluetooth* zur Kommunikation angeboten[4]. Das *Application Framework* bietet ähnlich umfangreiche Schnittstellen für diese Kommunikationskanäle, wie für die Verwendung von *TCP* und *UDP*. Auch hier wäre es möglich, den Aufwand der Datenerfassung zu reduzieren, indem direkt die verwendeten Systemaufrufe betrachtet werden. Da der Aufwand, alle verbleibenden Datenquellen zur Datensammlung nutzbar zu machen aber ohnehin groß ist, wird die Kommunikation über *NFC* und *Bluetooth* vom weiteren Verlauf der Arbeit vollständig ausgeschlossen. Darüber hinaus wäre für das Abspielen dieser Daten gesonderte Hardware notwendig, welche gesendete Daten entgegennimmt und entsprechende Antworten versendet.

Ein-/Ausgabe

Für Dateioperationen ist die Situation ähnlich wie diejenige beim Netzwerkverkehr. Neben der Schnittstelle zur Durchführung von Ein-/Ausgabeoperationen existieren diverse Komfortfunktionen, welche wiederum auf die Funktionen der Schnittstelle zugreifen. Diese Komfortfunktionen werden wieder vom `MediaPlayer` und dem `DownloadManager` bereitgestellt, welche im vorherigen Abschnitt bereits vorgestellt wurden. Dabei besteht der Komfort darin, dass heruntergeladene Dateien automatisch abgespeichert und Mediendateien automatisch geladen werden. Es existiert somit wieder eine Redundanz in den Daten, sodass die Menge an Funktionen, die instrumentiert werden müssen, ist sehr groß. *Android* unterteilt den Speicher in zwei Bereiche. Zum einen existiert der interne Bereich, in dem unter anderem installierte Applikationen zu finden sind. In diesem Bereich kann jede Applikation, ohne besondere Berechtigungen, eigene Daten ablegen, auf welche nicht von anderen Applikationen zugegriffen werden kann. Zum anderen gibt es den externen Bereich. Auch hier können Applikationen Daten ablegen, sofern sie über die notwendigen Berechtigungen verfügen. Der Unterschied besteht darin, dass die Daten nicht isoliert von anderen Applikationen sind und jede Applikation, mit den entsprechenden Berechtigungen, darauf zugreifen kann[5]. Ein weiterer wichtiger Unterschied zwischen den beiden Bereichen besteht darin, dass der externe Speicher durch *FUSE* bereitgestellt wird[6]. Hierbei handelt es sich um ein sogenanntes *Filesystem in Userspace*. Dies bedeutet, dass Dateisysteme durch Code im *Userspace* bereitgestellt werden können, ohne dass Modifikationen am *Kernel* notwendig sind. Ein spezielles, von *FUSE* bereitgestelltes *Kernel*-Modul stellt die notwendi-

[4] Siehe `https://developer.android.com/guide/topics/connectivity/nfc/nfc.html`, auferufen am 04.11.2016

[5] `https://developer.android.com/training/basics/data-storage/files.html`, auferufen am 13.10.2016

[6] Siehe `https://source.android.com/devices/storage/`, auferufen am 13.10.2016

gen Schnittstellen zur Kommunikation mit den Objekten des *Kernels* bereit. Häufig wird *FUSE* für virtuelle Dateisysteme verwendet[7]. Das bedeutet, dass bei einem Zugriff auf ein *FUSE*-Dateisystem potentiell andere Aktionen ausgeführt werden, als bei einem Zugriff auf ein klassisches Dateisystem. Dies trifft auf den internen Speicher nicht zu. Hierdurch ergibt sich ein Potential für Laufzeit- und Performanzunterschiede zwischen Dateioperationen auf dem internen und dem externen Speicher. Dabei bietet das *Application Framework* eine Schnittstelle zur Durchführung von Dateioperationen[8]. Zusätzlich existieren für die einzelnen Speicherbereiche teils mehrere spezielle Zugriffsfunktionen[9]. Außerdem bietet das *Android* Betriebssystem jeder Applikation den Zugriff auf eine exklusiv genutzte *SQLite* Datenbank[10]. Es ergibt sich somit wieder die Situation, dass viele verschiedene Funktionen relevante Funktionalitäten bieten und für ein möglichst vollständiges Profil einer Applikation instrumentiert werden müssten. Gleichzeitig werden bei vollständiger Instrumentierung redundante Daten aufgezeichnet, sofern die beobachtete Applikation die Komfortfunktionen verwendet. Dabei werden, analog zum Netzwerkverkehr, alle Funktionen zur Interaktion mit Dateien, auf einige wenige Systemaufrufe abgebildet. Aus diesen Gründen wird dieselbe Entscheidung wie für die Netzwerkdaten getroffen. Statt einer Erfassung aller Daten im *Application Framework*, werden die relevanten Systemaufrufe durch die Verwendung von *SystemTap* instrumentiert und nur die Nutzung von Komfortfunktionen direkt im *Application Framework* erfasst. Der hierdurch entstehende Mehraufwand zum Abspielen wird dabei ebenso in Kauf genommen, wie die Abstraktion vom Verhalten der beobachteten Applikation. Es sei noch darauf hingewiesen, dass neben den klassischen Dateioperationen (öffnen, schließen, lesen & schreiben) für jede verwendete Datei festgehalten werden soll, ob diese sich auf dem internen oder dem externen Speicher befindet. Hierdurch kann diese Information im Rahmen des Abspielens verwendet werden, um die entsprechenden Aufrufe durch *FUSE* zu leiten.

Da durch die Erfassung von Daten auf der Ebene des Kernels eine gewisse Abstraktion vom Verhalten der beobachteten Applikation stattfindet, wird die Erfassung der verbleibenden Datenquellen im *Application Framework* angesiedelt. Dabei wird zur Erfassung von Daten auf der Ebene des *Application Frameworks* kein spezielles Werkzeug, wie *SystemTap* für den *Kernel*, verwendet. Stattdessen werden die Daten erlangt, indem der Programmcode, der die relevanten Aktionen

[7] Siehe `https://www.win.tue.nl/~aeb/linux/lk/lk-8.html`, auferufen am 13.11.2016

[8] Siehe `https://developer.android.com/reference/java/io/File.html`, auferufen am 13.10.2016

[9] Siehe `https://developer.android.com/guide/topics/resources/more-resources.html#Id`, auferufen am 13.10.2016

[10] Siehe `https://developer.android.com/training/basics/data-storage/databases.html`, aufgerufen am 23.11.2016

ausführt, direkt instrumentiert wird. Dabei sollen die Daten im *Application Framework* gesammelt und anschließend an eine spezielle Applikation gesendet werden, welche die Verarbeitung der Daten vornimmt. Diese Applikation wird nach der Betrachtung sämtlicher Datenquellen vorgestellt. Somit wird die Last auf der Ebene des *Application Frameworks* möglichst minimal gehalten und auf die Ebene der Applikationen verschoben. Hierdurch soll erreicht werden, dass das Verhalten des *Application Frameworks* möglichst wenig verändert wird. Ein Blockieren der Ausführung, welches durch Code verursacht wird, welcher zur Instrumentierung eingefügt wurde, ist unter allen Umständen zu vermeiden.

Application-Lifecycle

Eine der wichtigsten Datenquellen ist der Lebenszyklus der beobachteten Applikation. Diese Wichtigkeit liegt darin begründet, dass die Daten, die durch diese Quelle anfallen, dazu genutzt werden, um in einer anschließenden Analyse Sitzungen zu erkennen, in denen der Benutzer die Applikation verwendet. Ebenfalls ist das anschließende Abspielen dieser Nutzungssitzungen ein elementarer Teil, wenn es darum geht, das Verhalten wiederzugeben. Für eine genauere Beschreibung des Lebenszyklus von Applikationen und wichtige Komponenten dieser sei auf Kapitel 2.2.5 verwiesen, da an der genannten Stelle eine genauere Beschreibung zu finden ist. Dabei reicht es nicht aus, die Applikation als ein Ganzes zu betrachten, da der Lebenszyklus einer Applikation es erlaubt, dass eine Applikation beliebig lange vor und nach der Interaktion mit dem Benutzer läuft. Der Lebenszyklus von Hintergrundkomponenten ist nur bedingt von Interesse. Dies liegt daran, dass die Aktionen, welche von den Hintergrundprozessen ausgelöst werden, durch die anderen Datenquellen erfasst werden, sofern es sich um relevante Aktionen handelt. Stattdessen bietet es sich an, die Datenquelle in die Richtung zu erweitern, dass der Lebenszyklus von Komponenten erfasst wird, welche genutzt werden, um dem Benutzer Inhalte zu präsentieren und mit diesem zu interagieren. Die zentrale Komponente, die diese Aufgabe erfüllt, ist die *Activity*. Durch die Aufnahme des Lebenszyklus von *Activities* in die Datenquelle wird es möglich, Zeiten, in denen der Nutzer mit der Applikation interagiert, wesentlich genauer zu bestimmen und abzuspielen. Denn der Lebenszyklus einer *Activity* enthält Phasen, in denen die jeweilige *Activity* in den Vordergrund kommt, bzw. für den Nutzer nicht mehr sichtbar ist. Aus diesem Grund wird neben Informationen zum Starten und Beenden von Applikationen, der Lebenszyklus von *Activities* erfasst. Außerdem soll die aufgezeichnete Applikation zu Beginn der Datenaufzeichnung einmal vollständig beendet werden. Dies hat den Zweck, dass die Aufzeichnung nicht mitten im Betrieb beginnt und hierdurch relevante Daten verloren gehen. Es wird somit sichergestellt, dass die Aufzeichnung den Beginn jeder Nutzerinteraktion erfasst.

Außerdem erleichtert diese Maßnahme sowohl das Abspielen als auch das Erfassen von Daten. Denn würde kein fester Zustand, in dem sich die Applikation zu Beginn eines erfassten Datensatzes befindet, definiert, wäre es notwendig, zum Beginn der Aufzeichnung Daten zu erfassen, die den aktuellen Zustand eindeutig beschreiben. Außerdem müsste dieser Zustand vor dem Beginn des Abspielens möglichst genau wiederhergestellt werden.

Bildschirmzustand

Um zu erfassen, zu welchen Zeitpunkten der Bildschirm ein- und abgeschaltet wird, ist keinerlei Anpassung am *Application Framework* notwendig. Wenn diese Aktionen ausgelöst werden, versendet *Android* automatisch *Intents*, welche die relevanten Informationen enthalten. Darüber hinaus sollten Veränderungen der Intensität der Hintergrundbeleuchtung erfasst werden. Hierfür stellt *Android* keine Daten zur Verfügung. Daraus folgt, dass die aktuelle Intensität periodisch erfasst werden muss, damit Änderung erkannt werden können. Hier stellt sich die Frage nach einer geeigneten Abtastrate. Darüber hinaus unterliegt die Steuerung der Hintergrundbeleuchtung im Regelfall dem Betriebssystem. Für diese Aufgabe sind spezielle Sensoren verbaut, die in der Lage sind, die Helligkeit zu messen. Somit wird die Hintergrundbeleuchtung nicht durch den Benutzer gesteuert sondern indirekt durch das Umgebungslicht. Somit ist die Steuerung der Hintergrundbeleuchtung in den seltensten Fällen direkter Teil des Verhaltens einer beobachteten Applikation. In Kombination mit der großen Last, welche durch regelmäßige Erfassung in kurzen Zeitabständen entstehen würde, ist die Entscheidung gefallen, die Hintergrundbeleuchtung und deren Intensität vom weiteren Verlauf der Arbeit auszuschließen.

Power Management

Die einzige Anforderung, die für die Funktionen des *Power Management* definiert wurde, ist die, dass die Nutzung von *Wakelocks* erfasst werden soll. Der `PowerManager` ist ein spezieller *Service*, welcher vom *Application Framework* bereitgestellt wird und diverse Funktionen für das *Power Management* bietet[11]. In diesen Funktionen ist auch eine Schnittstelle zum Anfordern, Belegen und Freigeben von *Wakelocks* enthalten. Darüber hinaus existieren Komfortfunktionen, die es erlauben sogenannte partielle *Wakelocks* anzufordern, welche nicht durch den `PowerManager` verwaltet werden. Diese Funktionen erlauben es, dass *Activities* si-

[11] Siehe `https://source.android.com/devices/tech/power/mgmt.html`, auferufen am 06.10.2016

cherstellen können, dass der Bildschirm nicht ausgeschaltet wird. Alle diese Funktionen verwenden gemeinsam die Verwaltung von *Wakelocks*, welche der *Kernel* bietet. Es ergibt sich somit die Möglichkeit, den Aufwand zu reduzieren, indem die Daten direkt im *Kernel* gesammelt werden. Dabei ergibt sich die Problematik, dass die *Wakelocks*, welche im *Kernel* verwendet werden, nicht identisch zu denen im *Application Framework* sind. Mehrere funktional identische *Wakelocks* des *Application Frameworks* können auf ein einzelnes *Wakelock* im Kernel abgebildet werden. Außerdem handelt es sich um eine sehr überschaubare Menge von Funktionen, die im *Application Framework* instrumentiert werden müssen, weswegen sich, zu Gunsten der Genauigkeit der Daten, gegen eine Erfassung im *Kernel* entschieden wird. Stattdessen werden direkt die Funktionen im *Application Framework* instrumentiert. Dabei soll sowohl die direkte Verwendung (belegen & freigeben) von *Wakelocks* im *Application Framework* erfasst werden, als auch die Verwendung der zuvor genannten Komfortfunktionen.

Batteriezustand

Es wurde die Anforderung definiert, dass der aktuelle Ladezustand der Batterie periodisch erfasst wird, um auf Basis dieser Daten Vergleiche durchzuführen. Das *Application Framework* bietet eine Funktion an, mit der genau diese Daten abgefragt werden können[12]. Aus diesem Grund ist an dieser Stelle keine Modifikation zur Instrumentierung notwendig und es werden die vorhandenen Funktionen zur Erfassung der Daten verwendet. Dabei wird die Erfassung direkt von der Applikation zur Datensammlung durchgeführt. Die Abfrage geschieht periodisch, weswegen es notwendig ist, eine geeignete Abtastrate zu wählen. Dabei muss ein Kompromiss eingegangen werden zwischen einer möglichst genauen Erfassung durch viele Messwerte und der Möglichkeit, das Gerät für längere Zeiten in Energiesparmodi zu versetzten, um die Daten nicht zu verfälschen und die Batterielaufzeit zu reduzieren.

GPS-Daten

Für die Verwendung von *GPS*-Daten wurde im Rahmen der Anforderungen keine Einschränkung gemacht. So soll sowohl die direkte Nutzung der Positionsdaten, als auch die Verwendung der diversen Komfortfunktionen protokolliert werden. Für diese Funktionalitäten bietet *Android* zwei, funktional sehr ähnliche, Schnittstellen an. Die erste Schnittstelle ist Teil des *Application Frameworks* und wird durch den

[12] Siehe `https://developer.android.com/training/monitoring-device-state/battery-monitoring.html`, auferufen am 15.10.2016

`LocationManager` bereitgestellt[13]. Diese Schnittstelle steht seit der ersten Version von *Android* zur Verfügung und bietet Zugriff auf Positionsdaten und damit verbundene Komfortfunktionen. Diese Schnittstelle soll zur Erfassung von Daten vollständig instrumentiert werden. Der `LocationManager` ist auch in aktuellen Versionen noch in *Android* enthalten, ist aber seit September 2012 als veraltet markiert. Zu dieser Zeit wurden die sogenannten *Google Play Services* veröffentlicht[14]. Bei den *Google Play Services* handelt es sich um eine System-Applikation, welche eine große Anzahl an Schnittstellen bündelt. Enthalten in dieser Menge sind Verbindungen zu diversen *Google*-Diensten (beispielsweise *maps* und *Google+*) und Schnittstellen für diverse Funktionalitäten (beispielsweise drahtloses Bezahlen). Ebenfalls ist eine Schnittstelle zur Verwendung von Positionsdaten in den *Google Play Services* enthalten. Seit der Veröffentlichung empfiehlt *Google* die Verwendung der Schnittstellen der *Google Play Services*, welche die bisher existierenden Schnittstellen ablösen sollen. Somit müsste die Schnittstelle für Positionsdaten, die in den *Google Play Services* enthalten ist (sogenannte *location API*) ebenfalls vollständig instrumentiert werden, um alle relevanten Daten zu erhalten. Dabei ergibt sich die Problematik, dass die *Google Play Services* nicht teil des *Android Open Source Project* sind[15]. Stattdessen handelt es sich bei den *Google Play Services* um ein proprietäres Programm. Daraus folgt, dass kein Zugriff auf den Programmcode möglich ist, wodurch eine Instrumentierung verhindert wird. Es existiert ein Projekt mit dem Namen *microG*, welches sich das Ziel gesetzt hat eine quelloffene Alternative zu den *Google Play Services* bereitzustellen[16]. Zum Zeitpunkt der Entstehung dieser Arbeit befindet sich das Projekt noch in Entwicklung[17]. Es werden aktuell erst wenige Schnittstellen der *Google Play Services* vollständig nachgebildet, aber die Schnittstelle zur Verwendung von Positionsdaten ist bereits vollständig nutzbar[18]. Es besteht somit die Möglichkeit die *Google Play Services* durch eine instrumentierte Version von *microG* zu ersetzten und so sämtliche Aufrufe von Funktionen zur Nutzung von Positionsdaten zu erfassen. Da der Funktionsumfang von *mi-*

[13] Siehe `https://developer.android.com/reference/android/location/LocationManager.html`, auferufen am 15.10.2016 & `https://developer.android.com/guide/topics/location/strategies.html`, auferufen am 15.10.2016

[14] Siehe `https://developers.google.com/android/guides/overview`, auferufen am 15.10.2016

[15] Siehe `https://commonsware.com/blog/2013/05/22/remember-google-play-services-proprietary.html`, auferufen am 15.10.2016

[16] Siehe `https://github.com/microg/android_packages_apps_GmsCore/wiki`, auferufen am 15.10.2016

[17] Siehe `https://github.com/microg/android_packages_apps_GmsCore/wiki/Implementation-Status`, auferufen am 15.10.2016

[18] Siehe `https://github.com/microg/android_packages_apps_UnifiedNlp`, auferufen am 15.10.2016

croG aktuell noch sehr rudimentär ist und bei der Nutzung einiger Schnittstellen Abstürze auftreten können, besteht die Gefahr, dass der Nutzungskomfort durch die Verwendung von *microG* stark eingeschränkt wird. Darüber hinaus sind viele Applikationen durch die entstehenden Abstürze nicht in Kombination mit *microG* nutzbar, wodurch die Menge der Applikationen, welche für die Datenerfassung genutzt werden können, aktuell stark eingeschränkt würde. Aus diesem Grund wurde sich gegen die Nutzung von *microG* entschieden und im Rahmen der vorliegenden Arbeit wird nur die Schnittstelle, welche vom `LocationManager` bereitgestellt wird, instrumentiert.

Prozessorlast

Das *Application Framework* bietet keine direkte Möglichkeit, um die aktuelle Prozessorlast zu erfassen. An dieser Stelle kann man sich zunutze machen, dass der *Kernel* des *Android*-Betriebssystems eine modifizierte Version des *Linux-Kernels* ist. Denn neben dem eigentlichen Kernel wird das *Android* Betriebssystem mit einigen Werkzeugen für die Kommandozeile, welche von klassischen *Linux* Systemen bekannt sind, ausgeliefert. Ein Beispiel für ein solches mitgeliefertes Werkzeug ist *top*[19]. Das Werkzeug *top* erlaubt es, Informationen über den aktuellen Betriebssystemzustand und laufende Prozesse dynamisch in Echtzeit abzufragen. Um relevante Daten über die aktuelle Prozessorlast in regelmäßigen Abständen zu erhalten, ist es notwendig, dass die Applikation, welche die Datenerfassung koordiniert, periodisch das angesprochene Werkzeug aufruft, welches anschließend die Daten aus dem *Kernel* extrahiert. Dabei besteht das Problem, dass eine Applikation keinen direkten Zugriff auf den *Kernel* besitzt. Unter anderem zu diesem Zweck bietet das *Application Framework* die `Runtime`[20]. Die `Runtime` bietet diverse Funktionen, um das Verhalten der virtuellen Maschine, in der eine Applikation läuft, zu manipulieren. Beispielsweise indem eine Ausführung des *Gabarage Collectors* ausgelöst wird, wodurch nicht mehr benötigte Elemente vollständig aus dem Speicher entfernt werden oder indem die laufende virtuelle Maschine zum Beenden gezwungen wird. Relevant zur Erfassung der Prozessorlast ist die Möglichkeit der `Runtime`, Kommandozeilenbefehle auszuführen. Durch die Verwendung dieser Funktion ist es möglich, periodisch die Prozessorlast zu erfahren. Darüber hinaus sollten zusätzliche Daten zum Prozessor erfasst werden, um die gesammelten Daten zur Prozessorlast in einen Kontext setzen zu können. Zum einen sind die Anzahl verfügbarer Prozessoren und zum anderen Daten über die aktuelle und die maximale Frequenz der einzelnen Prozessorkerne zu erfassen. Die Anzahl der Kerne kann direkt von der

[19] Siehe `http://man7.org/linux/man-pages/man1/top.1.html`, auferufen am 16.10.2016

[20] Siehe `https://developer.android.com/reference/java/lang/Runtime.html`, auferufen am 05.11.2016

`Runtime` abgefragt werden. Die Daten, welche die Frequenzen betreffen, können, wie bei einem klassischem *Linux*-Betriebssystem, über spezielle Dateien erlangt werden[21]. Durch Kombination dieser drei Methoden können alle Daten gesammelt werden, die für diese Datenquelle relevant sind.

Komfortfunktionen

An dieser Stelle sind alle zu erfassenden Datenquellen modelliert. Es wird nochmal gesondert darauf hingewiesen, dass die Funktionen der beiden Komponenten `DownloadManager` und `MediaPlayer` vollständig erfasst werden sollen. Beide Komponenten wurden bereits bei der Betrachtung von Netzwerkdaten und Ein-/Ausgabedaten betrachtet und werden hier erneut erwähnt, da ihre Erfassung durch Instrumentierung im *Application Framework* stattfindet. Für genauere Informationen über die Funktionalitäten, die diese Komponenten bieten, sei auf die Abschnitte zu Netzwerkdaten sowie Ein-/Ausgabedaten verwiesen. Die Daten, welche durch die Nutzung dieser Komponenten entstehen, erlauben es, die Genauigkeit des erfassten Verhaltens zu erhöhen. Dies ist notwendig, da die Erfassung von Daten im *Kernel* zu einer gewissen Abstraktion führt.

Applikation

Bis zu diesem Punkt wurden sämtliche Datenquellen, welche in die Datenerfassung einfließen sollen, im Rahmen der Modellierung betrachtet und ggf. eingegrenzt. Es bleibt somit noch die Applikation zur Entgegennahme der Daten übrig. Diese wird im Folgenden betrachtet. Dabei werden auch allgemeine Anforderungen, die sich auf alle Daten gleichermaßen beziehen, modelliert. Die Applikation soll dem Benutzer eine einfache Möglichkeit bieten, eine Auswahl an zu erfassenden Applikation(en) auszuwählen. Durch die Auswahl wird auch die eigentliche Datenerfassung gestartet. Sobald die Auswahl aufgehoben bzw. geändert wird, soll die Datenerfassung abgeschlossen und eine neue Erfassung gestartet werden, sofern dies notwendig ist.

Um die Last auf der Ebene des *Application Frameworks* möglichst gering zu halten, wird auf dieser Ebene keinerlei Filterung vorgenommen. Daraus folgt, dass zu jedem Zeitpunkt Daten über alle Applikationen erfasst und asynchron an die Applikation gesendet werden. Dies geschieht auch dann, wenn aktuell keine Aufnahme stattfindet. Somit ist es notwendig, dass die Applikation, welche die Daten empfängt, eine Filterung vornimmt, damit nur die Daten erfasst werden, welche die

[21] Siehe `https://www.kernel.org/doc/Documentation/cpu-freq/user-guide.txt`, auferufen am 16.10.2016

aktuell aufgezeichnete Applikation betreffen. Diese Filterung umfasst nur Daten, die im *Application Framework* gesammelt werden. Da *SystemTap Kernel*-Module zur Erfassung von Daten verwendet, ist es unabhängig vom *Application Framework*. Die Module, welche die Datenerfassung im *Kernel* vornehmen, führen selbst eine Filterung durch und schreiben relevante Daten in eigene Ausgabedateien. Die Verwendung von unterschiedlichen Ausgabedateien für *Systemtap* und Daten aus dem *Application Framework* ist auch aus dem Grund notwendig, da keinerlei Synchronisation zwischen den beiden Methoden zur Datenerfassung stattfindet. Eine solche Synchronisation würde in zusätzlicher Last resultieren. Durch den Verzicht wird diese Last in eine Vorverarbeitung verschoben, welche zwischen Aufnahme und Wiedergabe von Daten stattfinden muss.

Die Ausgabe einer einzelnen Aufnahme von Daten wird somit von drei Ausgabedateien gebildet. Zwei dieser Dateien entstehen durch die Daten, welche von *SystemTap* gesammelt werden. Dabei enthält die eine Datei sämtliche Netzwerkdaten und die andere alle Daten, welche sich auf die Ein- und Ausgabe beziehen. Die letzte Datei enthält alle übrigen Daten, welche im *Application Framework* gesammelt werden. Um das Abspielen der Daten zu vereinfachen bietet es sich an, diese drei Ausgabedateien vor dem Abspielen zusammenzuführen. Hierdurch entsteht auch kein großer Mehraufwand, da die Daten vor dem Abspielen ohnehin sortiert werden müssen. Dies liegt daran, dass die Daten aus dem *Application Framwork* asynchron an die Applikation gesendet werden und somit keine Aufzeichnung in der korrekten Reihenfolge sichergestellt ist. Dasselbe gilt für die Daten, welche von *SystemTap* gesammelt werden, da *SystemTap* auf allen Prozessorkernen ausgeführt wird und die Daten vor der Speicherung nicht extra sortiert werden. Dies geht zu Gunsten der Performanz, da ansonsten das Schreiben von manchen Daten verzögert werden müsste, um die korrekte zeitliche Reihenfolge zu gewährleisten. Auf Basis desselben Arguments werden auch die Daten, welche im *Application Framework* gesammelt werden, nicht sortiert. Die zusätzliche Last könnte die Aufzeichnung verfälschen, was möglichst zu vermeiden ist.

Für die Entwicklung des Verfahrens wäre es sinnvoll ein simples Format zur Ausgabe der Daten zu verwenden, welches einfach verarbeitet werden kann und im besten Fall sogar von Menschen lesbar ist. Hierdurch reduziert sich der Entwicklungsaufwand, da Fehler in der Ausgabe (sowohl semantische, als auch syntaktische) schneller gefunden werden können. Außerdem entsteht kein Aufwand für die Konvertierung zwischen Ausgabeformat und dem Format, das verarbeitet werden kann. Ein geeigneter Kandidat für ein solches Ausgabeformat wäre *XML* [2]. Ein weiterer Vorteil eines solchen Formats ist es, dass es auf einfache Art und Weise möglich ist, synthetische Daten zum Testen zu erzeugen. Es sei an dieser Stelle darauf hingewiesen, dass das entwickelte Verfahren die Aufnahme für mehrere Applikationen gleichzeitig unterstützt. Falls diese Möglichkeit genutzt wird, erhöht

sich der Aufwand für die Verarbeitung der Daten, bevor sie abgespielt werden können. Zwar werden die Daten, welche im *Application Framework* entstehen nach der verursachenden Applikation getrennt, dies trifft aber nicht auf die Daten zu, welche *SystemTap* liefert. Aus diesem Grund wäre eine zusätzliche Filterung der *SystemTap*-Daten, im Rahmen der Zusammenführung aller Datenquellen, notwendig. Die vorliegende Arbeit beschränkt sich auf die Aufzeichnung von einzelnen Applikationen. Dies betrifft aber ausschließlich die Werkzeuge zur Datenverarbeitung, welche in Kapitel 5.3 vorgestellt werden. Die dort beschriebenen Werkzeuge müssten angepasst werden, um die Daten verschiedener Applikationen zu trennen.

Abschließend für die Modellierung zeigt Abbildung 4.2 den Vorgang der Datenaufzeichnung in Form eines abstrahierten Sequenzdiagramms. Es ist zu erkennen, dass die Aufzeichnung aus dem *Applikation Framework* (bzw. durch den Nutzer, welcher die Aufzeichnungs-Applikation verwendet,) gestartet wird. Hierdurch wird für jede Applikation, die aufgezeichnet werden soll, eine eigene Instanz der Klasse `CapturedApplication` erzeugt. Dabei übernimmt jede Instanz die Verwaltung der Ausgabedatei für die jeweilige erfasste Applikation. Im Rahmen der Instanziierung wird jede Ausgabedatei mit dem ggf. notwendigen Rahmenkonstrukt des Ausgabeformats gefüllt. An dieser Stelle wird auch die von *SystemTap* verwendeten *Kernel*-Module gestartet. Dies ist in dem Diagramm nicht dargestellt um die Übersichtlichkeit zu wahren. Dabei würde eine Einbeziehung von *SystemTap* beinahe in einer Spiegelung des Diagramms resultieren. Denn *SystemTap* erhält für die Dauer der Datenaufzeichnung Daten aus dem *Kernel* und schreibt diese gefiltert in die Ausgabe. Dabei verwendet *SystemTap* aber keinen *Timer* zur Ausführung periodischer Aufzeichnung, da es hierfür auf der Ebene des *Kernels* keinen Bedarf gibt. Im Rahmen dieser Vorbereitungen werden die Applikationen, die im Folgenden aufgezeichnet werden sollen, auch vollständig beendet.

Nach dieser Initialisierung beginnt die eigentliche Datenaufzeichnung. Diese findet in einer Art von Schleife statt. Zu Beginn jeder Iteration wird ein *Timer* gestartet, welcher dafür sorgt, dass Prozessorlast und Batteriezustand periodisch erfasst werden. Zu jedem Zeitpunkt können beliebig viele Daten aus dem *Application Framework* an die Applikation gesendet werden. Dabei werden alle Daten zentral von einer Instanz des `CaptureService` entgegengenommen. Dieser führt die notwendige Filterung durch und gibt die Daten an die jeweilige `CapturedApplication`-Instanz, welche die Daten in die Ausgabe schreibt. Um sicherzustellen, dass die periodischen Daten den gesamten Zeitraum der Datenerfassung abdecken, werden diese Daten direkt zu Anfang und ganz am Ende der Datenaufzeichnung einmal, unabhängig der *Timer*, erfasst.

Der Benutzer hat durch die Verwendung der Aufzeichnungs-Applikation zu jedem Zeitpunkt die Möglichkeit, die Datenerfassung zu beenden. Ab diesem Zeitpunkt werden keine weiteren Daten mehr in die Ausgabe geschrieben und es wird

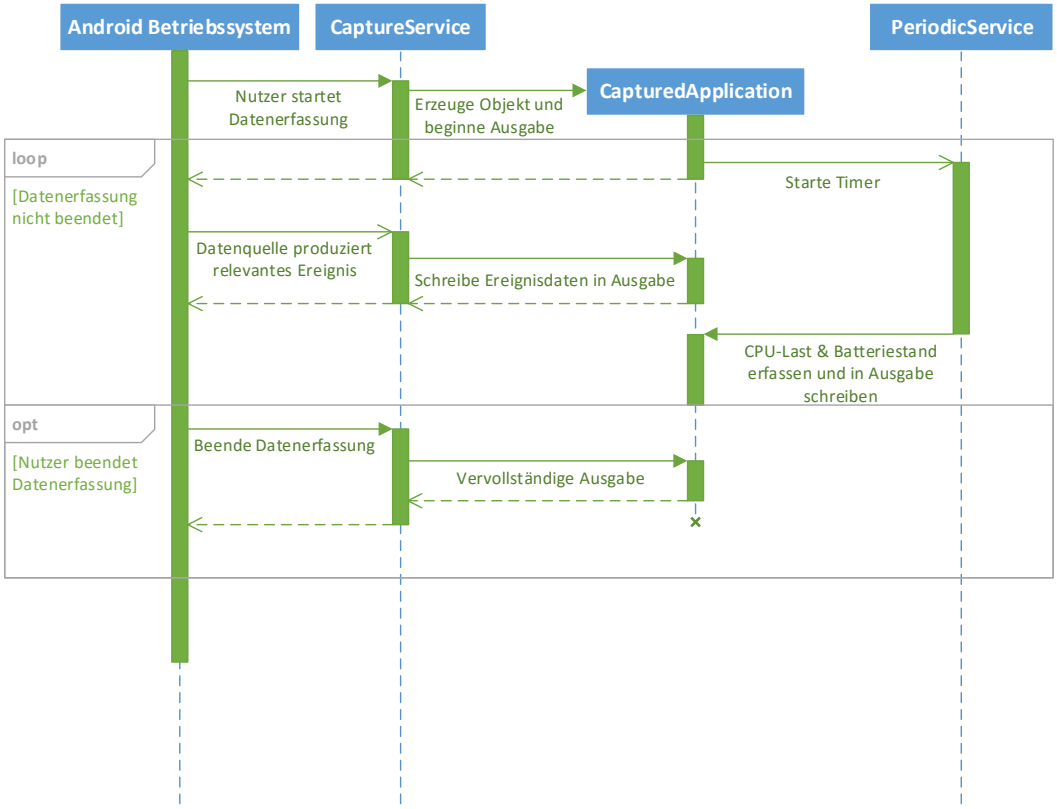

Abbildung 4.2: Vorgang der Datenerfassung, dargestellt als Sequenzdiagramm. Zur Übersichtlichkeit ist SystemTab nicht dargestellt.

nur noch die Ausgabe, unter Wahrung des Ausgabeformats, ordnungsgemäß beendet. Gleichzeitig wird die Ausführung der von *SystemTap* verwendeten *Kernel*-Module beendet.

4.2 Abspielen

Unter Berücksichtigung der Anforderungen aus Kapitel 3 und gleichzeitiger Einbeziehung von Rahmenbedingungen, die durch die Datenerfassung definiert wurden, wird in diesem Abschnitt das Verfahren zum Abspielen modelliert. Hierbei handelt es sich um den eigentlichen Benchmark, wogegen die Datenaufzeichnung die notwendigen Daten für diesen generieren soll. Die Applikation zum Abspielen wird

dabei so entworfen, dass sie nach der Installation auf einem beliebigen *Android*-Betriebssystem, welches die Anforderung an die Version erfüllt, lauffähig ist. Das einzige, was zur Aufnahme der Funktionalität bereitgestellt werden muss, sind Eingabedaten, welche abgespielt werden sollen. Nachfolgend werden zunächst Anforderungen an die Eingabedaten betrachtet und anschließend der Abspielprozess, welcher sich in zwei Phasen unterteilt, modelliert.

Anforderungen an Daten

An diese Eingabedaten werden einige Anforderungen definiert, welche erfüllt sein müssen, um ein ordnungsgemäßes Abspielen zu gewährleisten. Dabei wird für die Überführung der Ausgabe, welche von der Datenerfassung produziert wird, in das Eingabeformat der Wiedergabe ein spezielles Werkzeug verwendet, welches später in Kapitel 5.3 beschrieben wird.

Datenformat Die erste Anforderung an die Eingabedaten ist, dass diese Daten im korrekten Format vorliegen. Dies wird gesondert erwähnt, da die aufgenommenen Daten im Rahmen der Zusammenführung aller Datenquellen eine Transformation durchlaufen. Um den Aufwand dieser Transformation zu reduzieren wird das Dateiformat im Rahmen der Zusammenführung nicht verändert.

Datenreihenfolge Die nächste Anforderung ist für die korrekte Funktionsweise des Abspielens sehr wichtig. So ist es erforderlich, dass alle Einträge in der Eingabe chronologisch sortiert sind. Dies ist wichtig, da das Abspielen so entworfen wird, dass die zusätzliche Last durch die Abspiellogik möglichst klein gehalten wird. Aus diesem Grund werden die abzuspielenden Daten sequentiell verarbeitet und eventuelle Fehler in der zeitlichen Reihenfolge der Ereignisse werden ignoriert. Es wäre möglich, vor dem eigentlichen Abspielen die Daten zu analysieren und in diesem Zuge zu sortieren. Der Nachteil daran ist, dass hierdurch bei jedem Abspielen von Daten ein - unter Umständen nicht unerheblicher - Mehraufwand entsteht. Außerdem kann eine Datenaufzeichnung einen beliebigen Zeitraum umfassen. Hierdurch können sehr große Datenmengen entstehen, welche sortiert werden müssen. Da bei der Ausgabe der Datenerfassung keinerlei Sortierung gewährleistet ist, müsste der komplette Datensatz im Speicher gehalten werden oder eine nicht triviale Zwischenspeicherung in Dateien umgesetzt werden. Es ist möglich, dass das abspielende Gerät hierdurch überfordert ist. Aus diesen Gründen wird gefordert, dass die gesamte Eingabe bereits sortiert vorliegt.

Ausreichende Zustandsbeschreibung Ziel des Abspielens ist es das Verhalten einer Applikation möglichst genau nachzustellen. Hierfür ist es notwendig, dass die verwendeten Daten den Zustand der Applikation zu jedem Zeitpunkt möglichst genau darstellen. Beispielsweise wäre es für die Genauigkeit des Abspielens problematisch, wenn zu Beginn der Datenaufzeichnung die Applikation bereits läuft und *Activities* aktiv sind. Denn in diesem Fall müsste zu Beginn der Datenaufzeichnung der vollständige Zustand der Applikation erfasst und beim Abspielen wiederhergestellt werden. Es ergibt sich in dem Fall sogar eine potentielle Fehlerquelle, wenn der Zustand nicht vollständig erfasst wurde, wodurch gewisse Aktionen nicht mehr korrekt wiedergegeben werden können. Zum Beispiel könnte eine Applikation zu Beginn der Datenaufzeichnung eine Netzwerkverbindung Nutzen, welche nicht erfasst wurde, was darin resultieren kann, dass versucht wird, eine nicht vorhandene Netzwerkverbindung zu schließen bzw. zum Versenden von Daten zu verwenden. Der Zustand, welcher zu Beginn der Aufzeichnung erfasst werden muss, ist sehr groß, da Aspekte zu allen Datenquellen enthalten sein müssen. Dabei ist es nicht möglich, alle diese Daten auf einfache Art und Weise zu erhalten. Aus diesem Grund wurde die Datenaufzeichnung so modelliert, dass die aufgezeichnete Applikation zu Beginn der Erfassung vollständig beendet wird. Hierdurch und durch die anderen erfassten Daten wird sichergestellt, dass der aktuelle Zustand, welcher wiedergegeben werden soll, zu jedem Zeitpunkt eindeutig und vollständig definiert ist.

Vollständigkeit Außerdem wird gefordert, dass jeder Eintrag in dem abzuspielenden Datensatz vollständig ist und die enthaltenen Informationen ausreichen, um das Ereignis zu reproduzieren. Zum einen soll während/vor dem Abspielen keine Logik zur Analyse notwendig sein, die fehlende Informationen rekonstruiert und zum anderen können gewisse Informationen nicht aus dem Kontext gewonnen werden und müssen zwingend in den Daten enthalten sein. Beispielsweise kann bei einer Abfrage der aktuellen Position im `LocationManager` eine Genauigkeit übergeben werden, die eingehalten werden muss. Solche Informationen sind nachträglich nicht rekonstruierbar (und selbst wenn sie es sind, erzeugt die Reproduktion unnötige Last).

Semantische Korrektheit Zuletzt wird für die Daten gefordert, dass sie semantisch korrekt sind und nicht in ungültige (System-)Zustände führen oder Fehler produzieren. Beispielsweise kann auf eine Datei nur lesend und schreibend zugegriffen werden, wenn sie geöffnet, aber noch nicht geschlossen wurde. Auch eine Activity kann nur dann beendet werden, wenn sie zuvor gestartet wurde. Zugunsten der Performanz werden keinerlei Prüfungen während des Abspielens durchgeführt.

Stattdessen wird sich auf eine gewissenhafte Modellierung und Implementierung der Datenerfassung verlassen.

Zusammenfassend lässt sich sagen, dass die meisten Anforderungen, die an die Eingabe des Abspielens gestellt werden, erfordern, dass die Datenerfassung und die Werkzeuge zur Datenverarbeitung korrekt funktionieren und mit bedacht Entwickelt wurden.

Nun wird das eigentliche Verfahren zum Abspielen und die dazugehörige Applikation modelliert. Dabei fällt die Modellierung der Applikation einfach aus. Es soll einer der verfügbaren Datensätze ausgewählt werden können, welcher im Anschluss abgespielt wird. Dabei wird während des Abspielens die Benutzeroberfläche der Applikation geschlossen, da der Abspielprozess das Ein- und Ausblenden einer Benutzeroberfläche steuert. Der eigentliche Abspielprozess ist dabei in zwei Phasen unterteilt. In der ersten Phase werden die Daten einmal vollständig gescannt und Informationen gesammelt, die notwendig sind um das anschließende Abspielen möglichst einfach zu gestalten. Nach dieser Datenanalyse findet das eigentliche Abspielen der Daten statt. Diese beiden Phasen werden nun nacheinander genauer betrachtet.

Vorverarbeitung

Dem eigentlichen Abspielen von Daten geht eine Vorverarbeitung der Daten voraus. Diese hat den Zweck, die Last während des Abspielens zu reduzieren. Um dieses Ziel zu erreichen, werden Informationen, die für ein korrektes Abspielen notwendig sind, im Vorfeld aus den Daten extrahiert. Im Zuge dessen wird der Datensatz einmal vollständig durchlaufen und diverse Informationen gesammelt. Nachfolgend werden die Datenquellen benannt, die im Rahmen der Vorverarbeitung von Interesse sind und die Aktionen, die ausgeführt werden müssen, aufgezeigt.

Ein-/Ausgabe Zunächst werden alle Dateien, welche im Rahmen des Abspielens verwendet werden, identifiziert. Dies ist notwendig, da es für verschiedene Dateioperationen notwendig ist, dass die Dateien bereits im Vorfeld, mit der richtigen Größe, existieren. Diese Aktionen umfassen das Lesen sowie das Verschieben der aktuellen Lese-/Schreibposition. Der Inhalt der Dateien spielt keine Rolle, nur deren Existenz ist notwendig. Deshalb werden, nach dem Scannen des kompletten Datensatzes, sämtliche Dateien, die identifiziert wurden, mit der korrekten Größe erstellt. Dabei wird auch die Speicherposition (interner oder externer Speicher) miteinbezogen. Außerdem sollen sämtliche Dateien, die noch von anderen Abspielvorgängen vorhanden sind, gelöscht werden, um die Gefahr zu reduzieren, dass der verfügbare Speicher nicht mehr ausreicht.

Netzwerkdaten Ebenfalls werden sämtliche Verbindungen gesammelt, die für den Netzwerkverkehr verwendet werden. Dabei besteht eine Verbindung aus einem Netzwerkprotokoll, Sender- und Empfänger-Adresse sowie einer Menge von gesendeten bzw. empfangenen Nachrichten. Für diese Nachrichten sind die Größe und die Zeitpunkte, an denen sie auftreten, bekannt, nicht aber der Inhalt, da dieser keine Relevanz für das Verfahren hat. Die Extraktion der Netzwerkaktivitäten ist notwendig, um die Genauigkeit des Abspielens zu erhöhen. Weil die Daten während des Abspielens rein sequentiell durchlaufen werden, ist ein vorheriges Sammeln von relevanten Informationen notwendig. Bei der Applikation, deren Verhalten aufgezeichnet wird, treffen Netzwerkdaten auf Basis bestimmter Ereignisse ein. Kenntnis über diese Ereignisse kann nur erlangt werden, indem die Applikation ausführlich analysiert und der Inhalt von Netzwerkdaten betrachtet wird. Sofern die Möglichkeit dazu besteht (Quellcode für alle Teilnehmer der Kommunikation verfügbar), würde dies in einem erheblichem Aufwand resultieren, welcher der möglichst einfachen und generischen Nutzung des Verfahrens zuwiderläuft. Es stehen drei Möglichkeiten zur Verfügung, um ohne diese Kenntnisse die notwendigen Daten zu erfassen.

Bei der ersten Möglichkeit wird zu dem Zeitpunkt, an dem Daten empfangen werden sollen, eine Nachricht an einen speziellen Service geschickt. Diese Nachricht enthält die Größe der erwarteten Antwort und wird direkt versandt. Hierdurch ergibt sich eine gewisse Ungenauigkeit in den Daten. Zum einen ist es nicht möglich, Daten pünktlich zu empfangen, da diese zunächst immer angefordert werden müssen und somit eine Verzögerung entsteht. Viel problematischer an diesem Verfahren ist, dass für jede empfangene Nachricht eine Nachricht verschickt werden muss, welche nicht im abgespielten Datensatz enthalten ist, was in einer großen Abweichung vom aufgezeichneten Verhalten resultiert. Abbildung 4.3 illustriert diesen Sachverhalt. Die blauen Balken markieren die Zeit, in denen die Verbindung aktiv ist. Die grünen und roten Balken stellen die Zeiten dar, in denen Daten gesendet, bzw. empfangen werden. Die Länge der Balken repräsentiert die Größe der Nachrichten. Es ist gut zu erkennen, dass zwischen Datenanforderung und -empfang eine weitere Verzögerung entsteht, was in den Laufzeiten der Nachrichten begründet ist. Aus Gründen der Einfachheit ignoriert die Abbildung eventuelle Schwankungen in den Bandbreiten der Übertragungen.

Die zweite Möglichkeit besteht darin, dass bei jeder Nachricht, die gemäß des Datensatzes versandt wird, die Größe der Nachricht übermittelt wird, welche als nächstes empfangen werden soll. Ebenfalls wird in der Nachricht übermittelt, wann die Daten empfangen werden sollen. Dies ist die Differenz aus der aktuellen Zeit und der gewünschten Ankunftszeit. Hierdurch ergibt sich eine Mindestgröße für versendete Nachrichten, wodurch eine Differenz zum Datensatz entstehen kann, wenn dieser das Versenden von sehr kleinen Nachrichten vorsieht. Es ergibt sich

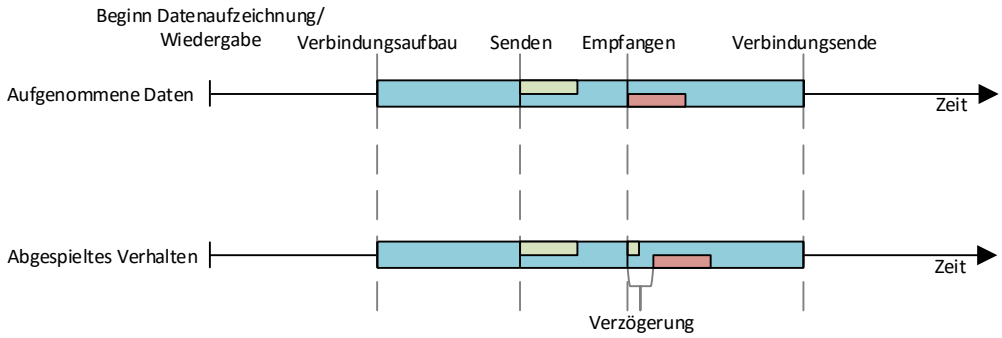

Abbildung 4.3: Verzögerung beim Empfang von Netzwerkdaten, verursacht durch die Datenanforderung.

somit auch bei diesem Verfahren eine Abweichung von dem Verhalten, dass wiedergegeben werden soll. Dies tritt aber nur dann auf, wenn besonders kleine Nachrichten versandt werden sollen. Die minimale Größe wird dabei durch die Größe der Zeichenkette bestimmt, welche notwendig ist, um die Nachrichtengröße in *Bytes*, die Verzögerung in Millisekunden und Trennzeichen abzubilden. Damit lässt sich abschätzen, dass die minimale Nachrichten Größe geringer als 1 *Kilobyte* ist. Darüber hinaus ist die Abweichung vom tatsächlichen Verhalten, im Vergleich zur ersten Variante, sehr gering, da nur geringfügig zu viele Daten verschickt werden. Denn in der ersten Möglichkeit entstehen vollständig neue Nachrichten, die nicht vorgesehen waren. Abbildung 4.4 veranschaulicht dieses Verfahren. Dabei ist die Bedeutung der Farben identisch zu Abbildung 4.3. Die Graphik zeigt deutlich, dass die Daten pünktlich eintreffen, dafür aber die gesendeten Daten größer sind, als in den aufgenommenen Daten verzeichnet ist.

Die dritte Möglichkeit besteht darin, aus den Daten, im Rahmen der Vorverarbeitung, ein vollständiges Profil des Netzwerkverkehrs zu erstellen und dieses an den Service, welcher Daten entgegennimmt und versendet, zu übertragen. Dieser Service würde dann automatisiert die Daten zu den passenden Zeitpunkten versenden, ohne dass die Gefahr besteht, dass zu große Dateien versendet werden. Problematisch an dieser Methode ist, dass es keine Möglichkeit gibt, eventuelle Verzögerungen zu berücksichtigen, welche durch Unterschiede in der *Hardware* von aufnehmendem und abspielendem Gerät entstehen.

Im direkten Vergleich zeigt sich, dass die Abweichung bei der zweiten Methode geringer als bei der ersten ist. Außerdem spiegelt die zweite Methode das Verfahren besser wieder, als die dritte. Aus diesem Grund wird die zweite Methode für das

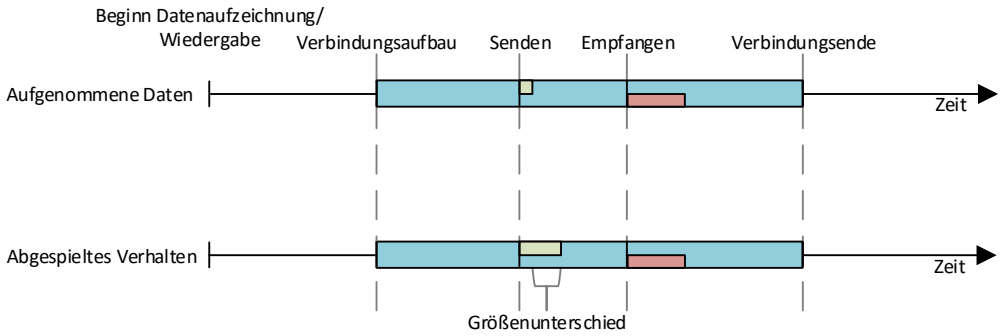

Abbildung 4.4: Größenunterschied beim Senden von Netzwerkdaten, verursacht durch die Datenanforderung.

Verfahren verwendet.

Außerdem ist es während der Vorverarbeitung möglich, beliebige Daten aus dem Datensatz zu extrahieren und mit diesen dann das Verfahren zu konfigurieren. Beispielsweise kann aus speziellen Ereignissen Wissen darüber erlangt werden, wo der interne und wo der externe Speicher zu finden sind. Da dies bei verschiedenen Versionen von *Android* unterschiedlich sein kann ist das Wissen darüber notwendig um ein akkurates Abspielen zu gewährleisten. Genauere Informationen über solche Daten werden in Kapitel 5 geliefert.

Wiedergabe

Die nachfolgende Modellierung des eigentlichen Abspielvorgangs geschieht unter Berücksichtigung der Daten, welche im Rahmen der Vorverarbeitung gesammelt wurden. Es wurde zuvor bereits darauf hingewiesen, dass die Daten während des Abspielens sequentiell durchlaufen werden. Dabei muss für jedes Ereignis festgelegt werden, wann es abgespielt werden soll. Es ist das Ziel, die erfassten Daten möglichst genau wiederzugeben. In diesem Zusammenhang wird die Gesamtlaufzeit der Aufzeichnung als wichtiges Kriterium betrachtet. Daher soll die Zeit, welche das Abspielen dauert, eine möglichst geringe Differenz zu der Zeitspanne, welche die Aufnahme umfasst, aufweisen. Aus diesem Grund werden abzuspielende Aktionen unter Umständen verzögert. Denn zur Planung, wann welches Ereignis wiedergegeben wird, wird die Zeitspanne zwischen dem Beginn der Datenerfassung und dem Auftreten eines Ereignisses verwendet. Die Zeit zwischen dem Beginn der Wiedergabe und der Wiedergabe eines Ereignisses soll möglichst identisch zu dieser Zeitspanne sein.

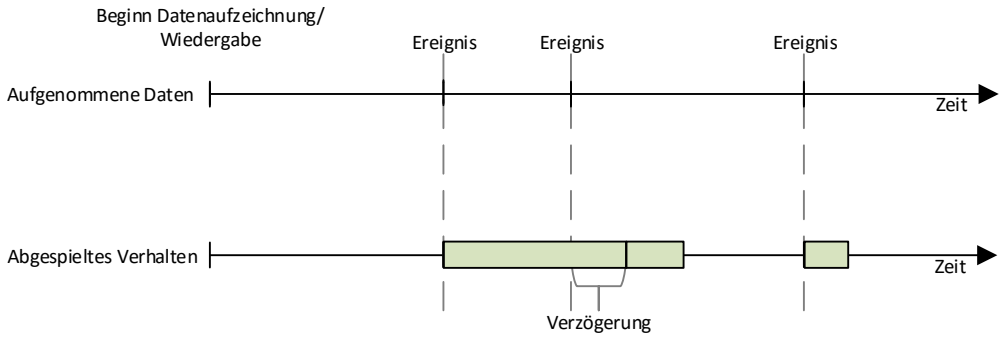

Abbildung 4.5: Abspielen eines Ereignisses wird durch die Ausführung eines vorherigen Ereignis verzögert.

Solange die Abspielzeit eines Ereignisses nicht länger dauert als die Zeit zwischen dem jeweiligen Ereignis und dem darauffolgenden, ist das Zeitverhalten beim Abspielen identisch zum aufgezeichneten Datensatz. In weniger optimalen Fällen kann es aber vorkommen, dass sich zu dem Zeitpunkt, zu dem ein Ereignis abgespielt werden soll, das Vorherige sich noch in Ausführung befindet. Dann wird das abzuspielende Ereignis sofort, wenn die vorherige Ausführung abgeschlossen ist, abgespielt um die Abweichung vom Datensatz minimal zu halten. Da sich die Ausführungszeitpunkte aller Ereignisse auf den Beginn des Abspielens beziehen und nicht auf das vorherige Ereignis, wird die Verzögerung, sofern möglich, wieder abgebaut. Solche Überlappungen können aus verschiedenen Gründen entstehen. Beispielsweise kann die aufgezeichnete Applikation viele Aufgaben parallel abarbeiten oder das Gerät, auf dem die Aufnahme vorgenommen wurde, ist performanter als das abspielende Gerät. In Abbildung 4.5 wird nochmal dargestellt, wie die Verzögerung bei einer Überlappung behandelt wird. Die farbigen Balken stellen dabei die Ausführungszeiten dar.

Eine Alternative wäre es, anstatt der Zeitspanne zwischen einem Ereignis und Beginn der Datenerfassung, die Zeit zwischen Ereignissen als Bezugswert zu nehmen. Dieses Verfahren bringt die Problematik mit sich, dass Verzögerungen im Laufe des Abspielens nicht ausgeglichen werden, sondern vollständig durch das Abspielen erhalten bleiben. Statt einen Ausgleich vorzunehmen, addieren sich sogar alle Verzögerungen, welche in einer längeren Gesamtlaufzeit des Abspielvorgangs resultieren. Bei einem entsprechend großen Datensatz, der abgespielt werden soll und dabei eine ausreichende Anzahl (kleiner) Verzögerungen erzeugt, ergibt sich am Ende des Abspielvorgangs ein erheblicher Laufzeitunterschied zwischen Datensatz

und abgespieltem Verhalten.

Um die entstehende Abweichung quantifizieren zu können, wird ein Test durchgeführt. Basis dieses Tests ist ein synthetischer Datensatz, welcher jedes unterstützte Ereignis mindestens einmal enthält. Insgesamt umfasst der Datensatz circa 70 Ereignisse, dessen Abspielen etwa 60 Sekunden dauert. Dabei unterschreitet die Zeit zwischen zwei Ereignissen niemals die Grenze von 20 Millisekunden. Nachdem dieser Datensatz fünfmal mit beiden Varianten wiedergegeben wurde, werden die Ausführungszeiten gemittelt und die Abweichung betrachtet. Bei der ersten Methode, welche die Startzeit der Datenaufzeichnung als Bezugspunkt wählt, liegt die mittlere Abweichung unter 1% bei einem Höchstwert von 1,2%. Bei der zweiten Variante, welche dafür sorgt, dass die Zeitspanne zwischen Ereignissen möglichst gleich zu denen im Datensatz ist, liegt die mittlere Abweichung bei knapp 80% mit einem Höchstwert von 83,3%. Somit dauert die Wiedergabe des Datensatzes mit dem zweiten Verfahren über 100 Sekunden. In einem gewissen Rahmen lässt sich die Abweichung durch die synthetische Natur des Datensatzes erklären, da kein besonderes Augenmerk auf die Realitätsnähe der Ankunftszeiten von Ereignissen gelegt wurde. Eine ähnliche Situation könnte bei realen Daten auftreten, wenn das abspielende Gerät (in Teilen) über schlechtere *Hardware* verfügt, als das Gerät, welches die Daten erfasst hat. Insgesamt zeigt sich, dass die erste Variante das Verhalten, welches im Datensatz beschrieben ist, wesentlich besser wiedergibt.

Auch wenn es Applikationen möglich ist, Aufgaben periodisch und/oder verzögert ausführen zu lassen, ist es der Regelfall, dass ausgeführte Aktionen direkt durch die Interaktion des Benutzers ausgelöst werden. Daraus folgt die Vermutung, dass die Ereignisse, welche in den Daten auftauchen, möglichst zeitnah zum angegebenen Zeitpunkt wiedergegeben werden sollen und nicht eine feste Zeitspanne nach dem vorherigen Ereignis. Als Beispiel können zwei unterschiedlich leistungsfähige Geräte betrachtet werden. Auf beiden Geräten wird eine Applikation identisch genutzt, wodurch identische Ereignisketten produziert werden. Auf dem Gerät mit der höheren Leistung können unter Umständen große Zeitspannen der Inaktivität zwischen den Ereignissen beobachtet werden, wogegen auf dem langsamen Gerät diese ggf. vollständig fehlen. Darüber hinaus ist es möglich, dass die Gesamtlaufzeit, bei identischen ausgeführten Aktionen, auf dem langsamen Gerät größer ist als auf dem schnelleren Gerät.

Aus diesen Gründen wird das zuvor genannte Verfahren, welches sich auf den Startzeitpunkt der Datenerfassung bezieht, verwendet, um ein möglichst genaues Abspielen zu erzielen. Um einen besseren Vergleich der beiden vorgestellten Abspiellogiken zu ermöglichen wird in Abbildung 4.6 gezeigt, wie Verzögerungen, durch den Bezug auf die Zeit zwischen Ereignissen, erhalten bleiben. Die Zeiten zwischen den Ereignissen sind dabei durch die blauen Balken repräsentiert, wogegen die grünen Balken Ausführungszeiten repräsentieren. Es lässt sich gut erkennen,

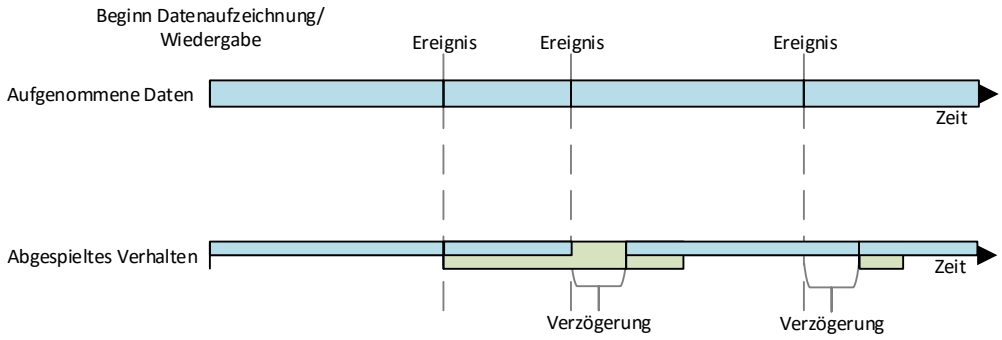

Abbildung 4.6: Verzögerungen bleiben durch die alternative Abspiellogik über die gesamte Dauer erhalten.

dass die Zeit zwischen den Ereignissen als untere Schranke die Werte hat, die der Datensatz vorgibt, aber durch die tatsächliche Ausführung verzögert werden kann.

Dabei werden zum Abspielen der einzelnen Ereignisse ausschließlich die Schnittstellen verwendet, welche vom *Application Framework* für alle Applikationen zur Verfügung gestellt werden. Es wird versucht, sofern dies Möglich ist, dieselben Funktionen zu verwenden, die für die Erzeugung der Ereignisse gesorgt haben. Hierdurch wird die beste Genauigkeit des Abspielens erreicht. Diese Möglichkeit ist nicht für Daten zu Dateioperationen und Netzwerkverkehr verfügbar, da bei der Erfassung dieser Daten vom tatsächlichen Verhalten der Applikation abstrahiert wird.

Für diese Daten werden möglichst simple Aktionen gewählt, die das Verhalten wiedergeben. Denn es soll zwar das originale Verhalten wiedergegeben werden, aber es sollen keine neuen Verhaltensaspekte hinzugefügt werden. Die Vereinfachung, die durch die Abstraktion entsteht ist nicht weiter kritisch. Denn wenn bei der Abstraktion relevante Informationen verloren gegangen sind (beispielsweise Prozessorlast), finden sich die Informationen darüber in anderen Ereignissen des Datensatzes. Aus diesem Grund wird für den Netzwerkverkehr einfaches Senden und Empfangen, ohne irgendwelche Komfortfunktionen gewählt. Dabei werden die Transferprotokolle verwendet, welche in den abzuspielenden Daten genannt sind (*TCP* oder *UDP*). Sofern auf einer Verbindung zuerst Daten gesendet und im Anschluss empfangen werden sollen, werden an die gesendeten Daten Informationen angehängt, wann eine Nachricht in welcher Größe als Antwort erwartet wird. Findet vor dem Empfang kein Versand statt, bzw. wenn Daten häufiger empfangen als verschickt werden, wird eine gesonderte Nachricht abgesetzt, die eine direkte Antwort in der

korrekten Größe erwartet. Wenn besonders kleine Nachrichten verschickt werden, kann dieses Verfahren zu verschickten Nachrichten führen die minimal größer, als im Datensatz spezifiziert, sind. Wie zuvor angesprochen liegt die minimale Größe im Bereich von wenigen *Kilobytes*. Um diese zukünftigen Nachrichten umzusetzen, werden die Daten, die in der Vorverarbeitung gesammelt wurden verwendet. Dabei muss sichergestellt werden, dass einzelne eingehende Nachrichten nicht mehrmals für die Zukunft angefordert werden. Für Dateioperationen werden simpelste Lese- und Schreiboperationen ausgewählt um das aufgenommene Verhalten wiederzugeben. Damit diese Operationen auf die richtigen Dateien angewandt werden, wird eine Abbildung von den Dateien, die im Datensatz erwähnt werden, auf die Dateien, welche in der Vorverarbeitung angelegt wurden, verwendet.

Besondere Beachtung sollte an dieser Stelle noch den Ereignissen, die sich auf den *Activity*-Lebenszyklus beziehen, gewidmet werden. Um diese Ereignisse wiederzugeben wird eine *Activity* erzeugt, die über keinerlei Benutzeroberfläche verfügt. Sie kann jedoch Nachrichten entgegennehmen, um gestartet zu werden und spezielle Aktionen auf der *Activity*-Ebene auszuführen. Beispielsweise um zu verhindern, dass der Bildschirm ausgeht. Darüber hinaus kann sich die *Activity* selber beenden, um den eigenen Lebenszyklus abzuschließen.

Das Abspielen eines Datensatzes ist unter Umständen eine lang andauernde Aufgabe, welche über längere Zeiträume inaktiv ist, da die aufgezeichnete Applikation während der Aufnahme nicht durchgehend genutzt wurde. Um sicherzustellen, dass das abspielende Gerät zur korrekten Zeit wieder Ereignisse abspielen kann, muss verhindert werden, dass das Gerät in einen Energiesparmodus wechselt, der die abspielende Applikation suspendiert. Zu diesem Zweck wird für die gesamte Dauer des Abspielens (Vorverarbeitung & Wiedergabe) ein *Wakelock* gehalten, welches tiefe Energiesparmodi des Prozessors verhindert. Hierdurch verlieren die Vergleiche des Batteriezustandes zwischen Datensatz und Wiedergabe an Bedeutung, aber ein ordnungsgemäßes Abspielen wird sichergestellt.

Abschließend zeigt Abbildung 4.7 den vollständigen Abspielvorgang in Form eines Sequenzdiagramms. Dabei wird die Vorverarbeitung nicht vollständig dargestellt, um die Übersichtlichkeit der Abbildung zu wahren. Es ist erkennbar, dass die Wiedergabe direkt auf die Vorverarbeitung folgt und dass die Wiedergabe sequentiell den Datensatz abarbeitet. Es ist zu sehen, dass verschiedene Ereignisse im Datensatz auf verschiedene Art und Weise wiedergegeben werden und dass zwischen den einzelnen Ereignissen ggf. gewartet wird, um die Zeiten gemäß des Datensatzes zu reproduzieren. Ebenfalls ist erkennbar, dass ein *Wakelock* über die gesamte Dauer des Abspielens gehalten wird.

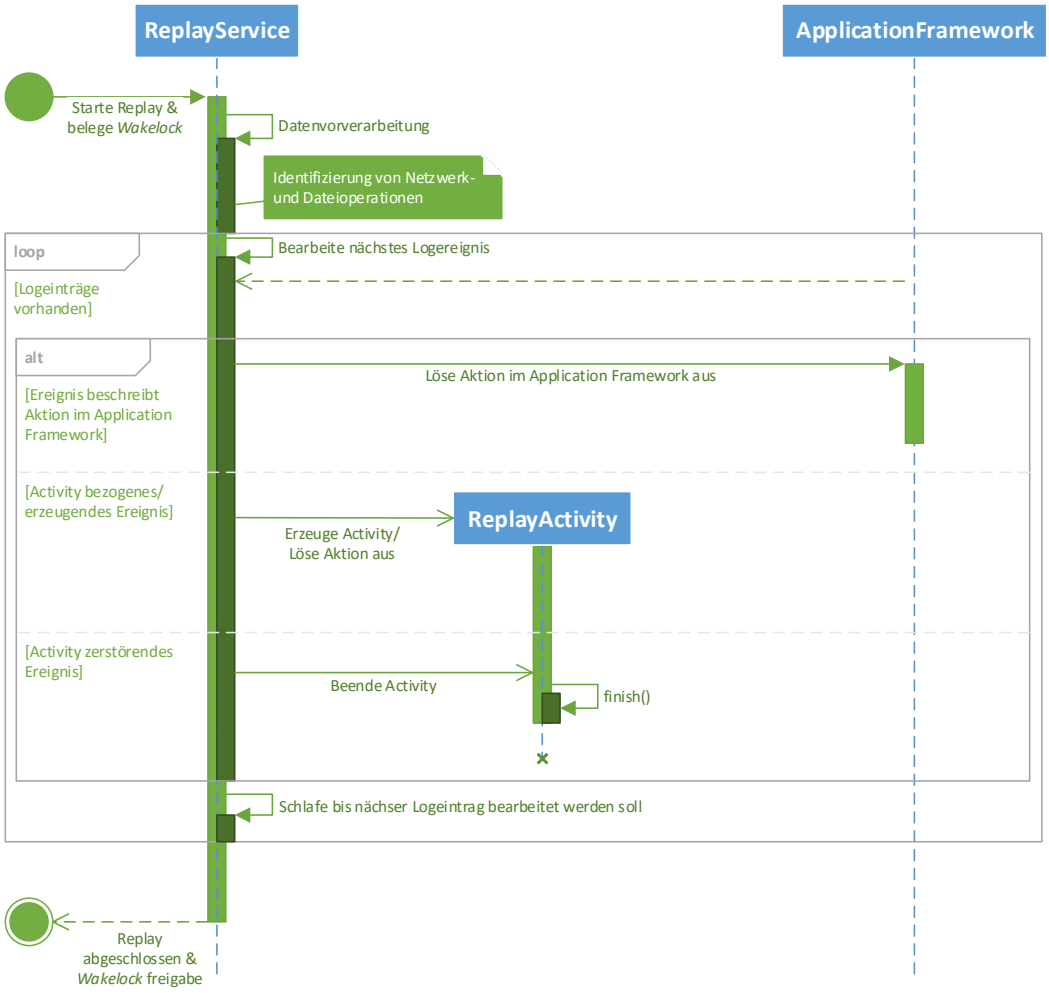

Abbildung 4.7: Vorgang der Datenwiedergabe, dargestellt als Sequenzdiagramm.

5 Implementierung

In diesem Kapitel werden Details der Implementierung betrachtet. Diese betreffen die Prozesse zur Datenerfassung und -wiedergabe. Im Anschluss daran werden die Werkzeuge vorgestellt, welche verwendet wurden, um die erfassten Daten aufzubereiten und/oder zu analysieren. Ebenfalls wird ein Werkzeug vorgestellt, welches notwendig ist, um die korrekte Funktionalität der Wiedergabe zu gewährleisten. Außerdem werden im Rahmen dieses Kapitels Problematiken aufgezeigt, welche eine vollständige Erfüllung sämtlicher Anforderungen verhindern und welche Einschränkungen hierdurch entstehen.

Für die Implementierung wird ein Smartphone des Typs *Galaxy S* des Herstellers *Samsung*[1] verwendet.

5.1 Aufzeichnen

Im Rahmen der Implementierung der Datenerfassung wurden einige Entscheidungen getroffen, welche diverse Auswirkungen auf das gesamte Verfahren besitzen. Aus diesem Grund sollen diese Entscheidungen und deren Folgen in diesem Abschnitt genauer betrachtet werden.

5.1.0.1 Berechtigungen

Eine wichtige Anforderung an das entwickelte Verfahren ist es, dass jede Applikation, die auf dem aufnehmenden Gerät installiert ist, zur Datenerfassung gewählt werden kann. Aus diesem Grund ist es notwendig, dass die Applikation, welche die Aufnahme durchführt, eine Liste aller Applikationen erhalten kann, wenn die Applikation zur Datenaufnahme gewählt wird. Die Information darüber, welche Applikationen auf dem Gerät installiert sind, wird von den Entwicklern als sensibel eingestuft und erfordert eine spezielle Berechtigung. Für den Zugriff auf einige besonders sensible Daten werden Berechtigungen benötigt, welche nur Systemapplikationen erhalten können. Hierzu gehört auch die Liste der installierten Applikationen.

[1] Siehe `http://www.gsmarena.com/samsung_google_nexus_s-3620.php`, aufgerufen am 08.12.16

Es ergibt sich somit die Notwendigkeit, die Applikation als Systemapplikation zu entwickeln. Hierfür ist es notwendig, dass die Applikation mit der Signatur des Betriebssystems signiert wird[2]. Aus Sicherheitsgründen ist es in aktuellen Versionen von *Android* nicht möglich zusätzliche Systemapplikationen zu installieren. Solche Applikationen müssen gemeinsam mit dem Betriebssystem ausgeliefert werden. Eine zusätzliche Applikation in die Installation von *Android* einzufügen ist ohne erheblichen Aufwand möglich[3], erschwert aber den Entwicklungs- und Testaufwand erheblich, da für jede neue Version der Applikation ein neues *Android* gebaut und installiert werden muss. Der Zeitaufwand der hierfür anfällt hängt von der verwendeten Maschine ab. Im vorliegenden Fall dauerte es in etwa 40 Minuten, was im Vergleich zu den 30 Sekunden, welche die Installation der App auf einem bestehenden System dauert, nicht akzeptabel ist. Aus diesem Grund wird dieser Sicherheitsmechanismus von *Android* für die Entwicklung deaktiviert. Darüber hinaus wird die finale Version der Applikation in das Betriebssystem integriert, wodurch es möglich ist den Sicherheitsmechanismus wieder zu aktivieren.

Der Codeabschnitt, welcher die Überprüfung bei der Installation neuer Applikationen durchführt, kann in der Datei `android/frameworks/base/services/core/java/com/android/server/pm/PackageManagerService.java` gefunden werden. Ein entfernen der genannten Überprüfung erlaubt es Systemapplikationen in ein laufendes System zu installieren. Da es Systemapplikationen möglich ist vollständige Kontrolle über das ausführende System zu erhalten wird aber davon abgeraten eine *Android* Version ohne diesen Mechanismus zu verteilen und produktiv zu nutzen. Es eignet sich lediglich zu Testzwecken.

5.1.0.2 Maximale Genauigkeit

Das Verhalten, das sich in den erfassten Daten befindet, soll möglichst genau die erfasste Applikation widerspiegeln. Deswegen sollte die Auslösung für die Information, wann Ereignisse aufgetreten sind, möglichst hoch gewählt werden. Das *Android* Betriebssystem bietet eine Möglichkeit die vergangene Zeit seit dem Systemstart, gemessen in Nanosekunden, zu erfahren. Hierbei ergeben sich mehrere Probleme. Zum einen ist es möglich, dass das Gerät während des Vorgangs der Datenerfassung neu gestartet wird, wodurch die notwendige Sortierung der Daten erschwert wird, da es nicht mehr möglich ist die Daten einfach nach Zeitstempeln zu sortieren. Es wäre somit notwendig eine andere Vorverarbeitung der Daten zu wählen (beispielsweise indem spezielle Steuerdaten erfasst werden), um die Daten für

[2]Siehe `https://source.android.com/devices/tech/ota/sign_builds.html`, aufgerufen am 21.11.2016

[3]Siehe `https://wiki.cyanogenmod.org/w/Doc:_adding_your_own_app`, aufgerufen am 25.11.2016

die Analyse und das Abspielen aufzubereiten. Zum anderen ist ein nicht unwesentlicher Anwendungszweck der Daten die Analyse des Nutzerverhaltens. Verwendet man reale Zeitangaben, wie beispielsweise *UNIX*-Zeitstempel, ist es möglich die Zeit mit in die Analysen mit einzubeziehen. Verwendet man stattdessen die Zeit seit dem Systemstart als Bezugspunkt fällt diese Möglichkeit weg, außer es werden zusätzliche Daten erfasst, um diese Information zu rekonstruieren. Ein weiterer (vernachlässigbarer) Nebeneffekt ist, dass durch die Verwendung von Nanosekunden für Zeitinformationen die Größe der Ausgabedateien schneller ansteigt, als im Vergleich zu Zeitstempeln mit einer Auflösung von Millisekunden. Außerdem muss beachtet werden, dass die Verwendung der Zeit in Nanosekunden nicht sicherstellt, dass diese Datenquelle auch mit einer Präzision von Nanosekunden aktualisiert wird. Guihot et al. stellten fest, dass die tatsächliche Präzision auf verschiedenen Geräten stark unterschiedlich ist [16].

Der nächste Schritt mit einer geringeren Auflösung wären Zeitinformationen mit einer Genauigkeit von Mikrosekunden zu speichern. Das *Android* Betriebssystem bietet keine Möglichkeit an, um die aktuelle Zeit bzw. verstrichene Zeit mit einer Mikrosekunden Auflösung zu erhalten, weswegen auch diese Auflösung sich nicht direkt anbietet. Es wäre notwendig die Zeiten, die in einer Auflösung von Millisekunden vorliegen, künstlich in Mikrosekunden umzuwandeln.

Die letzte Alternative für die feingranulare Erfassung von Daten eine Auflösung in Millisekunden. *Android* bietet sowohl einen *UNIX*-Zeitstempel, als auch die verstrichene Zeit seit Systemstart in Millisekunden an. Bei der Verwendung dieser Zeitstempel wären keine weiteren Anpassungen notwendig, um die Daten auch optimal für die Analyse verwenden zu können.

Ein zusätzlicher Aspekt sollte bei der Wahl der Genauigkeit von Zeitdaten betrachtet werden. Und zwar wie genau es möglich ist die Daten im Anschluss wieder abzuspielen. Es stellt sich somit die Frage, wie hoch die Genauigkeit ist, wenn ein *Thread* für eine bestimmte Zeitspanne schlafen gelegt werden soll, um im Anschluss das nächste Ereignis abzuspielen. Durch die Genauigkeit die für diese Aufgabe erreicht werden kann wird die maximale Genauigkeit bestimmt, welche das Abspielen erreichen kann. Denn das Warten auf einen bestimmten Zeitpunkt ist ein elementarer Punkt des Abspielens von Daten, welches ggf. bei jedem abzuspielenden Ereignis verwendet wird. Der Mechanismus, um Komponenten für eine gewisse Zeit schlafen zu lassen, verwendet intern den Unterbrechungsmechanismus des Betriebssystems[4]. Hierbei erhält das Betriebssystem periodisch Unterbrechungen, um wartende Komponenten pünktlich aufzuwecken. Dabei liegt die Periodenlänge typischerweise im Bereich einiger weniger Millisekunden. Somit reicht eine Erfassung der Daten mit

[4]Siehe `https://blogs.oracle.com/dholmes/entry/inside_the_hotspot_vm_clocks`, aufgerufen am 25.11.2016

einer Genauigkeit von Millisekunden aus, um die maximale Genauigkeit des Abspielens zu überschreiten.

Dem gegenüber steht, dass für eine Analyse der Daten eine genauere Auflösung der Zeit durchaus gewinnbringend ist. Denn erste Aufnahmen zu Testzwecken haben gezeigt, dass bei einer gewählten Auflösung im Bereich von Millisekunden bei den Daten, welche durch *SystemTap* erfasst werden, viele Ereignisse einen identischen Zeitstempel besitzen. Von diesem Phänomen sind nur Daten betroffen, die direkt im Kernel erfasst werden. Dies ist darin begründet, dass für den Versand einer Nachricht über das Netzwerk nur ein Aufruf im *Application Framework* notwendig ist, welcher intern einen Systemaufruf verwendet. Dieser Systemaufruf zerteilt die Nachricht ggf. in kleine Teile und ruft für jeden Teil eine Funktion im *Kernel* auf die unmittelbar mit der *Hardware* kommuniziert. Diese einzelnen Aufrufe liegen teilweise nur wenige Mikrosekunden auseinander. Diese Funktionen, die von Systemaufrufen verwendet werden, werden im Rahmen der vorliegenden Arbeit mit *SystemTap* beobachtet.

Somit besteht für die Daten, welche von *SystemTap* erfasst werden, ein Bedarf an einer höheren Auflösung, um bessere Analysen durchführen zu können. Dabei bietet *SystemTap* Methoden an, um einen *UNIX*-Zeitstempel mit einer Auflösung von wahlweise Millisekunden, Mikrosekunden oder Nanosekunden zu erhalten.

Aus diesen Gründen werden sämtliche Daten mit einer Auflösung von Mikrosekunden erfasst. Dabei verfügen die Daten, welche im *Application Framework* gesammelt werden, nur über eine Millisekunden Auflösung. Um das Zusammenführen der Datenquellen zu vereinfachen, werden aber in eine Mikrosekunden Auflösung umgewandelt.

5.1.0.3 Dateiformat

Es stehen zwei verschiedene Kategorien von Ausgabeformaten zur Verfügung. Beide können für das entwickelte Verfahren verwendet werden und bieten sowohl Vor- als auch Nachteile.

Die erste Gruppe umfasst Dateiformate, die durch den Menschen lesbar sind und auf Text basieren. Vorteilhaft bei einem solchen Dateiformat ist, dass im Rahmen der Entwicklung schneller Fehler und Inkonsistenzen in den Daten gefunden werden können, da diese nicht zuerst interpretiert werden müssen. Darüber hinaus ist es möglich, für spezielle Tests synthetische Daten ohne großen Aufwand zu erzeugen und zu verwenden. Der größte Nachteil bei solchen Dateiformaten liegt darin, dass mehr Speicherplatz als notwendig belegt wird. Dies liegt darin, dass die Daten als Text dargestellt werden und das der Text darüber hinaus oftmals Daten enthält, welche nur zur Interpretation durch den Menschen gedacht sind. Als Beispiel seien

die frei wählbaren Element- und Attributnamen des *XML*-Formats[5] genannt.

Die andere Gruppe wird von binären Dateiformaten gebildet. Die Verwendung eines solchen Dateiformats würde (ggf. erheblich) kleinere Ausgabedateien erzeugen. Dies geht zu Lasten der einfachen Lesbarkeit der Daten. Sowohl zum einfachen betrachten, als auch für die weitere Verarbeitung sind Werkzeuge notwendig, um aus dem binären Format die gewünschten Daten zu erzeugen.

Zugunsten des einfachen Umgangs mit den erzeugten Daten und einer Vereinfachung der Entwicklung, werden im Rahmen dieser Arbeit, sämtliche Daten im textbasierten *XML*-Format abgespeichert.

5.1.0.4 Instrumentierung

Im Programmcode des *Application Frameworks* ist es notwendig, eine große Anzahl von Stellen zu instrumentieren, um sämtliche geforderten Daten zu erfassen. Dabei wäre es im Hinblick auf Wartbarkeit und Komplexität sinnvoll einen möglichst generischen Ansatz zu wählen, welcher es erlaubt sämtliche Daten entgegenzunehmen. Dabei muss bei der Entwicklung beachtet werden, dass jede Datenquelle über eine unterschiedliche Anzahl von relevanten Parametern verfügt, welche alle beliebige Typen besitzen können. Da die Daten in einem Textformat abgespeichert werden, wird vereinbart, dass sämtliche zu erfassenden Daten auf der Ebene des *Application Frameworks* in eine Textrepräsentation überführt werden. Es werden also keine Zahlen und andere Objekte an die Applikation zur Datenerfassung verschickt, sondern alle Daten die verschickt werden sollen, werden noch vor dem Versand in eine Textform überführt, welche bei Bedarf wieder in das Objekt umgewandelt werden kann und darüber hinaus direkt in die Ausgabe geschrieben werden kann. Hierdurch wird sowohl ein generisches Erfassen, als auch das Abspeichern der Daten einfacher.

Jede versandte Nachricht muss mindestens drei Informationen enthalten: Die Aktion die aufgezeichnet wird, den Namen der verursachenden Applikation und den Zeitpunkt, wann das Ereignis aufgetreten ist. Darüber hinaus ist eine beliebige Anzahl weiterer Informationen möglich. Die Instrumentierung sollte möglichst leichtgewichtig sein und es ist unter allen Umständen zu vermeiden, dass durch den Nachrichtenversand die normale Ausführung blockiert wird. Hierdurch würde sowohl der Nutzerkomfort als auch das normale Verhalten des Betriebssystems verändert. Aus diesem Grund wird jeweils ein `Intent` für ein aufgetretenes Ereignis versendet. Ein `Intent` ist eine asynchrone Nachricht, welche von *Android* zur Verfügung gestellt wird. Dabei können diese Nachricht an einzelne Komponenten oder als *Broadcast* versendet werden.

[5] *Extensible Markup Language*, siehe [2]

```
1  private void sendIntent(String action, String... data) {
2      Bundle bundle = new Bundle();
3      bundle.putString("PACKAGE_NAME", mPackageName);
4      bundle.putString("CREATION_DATE",
           String.valueOf(System.currentTimeMillis()));
5      if (data != null && data.length > 0 && data.length % 2 == 0) {
6          for (int i = 0; i < data.length; i+=2)
7              bundle.putString(data[i], data[i+1]);
8          }
9      Intent intent = new Intent(action).putExtras(bundle);
10     if (ActivityManagerNative.getDefault() != null &&
           ActivityManagerNative.getDefault().isCallFinishBooting())
11         mContext.sendBroadcastAsUser(intent,
               android.os.Process.myUserHandle());
12 }
```

Abbildung 5.1: Der Code im *Application Framework*, welcher Daten an die Applikation zur Datenerfassung sendet.

Abbildung 5.1 zeigt die einfachste Version der Funktion, welche in jede instrumentierte Klasse eingefügt wird, um Daten zu versenden. Im Einzelfall kann ein Bedarf an Änderungen vorliegen. Beispielsweise kann es sein, dass der Paketname der erzeugenden Applikation nicht in der Variable `mPackageName` vorliegt, oder dass der Kontext der aktuellen Applikation zuerst angefordert werden muss. Viele Komponenten besitzen ohnehin einen eigenen Kontext, welcher verwendet wird, sofern er vorhanden ist. Sobald dieser Kontext aber fehlt muss er zunächst angefordert werden. Eine mögliche Stelle um einen Kontext zu erhalten, ist die aktuell laufende Applikation. Über diesen Kontext werden die Nachrichten versendet und der Paketname ist notwendig, damit die Daten in der Applikation zur Datenerfassung wieder einzelnen Applikationen zugeordnet werden können. Es muss vor dem Versandt sichergestellt werden, dass das Betriebssystem fertig hochgefahren ist und somit die Möglichkeit zum verschicken von Nachrichten gegeben ist. Das erfasste Ereignis wird in der Aktion der Nachricht kodiert. Um eine einfache und übersichtliche Verwaltung aller verwendeten Ereignisse zu haben werden die verwendeten Aktionen in der Klasse `Intent` als statische, konstante Werte eingetragen. An derselben Stelle befinden sich auch die Standardaktionen, welche von *Android* verwendet werden. Ein `Bundle` (*Key-Value*-Speicher) enthält sämtliche Parameter, die für ein Ereignis erfasst werden sollen und wird an die Nachricht angehängt.

5.1.0.5 Neustarts während der Aufnahme

Es ist möglich, dass das aufzeichnende Gerät während der Datenerfassung neu gestartet wird. Die Erfassung soll über diese möglichen Neustarts weiterlaufen. Aus diesem Grund ist es notwendig, vor dem Erfassen von Daten zu prüfen, ob das Betriebssystem vollständig hochgefahren ist, sobald ein Ereignis auftritt. Wird eine solche Überprüfung nicht durchgeführt kann es passieren, dass während des Datenversands auf Komponenten des Betriebssystems zugegriffen wird, welche noch nicht gestartet wurden und deswegen nicht betriebsbereit sind. Dies kann im schlimmsten Fall darin resultieren, dass das Betriebssystem nicht mehr gestartet werden kann, wenn ein Ereignis während des Startvorgangs auftritt.

Dabei stellt das *Application Framework* keine Informationen darüber bereit, ob das Betriebssystem vollständig gestartet ist, zur Verfügung. Aus diesem Grund muss, für den entwickelten Ansatz, eine Funktionalität nachgeliefert werden, welche zumindest überprüfen kann, ob die Komponenten, die im *Application Framework* zur Erfassung verwendet werden, betriebsbereit sind. Konkret bedeutet dies, dass beim Auftreten eines Ereignisses nur dann Daten erfasst werden können, wenn es möglich ist diese Daten durch einen `Intent` zu versenden. Dabei wird die Verwaltung und Zustellung von `Intent`s in *Android* durch den `ActivityManager` durchgeführt. Das bedeutet, dass Ereignisse erfasst werden können, sobald der `ActivityManager` betriebsbereit ist.

Durch eine genauere Betrachtung des Startvorgangs von *Android* zeigt sich folgendes Verhalten. Sobald das *Application Framework* gestartet wird, wird für den `ActivityManager` (und diverse andere Komponenten) eine Instanz eingefügt, welche über keinerlei Funktionalität verfügt, aber die notwendigen Verknüpfungen zwischen den Komponenten realisiert. Während des Startvorgangs werden schrittweise an alle diese funktionslosen Instanzen *Services* angehängt, welche dann auch alle Funktionalitäten bereitstellen[6]. Mit diesem Wissen ist es ausreichend, in der tatsächlichen Klasse und in dem Platzhalter eine Methode einzufügen, welche hart einen Wahrheitswert zurückgibt und damit die Einsatzbereitschaft signalisiert.

5.1.0.6 Verhindern von Suspendierungen

Während der Datenerfassung interagiert der Nutzer im Regelfall nicht mit der Applikation zur Datenerfassung. Hierdurch sieht das Betriebssystem die Applikation als inaktiv an und suspendiert möglichst viele ihrer Komponenten. Zu den suspendierten Komponenten gehören auch einfache Hintergrundprozesse wie *Services*. Aus diesem Grund ist ein normaler *Service* nicht geeignet, um alle Daten zu erfassen.

[6] Siehe `https://community.nxp.com/docs/DOC-102546`, aufgerufen am 25.11.2016 & `https://community.nxp.com/docs/DOC-102546`, aufgerufen am 26.11.2016

Dies wurde im Rahmen der Implementierung getestet und es stellte sich heraus, dass nach der ersten Suspendierung keine Daten mehr aufgezeichnet wurden, da kein automatischer Neustart von suspendierten Komponenten durchgeführt wird.

Speziell für Hintergrundaktivitäten, welche nicht suspendiert werden sollen bietet das *Android* Betriebssystem sogenannte Vordergrund-*Services* an. Diese werden wie ein normaler Vordergrundprozess betrachtet und verhindern somit ein Suspendieren der Komponenten, da die Applikation nicht als inaktiv betrachtet wird. Für den Benutzer ist ein Vordergrund-*Service* daran erkennbar, dass er in der Benachrichtigungsleiste eine Möglichkeit zur Interaktion für den Nutzer bereitstellt.

Eine verbleibende Problematik besteht darin, dass es bei langen Zeiträumen der Datenerfassung möglich ist, dass Zeiten vorliegen, in denen der Nutzer in keiner Weise mit dem aufzeichnenden Gerät interagiert. In diesen Zeiten der Inaktivität betritt das Betriebssystem tiefe Energiesparmodi. In diesen Modi werden auch Vordergrund-*Services* zugunsten des Energieverbrauchs suspendiert, sofern sie keine Aktionen ausführen. Falls die aufgezeichnete Applikation nicht selber über einen solchen *Service* verfügt, welcher aktiv eine Aufgabe ausführt, ist diese Applikation bereits suspendiert worden. Genauer ausgedrückt bedeutet dies, dass jede Applikation die zu diesem Zeitpunkt keine aktive Vordergrundkomponente besitzt bereits suspendiert worden ist und nur Applikationen mit aktiven Vordergrundkomponenten noch nicht verdrängt worden sind. Dies resultiert darin, dass auch der Vordergrund-*Service*, welcher die Daten erfasst, aufgrund inaktiv ist, da aktuell keine Daten eintreffen und aus diesem Grund suspendiert wird. Dies ist prinzipiell nicht problematisch, da in diesen Zeiträumen keine relevanten Daten anfallen können, da auch die Applikation, die aufgezeichnet wird, suspendiert ist. Es muss jedoch sichergestellt werden, dass der *Service* zur Datenerfassung wieder gestartet wird, sobald das Gerät nicht mehr inaktiv ist bzw. relevante Daten erzeugt werden. Aus diesem Grund wird bei jedem relevanten Ereignis ein Vordergrund-*Service* zur Datenerfassung erzeugt, sofern noch keiner existiert. Hierdurch wird sichergestellt, dass sämtliche Daten erfasst werden. Gleichzeitig muss berücksichtigt werden, dass durch dieses Verfahren in den periodisch erfassten Daten (Prozessorlast und Batteriezustand) Lücken entstehen.

5.1.0.7 SystemTap Namensauflösung

Netzwerkdaten und Informationen zur Ein-/Ausgabe werden von *SystemTap* direkt im *Kernel* erfasst. Daraus folgt, dass die Objekte, welche die Ereignisse auslösen, nicht mehr Applikationen im Sinne von *Android* sind. Stattdessen treten Prozesse an diese Stelle, wobei jede Applikation in der Regel über genau einen Prozess verfügt. Dabei trägt der Prozess den Paketnamen der jeweiligen Applikation als Namen. Jeder dieser Prozesse kann eine beliebige Anzahl an Kind-*Threads* besitzen,

welche über beliebige Namen verfügen können. Auch diese *Threads* können über eigene Kind-*Threads* verfügen, wodurch sich potentiell eine beliebige Verschachtlung von *Threads* ergibt.

Es besteht somit ein Bedarf an einer Abbildung von Prozessen auf Applikationen. Dies gestaltet sich schwierig, da jeder *Thread* einen beliebigen Namen haben kann. Die einzige Ausnahme wird durch den *Thread* auf oberster Ebene gebildet, welcher immer den Paketnamen als Namen verwendet. Für die vorliegende Arbeit wurde ein Kompromiss zwischen der Genauigkeit der Abbildung und der Laufzeit, welche die Bestimmung einer Applikation dauert, eingegangen. Dabei wird der jeweilige *Thread* und falls notwendig der Eltern-*Thread* überprüft. Falls keiner von beiden den Paketnamen einer aufgezeichneten Applikation als Namen verwendet, wird das Ereignis als nicht relevant betrachtet. Durch dieses Verfahren kann nicht garantiert werden, dass sämtliche anfallenden relevanten Daten auch wirklich durch die Aufzeichnung erfasst werden. Gegenüber der entstehenden Last, wenn für jedes Ereignis die Kette an *Threads* vollständig durchlaufen wird, ist dies vertretbar. Denn *SystemTap* überprüft für jedes eingehende Ereignis (jeder Applikation), ob es von Relevanz ist, was zu einer erheblichen Last führen würde. Hier kommt ebenfalls die Tatsache ins Spiel, dass *SystemTap* die Verarbeitung eines Ereignisses abbricht, falls diese zu lange dauert. Hierdurch könnte sich die Genauigkeit der Erfassung noch weiter verschlechtern.

5.1.0.8 Wakelocks

Im Rahmen von ersten Testläufen zeigte sich eine Problematik in der Erfassung von Ereignissen, welche sich auf *Wakelocks* beziehen. Ähnlich wie bei den Positionsdaten, wird durch die *Google Play Services* eine Schnittstelle zum Umgang mit *Wakelocks* angeboten. Diese Schnittstelle verwendet die Mechanismen für das Belegen und Freigeben von *Wakelocks*, welche im Rahmen der Instrumentierung modifiziert wurden. Das Problem besteht darin, dass die Aufrufe nicht mehr aus der aufgezeichneten Applikation geschehen, sondern aus den *Google Play Services*. Zusammen mit der fehlenden Möglichkeit die *Google Play Services* zu instrumentieren, ist es nicht mehr möglich die *Wakelocks* zu Applikationen zuzuordnen.

Die Verwendung dieser Schnittstelle ist nicht verpflichtend, wird aber empfohlen, da hier noch weitere Optimierungen vorgenommen werden. Beispielsweise indem gleichartige *Wakelocks* zusammengeführt werden, wodurch die Gesamtzahl an *Wakelocks* reduziert wird. Gleichzeitig folgt hieraus, dass ein unbekannter Anteil von *Wakelocks* nicht durch die aufzeichnende Applikation erfasst werden können.

5.2 Abspielen

Bei der Implementierung des Abspielens von aufbereiteten Daten sind Einschränkungen sichtbar geworden, welche die Qualität des Abspielens in einem gewissen Grad schmälern. Diese Einschränkungen und getroffene Entscheidungen werden in diesem Abschnitt aufgezeigt und diskutiert.

5.2.0.1 Verhindern von Suspendierung

Wie bereits für die Datenaufnahme erläutert, besteht das Risiko, dass eine Applikation suspendiert wird, sobald Energiesparmodi betreten werden. Dies war auf der Seite der Datenerfassung noch unproblematisch, führt aber zu erheblichen Problemen beim Abspielen. Zunächst gehen sämtliche Informationen verloren, welche während der Vorverarbeitung gesammelt wurden und auch das nächste abzuspielende Ereignis ist nicht mehr bekannt, da im Rahmen der Suspendierung ggf. Objekte welche die relevanten Informationen enthalten zerstört werden. Außerdem müsste sichergestellt werden, dass das Gerät pünktlich zum Abspielen des nächsten Ereignisses wieder aktiv ist und zu diesem Zeitpunkt die Daten der Vorverarbeitung wieder vorliegen. Es wäre möglich, die erneute Vorverarbeitung zu verhindern, indem vor der Suspendierung die Ergebnisse persistent gespeichert werden und anschließend wieder gelesen werden. Sowohl die erneute Vorverarbeitung als auch das Speichern und Laden der Daten aus der Vorverarbeitung würde zu einer zusätzlichen Last führen, welche abhängig von der Eingabegröße, sehr groß ausfallen kann. Hierdurch entstehen signifikante Abweichungen vom aufgezeichneten Verhalten, insbesondere dann, wenn der abspielende Prozess häufiger suspendiert wird. Um eine hohe Genauigkeit zu gewährleisten wäre es darüber hinaus notwendig, abhängig von der Größe der gesicherten Daten, das Gerät frühzeitig aufzuwecken, damit die Daten zum Abspielzeitpunkt bereits zur Verfügung stehen.

Um diese Problematik zu lösen, wird für den gesamten Zeitraum der Wiedergabe ein *Wakelock* gehalten, welches das Betreten von tiefen Energiesparmodi verhindert. Auch hierdurch wird eine Abweichung vom aufgezeichneten Verhalten erzeugt. Diese Abweichung ist aber weitaus geringer und weniger kritisch als die Abweichung durch das Suspendieren. Dies liegt auch daran, dass sich die Abweichung, welche ohne *Wakelock* entsteht, nicht im Vorfeld bestimmen lässt und eine deutliche Ungewissheit in das abgespielte Verhalten einführt. Denn potentiell kann das abspielende Gerät zwischen jedem Paar von Ereignissen suspendiert werden und somit eine Verzögerung verursachen. Somit wird durch die geplante Abweichung, welche das *Wakelock* verursacht, die Genauigkeit in der Summe deutlich erhöht.

5.2.0.2 Bildschirmzustand

Die aufgezeichneten Daten enthalten auch Informationen darüber, zu welchen Zeiten der Bildschirm an- bzw. ausgeschaltet wurde. Um diese Daten wieder abspielen zu können3 ist es notwendig die den Bildschirm steuern zu können. Dies steht im Widerspruch zu der Anforderung, dass die Applikation zur Wiedergabe eine normale Applikation sein soll. Denn um die Kontrolle über die Bildschirmsteuerung zu erhalten, werden Berechtigungen benötigt, welche nur für Systemapplikationen zur Verfügung stehen.

Diese Einschränkung kann umgangen werden, indem ein Mechanismus verwendet wird, welcher für den Einsatz von *Android* im Unternehmensumfeld gedacht ist. Für diesen Einsatzbereich können sich Applikation als Geräte-Administrator registrieren[7]. Diese Applikation hat dann den Zweck, die Nutzung des Gerätes so einzuschränken, dass die *Compliance*-Richtlinien des verwendenden Unternehmens eingehalten werden. Darüber hinaus kann der Zugriff aus der Ferne verhindert werden und Daten im Notfall gelöscht werden. Dabei kann jedes Gerät über keinen, oder genau einen Geräte-Administrator verfügen. Der Benutzer muss bestätigen, dass eine bestimmte Applikation als Geräte-Administrator registriert wird und im Anschluss besitzt diese Applikation weitreichende Befugnisse. Unter den Möglichkeiten, die der Applikation dann zur Verfügung stehen, befindet sich auch eine Möglichkeit den Bildschirmzustand zu steuern.

Somit wird es möglich auch die Ereignisse zum Ein- und Ausschalten des Bildschirms wiederzugeben, sofern die abspielende Applikation als Geräte-Administrator registriert ist. Durch dieses Vorgehen wird keine der Anforderungen verletzt. Dennoch muss beachtet werden, dass durch diese Lösung Geräte vom Abspielen ausgeschlossen werden, die in Unternehmen eingesetzt werden und aus diesem, oder einem beliebigen anderem Grund die Verwendung eines Geräte-Managers vorschreiben.

5.2.0.3 Lesen der abzuspielenden Daten

Um die aufgenommenen Daten abzuspielen, ist es zwingend notwendig sie einzulesen. Dabei werden während des Abspielens immer wieder kleine Abschnitte, welche die nächsten abzuspielenden Ereignisse enthalten, eingelesen und abgespielt. Hierdurch entsteht eine Abweichung vom aufgenommenen Verhalten, welche nicht vermeidbar ist. Die einzige Alternative, die es erlaubt die Daten im vorliegenden Format für eine Wiedergabe zu nutzen, besteht darin sämtliche Daten vor dem Abspielen vollständig einzulesen und im Speicher zu halten. Diese alternative ist

[7] Siehe `https://developer.android.com/guide/topics/admin/device-admin.html`, aufgerufen am 27.11.2016

nur theoretischer Natur, da die abzuspielenden Daten beliebig groß sein können und damit schnell die Größe des verfügbaren Speichers überschreiten.

Auch vollständig andere Methoden, wie beispielsweise das Empfangen der abzuspielenden Daten über das Netzwerk, würden eine zusätzliche Last verursachen, welche in einer Abweichung vom aufgezeichneten Verhalten resultiert. Im genannten Beispiel würde dies in einer erhöhten Menge von Netzwerkverkehr resultieren, da die abzuspielenden Daten empfangen werden. Die einzige Möglichkeit, mit der diese zusätzliche Last vermieden werden kann, besteht darin, die aufgezeichneten Daten fest im Programmcode des Abspielens zu verankern. In diesem Falle entsteht keine Abweichung, welche das wiedergegebene Verhalten, im Vergleich zur Aufnahme, verfälscht. Da aber keine Möglichkeit vorgesehen ist, die aufgenommenen Daten automatisch in Programmcode zu transformieren, widerspricht dieses Verfahren dem Ziel der Arbeit ein generisches und simpel nutzbares Verfahren zu entwickeln.

Aus diesem Grund muss diese Abweichung beim Abspielen hingenommen werden. Dadurch, dass die Abweichung bekannt ist und durch die Größe der Eingabe grob quantifiziert werden kann, ist es möglich sie zu berücksichtigen, wenn aus der Wiedergabe von Daten Schlussfolgerungen zur Performanz des Gerätes gemacht werden sollen.

5.2.0.4 Activity Lebenszyklus

Der vollständige Lebenszyklus einer *Activity* wurde bereits in Kapitel 2.2.5 beschrieben. Dieser wird im Rahmen der Datenerfassung vollständig aufgezeichnet und soll während der Wiedergabe wieder abgespielt werden. Hierfür ist es notwendig, dass die Transitionen zwischen den einzelnen Phasen des Lebenszyklus von der abspielenden Applikation ausgelöst werden können. Dabei wird jede Transition durch eine spezielle Methode ausgeführt.

Die Funktionen, welche die Transitionen im Lebenszyklus einer *Activity* ausführen, können von jeder Stelle aus aufgerufen werden. Aber nur wenn diese Aufrufe vom Betriebssystem und nicht von der Applikation selber kommen, ist sichergestellt, dass der Lebenszyklus ordnungsgemäß durchlaufen wird und das intendierte Verhalten auftritt. Dies liegt daran, dass die Funktionen normalerweise von Betriebssystem in einem Kontext aufgerufen werden, welcher das notwendige Umfeld, in dem die *Activity* läuft, sicherstellt. Beispielsweise wird sichergestellt, dass eine *Activity* sichtbar im Vordergrund ist, wenn sie in den Zustand *Resumed* wechselt.

Abbildung 5.2 zeigt nochmals den Lebenszyklus einer *Activity*, damit die nachfolgenden Beschreibungen besser nachvollzogen werden können. Dabei wurden die Transitionen, entsprechend der Möglichkeit, sie während der Wiedergabe auszulösen, farblich markiert. Blaue Transitionen können ohne Probleme ausgelöst werden.

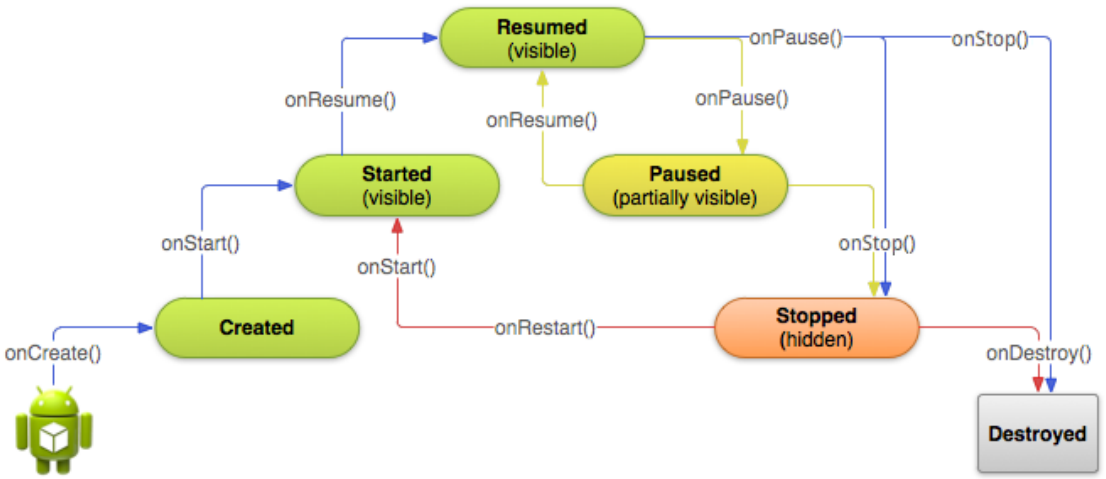

Abbildung 5.2: Darstellung des *Activity*-Lebenszyklus. Transitionen sind nach der Möglichkeit zur manuellen Auslösung markiert. Angelehnt an eine Graphik der offiziellen *Android*-Dokumentation[8]

Es ist sichtbar, dass jeder Zustand, außer *Paused* durch diese Transitionen erreicht werden kann. Die roten Transitionen gehen ausschließlich vom Zustand *Stopped* aus und können nicht gezielt ausgelöst werden. Bei der Transition zwischen *Stopped* und *Started* bestimmt das Betriebssystem, ob sie während der Wiedergabe wie gewünscht ausgelöst werden kann. Es ist sogar möglich, dass sie an Stellen ausgelöst wird, an denen eigentlich eine neue *Activity* erzeugt werden soll. Dies liegt zum einen daran, dass das Betriebssystem selber festlegt, wann eine gestoppte *Activity* zerstört werden soll. Außerdem ist es möglich, dass in den aufgezeichneten Daten eine *Activity A* geschlossen wird, weil eine *Activity B* geöffnet wird, was während der Wiedergabe durch eine einzelne *Activity* wiedergegeben wird. Gelb markierte Transitionen haben alle entweder ihren Anfang oder ihre Ende im Zustand *Paused* und können mit dem Verfahren, welches Ergebnis der vorliegenden Arbeit ist, nicht wiedergegeben werden. Durch entsprechenden Mehraufwand wäre es aber möglich, sie in die Wiedergabe einzubeziehen.

Es ist möglich, eine *Activity* in den Vordergrund zu holen, wodurch sie in den Zustand *Resumed* versetzt wird. Dabei werden unter Umständen noch andere Zustände durchlaufen, falls die *Activity* zunächst erzeugt werden muss. Ebenfalls ist es möglich, dass eine *Activity*, welche sich im Zustand *Resumed* befindet, sich selber pausiert, anschließend beendet und sich zuletzt in den Zustand *Destroyed* überführt. Eine Besonderheit muss für die Transition zwischen den Zuständen *Stopped*,

[8]Siehe `http://source.android.com/security/index.html`, aufgerufen am 13.10.2016

Started & *Destroyed* erwähnt werden. Eine gestoppte *Activity* kann vom Betriebssystem zu jedem Zeitpunkt zerstört werden, um zusätzlichen Speicher verfügbar zu machen. Deshalb verfügt die abspielende Anwendung keinerlei Kontrolle über diese Transition. Aus diesem Grund kann kein Einfluss darauf genommen werden, ob eine (gemäß den abgespielten Daten) gestoppte *Activity* während der Wiedergabe neu gestartet oder neu erzeugt wird. Hierdurch ergibt sich die erste potentielle Abweichung vom erfassten Verhalten.

Eine laufende *Activity* kann auf einfache Art gestoppt werden, indem sie in den Hintergrund geschickt wird. Dies geschieht, indem die Standardansicht des Betriebssystems in den Vordergrund geholt wird. Dabei wird die *Activity* implizit zunächst pausiert und im Anschluss direkt gestoppt.

Es verbleiben die Transitionen zwischen den Zuständen *Resumed*, *Stopped* & *Paused*. Auch hier ergeben sich Besonderheiten. Wenn eine *Activity*, welche sich im Vordergrund befindet, pausiert wird kann dies mehrere Gründe haben. Es kann zum einen daran liegen, dass eine andere *Activity* sie verdeckt, oder zum anderen daran, dass die *Activity* grade gestoppt wird. Zunächst müsste dieser Grund aus den abzuspielenden Daten extrahiert werden, indem die Nachfolgenden Ereignisse untersucht werden. Stellt sich dabei heraus, dass die *Activity* wegen einer temporären Überdeckung pausiert wird, könnte dies genau so abgespielt werden. Hierbei ergibt sich das Problem, dass die überdeckende *Activity* ihrerseits wieder von einer anderen *Activity* überdeckt werden kann. Dies lässt sich beliebig weiterführen. Da für jede *Activity* zu jedem Zeitpunkt nur eine Instanz aktiv sein kann, werden theoretisch unendlich viele *Activities* benötigt, um das exakte Verhalten abzuspielen. Da sich nur durch die Analyse einer großen Anzahl von Applikationen eine realistische obere Schranke für die Anzahl der Überdeckungen bestimmen lässt, wird der Fall der Überdeckung und temporären Überdeckung von der Wiedergabe ausgeschlossen. In dem Fall, dass die *Activity* pausiert und im Anschluss direkt gestoppt wird, wird aus zwei Gründen ignoriert. Der erste Grund ist, dass bereits im vorherigen Absatz erwähnt wurde, wie eine laufende *Activity* direkt gestoppt werden kann. Der zweite Grund ist der, dass es keine Möglichkeit, abseits der Überdeckung, gibt, welche es erlaubt eine *Activity* zu pausieren. Denn dies würde bedeuten, dass sich die *Activity* noch im Vordergrund befindet, aber nicht mehr aktiv ist. Hierdurch würde der Nutzungskomfort massiv abnehmen, da der Nutzer nicht mehr mit der *Activity* interagieren kann.

Es ist zu beachten, dass durch dieses Verfahren und die vorhandenen Probleme der gesamte Lebenszyklus einer *Activity* immer mit einer gewissen Abweichung abgespielt wird. Dies liegt auch daran, dass die aufgezeichnete Applikation in den Funktionen, welche die Transitionen durchführen, beliebige Aktionen ausführen können, wogegen die *Activity*, welche für die Wiedergabe verwendet wird, in diesen Funktionen nichts tut. Die Prozessorlast und Nutzung anderer Ressourcen, die

eventuell durch die Aktionen der Transitionen erzeugt wird, wird separat abgespielt.

In Abbildung 5.3 wird dargestellt, wie der aufgezeichnete Lebenszyklus einer Applikation abgespielt wird. Die schwarz gestrichelte Linie stellt die Aufzeichnung dar, wogegen die blaue Linie das abgespielte Verhalten repräsentiert. Es ist gut zu erkennen, dass die Zeit zwischen der Erzeugung und dem Erreichen des Zustands *Resumed* in der Wiedergabe kürzer ist als in der Aufnahme. Dies liegt daran, dass während der Wiedergabe keine Aktionen in den einzelnen Phasen ausgeführt werden. Hierdurch wird eine ausgelöste Transition direkt durchgeführt, wobei nur das Standardverhalten einer *Activity* durchgeführt wird. Es ist ebenfalls zu erkennen, dass der Zustand *Paused* nicht berücksichtigt wird. Zum einen ist dies daran erkennbar, dass eine visuelle Überdeckung vollständig ignoriert wird und zum anderen daran, dass in der Wiedergabe erst dann das Stoppen eingeleitet wird, wenn in der Aufzeichnung bereits der Zustand *Stopped* erreicht ist. Auch an dieser Stelle lässt sich erkennen, dass sich die abgespielte *Activity* nur kurz in den Zuständen befindet, sofern diese nur eine Zwischenstation darstellen. Dies resultiert darin, dass der Zustand *Stopped* in der Wiedergabe nur mit einer geringen zeitlichen Differenz zur Aufnahme erreicht wird. Ab dem Zeitpunkt, wo der Zustand *Stopped* erreicht wird, wird der Zustand der abgespielten *Activity* als unbekannt angesehen, was durch die gestrichelte blaue Linie dargestellt wird.

5.2.0.5 Positionsdaten

Bereits während der Modellierung haben die Positionsdaten eine besondere Stellung eingenommen, da es mehrere, funktional ähnliche Schnittstellen gibt, wobei nur eine für die Datenerfassung verfügbar ist. Eine genauere Beschreibung kann in Abschnitt 4.1 gefunden werden. Dabei wird die zur Verfügung stehende Schnittstelle vollständig instrumentiert. In der Menge der erfassten Funktionalitäten sind auch interne Funktionen enthalten, welche nicht direkt in Applikationen verwendet werden können. Hieraus folgt, dass Ereignisse, welche die Nutzung dieser Funktionen beschreiben, auch nicht abgespielt werden können.

Die Erfassung dieser internen Methoden wird durch dadurch motiviert, dass die *Play Services*, welche die andere Schnittstelle für Positionsdaten bereitstellen, intern auf diese Funktionen zugreift. Hierdurch wäre es möglich gewesen indirekte Schlüsse auf die Nutzung dieser Schnittstelle zu machen. Dabei war es nicht möglich diese Annahme auf Basis der erhobenen Daten zu bestätigen. Gleichzeitig reichen die Daten, welche im Rahmen dieser Arbeit erfasst wurden, nicht aus, um die Annahme vollständig zu widerlegen.

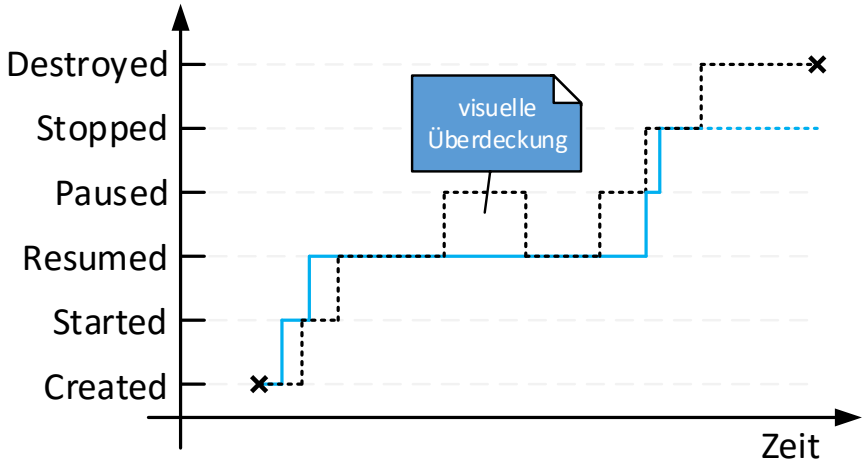

Abbildung 5.3: Vergleich zwischen aufgezeichnetem und abgespielten Lebenszyklus einer *Activity*.

5.2.0.6 Wiedergabe von Komfortfunktionen

Im Rahmen der Modellierung wurde festgelegt, dass auch die Nutzung der Komfortfunktionen, welche für den Netzwerkverkehr und die Ein-/Ausgabe zur Verfügung stehen, erfasst werden sollen. Durch diese Informationen sollten die Daten, welche von *SystemTap* erfasst werden, mit einem Kontext verbunden werden, um die Abstraktion durch die Erfassung im *Kernel* in einem gewissen Rahmen rückgängig zu machen. In der Praxis gestaltet sich dies aus mehreren Gründen als problematisch. Die *Threads*, welche von den Komfortfunktionen genutzt werden, verwenden als Namen den Paketnamen der Applikation, welche sie verwenden. Es bestand die Annahme, dass diese *Threads* die Namen der Komfortfunktionen verwenden, welche sie aktuell ausführen. Namentlich wären das der `DownloadManager` und der `MediaPlayer`. Da dies nicht der Fall ist, ergibt sich der Bedarf einer zusätzlichen Zuordnung der Daten, welche nicht, oder nur schwer, mit den aktuell erfassten Daten durchführbar ist. Das andere große Problem betrifft ausschließlich den `MediaPlayer`. Unter der Annahme, dass die zuvor angesprochene Analyse der Daten durchgeführt wurde und somit bekannt ist, welche Daten in welchen Größen vorliegen müssen, ist es notwendig diese Daten zu erzeugen. Dabei führt der `MediaPlayer` seine Funktionalität nur dann aus, wenn ihm valide Mediendateien vorliegen. Dabei ist es mit dem Verfahren, dass im Rahmen dieser Arbeit entwickelt

wurde, nicht möglich valide Mediendateien in beliebiger Größe zu erstellen.

Auf Basis dieser Problematik wird vom Abspielen der genannten Komfortfunktionen vollständig verzichtet. Dennoch werden die Daten erfasst und stehen für Analysezwecke zur Verfügung.

5.2.0.7 Parallelisierung

Während der Datenerfassung kann die aufgezeichnete Applikation ein beliebiges Maß an Parallelverarbeitung nutzen, um ihre Aufgaben zu erledigen. Dabei werden keinerlei Informationen über den Grad der Parallelität in der Ausgabe gespeichert. Dies liegt daran, dass es nicht auf simple Art und Weise möglich ist eine Zuordnung zwischen Ereignissen, den erzeugenden Komponenten und dem jeweils ausführenden *Thread* zu machen. Der hierdurch entstehende Aufwand würde den Umfang der Arbeit übersteigen, weswegen auf diese Informationen verzichtet wird.

Das Abspielen findet ohne Parallelverarbeitung in einem einzelnen *Thread* statt. Die einzige Ausnahme wird dabei von der Prozessorlast gebildet, welche in einem separaten *Thread* erzeugt wird, um nicht das übrige Abspielen zu blockieren. Da keine Informationen darüber vorliegen, wie sehr die aufgezeichneten Daten durch Parallelverarbeitung beeinflusst sind, wird für die abgespielten Daten angenommen, dass sie keine Parallelverarbeitung enthalten. Durch dieses Vorgehen lässt sich eine Abweichung zwischen aufgezeichnetem und abgespieltem Verhalten nicht vermeiden, sofern die aufgezeichneten Daten durch Parallelverarbeitung entstanden sind. Bei exzessiver paralleler Verarbeitung über längere Zeiträume können diese einzelnen Abweichungen eine große Gesamtabweichung ergeben.

5.2.0.8 Prozessorlast

Im Rahmen der Modellierung wurden zusätzliche Daten, neben der reinen Prozessorlast, definiert, welche aufgezeichnet werden sollen, um die Prozessorlast mit der jeweiligen *Hardware* in Verbindung zu setzten (vgl. Kapitel 4.1). In diesen zusätzlichen Daten sind die Anzahl der verfügbaren Prozessorkerne, deren aktuell, sowie die maximale Frequenz enthalten. Zu diesem Zweck müsste ein Verfahren modelliert werden, welches während der Wiedergabe die Anzahl an Prozessoren, sowie deren Frequenzen mit einbezieht und somit das aufzeichnende und das abspielende Gerät vergleicht. Auf Basis dieses Vergleichs könnte die Prozessorlast, sofern notwendig, angepasst werden, um sie an das abspielende Gerät anzupassen.

Dabei müsste das entwickelte Verfahren gründlich evaluiert werden, um die Eignung nachzuweisen. Alle diese Maßnahmen würden viel Aufwand erzeugen. Aus zeitlichen Gründen war es nicht möglich, die Entwicklung und Evaluation eines solchen Verfahrens in die vorliegende Arbeit einfließen zu lassen.

Stattdessen wird ein weniger komplexes Verfahren verwendet, welches ausschließlich die Prozessorlast betrachtet. Dabei wird aus den abzuspielenden Daten extrahiert, über welchen Zeitraum eine gewisse (prozentuale) Last erzeugt werden muss. Dieser Zeitraum wird anschließend in viele, sehr kleine Intervalle (100 Millisekunden) aufgeteilt. In jedem dieser Intervalle wird entsprechend der zu erzeugenden Last ein Mix aus Instruktionen ausgeführt. Beispielsweise werden bei 20% Last für 20 Millisekunden Operationen ausgeführt. Dieser Mix aus Instruktionen umfasst Variablendeklarationen und diverse Rechenoperationen mit Fließkommazahlen in einfacher und doppelter Genauigkeit, sowie 32-Bit und 64-Bit Ganzzahlen. Diese Mix wird verwendet, um zu verhindern, dass der Prozessor durch eine einzelne Instruktion nur punktuell beansprucht wird.

Testläufe über verschiedene Zeiträume zwischen fünf Sekunden und einer Minute ergaben für diverse Ziellasten Abweichungen im Bereich von etwa 2%. Diese Abweichung wird für das Verfahren als hinreichend genau akzeptiert. Es muss aber zwingend beachtet werden, dass dieses Verfahren nur dann sinnvoll eingesetzt werden kann, wenn aufzeichnendes und abspielendes Gerät über identische Prozessoren verfügen.

Allgemein und unabhängig vom Verfahren zur Wiedergabe muss beachtet werden, dass die Prozessorlast während der Wiedergabe höher ausfällt als während der Aufzeichnung. Zum einen liegt dies daran, dass die Logik, welche den Abspielvorgang durchführt, auch eine gewisse Last erzeugt, welche nicht vermieden werden kann. Zum anderen erzeugen viel der abgespielten Ereignisse eine geringe eigene Last. Diese Abweichung lässt sich nicht vollständig vermeiden und wird für das Abspielen akzeptiert.

Ein möglicher Ansatz, welcher die jeweilige Konfiguration der *Hardware* einbezieht wird im folgenden umrissen. Grundgedanke ist es, sämtliche Kerne des Prozessors als einen virtuellen Kern zu betrachten. Beispielsweise würde ein aufnehmendes Gerät mit vier Kernen, welche jeweils mit einer Frequenz von einem Gigahertz arbeiten zu einem einzelnen Kern mit vier Gigahertz betrachtet. Für das Beispiel gilt die Annahme, dass eine Last von 50% aufgezeichnet wird. Diese soll nun auf zwei verschiedenen Geräten abgespielt werden.

Das erste Gerät verfügt über 4 Kerne mit einer Frequenz von zwei Gigahertz. Dies resultiert in einem virtuellen Kern mit einer Frequenz von acht Gigahertz. Das abspielende Gerät hat somit die doppelte Frequenz im Vergleich zum aufzeichnenden. Daraus folgt, dass die abzuspielende Last halbiert werden muss. Somit wird während der Wiedergabe eine Last von 25% erzeugt. Dies kann mit dem zuvor beschriebenen Verfahren geschehen.

Das zweite Gerät verfügt über einen einzelnen Kern mit einer Frequenz von einem Gigahertz. Das bedeuted, dass das aufzeichnende Gerät eine viermal so große virtuelle Frequenz besitzt. Aus diesem Grund müsste auf dem Gerät während

der Wiedergabe die vierfache Last erzeugt werden. Dies wären 200%, welche nicht erzeugt werden können. In diesem Fall würde eine Last von 100% erzeugt und der Wiedergabezeitraum entsprechend verlängert, um dieselbe Last wie während der Aufgabe zu erzeugen. In diesem Beispiel würde also für die doppelte Dauer des Aufnahmeintervalls eine Last von 100% erzeugt.

Vor der Verwendung dieses Verfahrens wäre es notwendig, zu evaluieren, ob diese Vereinfachung durch die Bildung eines virtuellen Kerns zu sehr von der Realität abstrahiert, oder ob die produzierten Werte hinreichend genau sind. Ebenfalls ignoriert dieses Verfahren Unterschiede in den Kernen, welche nicht die Frequenz betreffen. Es ist durchaus möglich, dass diese Unterschiede schwer ins Gewicht fallen und eine solche Abstraktion, wie zuvor beschrieben, unbrauchbar machen.

5.2.0.9 Dateioperationen

Während der Datenaufzeichnung werden sämtliche Dateioperationen erfasst, die von der aufgezeichneten Applikation erzeugt werden. In den erfassten Daten sind somit auch Zugriffe auf Bibliotheken und andere Dateien enthalten, welche zusammen mit der Applikation ausgeliefert werden. Da es sich hierbei nicht um eine charakteristische Nutzung der verfügbaren Ressourcen handelt, werden diese Daten während der Wiedergabe nicht betrachtet. Genauer werden die Daten während der Vorverarbeitung nicht behandelt. Hierdurch stehen während der Wiedergabe keine Informationen zur Verfügung, auf welche Dateien der Wiedergabe sich die Ereignisse beziehen, weswegen sie dann ignoriert werden.

Außerdem sei darauf hingewiesen, dass während der Vorverarbeitung für jede Datei, die in den aufgezeichneten Daten auftaucht, eine Datei erzeugt wird, welche über eine identische Größe verfügt und während der Wiedergabe genutzt wird. Dabei wird unterschieden, ob sich die jeweilige Datei während der Aufnahme im internen oder im externen Speicher (vgl. Kapitel 4.1) befindet und an der entsprechenden Position erzeugt.

5.3 Werkzeugkette

Im Rahmen der vorliegenden Arbeit wurden diverse Werkzeuge entwickelt. Diese lassen sich in zwei Gruppen einteilen. Die erste Gruppe enthält alle Werkzeuge, welche weder Teil der Applikation zur Datenerfassung, noch der Applikation zur Wiedergabe sind aber dennoch notwendig sind, um ein ordnungsgemäßes Abspielen der Daten zu gewährleisten. Diese Gruppe wird im ersten Unterabschnitt vorgestellt. Im Anschluss wird die zweite Gruppe von Werkzeugen vorgestellt. Diese enthalten diverse Werkzeuge, welche die Analyse der Daten vereinfachen oder erst

ermöglichen. Unter diesen Werkzeugen finden sich sowohl sehr allgemeine Werkzeuge, die einfach alle Daten zusammentragen, als auch spezielle Werkzeuge, welche einzelne Aspekte aus den Daten extrahieren.

5.3.1 Vorverarbeitung und Wiedergabe

Die Daten liegen nach der Aufnahme in mehreren Dateien vor und sind nicht sortiert. Dies verhindert ein ordnungsgemäßes Abspielen der Daten und erfordert eine Aufbereitung. Während der anschließenden Wiedergabe ist es möglich, dass Netzwerkdaten abgespielt werden sollen. Dabei können nicht dieselben Adressen verwendet werden, welche in der Aufnahme auftauchen. Dies liegt daran, dass diese Adressen unter Umständen nicht mehr erreichbar sind, oder das durch die Nutzung gegen Nutzungsbedingungen verstoßen wird. Aus diesem Grund ist ein spezieller Dienst notwendig, welcher den Kommunikationspartner für sämtliche Netzwerkkommunikation darstellt.

Im Folgenden wird zunächst das Werkzeug zur Vorverarbeitung der Daten vorgestellt und im Anschluss der spezielle Netzwerkservice.

Datenzusammenführung

Teil der Datenaufbereitung ist es sämtliche Dateien, welche im Rahmen der Aufnahme erzeugt wurden, zu einer einzelnen Datei zusammenzuführen. Dies ist nicht nur aus Gründen der einfachen Handhabung der Daten sinnvoll. Ebenfalls wird die Komplexität des Abspielvorgangs und der Analysen reduziert, da durch mögliche Neustarts des aufnehmenden Gerätes während der Aufnahme beliebig viele Dateien zu einer einzelnen Datenerfassung gehören können.

Wesentlich wichtiger als die Zusammenführung der einzelnen Dateien ist es aber, dass im Rahmen der Vorverarbeitung sämtliche Ereignisse chronologisch sortiert werden und somit in korrekter Reihenfolge für die sequentielle Arbeitsweise der Wiedergabe vorliegen.

Dabei wird die Aufbereitung der Daten durch eine *XSLT*-Transformation durchgeführt [5]. Diese Transformation führt die Daten aus den verschiedenen Quellen zusammen und sortiert sie chronologisch. Es wird an dieser Stelle nochmal darauf hingewiesen, dass während dieser Vorverarbeitung keinerlei Filterung der Daten vorgenommen wird. Diese wäre notwendig, falls mehrere Applikationen zur selben Zeit aufgezeichnet werden und somit die Daten, welche von *SystemTap* erfasst werden, vermischt werden.

Netzwerkservice

Das entwickelte Verfahren ermöglicht die Erfassung von Netzwerkdaten, welche durch die Nutzung der Protokolle *TCP* und *UDP* entstehen. Damit es möglich ist, diese Daten auch wieder abzuspielen zu können, wird ein Netzwerkservice benötigt, welche den Kommunikationspartner für das abspielende Gerät darstellt.

Bei diesem Dienst handelt es sich um eine minimale *Java*-Applikation, welche Verbindungen entgegennimmt und Nachrichten zu gewünschten Zeitpunkten versenden kann. Dabei wird jede Verbindung in einen eigenen *Thread* ausgelagert und individuell gesteuert. Jede empfangene Nachricht kann Informationen enthalten, zu welchem Zeitpunkt eine Antwort in welcher Größe für die Verbindung erwartet wird. Diese Informationen werden genutzt, um das aufgezeichnete Netzwerkverhalten möglichst genau wiederzugeben.

Die einzelnen Verbindungen werden solange aufrecht erhalten, bis sie explizit vom abspielenden Gerät geschlossen werden. Deshalb ist es möglich, dass am Ende eines Abspielvorgangs auf der Serverseite noch offene Verbindungen bestehen. Dies ist für den Abspielvorgang nicht von Relevanz und im Kontext der vorliegenden Arbeit ist es akzeptabel, für jeden Abspielvorgang den Dienst neu zu starten.

Die Adresse unter der der Dienst erreichbar ist, ist aktuell fest im Programmcode der Applikation zur Wiedergabe verankert. Wenn das Verfahren in einem größerem Rahmen ggf. mit zentralem Netzwerkservice eingesetzt werden soll, ist hier eine bessere Lösung notwendig. Auch weil der Dienst in der aktuellen Fassung in keiner Weise geschützt ist und aus dem Grund anfällig für Angriffe ist.

5.3.2 Nachbearbeitung zum Erkenntnisgewinn

Bereits für eine Datenaufzeichnung über sehr kurze Zeiträume unter 30 Minuten ist es möglich, dass die erfassten Daten mehrere hunderttausend Ereignisse umfassen. Diese Daten sind nicht einfach handhabbar. Aus diesem Grund ist es notwendig, einzelne Datenaspekte mit speziellen Werkzeugen zu extrahieren und aufzubereiten. Dabei befassen sich manche Werkzeuge mit ganzen Kategorien von Ereignissen, wogegen andere nur einzelne, wenige Ereignisse betrachten. Nachfolgend werden alle verwendeten Werkzeuge kurz in ihrer Funktion vorgestellt und der Grund für ihre Verwendung aufgezeigt. Sämtliche Werkzeuge, die in diesem Abschnitt vorgestellt werden, wurden, sofern nicht anders angegeben, unter der Verwendung von *Python* umgesetzt.

Gruppierung und Kenngrößen Bestimmung

Ein zentrales Werkzeug dient dem Zweck die großen Datenmengen zu gruppieren und gewisse Größen zu extrahieren. Das Werkzeug wird in zwei verschiedenen Fassungen benutzt. Die eine Fassung betrachtet den Netzwerkverkehr und die andere Dateioperationen. Nachfolgend wird nur das Werkzeug für die Netzwerkdaten betrachtet, da das Werkzeug für Ein-/Ausgabedaten funktional äquivalent ist und nur andere Gruppierungen vornimmt und andere Werte extrahiert. So werden Verbindungen zu Dateien und die gesendete & empfangene Daten werden zu gelesenen & geschriebenen Dateien. Die Daten werden sequentiell durchlaufen und aus relevanten Ereignissen werden Informationen gesammelt und zusammengeführt. Dabei wird die Ausgabe im *JSON*-Dateiformat abgelegt [7].

Abbildung 5.4 zeigt einen Ausschnitt für das Ergebnis der Gruppierung für Netzwerkdaten. Es ist zu erkennen, dass Verbindungen anhand der Kombination von *IP*-Adresse und verwendetem *Port* identifiziert werden. Dabei werden für jede Verbindung Daten wie die Summe aller gesendeten und empfangenen Daten extrahiert und gespeichert. Bei Bedarf ist es auch möglich tiefer in eine einzelne Verbindung zu gucken und so genauere Daten zu erhalten, beispielsweise einzelne Sende- und Empfangsoperationen.

Die so zusammengefassten Daten können später im Rahmen von Datenanalysen verwendet werden, um Schlüsse aus den Daten zu ziehen oder Informationen zu beziehen. Beispielsweise lässt es sich mit den Daten schnell die Frage beantworten wie viele Verbindungen zu einer bestimmten Adresse aufgebaut wurden und welche Datenmengen über diese Verbindung gesendet bzw. empfangen wurden.

Extraktion einzelner Eigenschaften

Selbst die gruppierten Daten, welche das Ergebnis des zuvor betrachteten Werkzeugs sind, haben einen enormen Umfang. Durch die Redundanzen, die durch das Einfügen von Kennzahlen eingeführt werden, können sie sogar die Größe des ursprünglichen Datensatzes übersteigen. Diese Kennzahlen können Informationen sein, wie die Anzahl Ereignisse einer bestimmten Gruppe (beispielsweise Dateioperationen), die Summe aller empfangenden Netzwerkoperationen, oder wie viele Daten insgesamt persistent gespeichert wurden. Somit sind die Daten noch immer nicht für jeden Zweck einfach zu handhaben. Aus diesem Grund werden diverse kleine Werkzeuge verwendet, welche einzelne bzw. einige wenige Ereignisse betrachten und Daten aus ihnen extrahieren. Beispiele hierfür wäre eine Extraktion des Lebenszyklus von *Activities* und der Zustand des Bildschirms.

```
{
    "104.104.199.80:80": {
        "conData": [
            {
                "duration": "130",
                "end": "1475800368485",
                "protocol": "TCP",
                "recData": 53446,
                "recDetails": [
                    {
                        "payloadSize": "1448",
                        "timestamp": "1475800368355"
                    },
                    ...
                ],
                "sendData": 933,
                "sendDetails": [
                    {
                        "payloadSize": "405",
                        "timestamp": "1475800368372"
                    },
                    ...
                ],
                "start": "1475800368355"
            },
            ...
        ],
        "numConnections": 5,
        "recData": 3748561,
        "sendData": 93786
    },
    ...
}
```

Abbildung 5.4: Beispiel für gruppierte Netzwerkdaten

6 Evaluation

An dieser Stelle wurde das entwickelte Verfahren vollständig vorgestellt und umgesetzt. Die resultierende Implementierung soll im Folgenden genutzt werden, um die anfallenden Daten auf verschiedene Arten zu evaluieren. Dabei soll zunächst das Verfahren verwendet werden, um sich selber zu evaluieren und so Aussagen über die Güte des Verfahrens machen zu können.

Im Anschluss werden die aufgezeichneten Daten verwendet, um Informationen über die erfassten Applikationen zu sammeln. Auf Basis dieser Daten sollen, sofern möglich, Rückschlüsse auf das Verhalten der Applikation gemacht werden.

6.1 Bewertung durch Introspektion

Zunächst ist es notwendig, dass das Verfahren, welches im Rahmen der vorliegenden Arbeit entwickelt wurde, im Hinblick auf seine Güte untersucht wird. Zu diesem Zwecke wurden zwei Applikationen über einen Zeitraum von circa 15 Minuten aufgezeichnet und währenddessen nach einem, zuvor definierten, Schema verwendet. Die dabei anfallenden Daten wurden für eine Wiedergabe aufbereitet und anschließend wiedergegeben. Dabei wird der Vorgang der Wiedergabe wieder aufgezeichnet. Somit liegen für beide Applikationen zwei Datensätze vor, welche identische Daten enthalten sollten.

Durch dieses Verfahren ist es möglich, die Genauigkeit des Abspielens zu bewerten. Die Qualität und Vollständigkeit der eigentlichen Datenerfassung kann hierdurch nicht beurteilt werden. Zu diesem Zwecke wurden diverse Applikationen während der Implementierung verwendet, um einzelne Ereignisse auszulösen und deren Erfassung sicherzustellen. Teilweise wurden diese Applikationen extra für diese Test entwickelt. Dennoch ist es ohne großen Aufwand nicht möglich die Vollständigkeit der Datenerfassung nachzuweisen. Beispielsweise wäre es notwendig, alle Szenarien, die zu einem bestimmten Ereignis führen können, zu identifizieren und mindestens einmal zu erfassen, um die korrekte Funktionsweise nachzuweisen. Besonders bei den Daten, welche von *SystemTap* erfasst werden, existieren sehr viele Szenarien, die zu Aufrufen der beobachteten Funktionen bzw. Systemaufrufe führen. Da dies den Umfang der vorliegenden Arbeit übersteigen würde, beschränkt sich die Evaluation auf die Genauigkeit der Wiedergabe und verlässt sich auf die

5 Minuten	Nutzung einer Webseite, welche automatisch neuen Inhalt lädt, sobald der bereits geladene durchgesehen wurde (vgl. *DeviantArt*[1])
1 Minute	Inaktivität bei geschlossener Applikation mit eingeschaltetem Bildschirm
3 Minuten	Nutzung zuvor genannter Seite
1 Minute	Inaktivität bei geschlossener Applikation mit ausgeschaltetem Bildschirm
5 Minuten	Nutzung einer Webseite, welche auf Knopfdruck neu geladen wird und anderen, zufälligen Inhalt anzeigt (vgl. *Chefkoch*[2])

Tabelle 6.1: Schema, wie die Nutzung im ersten Testszenario aufgeteilt wird.

ausgiebigen Tests während der Implementierung. Dennoch können Vergleiche zwischen den erfassten Daten und der Nutzung der Applikation vorgenommen zu werden um zumindest ein Indiz für die korrekte Erfassung zu erhalten.

Nachfolgend werden zunächst die Szenarien, welche zur Evaluation verwendet werden, vorgestellt. Im Anschluss werden einzelne Ereignisgruppen betrachtet, um die Genauigkeit der Wiedergabe zu beurteilen.

6.1.1 Testszenarien

Es existieren zwei Testszenarien. Bei der Aufzeichnung beider Applikationen wurde ein grobes Nutzungsschema vorgegeben, um einzelne Nutzungsphasen besser identifizieren zu können. Diese Szenarien sollen im Folgenden vorgestellt werden und einige ihrer Charakteristiken aufgezeigt werden. Ebenfalls werden die Ergebnisse der Wiedergabe grob umrissen.

Browser

Im ersten Szenario wird der *Webbrowser* verwendet, welcher mit *CyanogenMod* mitgeliefert wird. Die Datenerfassung umfasst einen Zeitraum von grob 15 Minuten. Diese sind, wie in Tabelle 6.1 angegeben, aufgeteilt.

Die Aufnahme umfasst einen Zeitraum von 15 Minuten und 38 Sekunden und enthält knapp 81000 Ereignisse. Die anschließende Aufnahme der Wiedergabe dauert 16 Minuten und 24 Sekunden und enthält in etwa 41000 Ereignisse. Tabelle 6.2 zeigt diverse Kennzahlen, die für die ursprüngliche Aufnahme und deren Wiedergabe erfasst wurden und zeigt Abweichungen zwischen den beiden. Es ist erkennbar,

	Aufnahme *Browser*	Wiedergabe	Abweichung
Dauer in Millisekunden	938597	984665	+4,9%
Anzahl Ereignisse	80955	41304	−49,0%
gesendete Daten in Byte	1362229	915235	−32,8%
empfangene Daten in Byte	24515996	13531316	−44,8%
gelesene Daten in Byte	10444996	27215955	+160,6%
geschriebene Daten in Byte	133973647	103673911	−22,6%

Tabelle 6.2: Kennzahlen für das *Browser* Testszenario. Inklusive Abweichungen der Wiedergabe.

6,5 Minuten	Betrachtung verschiedener Städte und Verfolgung einzelner Straßen
2 Minuten	Inaktivität bei geschlossener Applikation mit ausgeschaltetem Bildschirm
6,5 Minuten	Nutzung nach zuvor genanntem Schema

Tabelle 6.3: Schema, wie die Nutzung im zweiten Testszenario aufgeteilt wird.

dass die Wiedergabe in etwa 5% länger dauert, gleichzeitig aber nur knapp die Hälfte an Ereignissen erzeugt.

Generell zeigen sich große Abweichungen in den einzelnen Kategorien, wobei besonders auffällt, dass während der Wiedergabe mehr als das 2,5-fache an Daten, im Vergleich zur Aufnahme, gelesen wird. Diese Abweichungen und deren Ursachen werden im weiteren Verlauf des Kapitels einzeln betrachtet und untersucht.

Google Maps

Für das zweite Szenario wurde die Applikation *Google Maps* ausgewählt. Dabei wurde nicht die Funktion zur Navigation verwendet, sondern nur verschiedene Bereiche betrachtet und über die Karte gewandert. Die Unterteilung der Aufnahme, in einzelne Nutzungsphasen, kann Tabelle 6.3 entnommen werden.

Die gesamte Aufnahme umfasst einen Zeitraum von 15 Minuten und 24 Sekunden, genauso wie die Wiedergabe. Tabelle 6.4 zeigt für dieses Szenario Kennzahlen und Vergleiche zwischen Aufnahme und Wiedergabe. Auch in diesem Szenario umfasst die Wiedergabe nur knapp die Hälfte der Ereignisse der Aufnahme. Ebenfalls

	Aufnahme *Maps*	Wiedergabe	Abweichung
Dauer in Millisekunden	924494	924530	+0,0%
Anzahl Ereignisse	75340	35252	−53,2%
gesendete Daten in Byte	1150377	1200011	+4,3%
empfangene Daten in Byte	19863918	24326192	+22,5%
gelesene Daten in Byte	16842527	22584083	+34,1%
geschriebene Daten in Byte	41881959	2126585	−94,9%

Tabelle 6.4: Kennzahlen für das *Maps* Testszenario. Inklusive Abweichungen der Wiedergabe.

zeigen sich auch in diesem Szenario, teils extreme, Abweichungen zwischen Aufnahme und Wiedergabe, welche im Folgenden betrachtet werden.

6.1.2 Lebenszyklus und Bildschirmzustand

Ein wichtiges Kriterium, welches den Grad der Genauigkeit beeinflusst, ist, wie genau der Lebenszyklus der aufgezeichneten *Activities* wiedergegeben wird und wie genau die Phasen, in denen der Bildschirm ein- bzw. ausgeschaltet ist, nachgestellt werden können. Die Abbildung 6.1 zeigt diese Informationen für die Aufnahme und die Wiedergabe des *Webbrowsers*, wogegen Abbildung 6.2 die Daten für das Szenario mit *Google Maps* darstellt.

In beiden Graphiken wird der *Activity* Lebenszyklus der Aufnahme durch eine schwarze und der *Activity* Lebenzyklus der Wiedergabe durch eine blaue Linie repräsentiert. Die nach rechts laufende, grüne Schattierung des Hintergrundes zeigt die Zeiten an, in denen der Bildschirm während der Wiedergabe eingeschaltet war. Die nach links laufende, rote Schattierung markiert die entsprechenden Zeiträume der Wiedergabe.

Es ist zu erkennen, dass in beiden Szenarien eine sehr genaue Wiedergabe erfolgt ist, da die Darstellungen für Aufnahme und Wiedergabe nahezu identisch sind. Aus diesem Grund ist die schwarze Linie nicht bzw. nur schwer erkennbar, da das Verhalten der Wiedergabe nahezu identisch zur Aufzeichnung ist. Dabei ist die Genauigkeit bei dem *Google Maps* Szenario noch wesentlich besser, was sich auch in der quasi identischen Laufzeit von Wiedergabe und Aufnahme bemerkbar macht. Beim Szenario mit dem *Webbrowser* ist erkennbar, dass die *Activity* in der zweiten Nutzungsphase zu lange aktiv ist und der Bildschirm ebenfalls zu lange

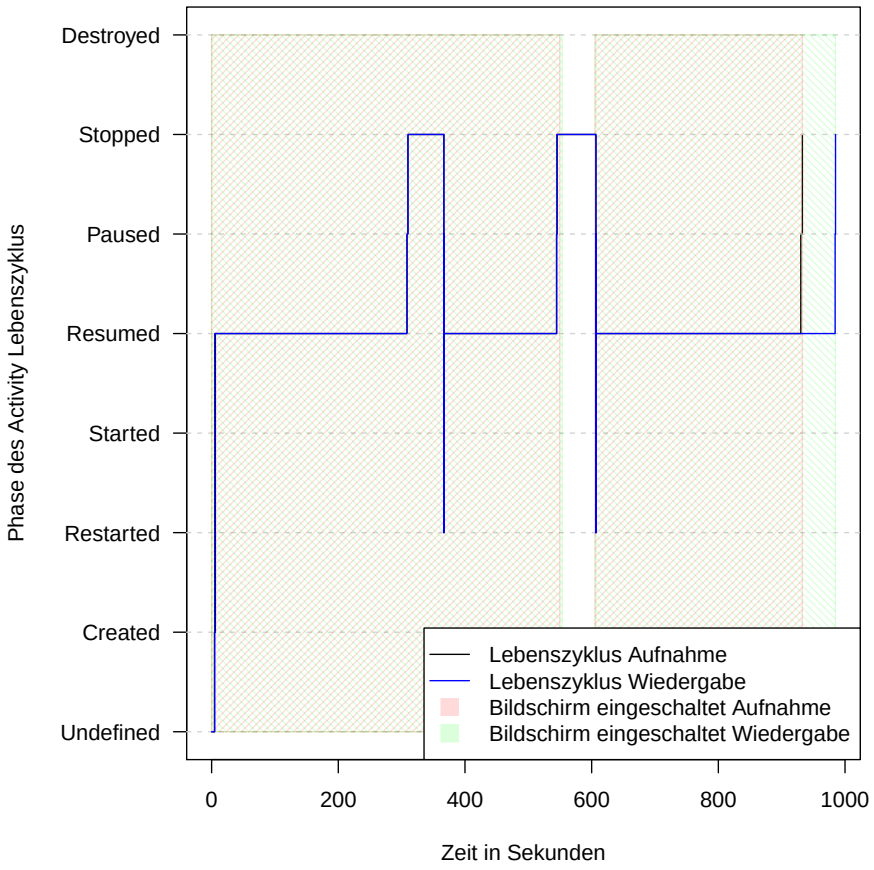

Abbildung 6.1: Vergleich des Bildschirmzustands und Lebenszyklus von *Activities* zwischen der Aufnahme des *Webbrowsers* und anschließender Wiedergabe der Daten.

eingeschaltet ist. Dies spiegelt auch die Differenz in den Laufzeiten von Wiedergabe und Aufnahme wieder. Denn die Wiedergabe dauert circa 45 Sekunden länger, was der Differenz im Lebenszyklus entspricht.

Ebenfalls kann den Abbildungen entnommen werden, dass in beiden Szenarien das beschriebene Nutzungsschema sehr gut und genau sichtbar ist. Die Laufzeitabweichung bei den Daten zum *Webbrowser* stellt die einzige Abweichung vom beschriebenen Verhalten dar.

An dieser Stelle ist es noch nicht möglich für die Differenz in den Laufzeiten des *Webbrowser* Szenarios Ursachen zu finden.

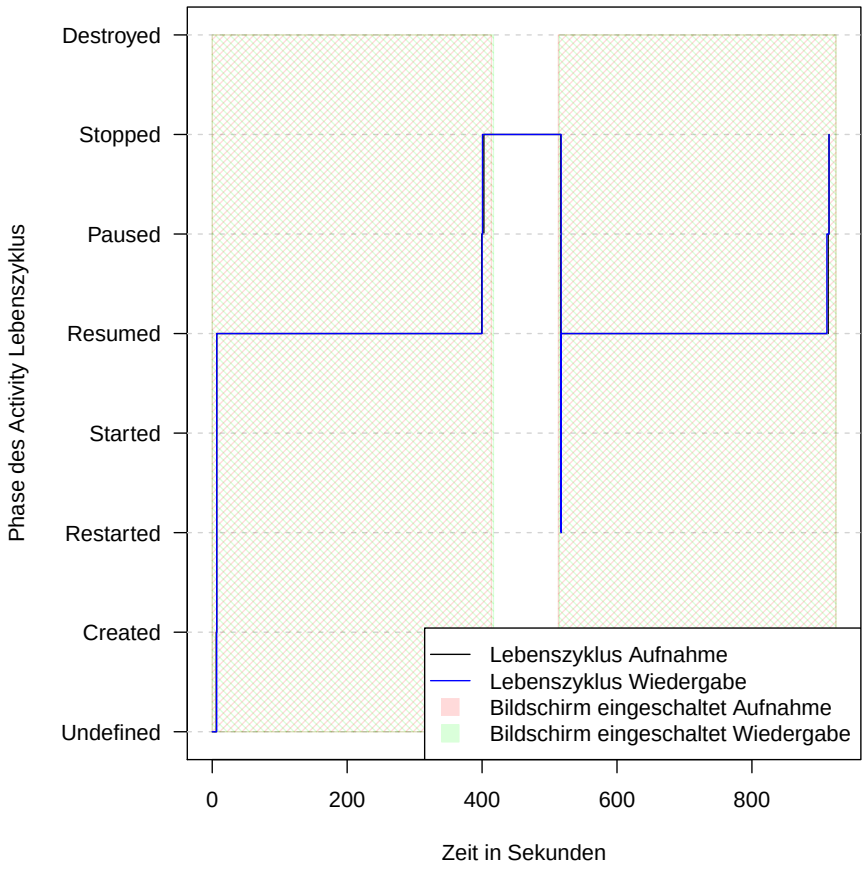

Abbildung 6.2: Vergleich des Bildschirmzustands und Lebenszyklus von *Activities* zwischen der Aufnahme von *Google Maps* und anschließender Wiedergabe der Daten.

6.1.3 Prozessorlast

Die Prozessorlast wurde im Rahmen der Modellierung als ein besonders wichtiges Merkmal für das Verhalten einer Applikation identifiziert (vgl. Kapitel 4.1). Aus diesem Grund sollte die Prozessorlast möglichst genau wiedergegeben werden können, um das Verhalten der aufgezeichneten Applikation nachzustellen. Da aus zeitlichen Gründen ein naiver Ansatz zur Erzeugung der Prozessorlast gewählt wurde, kann an dieser Stelle ein Vergleich zwischen Aufnahme und Wiedergabe vorgenommen werden, um die Eignung des gewählten Verfahrens zu bestimmen.

Abbildung 6.3 und Abbildung 6.4 zeigen diesen Vergleich für den *Browser* bzw. *Google Maps*. Dabei stehen die schwarzen Linien für die Daten der Aufnahme und

die blauen für die Wiedergabe.

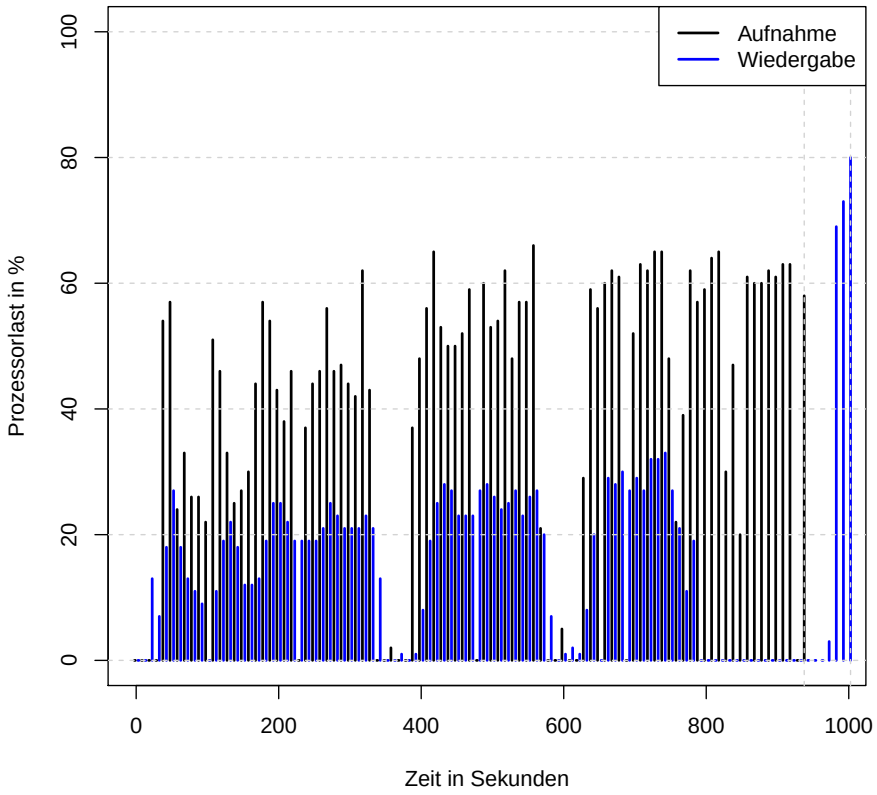

Abbildung 6.3: Vergleich der Prozessorlast zwischen der Aufnahme des *Webbrowsers* und anschließender Wiedergabe der Daten.

Bei beiden Szenarien kann erkannt werden, dass die, während der Wiedergabe, erzeugte Last circa 50% zu gering ist, aber der allgemeine Verlauf der ursprünglichen Daten erhalten bleibt. Dabei muss berücksichtigt werden, dass durch den Vorgang, welcher das Abspielen koordiniert, selber eine gewisse Last erzeugt wird, welche nicht vermieden werden kann und zu einem gewissen Rauschen in den Daten führt. Aus diesem Grund ist die erzeugte Last nicht mit einer höheren Genauigkeit um 50% zu gering. Wird dies berücksichtigt, festigt sich die Annahme, dass doppelt so viel Last erzeugt werden müsste, um das aufgezeichnete Verhalten nachzubilden. Es sei an dieser Stelle darauf hingewiesen, dass die ursprüngliche Datenerfassung und die anschließende Wiedergabe auf demselben Gerät vorgenommen wurden und

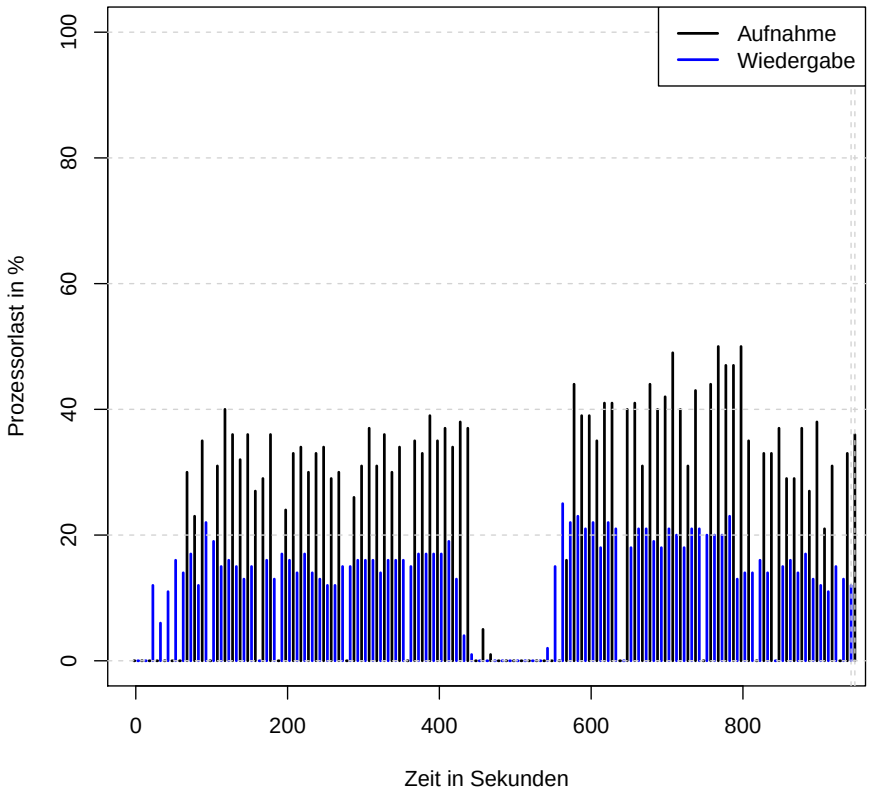

Abbildung 6.4: Vergleich der Prozessorlast zwischen der Aufnahme von *Google Maps* und anschließender Wiedergabe der Daten.

somit keine Unterschiede in der Hardware vorliegen. Bei dem verwendeten Gerät handelt es sich um das *Samsung Nexus S*, welches über zwei Prozessorkerne verfügt.

Somit wird bei dem verwendeten trivialen Verfahren zur Lasterzeugung in der aktuellen Form eine Untauglichkeit festgestellt, welche beseitigt werden muss, um die allgemeine Qualität des Abspielens zu erhöhen. Eine Möglichkeit das verwendete Verfahren zu verbessern besteht darin, die erzeugte Last auf die verfügbaren Prozessorkerne zu verteilen. Die Last wird aktuell auf einem einzelnen Kern erzeugt. Da das abspielende Gerät zwei Kerne zur Verfügung hat ergibt sich somit eine Halbierung der abgespielten Last gegenüber der Aufnahme. Beispielsweise wird eine erfasste Last von 50% aktuell auf einem Kern wiedergegeben, was zu einer Gesamtauslastung von 25% führt. Stattdessen müsste auf jedem Kern eine Last von 50% erzeugt werden, um die erfassten Daten korrekt wiederzugeben.

Aus Zeitgründen ist dieser Umstand im Rahmen der Entwicklung nicht aufgefallen und korrigiert worden. Im Rahmen der Entwicklung wurde das Verfahren, in der verwendeten Form, auf einem klassischen *Linux*-System in *Java* umgesetzt und getestet. Dabei wurde die erzeuge Prozessorlast mit dem Werkzeug *top*, welches auch in der Datenaufzeichnung verwendet wird, überprüft. Hierbei wurden hinreichend genaue Werte erzielt, welche Abweichungen im Bereich von ±5% enthalten. Hierdurch blieb der Fehler auf dem Smartphone für lange Zeit unentdeckt. Eine Mögliche Ursache liegt darin, dass die beiden Versionen von *top* unterschiedliche Standardparameter verwenden, was zu unterschiedlichem Verhalten führt.

Es ist sichtbar, dass in beiden Szenarien das definierte Nutzungsverhalten gut wiedergefunden werden kann. Einzig in der Wiedergabe der Aufzeichnung des *Browsers* ist eine Lücke von circa 3,5 Minuten zu erkennen, gefolgt von einer starken Lastspitze. Diese übersteigt auch die Daten der ursprünglichen Aufnahme, welche für den Zeitraum der Lastspitze gar keine Last verzeichnet. Hierbei handelt es sich um ein erstes Indiz für eine Phase der vollständigen Inaktivität im ersten Testszenario, welche zu den Laufzeitunterschieden führt.

6.1.4 Dateioperationen

Bei beiden Applikationen, welche im Rahmen der Gütebewertung verwendet werden, sind Dateioperationen ein wichtiges Merkmal, welches die Applikation auszeichnet. Der *Browser* verwaltet einen Zwischenspeicher, in welchem Objekte abgelegt werden, um die Netzwerkkommunikation zu verringern. Auf dieselbe Art und Weise verwendet *Google Maps* den persistenten Speicher, damit einzelne Kartensegmente nicht immer wieder neu empfangen werden müssen. Auch im Allgemeinen ist die persistente Speicherung von Daten und der Zugriff auf diese für viele Applikationen eine wichtige, wenn nicht sogar zentrale Funktion.

gelesene Daten

Wie bereits in Kapitel 5.2 erwähnt, werden Zugriffe auf die Daten, welche gemeinsam mit einer Applikation ausliefert werden, von der Wiedergabe ausgeschlossen, da es sich nicht um eine charakteristische Nutzung der verfügbaren Ressourcen handelt.

In den Abbildungen 6.5 und 6.6 ist die insgesamt gelesene Datenmenge im Verlauf der Zeit für die Aufnahme und die Wiedergabe des *Browsers* bzw. von *Google Maps* zu sehen. Dabei werden die Daten der Aufnahme durch eine schwarze Linie und die der Wiedergabe durch eine blaue Linie dargestellt. Dabei werden für beide Datensätze die Gesamtmenge gelesener Daten, sowie der Zeitpunkt zu dem diese Datenmenge gelesen wurde, durch gestrichelte Linien markiert.

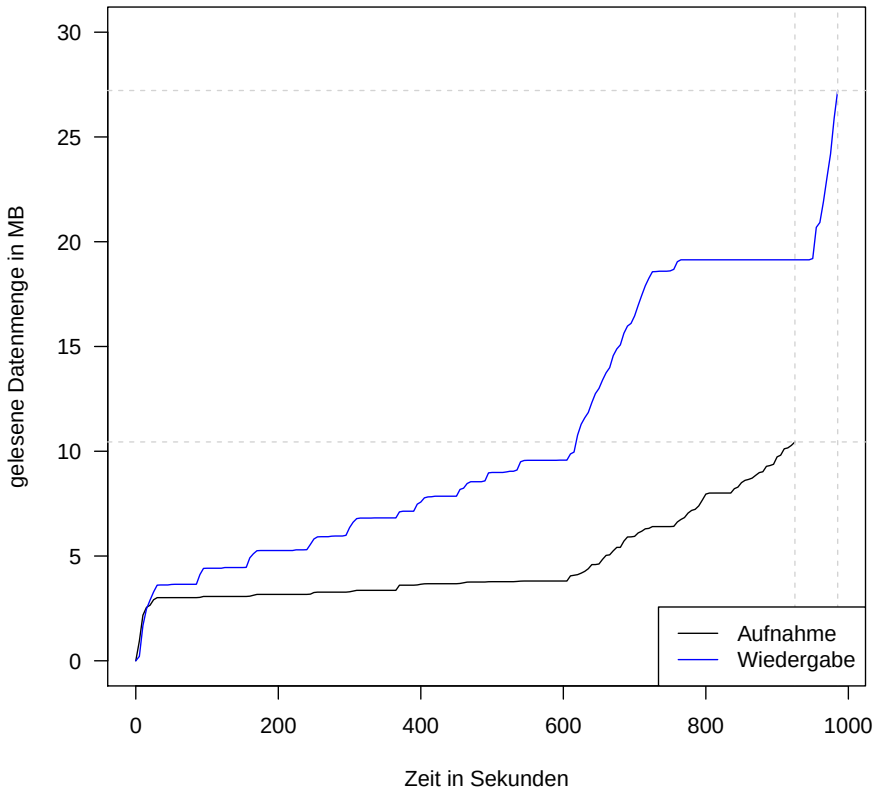

Abbildung 6.5: Vergleich gelesener Daten zwischen der Aufnahme des *Webbrowsers* und anschließender Wiedergabe der Daten.

Am auffälligsten ist, dass in beiden Szenarien, während der Wiedergabe, wesentlich mehr Daten gelesen werden, als während der Aufnahme. Die Abweichung beim *Browser* Szenario liegt dabei im Bereich, welcher der Größe des abgespielten Datensatzes entspricht. Wenn man dies berücksichtigt wird hierbei eine sehr hohe Genauigkeit erreicht, da sich das Lesen des abzuspielenden Datensatzes nicht vermeiden lässt. Bei dem zweiten Szenario entspricht die Differenz in etwa einem Drittel der Größe des Datensatzes. Dies lässt darauf schließen, dass zu wenig Daten gelesen werden. Eine Ursache hierfür kann auf Basis der bisher betrachteten Daten nicht festgestellt werden. Aber die Ausgabe der Wiedergabe legt die Vermutung nahe, dass das Problem in der Vorverarbeitung der Daten liegt. Dies wird im nächsten Abschnitt nochmals aufgegriffen.

Wie in den vorherigen Abbildungen lassen sich Laufzeitunterschiede zwischen

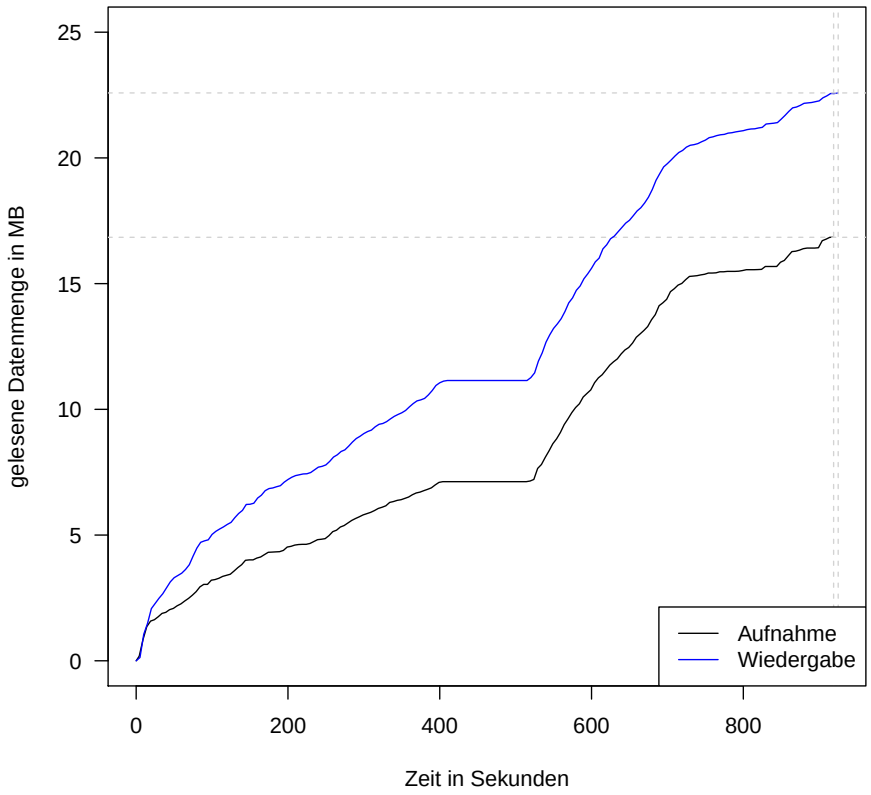

Abbildung 6.6: Vergleich gelesener Daten zwischen der Aufnahme von *Google Maps* und anschließender Wiedergabe der Daten.

Aufnahme und Wiedergabe sehr gut erkennen. Im Gegensatz dazu steht das Verhalten, mit dem die Applikationen während der Aufzeichnung verwendet wurde. Dieses lässt sich nur bei den Daten für *Google Maps* gut und eindeutig erkennen. Bei den Daten für den *Browser* lassen sich die Nutzungspausen in der Aufnahme nicht erkennen, was in der Verteilung von Leseoperationen begründet ist. In der Wiedergabe lässt sich das Nutzungsschema nur erahnen, da die Menge der gelesenen Daten stufenweise ansteigt und die Nutzungspausen jeweils nur durch eines der vielen Plateaus repräsentiert werden. Auch hier lässt sich beim *Browser*, während der Wiedergabe, eine Phase der vollständigen Inaktivität erkennen. Im Anschluss an diese Phase wird eine große Datenmenge gelesen.

Ebenfalls kann in der Aufzeichnung des *Browsers* erkannt werden, wie die Applikation mit unterschiedlichen Webseiten umgeht. Während bei der Nutzung der

ersten Webseite, welche immer nur neuen Inhalt nachlädt, der Zwischenspeicher quasi gar nicht benutzt wird, werden bei der zweiten Webseite, welche häufig neu geladen wird, viele Daten aus dem Zwischenspeicher gelesen.

geschriebene Daten

Die Daten, welche von einer Applikation gelesen werden, müssen zu irgendeinem Zeitpunkt von einer beliebigen Applikation abgespeichert werden. Wenn die gelesenen Daten in dem Speicherbereich liegen, welcher für eine Applikation exklusiv zur Verfügung steht, ist es sogar notwendig, dass die Applikation die Daten selber in den Speicher schreibt. Beide Applikationen, die für die Testszenarien verwendet werden, empfangen Daten aus dem Internet und speichern diese in ihren Zwischenspeicher.

In den Abbildungen 6.7 bzw. 6.8 ist für den *Browser* bzw. *Google Maps* die Gesamtmenge geschriebener Daten im Verlauf der Zeit dargestellt. Die schwarze Linie repräsentiert die Aufnahme und die blaue Linie die Wiedergabe.

Als erstes wird das *Browser* Szenario betrachtet. Wie bei den gelesenen Daten ist es nur schwer bis gar nicht möglich das Nutzungsverhalten aus den Daten abzulesen. Dies trifft in diesem Fall auf die Aufnahme und die Wiedergabe zu. Das einzige, was eindeutig erkennbar ist, ist die Tatsache, dass während der Nutzung der zweiten Webseite deutlich mehr Daten in den Zwischenspeicher geschrieben werden. Wie bisher in jedem untersuchten Datenmerkmal ist hier ein Plateau der Inaktivität erkennbar. Bis zu dieser abrupten Pause zeigt sich eine gute Übereinstimmung zwischen den aufgezeichneten und den wiedergegebenen Daten. Dennoch ist erkennbar, dass während der Wiedergabe die geschriebene Datenmenge zu jedem Zeitpunkt zu gering ist, um das aufgezeichnete Verhalten vollständig wiederzugeben. Dies lässt darauf schließen, dass die Vorverarbeitung der Daten, welche vor der tatsächlichen Wiedergabe durchgeführt wird, nicht korrekt arbeitet und einzelne Dateioperationen aus diesem Grund ignoriert werden.

Ein Blick auf die Daten für das Szenario mit *Google Maps* unterstreicht diese Annahme, genauso wie die zuvor aufgezeigte Abweichung in den gelesenen Daten. Im Szenario mit *Google Maps* weist die Wiedergabe keinerlei Ähnlichkeit mit der ursprünglichen Aufnahme auf. Es ist somit notwendig die Vorverarbeitung der Daten genau zu untersuchen und zu überprüfen, ob die fehlerhafte Wiedergabe ggf. nur auf einem Fehler in der Implementierung beruht, oder ob das modellierte Verfahren fehlerhaft ist.

Beispielsweise ist es möglich, dass die, aktuell verwendete, bidirektionale Abbildung von Datei Deskriptoren der Aufnahme auf Dateien der Wiedergabe so nicht anwendbar ist und zu fehlerhaft abgespielten Daten führt. Ebenso ist es möglich, dass die Filterung, welche nicht erwünschte Dateioperationen entfernt, (teilweise)

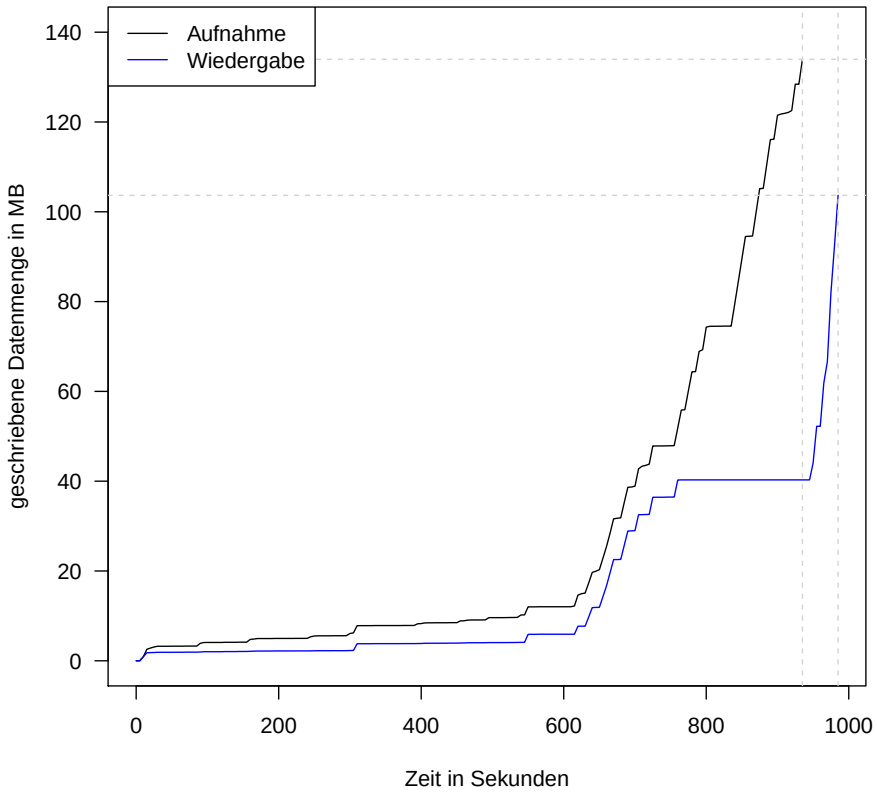

Abbildung 6.7: Vergleich geschriebener Daten zwischen der Aufnahme des *Webbrowsers* und anschließender Wiedergabe der Daten.

erwünschte Dateioperationen aus der Wiedergabe entfernt. Auch spezielle Ausgaben während der Wiedergabe lassen auf Probleme in der Vorverarbeitung schließen, wodurch Dateioperationen nicht mehr korrekt durchgeführt werden können. Aus zeitlichen Gründen ist es nicht mehr möglich gewesen die Ursache für dieses Fehlverhalten im Kontext der vorliegenden Arbeit zu ermitteln und zu beheben.

Es stellt sich die Frage, wieso die Wiedergabe im ersten Szenario nur eine geringe Abweichung aufweist, wogegen im zweiten Szenario ein vollkommen anderes Verhalten abgespielt wird. Dies lässt sich beantworten, indem das Verhalten der aufgezeichneten Applikationen genauer betrachtet wird. Von *Google Maps* werden circa 40 Dateien verwendet, wovon viele einige Megabyte groß sind, wogegen der *Browser* fast 20-mal so viele Dateien verwendet, die oftmals nur einige Kilobyte groß sind. Wenn während der Wiedergabe der *Browser* Daten nun einige Dateien

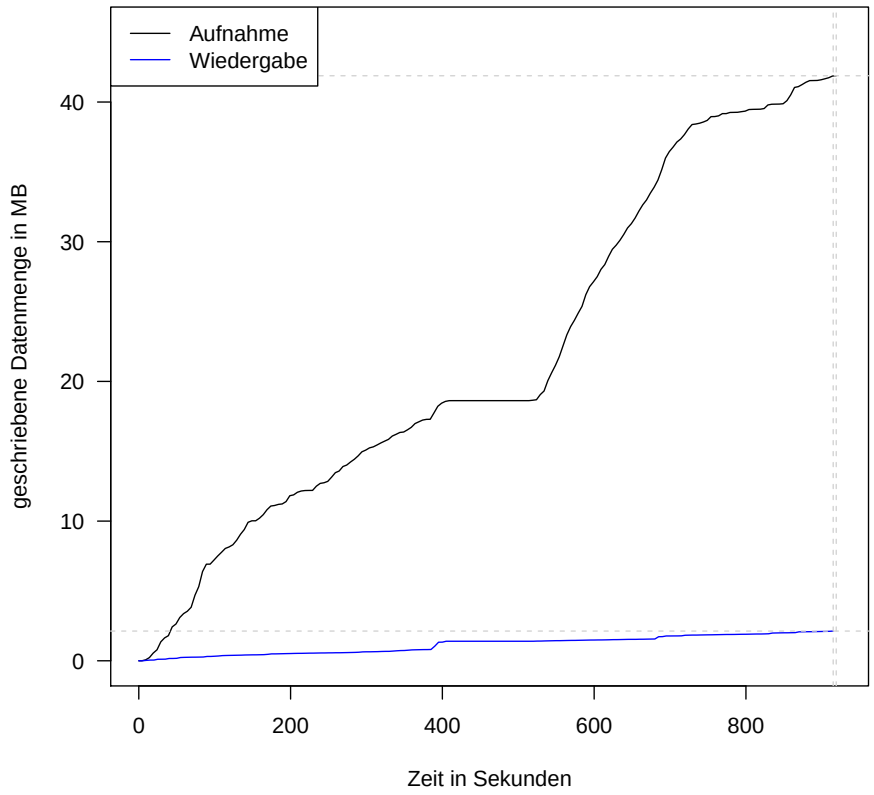

Abbildung 6.8: Vergleich geschriebener Daten zwischen der Aufnahme von *Google Maps* und anschließender Wiedergabe der Daten.

ignoriert werden, wird hierdurch keine so große Abweichung erzeugt wie bei *Google Maps*.

6.1.5 Netzwerkverkehr

Die Erzeugung von Netzwerkverkehr ist für viele Applikationen genauso wichtig, wie die Durchführung von Dateioperationen. Beide Applikationen, die für die Testszenarien ausgewählt wurden empfangen die Daten, welche sie verwenden und dem Nutzer anzeigen, aus dem Internet. Somit ist für diese Applikationen gegeben, dass der erzeugte Netzwerkverkehr ein wichtiges Merkmal ihres Verhaltens darstellt.

empfangene Daten

Bei beiden betrachteten Applikationen wird ein großer Teil des Netzwerkverkehrs durch empfangene Daten verursacht. Dies liegt daran, dass nur dann Daten gesendet werden, wenn neue Daten angefordert werden. Diese übersteigen in ihrer Größe die Anfragen um ein Vielfaches.

In den Abbildungen 6.9 bzw. 6.10 ist für den *Browser* bzw. *Google Maps* die Gesamtmenge empfangener Daten im Verlauf der Zeit dargestellt. Die schwarze Linie repräsentiert die Aufnahme und die blaue Linie die Wiedergabe.

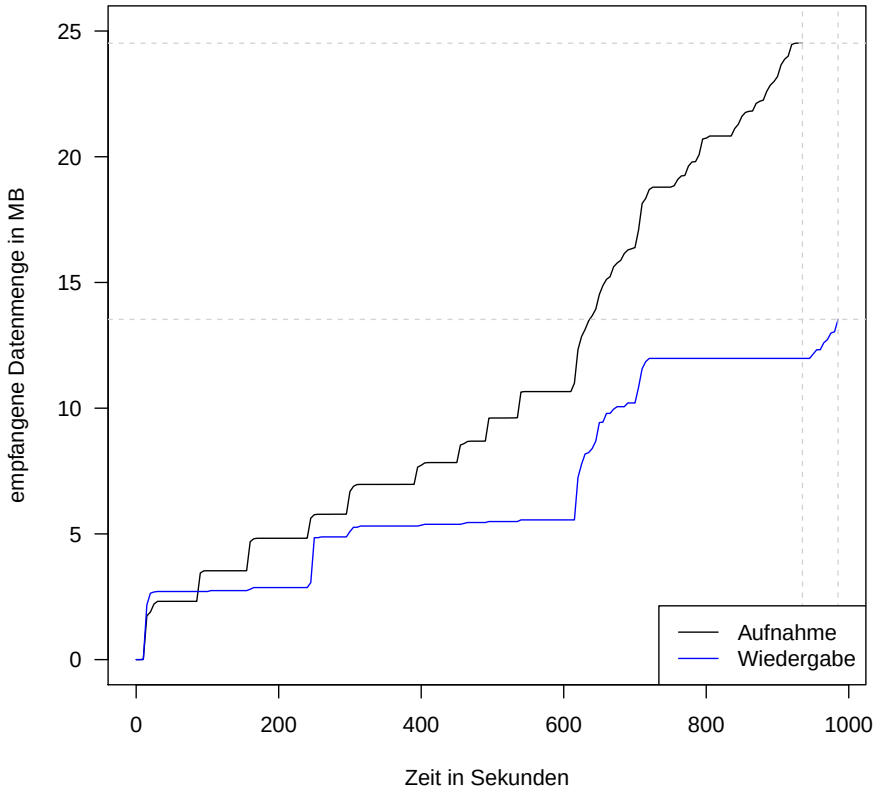

Abbildung 6.9: Vergleich empfangener Daten zwischen der Aufnahme des *Webbrowsers* und anschließender Wiedergabe der Daten.

Bei der Betrachtung der beiden Szenarien zeigt sich wieder ein unterschiedliches Bild. Im Szenario, welches sich auf den *Browser* bezieht, werden zu wenig Daten

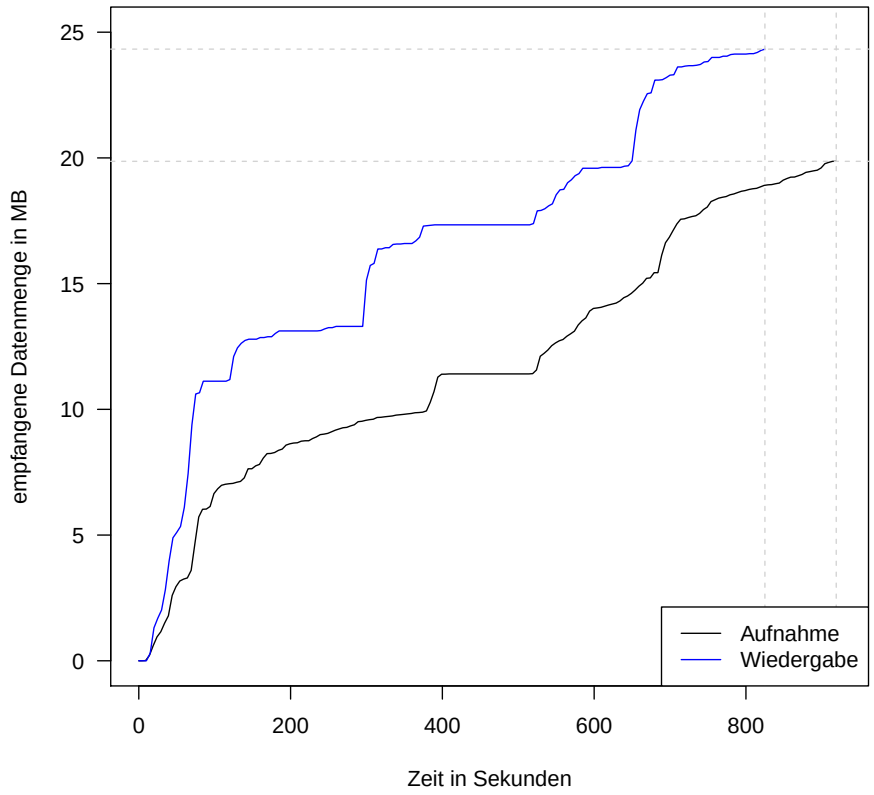

Abbildung 6.10: Vergleich empfangener Daten zwischen der Aufnahme von *Google Maps* und anschließender Wiedergabe der Daten.

empfangen. Es werden sogar über längere Zeiträume sehr wenig Daten empfangen, obwohl größere Mengen vorgesehen sind. Ebenfalls ist die längere Inaktivität zu erkennen, welche in allen Abbildungen zum *Browser* Szenario zu sehen war. Der allgemeine Verlauf der Kurve entspricht in den Phasen, wo keine Inaktivität vorliegt, aber mit einer akzeptablen Genauigkeit den aufgezeichneten Daten.

Im Gegensatz dazu werden bei der Aufzeichnung des *Google Maps* Szenarios zu viele Daten empfangen. Dabei ist der allgemeine Verlauf der Wiedergabekurve sehr ähnlich zu den aufgezeichneten Daten. Dabei ist die Kurve der Wiedergabe aber leicht gestaucht. Somit wird hier, abgesehen von der zu großen Datenmenge und der Stauchung, eine sehr hohe Genauigkeit erreicht. Darüber hinaus ist auffällig, dass während der Wiedergabe knapp anderthalb Minuten vor dem eigentlichen Ende der Aufzeichnung die maximale Datenmenge erreicht ist, was an dem zu frühen

Ende der Kurve erkannt werden kann.

Diese Problempunkte haben eine Reihe von Ursachen. Dass die Wiedergabedaten im zweiten Szenario zu früh enden, ist eine direkte Folge der gestauchten Kurve. Dabei ist die Ursache für diese Stauchung, dass die empfangenen Daten zu früh von dem Dienst, der die Daten bereitstellt, versendet werden. Aus diesem Grund müsste der Dienst überarbeitet werden, um sicherzustellen, das die Daten zu den richtigen Zeiten versendet werden. Ebenfalls sollte überprüft werden, dass die Daten für die richtigen Zeitpunkte angefordert werden und hier nicht ein Fehler in der Berechnung vorliegt (vgl. Kapitel 4.2). Ein weiteres Problem, das im zuvor genannten Dienst identifiziert werden kann, zeigt sich in den zu wenig empfangenen Daten im *Browser* Szenario. Die Aufnahme des *Browsers* umfasst fast 40-mal so viele Netzwerkverbindungen wie die von *Google Maps*. Dabei sind phasenweise viele Verbindungen gleichzeitig geöffnet, womit der Dienst nicht umgehen kann. Dies zeigt sich auch an speziellen Ausgaben von diesem Dienst. Hier müsste die Robustheit deutlich verbessert werden, da die einfache Methodik, die zur Entwicklung des Verfahrens verwendet wurde, nicht der anfallenden Belastung standhält. Das letzte Problem besteht darin, dass die Wiedergabe der *Google Maps* Daten zu viele Daten empfängt. Dabei werden die zu empfangenden Daten aus den Daten gelesen und direkt an den Server gesendet, weswegen hier keine große Fehlerwahrscheinlichkeit vorhanden ist. Wesentlich wahrscheinlicher ist, dass das Verfahren, welches während des Abspielens intelligent versucht zukünftige Daten anzufordern, unter gewissen Umständen Daten doppelt anfordert. Beispielsweise könnte es passieren, dass während der Wiedergabe, zum Zeitpunkt x eine Nachricht versendet wird, welche eine Nachricht für den Zeitpunkt y anfordert. Die Anforderung würde dann die Zeit $y - x$ enthalten. Dem zuvor geschilderten Problem nach wird später während der Wiedergabe, zum Zeitpunkt x' versendet, welche erneut Daten für den Zeitpunkt y anfordert. Diese Anforderung würde die Zeit $y - x'$ enthalten. Dies führt dazu, dass zum Zeitpunkt y zwei identische Nachrichten von dem Dienst an das Smartphone sendet, wodurch die doppelte Datenmenge empfangen wird.

Dabei war es aus zeitlichen Gründen nicht mehr möglich im Rahmen der vorliegenden Arbeit die erkannten Probleme zu bearbeiten.

gesendete Daten

In beiden Testszenarien ist es für die gewählten Applikationen nur möglich Daten zu empfangen, wenn diese zuvor durch die aufgezeichnete Applikation angefordert werden, indem selber Daten versendet werden. Dabei ist die gesendete Datenmenge wesentlich kleiner als die empfangene. Dies kann für andere Applikationen durchaus anders sein, beispielsweise wenn irgendwelche Ergebnisse an einen Server übertragen werden sollen. Auch für den *Browser* existieren Nutzungsszenarien, in denen

mehr Daten gesendet als empfangen werden, beispielsweise wenn Bilder oder andere Daten hochgeladen werden. Dies ist aber im Testszenario nicht gegeben, weswegen der Datenempfang nur einen kleinen Teil der Daten ausmacht.

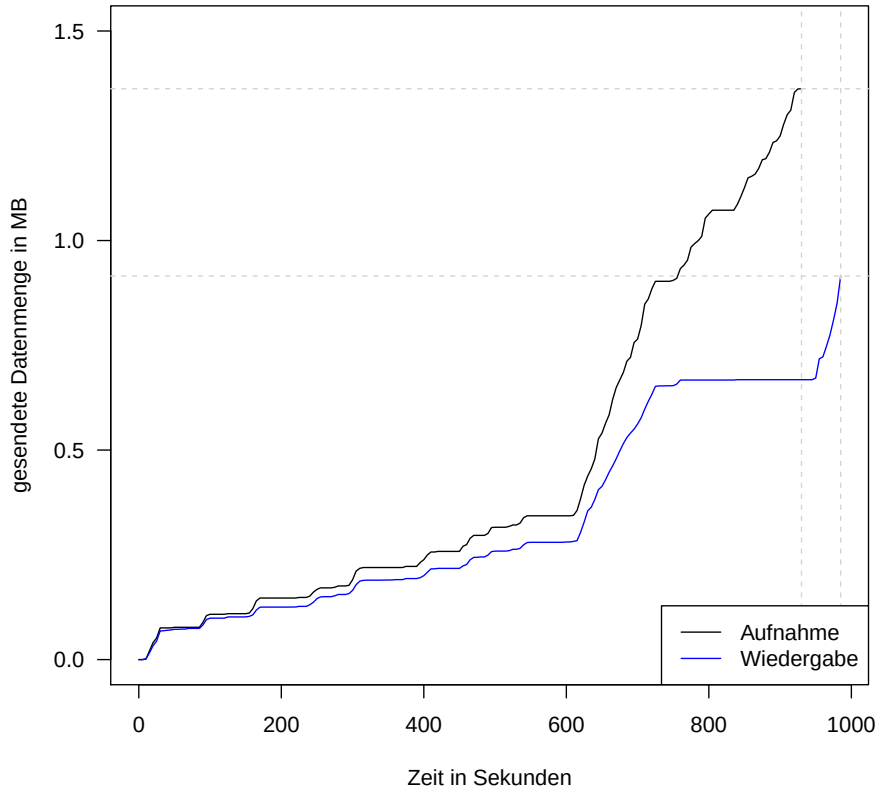

Abbildung 6.11: Vergleich gesendeter Daten zwischen der Aufnahme des *Webbrowsers* und anschließender Wiedergabe der Daten.

In den Abbildungen 6.11 bzw. 6.12 ist für den *Browser* bzw. *Google Maps* die Gesamtmenge gesendeter Daten im Verlauf der Zeit dargestellt. Die schwarze Linie repräsentiert die Aufnahme und die blaue Linie die Wiedergabe.

Zunächst lässt sich für das *Browser* Szenario eine Charakteristik feststellen, die sich durch alle untersuchten Merkmale zieht. Nämlich eine längere Unterbrechung der Wiedergabe, in der keinerlei Aktivität stattfindet.

Abgesehen davon wird in beiden Szenarien eine hohe Genauigkeit der Wiedergabe erreicht. Die minimale Überschreitung der gewünschten Datenmenge, welche in

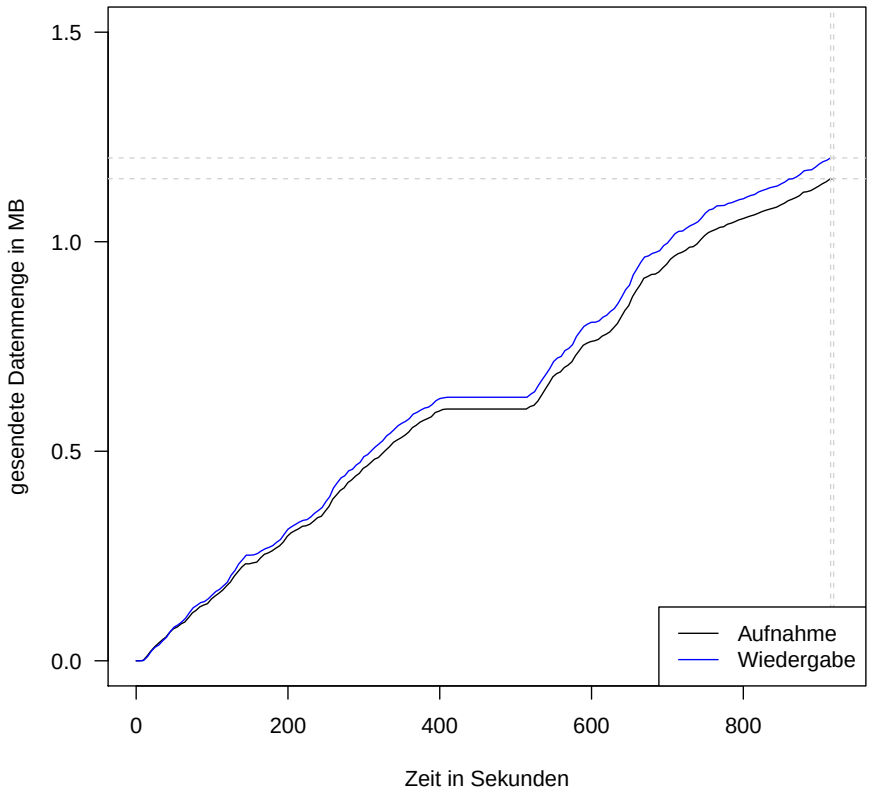

Abbildung 6.12: Vergleich gesendeter Daten zwischen der Aufnahme von *Google Maps* und anschließender Wiedergabe der Daten.

der Wiedergabe der Daten von *Google Maps* sichtbar ist, kann damit kann einfach begründet werden. Hierfür muss beachtet werden, dass die Nachrichten, welche die Daten anfordern und von der Applikation zur Wiedergabe versendet werden, eine Minimalgröße besitzen, welche in geringem Maß die Größe der ursprünglichen Nachrichten übersteigen kann. Dies lässt sich nicht vermeiden. Dem gegenüber steht das *Browser* Szenario, in welchem zu wenig Daten versendet werden. Der Grund hierfür ist die mangelnde Robustheit des zuvor angesprochenen Web-Dienstes. Diese sorgt dafür, dass zeitweise keine Verbindungen aufgebaut werden können und somit auch keine Daten versendet werden können.

6.1.6 Ergebnis

Die Evaluation hat verschiedene Erkenntnisse über das entwickelte Verfahren zum Vorschein gebracht. Diese werden an dieser Stelle nochmals zusammengefasst betrachtet und notwendige, zukünftige Handlungen aufgezeigt.

Auch wenn keine Korrektheit und Vollständigkeit der Datenerfassung nachgewiesen werden konnte, hat sich gezeigt, dass die aufgezeichneten Daten plausibel mit dem Verhalten erklärt werden kann, welches während der Aufzeichnung der Applikation verwendet wurde. In Kombination mit den Tests, die während der Implementierung durchgeführt wurden, kann stark davon ausgegangen werden, dass die Datenerfassung nahezu vollständig und korrekt ist. Um eine formale Korrektheit und Vollständigkeit nachzuweisen, könnte eine spezielle Applikation entwickelt werden, welche alle Ereignisse nach einem festen Schema und auf allen möglichen Wegen auslöst. Diese Applikation müsste dann aufgezeichnet werden und die entstehende Ausgabe einer, ebenfalls definierten, Vorlage entsprechen.

Im Rahmen der Evaluation wurden weder *Wakelocks* noch Positionsdaten betrachtet. Dies liegt daran, dass diese Datenquellen in beiden Testszenarien nicht auftauchen. Dies kann zwei verschiedene Gründe haben. Entweder verwenden die untersuchten Applikationen diese Funktionen nicht, oder es werden die Alternativen Schnittstellen der *Google Play Services* verwendet. Zumindest für *Google Maps* kann davon ausgegangen werden, dass die Nutzung von Positionsdaten sehr wahrscheinlich ist. Es ist für diese Applikation aber ebenso wahrscheinlich, dass sie die alternative Schnittstelle verwendet (siehe Kapitel 4.1).

Ebenfalls wurden Daten zum Batteriezustand von der Betrachtung ausgeschlossen. Der Grund hierfür ist, dass das Gerät, welches zur Evaluation genutzt wurde, den gesamten Zeitraum über an den Computer angeschlossen war, um die Ausgabe zu überwachen und eventuelle Auffälligkeiten zu identifizieren. Somit wurde das Gerät permanent geladen und die anfallenden Daten zum Batteriezustand sind wenig aussagekräftig.

Bei allen anderen Aspekten wurde eine hohe Genauigkeit festgestellt. Es existieren aber auch Problemstellen, die im Einzelfall ein erreichen dieser Genauigkeit verhindern können. So ist eine kleine Anpassung an dem Verfahren notwendig, welches während der Wiedergabe die Prozessorlast erzeugt. Indem die notwendige Last auf allen Kernen gleichzeitig erzeugt wird, erhöht sich die erreichte Genauigkeit enorm. Darüber hinaus wurde eine Schwachstelle in der Vorverarbeitung der Daten entdeckt, welche dafür sorgt, dass Dateioperationen in manchen Fällen nicht korrekt wiedergegeben werden können. Hier ist eine Überarbeitung notwendig, in dessen Zuge die Eignung des verwendeten Verfahrens festgestellt werden muss. Das letzte Problem der Datenwiedergabe liegt im externen Netzwerk Dienst, welcher alle Netzwerkdaten für die Wiedergabe bereitstellt und entgegen nimmt. Es hat

sich herausgestellt, dass das verwendete Verfahren schlecht mit der anfallenden Last skaliert. Konkret ist der Dienst während der Wiedergabe der *Browser* Daten zeitweise gar nicht erreichbar gewesen, was zu großen Abweichungen in den wiedergegebenen Daten geführt hat. Es hat sich hierdurch während der Wiedergabe eine Phase der vollkommenen Inaktivität ergeben. Die abzuspielenden Ergebnisse haben sich dabei angestaut, was zu einer intensiven Ressourcennutzung nach der Inaktivität und einer verlängerten Dauer der Wiedergabe geführt hat.

Zusammenfassend lässt sich sagen, dass der Abspielvorgang eine hohe Genauigkeit erreicht, aber gleichzeitig noch kleinere Mängel bei Dateioperationen und Prozessorlast aufweist. Das externe Hilfswerkzeug, welches die Wiedergabe des Netzwerkverkehrs ermöglicht, ist dagegen für Szenarien mit viel Datenverkehr vollständig ungeeignet und muss auf andere Art realisiert werden.

6.2 Erkenntnisse aus gesammelten Daten

Nach der Gütebewertung des Abspielvorgangs sollen an dieser Stelle weitere aufgezeichnete Datensätze untersucht werden. Dabei geschieht dies nur im Hinblick auf die aufgezeichneten Daten. Es findet somit keine Wiedergabe mit anschließendem Vergleich zur Aufnahme statt. Im Folgenden werden die Datensätze vorgestellt und im Anschluss relevante Aspekte von ihnen herausgestellt.

6.2.1 WhatsApp

Bei *WhatsApp* handelt es sich um eine Applikation zum Nachrichtenaustausch, welche weltweit die größte Verbreitung unter solchen Applikationen besitzt[3]. Neben simplen Textnachrichten, ist es möglich diverse Medien an Kontakte zu verschicken. Darin enthalten sind Bilder, Videos und Sprachnachrichten.

Ursprünglich war *WhatsApp* eine reine Smartphone-Applikation, steht inzwischen aber auch als klassische Applikation für den Computer zur Verfügung. Dabei ist die Version für den Computer nur eine Erweiterung der Smartphone Applikation und keine Alternative. So ist für die Nutzung von *WhatsApp* ein Smartphone mit aktiver Internetverbindung, sowie *WhatsApp* mit aktiviertem Benutzerkonto notwendig[4]. Dies liegt daran, dass der eigentliche Nachrichtenaustausch noch immer über das Smartphone abgewickelt wird und lediglich die Version für den Computer mit den Daten des Smartphones synchronisiert wird.

[3]Siehe `https://www.similarweb.com/blog/worldwide-messaging-apps`, aufgerufen am 11.12.2016

[4]Siehe `https://www.whatsapp.com/faq/de/web/28080003`, aufgerufen am 11.12.2016

Für *WhatsApp* wurden drei verschiedene Datensätze erfasst, welche jeweils einen Zeitraum von circa 24 Stunden abdecken. Dabei wurde bei einem Datensatz ausschließlich die Applikation für den Computer zur Kommunikation verwendet. Bei den anderen beiden Datensätzen wurde sowohl die Computer- als auch die Smartphone-Applikation, in nicht genauer bekannten Anteilen, genutzt. Der Datensatz, welcher nur durch die Nutzung der Computer-Applikation entstanden ist, wurde von einem Nutzer erstellt. Die beiden anderen Datensätze wurden von einem anderen Nutzer erfasst.

Nachfolgend werden der Netzwerkverkehr und durchgeführte Dateioperationen der Datensätze genauer betrachtet und analysiert. Dabei wird versucht, soweit möglich, Informationen über *WhatsApp* aus den Daten zu ziehen.

6.2.1.1 Dateioperationen

Damit der Nutzer die empfangenen und versendeten Nachrichten auch im weiteren Verlauf betrachten kann, ist es notwendig die Nachrichten zu persistieren. Eine Möglichkeit besteht darin, die Daten auf zentralen Servern abzuspeichern und bei Bedarf wieder abzurufen. Dies wird von *WhatsApp* so nicht durchgeführt. Stattdessen werden sämtliche Nachrichtenverläufe lokal auf dem Smartphone abgespeichert.

Der Abbildung 6.13 kann für jeden Datensatz die geschriebene Datenmenge im Verlauf der Zeit entnommen werden. Die schwarze Linie repräsentiert den Datensatz welcher durch ausschließliche Nutzung der Computer-Applikation von *WhatsApp* entstanden ist.

Es ist erkennbar, dass bei allen drei Datensätzen am Anfang ein ähnliches Verhalten auftritt. In dem Moment, wo die Applikation mit schreibenden Dateioperationen beginnt, wird eine Menge von circa 30 Megabyte geschrieben. Der Ursprung dieser Daten kann nicht für alle Datensätze klar benannt werden. Für den Datensatz, welcher durch reine Nutzung der Computer-Applikation entstanden ist, kann die Ursache benannt werden. *WhatsApp* führt standardmäßig eine tägliche Sicherung der Daten durch. Das aufzeichnende Gerät hat *WhatsApp* so konfiguriert, dass diese Sicherung um zwei Uhr in der Nacht durchgeführt wird. Da die Aufzeichnung für diesen Datensatz, mit einer Abweichung von weniger als einer Minute, um Mitternacht begonnen wurde lassen sich die erwähnten Daten mit dem Sicherungsvorgang in Verbindung bringen. In dem Zuge der Sicherung werden die neuen, zu sichernden Nachrichten aus dem persistenten Speicher geladen. Für die anderen Datensätze kann eine solche Beziehung nicht mit Sicherheit festgestellt werden, da die notwendigen Informationen über die Konfiguration von *WhatsApp* nicht vorliegen.

Eine andere Möglichkeit, wo diese Daten herstammen ist, dass es sich um die Persistierung der Daten handelt, welche beim Start der Applikation empfangen

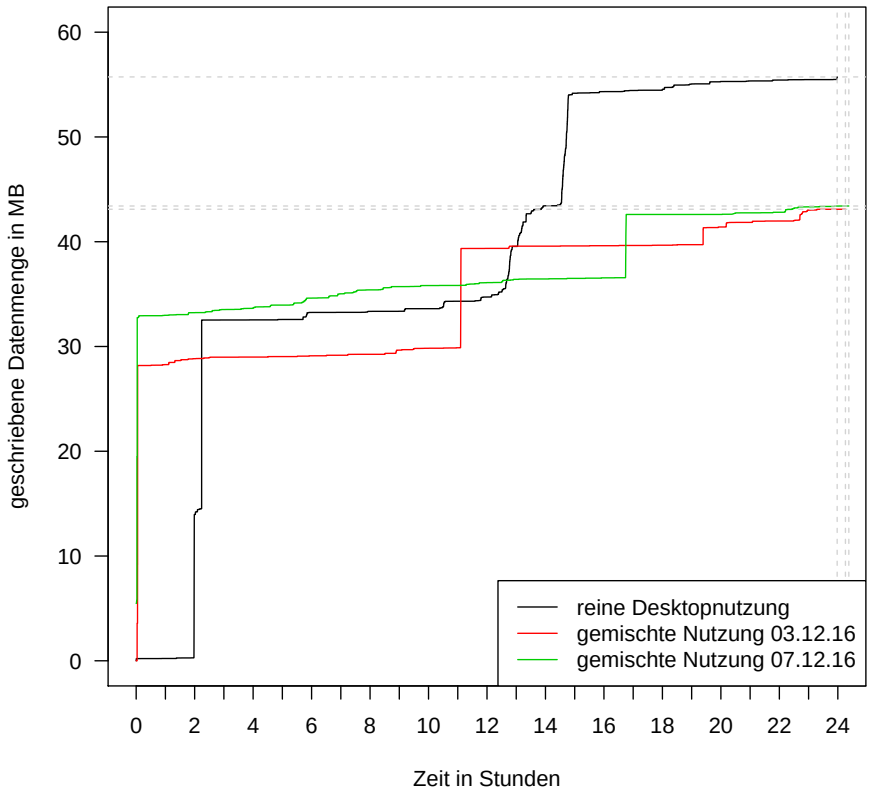

Abbildung 6.13: Darstellung der Menge geschriebener Daten von verschiedenen Datensätzen für *WhatsApp*

werden. Dagegen spricht, dass in allen Datensätzen eine ähnliche Menge an Daten geschrieben werden. Dies spricht dafür, dass *WhatsApp* irgendwelche anderen Daten abspeichert, die an dieser Stelle nicht näher benannt werden können.

Darüber hinaus kann erkannt werden, dass bei allen Datensätzen im Laufe der Zeit immer wieder Daten geschrieben werden. Teilweise sehr wenige, aber es lassen sich auch Situationen identifizieren wo in sehr kurzer Zeit mehrere Megabyte an Daten geschrieben werden. Auch wenn dies nicht mit Sicherheit nachgewiesen werden kann, liegt die Vermutung nahe, dass es sich hierbei um die gesendeten und empfangenen Nachrichten handelt. Da jede Nachricht die versandt bzw. empfangen wird, auch abgespeichert wird, liegt die Annahme nahe, dass zwischen der Menge von geschriebenen Daten und der Summe von gesendeten und empfangenen Daten eine Korrelation besteht. Diese Annahme kann erst dann untermauert bzw.

widerlegt werden, wenn der Netzwerkverkehr analysiert wurde.

Unter der vorherigen Annahme kann davon ausgegangen werden, dass bei dem Datensatz, welcher ohne Nutzung der Smartphone-Applikation erzeugt wurde, größere bzw. mehr Nachrichten gesendet und empfangen werden, als bei den anderen Datensätzen. Ebenfalls ist auffällig, das bei den beiden anderen Datensätzen die Gesamtmenge geschriebener Daten nahezu identisch ist. Dies ist ein mögliches Indiz dafür, dass *WhatsApp* in beiden Zeiträumen der Datenerfassung relativ ähnlich genutzt wird. Diese Annahme kann aufgrund der geringen Datenbasis nicht weiter bewiesen oder widerlegt werden.

In Abbildung 6.14 ist die gelesene Datenmenge im Verlauf der Zeit dargestellt. Dabei steht die schwarze Linie für den Datensatz, bei dem *WhatsApp* ausschließlich am Computer verwendet wurde. Die rote und die grüne Linie repräsentieren die anderen Datensätze.

Als erstes fällt auf, dass die Menge an gelesenen Daten die geschriebenen massiv übersteigt und zwar mindestens um das gut 40-fache. Auch hier ist erkennbar, dass im Rahmen der ersten lesenden Operation eine große Datenmenge verarbeitet wird, welche mehr als die Hälfte der Gesamtmenge ausmacht. Wie bei den geschriebenen Daten kann auch hier abgelesen werden, dass der Datensatz ohne Nutzung des Smartphones circa zwei Stunden nach den anderen Datensätzen beginnt Aktivitäten zu zeigen.

Der Ursprung dieser Daten kann nicht genau bestimmt werden. Dennoch ist es möglich eine Vermutung anzustellen. So liegt die Vermutung nahe, dass zur Startzeit der Applikation eine größere Menge Daten gelesen wird, um Teile der vorhandenen Nachrichtenverläufe zu laden und für den Benutzer verfügbar zu machen. Öffnet der Nutzer, während der Nutzung, Medien aus diesen Nachrichtenverläufen, oder möchte bisher noch nicht geladene Nachrichten betrachten ist es notwendig weitere Daten zu lesen.

Unter der Annahme, dass ein Großteil der gelesenen Daten auf gespeicherte Nachrichtenverläufe entfällt, kann auch festgestellt werden, dass auf dem Gerät, welches zur Erfassung der gemischten Nutzungsszenarien verwendet wurde, eine beinahe doppelt so große Menge von Nachrichten gespeichert ist, als im Vergleich zum Szenario der Nutzung ohne Interaktionen mit dem Smartphone. Dies kann auf mehr bzw. größere Nachrichten deuten.

In diesem Zusammenhang wird noch die Gesamtmenge gelesener Daten betrachtet, welche den aufgezeichneten Datensätzen entnommen werden kann. Dabei werden nur die beiden Datensätze betrachtet, bei denen eine gemischte Nutzung vorliegt. Die Differenz zwischen der Menge gelesener Daten beträgt zwischen den beiden Datensätzen grob 200 Megabyte. Zwischen den beiden Aufzeichnungen lagen vier Tage und in beiden Datensätzen wurden insgesamt circa 45 Megabyte an Daten geschrieben. Geht man davon aus, dass *WhatsApp* an den nicht aufgezeichneten

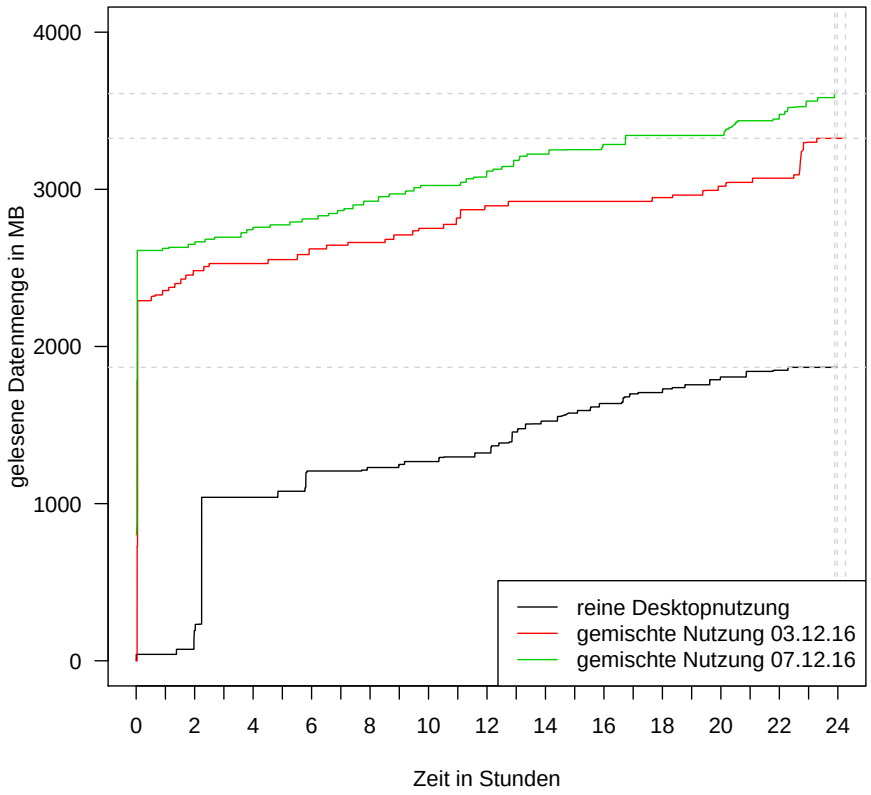

Abbildung 6.14: Darstellung der Menge gelesener Daten von verschiedenen Datensätzen für *WhatsApp*

Tagen ähnlich genutzt wurde, ergibt sich eine Übereinstimmung in der Differenz der gelesenen Datenmengen und den geschriebenen Daten.

Zusammenfassend lässt sich für die Dateioperationen sagen, dass zwar keine definitiven Aussagen über die Ursprünge der Daten gemacht werden konnten, es aber dennoch möglich war das erwartete Verhalten von *WhatsApp* anhand der Daten zu erklären und zu belegen, ohne Korrektheit nachzuweisen.

6.2.1.2 Netzwerkverkehr

Sowohl das Senden, als auch das Empfangen von Daten ist für *WhatsApp* eine zentrale Funktion, da die einzige Aufgabe der Applikation die Kommunikation mit anderen Benutzern ist. Somit entsteht für jede gesendete und für jede empfan-

gene Nachricht Netzwerkverkehr. Sofern die Computer-Applikation von *WhatsApp* verwendet wird entsteht zusätzlicher Netzwerkverkehr bei der Synchronisation zwischen Smartphone und Computer. Ebenfalls muss irgendeine Art von Sitzungsverwaltung durchgeführt werden, damit Nachrichten richtig zugestellt und versendet werden können.

Als erstes wird betrachtet mit welchen Adressen *WhatsApp* im Zeitraum der Datenerfassung kommuniziert hat. Dabei wurden aus den Datensätzen 37, 48 und 51 verschiedene Adressen extrahiert. Eine Filterung ergibt 123 verschiedene Adressen mit denen *WhatsApp*, während der Erfassung aller Datensätze, Verbindungen aufgebaut hat. Dabei hat jeder Datensatz zwischen 60 und 65 Verbindungen aufgebaut und zwischen 55 und 61 geschlossen. Die geringe Abweichung kann damit begründet werden, dass Verbindungen über das Ende der Aufzeichnung erhalten bleiben können, da die Applikation an dieser Stelle nicht beendet wird. Eine Stichprobenartige Überprüfung dieser Adressen hat ergeben, dass es sich bei allen Servern um die *WhatsApp* Server handelt[5], was zu erwarten war.

Darüber hinaus wurden drei verschiedene Ports identifiziert, die zur Abwicklung sämtlicher Kommunikation verwendet werden: 80, 443 & 5222. Dies sind die Standardports für *HTTP*, *HTTPS* und *XMPP*. Eine Recherche hat ergeben, dass dies die (eine Teilmenge der) Standardsports sind, welche *WhatsApp* verwendet[6].

Die Abbildungen 6.15 und 6.16 zeigen die empfangenen und die gesendeten Daten für alle *WhatsApp* Datensätze im Verlauf der Zeit. Die schwarze Linie repräsentiert den Datensatz welcher durch ausschließliche Nutzung der Computer-Applikation von *WhatsApp* entstanden ist.

Als erstes werden die empfangenen Daten betrachtet. Es fällt sofort auf, dass eine sehr geringe Datenmenge empfangen wird. Dies trifft auf alle Datensätze gleichermaßen zu. Da bei zwei der drei Datensätze auch lange Plateaus zu sehen sind, in denen keine Daten empfangen werden, liegt die Vermutung nahe, dass nicht sämtlicher Netzwerkverkehr von der Applikation aufgezeichnet wird. Somit ist es nicht möglich aus den erfassten Daten Schlüsse über die Nutzung von *WhatsApp* bzw. *WhatsApp* im Allgemeinen zu ziehen.

Bei der Betrachtung der gesendeten Daten ergibt sich ein ähnliches Bild. Es wurden wesentlich mehr Daten empfangen als gesendet. Die Gesamtmenge der empfangenen Daten ließe sich auch plausibel mit empfangenen Nachrichten erklären und mit der geschriebenen Datenmenge in Verbindung bringen. Dies ist aber dennoch nicht möglich, da der Verlauf der Daten nicht plausibel mit dem Verhalten von *WhatsApp* erklärt werden kann. Zu mindestens 80% der Zeit findet keinerlei

[5]Siehe `https://www.whatsapp.com/cidr.txt`, aufgerufen am 11.12.2016

[6]Siehe `https://github.com/ukanth/afwall/wiki/HOWTO-blocking-WhatsApp`, aufgerufen am 11.12.2016

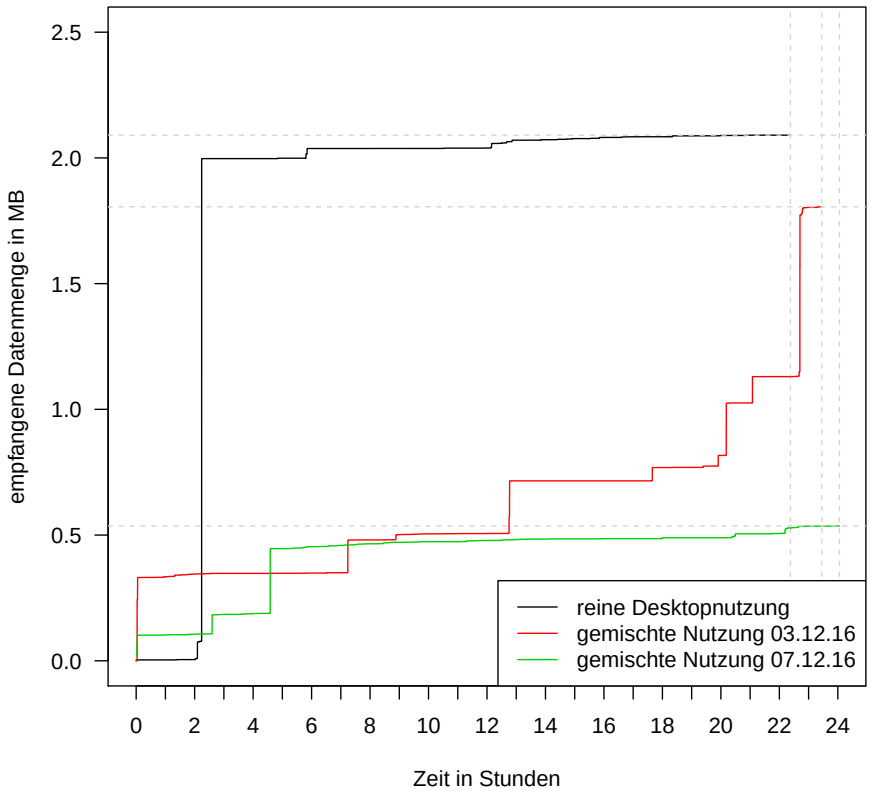

Abbildung 6.15: Darstellung der Menge empfangener Daten von verschiedenen Datensätzen für *WhatsApp*

Datenversand statt und das obwohl im Laufe der Datenaufzeichnung immer wieder Nachrichten verschickt wurden.

Es liegt somit offensichtlich ein Problem mit der Erfassung von Netzwerkverkehr vor. Denn neben der nicht plausiblen Menge an gesendeten und empfangenen Daten ist die Anzahl der aufgebauten Verbindungen sehr gering. Dabei verrät ein genauer Blick in die aufgezeichneten Daten, dass einige der Verbindungen für sehr lange Zeiträume (teilweise über 18 Stunden) geöffnet sind, was der Anzahl der Verbindungen eine gewisse Glaubwürdigkeit gibt. Dennoch sollte dies genau überprüft werden.

Im Rahmen der Implementierung wurden diverse iterative Änderungen am *SystemTap*-Skript für die Erfassung von Netzwerkdaten vorgenommen, um die Robustheit und die Genauigkeit der Erfassung zu verbessern. Dabei wurden diverse Tests

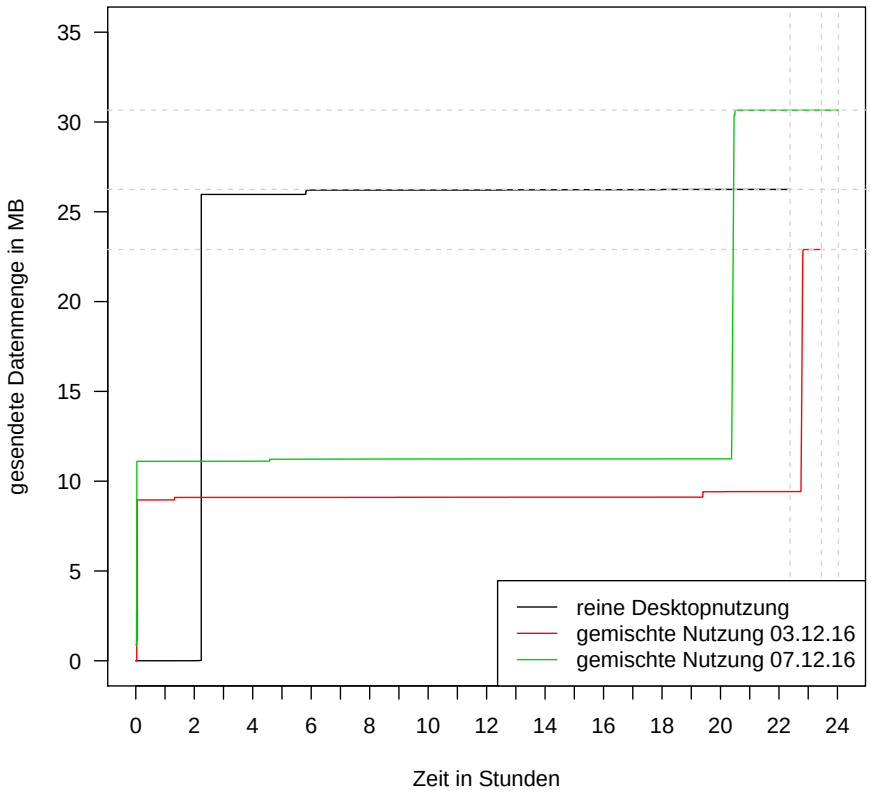

Abbildung 6.16: Darstellung der Menge gesendeter Daten von verschiedenen Datensätzen für *WhatsApp*

durchgeführt um die Funktionsweise zu überprüfen. Aus diesem Grund kann mit relativer Sicherheit davon ausgegangen werden, dass die Erfassung der Netzwerkdaten ordnungsgemäß funktioniert, sofern die beobachteten Funktionen verwendet werden. Da der *Kernel* von *Android* beliebige Modifikationen gegenüber dem *Linux-Kernel* besitzen kann, liegt die Vermutung nahe, dass noch weitere Funktionen zur Durchführung von Netzwerkverkehr existieren, welche von dem aktuellen *SystemTap*-Skript nicht erfasst werden.

Aus diesem Grund es notwendig eine weitere Überarbeitung des entsprechenden *SystemTap*-Skripts durchzuführen und in diesem Zuge den *Android-Kernel* genauer zu untersuchen. Die zeitlichen Beschränkungen, denen die vorliegende Arbeit unterliegt, haben verhindert, dass dies noch im Kontext der Arbeit geschehen konnte. In diesem Zuge sollte nochmals das *SystemTap*-Skript überarbeitet werden, da

die Ausgabe von *SystemTap* signalisiert, dass es immer wieder zu übersprungenen Behandlung aufgrund abgebrochener *kprobes* kommt. Dies liegt daran, dass *SystemTap* die Behandlung abbricht, sollte sie zu lange dauern (vgl. Kapitel 5.1.0.7).

Als Ergebnis für die Analyse der Netzwerkdaten von *WhatsApp* kann festgestellt werden, dass zwar keine Informationen über das Verhalten von *WhatsApp* erlangt werden konnten, es aber gelungen ist eine Schwäche in der Datenerfassung zu bestimmen.

6.2.2 Facebook

Bei *Facebook* handelt es sich um das soziale Netzwerk, welches weltweit die größte Verbreitung besitzt [17]. Es wurde im Jahr 2004 veröffentlicht und verzeichnet seitdem immer weiter steigende Nutzerzahlen [28].

Durch die große Verbreitung von *Facebook* ist es für eine große Anzahl an Nutzern relevant, wie sich die Applikation auf dem ausführenden Gerät verhält. Aus diesem Grund wurde *Facebook* für eine Analyse des Verhaltens ausgewählt.

Dabei liegt für *Facebook* ein einzelner Datensatz vor, welcher einen Zeitraum von circa 24 Stunden umfasst. Im Folgenden werden einzelne Aspekte der Daten betrachtet und miteinander in Verbindung gesetzt.

6.2.2.1 Activity-Lebenszyklus

In Abbildung 6.17 sind, für den gesamten Zeitraum der Datenerfassung, die Phasen eingezeichnet, in denen der Bildschirm eingeschaltet war. Darüber hinaus ist der Lebenszyklus, von den *Activities* eingezeichnet, die *Facebook* verwendet hat.

Die Zeiträume, in denen eine *Activity* aktiv genutzt wird, sind im Verhältnis zur Dauer der Datenerfassung sehr klein. Aus diesem Grund lassen sich aus der Abbildung nur schwer Informationen zum Lebenszyklus von *Activities* ablesen. Da diese Daten im Rahmen der Analyse aber nur verwendet werden, um sehr grobe Informationen zu erhalten, ist dies nicht weiter problematisch. Für genaue Analysen wäre es notwendig die Daten anders bzw. stückweise darzustellen.

In den ersten 9 Stunden der Datenerfassung sind viele und häufige Änderungen im *Activity*-Lebenszyklus sichtbar. Dies lässt darauf schließen, dass die Applikation nur in diesem Zeitraum aktiv genutzt wurde. Darüber hinaus ist eine weitere kurze Nutzungsphase gegen Ende der Datenaufzeichnung zu erkennen.

6.2.2.2 Dateioperationen

Facebook dient dazu dem Benutzer Inhalte zu präsentieren, welche von anderen Mitgliedern des Netzwerkes erzeugt wurden. Außerdem kann der Nutzer selber Inhalte

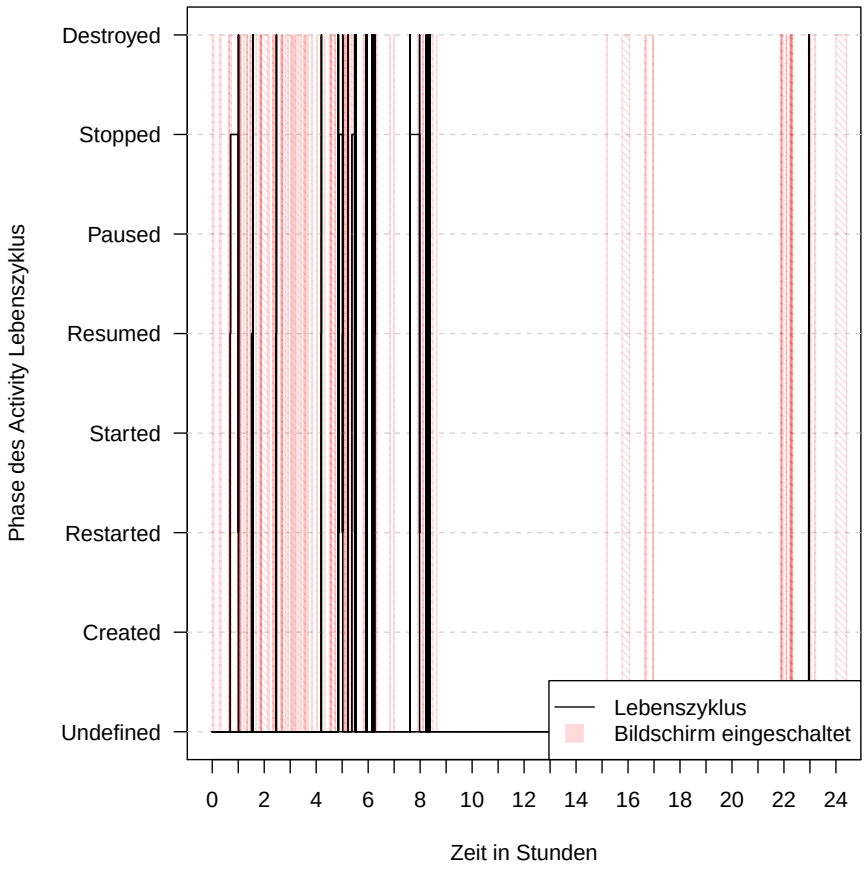

Abbildung 6.17: Darstellung des *Activity*-Lebenszyklus und Phasen in denen der Bildschirm eingeschaltet ist für den *Facebook* Datensatz

mit den anderen Mitgliedern Teilen. Deshalb kann davon ausgegangen werden, dass die Daten aus dem Internet empfangen werden und nicht auf dem Gerät vorliegen. Dennoch ist es für die Applikation sinnvoll, ähnlich wie der *Browser*, einen Zwischenspeicher zu verwenden, um Inhalte die häufig angezeigt werden, persistent zu speichern. Hierdurch wird der anfallende Netzwerkverkehr reduziert. Es besteht somit eine Ähnlichkeit zwischen der Datennutzung von *Facebook* und dem zuvor betrachteten *Browser*.

In den Daten, die zwischengespeichert werden sind beispielsweise die Profilbilder von Nutzern und bereits betrachtete Medien enthalten.

Die Abbildungen 6.18 und 6.19 zeigen die Gesamtmenge gelesener bzw. geschriebener Daten im Verlauf der Zeit.

Es ist gut zu erkennen, dass die Zeiten, in denen ein Großteil der Daten gele-

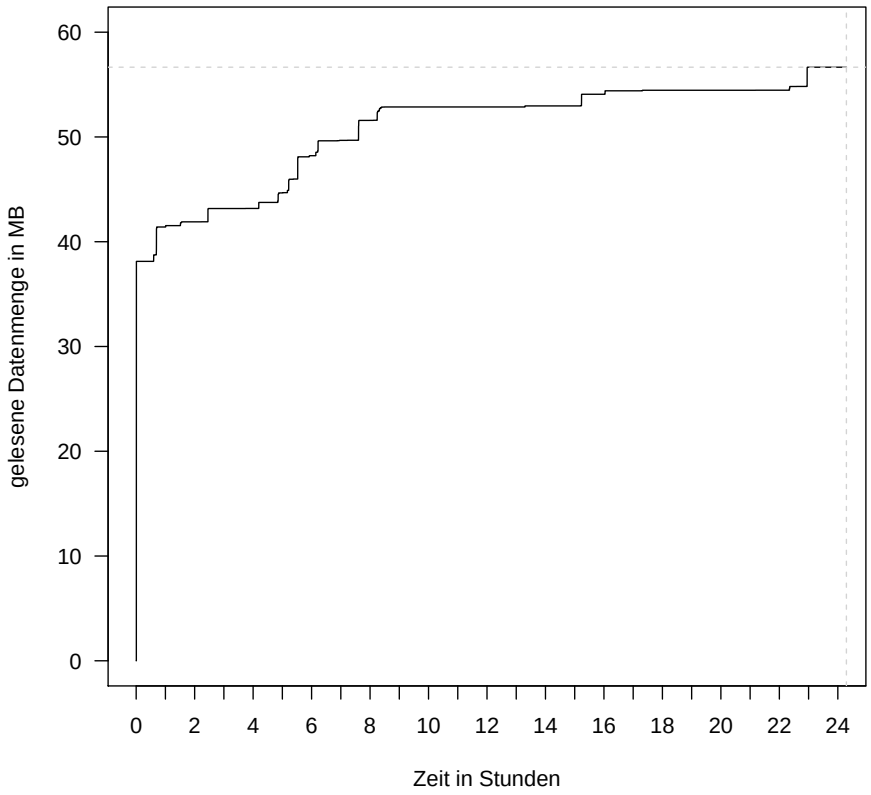

Abbildung 6.18: Darstellung der Menge gelesener Daten für den *Facebook* Datensatz

sen bzw. geschrieben wird, mit den Zeiten übereinstimmen die im Vorfeld als die Zeiträume identifiziert wurden, in denen die Applikation verwendet wird. Dabei übersteigt die geschriebene Datenmenge die gelesene um mehr als das vierfache, was dafür spricht, dass empfangene Daten für eine erneute Anzeige persistiert werden und nur ein geringer Teil der Daten auf dem Gerät vorliegt.

Außerdem kann erkannt werden, dass direkt zu Beginn der Datenaufzeichnung circa zwei Drittel der Gesamtmenge der Daten gelesen bzw. geschrieben wird. Auf Basis der vorliegenden Daten kann nicht genau bestimmt werden, was die Ursache für diese starke Aktivität ist. Eine Möglichkeit besteht darin, dass beim ersten Start der Applikation eine große Menge an gespeicherten Daten gelesen wird und für die Anzeige im Hauptspeicher gehalten wird. Werden während der Nutzung weitere Daten benötigt, die im Zwischenspeicher vorhanden sind, werden diese

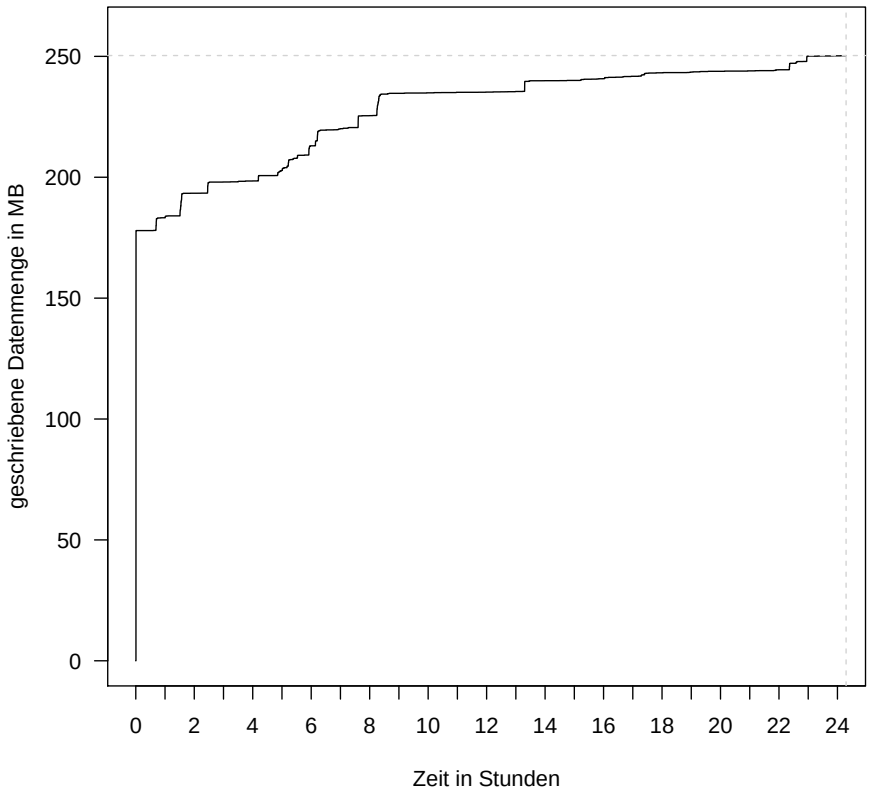

Abbildung 6.19: Darstellung der Menge geschriebener Daten für den *Facebook* Datensatz

nachgeladen. Analog dazu ist es möglich einen Teil der Daten, die direkt zu Beginn der Aufzeichnung geschrieben werden, den neuen Daten zuzuordnen, welche empfangen und dem Nutzer angezeigt werden sollen.

Des weiteren ist auffällig, das die Verläufe der gelesenen und geschriebenen Datenmenge sehr ähnlich sind und der einzige signifikante Unterschied in der Gesamtmenge besteht. Eine Mögliche Ursache hierfür ist, dass neue Daten, die empfangen werden, zuerst gespeichert und im Anschluss direkt für die Anzeige geladen werden.

6.2.2.3 Netzwerkverkehr

Das Empfangen und Versenden von Daten ist eine elementare Funktionalität von *Facebook*. Dies liegt daran, da das zentrale Nutzungsszenario von *Facebook* dar-

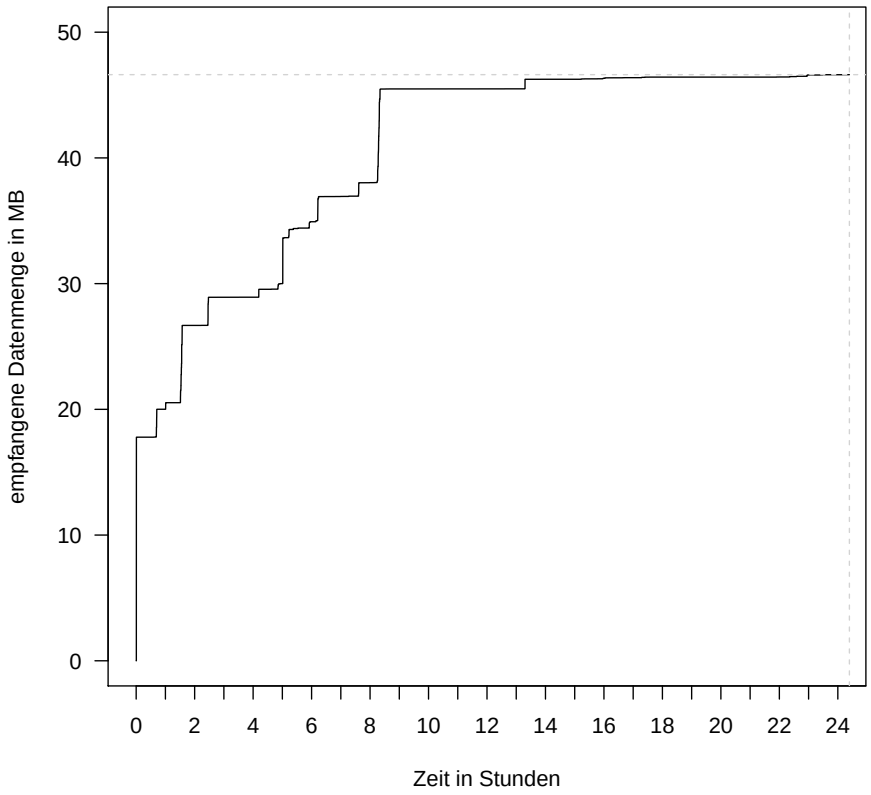

Abbildung 6.20: Darstellung der Menge empfangener Daten für den *Facebook* Datensatz

in besteht, die Inhalte, die von anderen Nutzern erzeugt wurden, zu betrachten und selber Inhalte mit anderen Benutzern zu teilen bzw. auf bestehende Inhalte zu reagieren. Es sei darauf hingewiesen, dass *Facebook* eine Möglichkeit zum Nachrichtenaustausch bietet, die in der Funktionalität sehr ähnlich zu *WhatsApp* ist. Diese steht seit 2011 als eigenständige Applikation, mit dem Namen *Facebook Messenger*, zur Verfügung[7]. Sollte diese Applikation im Rahmen der Datenerfassung genutzt worden sein, sind hierzu keine Daten erfasst worden. Der vorliegende Datensatz beschränkt sich somit vollständig auf das soziale Netzwerk *Facebook*.

In den Abbildungen 6.20 und 6.21 ist die Menge gelesener bzw. empfangener

[7]Siehe `https://newsroom.fb.com/news/2011/08/a-faster-way-to-message-on-mobile/`, aufgerufen am 13.12.2016

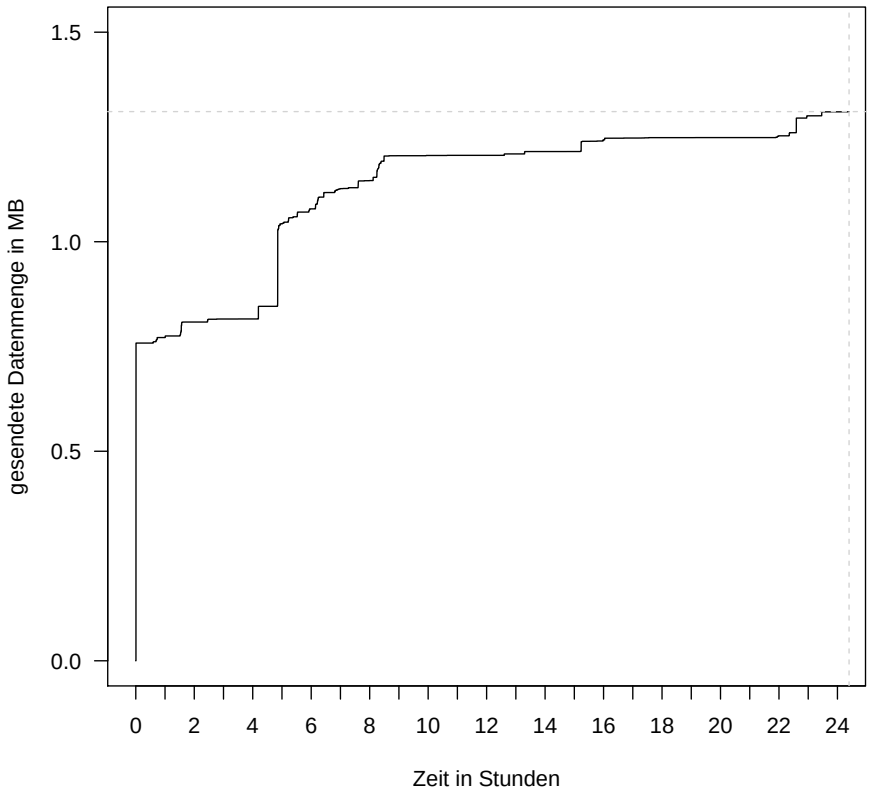

Abbildung 6.21: Darstellung der Menge gesendeter Daten für den *Facebook* Datensatz

Daten im Verlauf der Zeit zu sehen.

Auch bei den Netzwerkdaten lässt sich erkennen, dass die Hauptaktivität in den Zeiten stattfinden, in denen die Applikation genutzt wurde. Dabei erscheint die Gesamtmenge der gesendeten Daten gering, aber unter der Annahme, dass während der Datenerfassung nur Inhalte betrachtet wurden, kann die Datenmenge erklärt werden. Insbesondere wenn beachtet wird, dass die eventuelle Nutzung des *Facebook Messengers* nicht in den Daten auftaucht, da es sich um eine separate Applikation handelt. Denn in diesem Fall müssten nur kleine Nachrichten versendet werden um neue Inhalte anzufordern. Darüber hinaus kann bei circa 5 Stunden erkannt werden, dass eine größere Menge an Daten versendet wird. Dies ist ein Indiz dafür, dass zu diesem Zeitpunkt (einmalig) ein zu teilender Inhalt an *Facebook* übermittelt wurde.

Betrachtet man den Verlauf der empfangenen Daten fallen einige größere Sprünge auf. Dies ist ein Indiz dafür, dass zu diesen Zeitpunkten Medien - beispielsweise ein Bild - von *Facebook* empfangen und betrachtet wird. Die übrigen empfangenen Daten können damit erklärt werden, dass immer neue Inhalte empfangen werden, die dem Benutzer präsentiert werden sollen.

Es stehen keine näheren Informationen über die genaue Nutzung von *Facebook*, im Zeitraum der Datenerfassung, zur Verfügung. Aus diesem Grund ist es nicht möglich die Annahmen zu bestätigen oder zu widerlegen.

Es ist in den Daten sichtbar, dass zu Beginn der Aufzeichnung eine größere Menge an Daten gesendet bzw. empfangen wird. Dabei handelt es sich bei den empfangenen Daten wahrscheinlich um die neuen Inhalte, die *Facebook* dem Nutzer ausliefert, wogegen die gesendeten Daten eine Anmeldung und erste Datenanforderung enthalten. Dabei entspricht die Datenmenge, die zu Beginn empfangen wird circa 10% der Daten, die zu Beginn der Datenaufzeichnung abgespeichert werden. Es verbleibt somit eine große Menge an Daten, die zu Beginn der Aufnahme geschrieben werden und nicht mit den aufgestellten Vermutungen und erfassten Daten erklärt werden können.

Zusammenfassend lässt sich sagen, dass es möglich ist auf Basis der Daten einige Zusammenhänge herzustellen und diverse Vermutungen anzustellen. Dennoch war es nicht Möglich den Ursprung sämtlicher Daten mit Sicherheit festzustellen und anhand der Daten zu belegen.

Es lässt sich aus den erfassten Daten noch eine Schlussfolgerung ziehen, welche nicht das Verhalten von *Facebook* betrifft. Diese Schlussfolgerung ist auch nicht auf Basis dieses Datensatzes erfolgt, sondern aus der Gesamtheit aller erfassten Daten. Und zwar, dass die Erfassung von Netzwerkdaten funktionsfähig ist und plausible Daten erfasst. Im Rahmen der vorliegenden Arbeit wurden vier verschiedene Applikationen analysiert und alle diese Applikationen verwenden Netzwerkdaten in größeren Mengen. Bei drei der vier Applikationen wurden plausible Netzwerkdaten erfasst. Einzig bei der Aufzeichnung von *WhatsApp* hat die Erfassung von Netzwerkdaten versagt. Dies bestätigt die Annahme, dass es mindestens eine Möglichkeit zum Versenden und Empfangen von Netzwerkdaten gibt, welche durch das entwickelte Verfahren aktuell nicht erfasst wird.

7 Fazit

Abschließend sollen in diesem Kapitel die Ergebnisse der vorliegenden Arbeit zusammengefasst werden. Dabei werden Stärken und Schwächen nochmals herausgestellt. Im Anschluss daran wird ein Ausblick auf mögliche zukünftige Arbeiten, welche im Kontext dieser Arbeit angesiedelt sind, gegeben.

Das Ziel der Arbeit war es, ein generisches Verfahren zur Leistungsbewertung von *Android*-Smartphones zu entwickeln. Dieses Verfahren sollte auf einfache Art und Weise um zusätzliche Datensätze, welche zur Leistungsbewertung verwendet werden, erweitert werden können.

Dieses Ziel wurde im Rahmen der vorliegenden Arbeit erfüllt. Das Ergebnis der Entwicklung ist ein zweigeteiltes Verfahren. Der erste Teil des Verfahrens ermöglicht es, die Nutzung einer beliebigen Applikation aufzuzeichnen und für die spätere Nutzung zu persistieren. In dem aufgezeichneten Verhalten sind viele verschiedene Aspekte enthalten. Diese umfassen unter anderem durchgeführte Dateioperationen, Netzwerkverkehr sowie Informationen über den Lebenszyklus von *Activities*, welche ein zentrales Element von *Android*-Applikationen sind. Dabei wurde das Verfahren so entwickelt, dass es mit überschaubarem Aufwand um zusätzliche Datenquellen erweitert werden kann.

An dem entwickelten Verfahren zur Datenerfassung besteht zwar noch Optimierungsbedarf - beispielsweise bei der Vollständigkeit von erfassten Positionsdaten -, aber dennoch enthalten die aufgezeichneten Daten eine gute Darstellung des Verhaltens der beobachteten Applikation. Denn die erzeugten Datensätze enthalten ausreichend viele Datenquellen, um eine große Anzahl verschiedener Applikationen mit hoher Genauigkeit zu beschreiben. Trotzdem bestehen Einschränkungen bei der Erfassung von Positionsdaten und der Verwendung von *Wakelocks*. Die *Google Play Services* bieten eine Alternative zu der Schnittstelle welche das *Application Framework* bietet. Diese Alternativen Schnittstellen sind nicht quelloffen und können somit nicht für die Datenerfassung angepasst werden.

Die auf diese Weise erzeugten Datensätze werden aufbereitet und dienen im Anschluss dem zweiten Teil des Verfahrens als Eingabe. Hierbei handelt es sich um die eigentliche Leistungsbewertung, welche die Datensätze möglichst originalgetreu abspielt und so das ursprünglich aufgezeichnete Verhalten nachstellt. Der Vorgang der Wiedergabe ist selber in zwei Aktivitäten unterteilt. Im ersten Teil findet eine Vorverarbeitung der Daten statt, die den zweiten Teil, die eigentliche Wiedergabe,

vereinfacht, indem Zusammenhänge in den Daten extrahiert werden. Beispielsweise werden Netzwerkverbindungen identifiziert und gesendete bzw. empfangene Nachrichten diesen zugeordnet.

Dabei wurde im Rahmen der Evaluation eine hohe Genauigkeit für die Wiedergabe festgestellt. Darüber hinaus wurde auch festgestellt, dass es noch Aspekte gibt, die nicht fehlerfrei Arbeiten. Beispielsweise sei hier der Dienst genannt, welcher während der Wiedergabe als Sender und Empfänger für die Netzwerkkommunikation verwendet wird. Dieser Dienst bricht unter hoher Last zusammen und müsste dementsprechend im Hinblick auf die Robustheit verbessert werden.

Ebenfalls konnte während der Evaluation gezeigt werden, dass sich das Nutzungsverhalten, welches während der Datenerfassung genutzt wird, anschließend in den Daten wiedererkennen lässt. Darüber hinaus lässt sich erkennen, wie sich die beobachtete Applikation in verschiedenen Szenarien verhält. Somit sind die aufgezeichneten Daten auch geeignet um Analysen der erfassten Applikationen durchzuführen.

7.1 Ausblick

In diesem Abschnitt wird ein Überblick über einige mögliche Arbeiten gegeben, welche zukünftig durchgeführt werden können und im Kontext dieser Arbeit angesiedelt sind.

Zunächst sollten die, im aktuellen Stand vorhandenen, Mängel beseitigt werden, um die allgemeine Qualität des Verfahrens noch weiter zu erhöhen. So ist es beispielsweise notwendig, den Netzwerkdienst für die Wiedergabe zu überarbeiten, um die Robustheit zu erhöhen und die Genauigkeit, wann angeforderte Daten versendet werden, zu erhöhen. Dies würde es erlauben auch Datensätze abzuspielen, die eine intensive Nutzung von Netzwerkdaten aufweisen.

Ebenfalls sollte die Vorverarbeitung, welche vor der eigentlichen Wiedergabe stattfindet, nochmals überarbeitet werden, da hier offensichtlich nicht alle Dateioperationen richtig zugeordnet werden können, was zu einer ungenauen Wiedergabe führt. Eine Verbesserung in diesem Bereich wäre durchaus wichtig, da die Evaluation gezeigt hat, das die bisherige Problematik sowohl kleine Abweichungen, als auch die Wiedergabe eines vollständig anderen Verhaltens bewirken kann.

In der aktuellen Fassung besitzt das Verfahren zur Wiedergabe eine nicht ausreichende Genauigkeit beim erzeugen der Prozessorlast. Dabei wurde im Rahmen der vorliegenden Arbeit nur ein naives Verfahren verwendet, um diese Last zu erzeugen. Statt dieses Verfahren zu verbessern, bietet sich die Entwicklung eines neuen Verfahrens an, welche alle zur Verfügung stehenden Daten, wie beispielsweise die Taktfrequenz der einzelnen Kerne, verwendet. Hierdurch würde es möglich, dass

die Wiedergabe auf unterschiedlichen *Hardware*-Konfigurationen plausible Daten abspielt. Beispielsweise wenn das abspielende Gerät schneller bzw. langsamer als das aufzeichnende Gerät ist. Dies würde das aktuell verwendete Verfahren nicht unterstützen, selbst wenn die Genauigkeit verbessert wird.

Im Rahmen der Evaluation hat sich gezeigt, dass im Bereich des Netzwerkverkehrs Lücken existieren. Dies resultiert darin, dass für manche Applikationen nicht der gesamte Netzwerkverkehr aufgezeichnet wird. Diese Problematik sollte in jedem Fall behoben werden, da hierdurch eine mögliche Abweichung, unbekannter Größe, in den aufgezeichneten Daten vorliegt.

Neben diesen notwendigen Verbesserungen gibt es noch diverse Möglichkeiten zur Erweiterung des Verfahrens, welche die Nutzung vereinfachen oder andere Aspekte verbessern würden. Zum Beispiel besteht, in der aktuellen Fassung des Verfahrens, eine Hürde zur Nutzung. Dies liegt daran, dass eine speziell angepasste Version von *Android* notwendig ist, um Daten zu erfassen. Dabei muss diese Version für jedes Gerätemodell, mit dem Daten aufgezeichnet werden sollen, portiert werden. Dieser Aufwand lässt sich nicht vermeiden, wodurch die potentielle Nutzerbasis für die Datenerfassung stark eingeschränkt wird. Der Abspielvorgang dagegen kann von jedem beliebigen Gerät ausgeführt werden, sofern es die Anforderung an die *Android*-Version erfüllt. Aus diesem Grund bietet es sich an, dass aufgezeichnete Daten, welche für die Wiedergabe aufbereitet wurden, an einer zentralen Stelle angeboten werden und ggf. direkt aus der Applikation zur Wiedergabe heruntergeladen werden können. Hierdurch würde die Menge an Nutzern, die das entwickelte Verfahren zur Leistungsbewertung verwenden können, stark gesteigert und losgelöst von der Nutzermenge, die in der Lage ist neue Daten aufzuzeichnen.

In diesem Zusammenhang wäre es ebenfalls sinnvoll, wenn von dem aktuell verwendeten *XML*-Format für Daten auf ein binäres Format gewechselt wird. Die Datensätze für *WhatsApp*, die während der Evaluation verwendet wurden, umfassen einen Zeitraum von grob 24 Stunden und übersteigen teilweise die Größe von einem Gigabyte. Hier besteht ein Einsparungspotential durch die Verwendung von binären Daten. Außerdem würde das Lesen von binären Daten während der Wiedergabe eine geringere Last verursachen, als *XML*-Daten. Dies liegt daran, dass die Daten nicht zuerst aus einer Textrepräsentation in die eigentlichen Daten umgewandelt werden müssen, sondern direkt eingelesen werden können. Durch diese Entlastung würde die Genauigkeit im Bereich der Wiedergabe von Prozessorlast erhöht.

Die Qualität des Verfahrens könnte weiter gesteigert werden, indem weitere Datenquellen in die Aufnahme und die Wiedergabe einbezogen werden. Dabei stehen viele Datenquellen als potentielle Kandidaten zur Verfügung. Einige dieser Datenquellen wurden bereits im Rahmen der Anforderungsdefinition genannt, aber für die vorliegende Arbeit ausgeschlossen (siehe Kapitel 3.1).

Einen positiven Einfluss auf die aufgezeichneten Daten hätte es, wenn im Rahmen einer zukünftigen Arbeit erneut die Datenquellen für Positionsdaten und die Verwendung von *Wakelocks* betrachtet werden und die aktuell bestehenden Probleme gelöst werden. Für beide Datenquellen stehen mehrere Schnittstellen zur Verfügung, wobei nicht alle diese Schnittstellen quelloffen sind. Bei den Schnittstellen, für die kein Programmcode verfügbar ist, kann keine Instrumentierung zum Zwecke der Datenerfassung durchgeführt werden. Zum dem Zeitpunkt, als die vorliegende Arbeit erstellt wurde, gab es keine quelloffene Alternative, die den selben Funktionsumfang wie die proprietären Schnittstellen bietet. Sollte sich dies in der Zukunft ändern wäre es sinnvoll eine angepasste Version dieser quelloffenen Alternative zu verwenden, um die Erfassung aller Daten zu ermöglichen.

Literatur

[1] V. Aslot u. a. “SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance”. In: *OpenMP Shared Memory Parallel Programming: International Workshop on OpenMP Applications and Tools, WOMPAT 2001 West Lafayette, IN, USA, July 30–31, 2001 Proceedings*. Hrsg. von R. Eigenmann und M. J. Voss. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, S. 1–10. ISBN: 978-3-540-44587-6. DOI: 10.1007/3-540-44587-0_1. URL: http://dx.doi.org/10.1007/3-540-44587-0_1.

[2] J. Boyer. *Canonical XML version 1.0*. Techn. Ber. 2001.

[3] A. Carroll und G. Heiser. “An Analysis of Power Consumption in a Smartphone.” In: *USENIX annual technical conference*. Bd. 14. Boston, MA. 2010.

[4] C. Carvalho. “The gap between processor and memory speeds”. In: *Proc. of IEEE International Conference on Control and Automation*. 2002.

[5] J. Clark u. a. “Xsl transformations (xslt)”. In: *World Wide Web Consortium (W3C). URL http://www. w3. org/TR/xslt* (1999), S. 103.

[6] W. Cohen und D. Domingo. *Red Hat Enterprise Linux 6 SystemTap Tapset Reference*. 2010.

[7] D. Crockford. “The application/json media type for javascript object notation (json)”. In: (2006).

[8] A. Developers. *What is android*. Aufgerufen: 2016-09-26.

[9] K. M. Dixit. “New CPU benchmark suites from SPEC”. In: *Compcon Spring ’92. Thirty-Seventh IEEE Computer Society International Conference, Digest of Papers*. 1992, S. 305–310. DOI: 10.1109/CMPCON.1992.186729.

[10] F. C. Eigler u. a. *Architecture of systemtap: a Linux trace/probe tool*. 2005.

[11] H. Falaki u. a. “Diversity in Smartphone Usage”. In: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. MobiSys ’10. San Francisco, California, USA: ACM, 2010, S. 179–194. ISBN: 978-1-60558-985-5. DOI: 10.1145/1814433.1814453. URL: http://doi.acm.org/10.1145/1814433.1814453.

[12] *Gartner Says Worldwide Smartphone Sales Grew 3.9 Percent in First Quarter of 2016*. http://www.gartner.com/newsroom/id/3323017. Aufgerufen: 2016-10-04.

[13] A. Gattiker u. a. "Big Data text-oriented benchmark creation for Hadoop". In: *IBM Journal of Research and Development* 57.3/4 (2013), 10:1–10:6. ISSN: 0018-8646. DOI: 10.1147/JRD.2013.2240732.

[14] L. Goasduff und C. Pettey. *Gartner says worldwide smartphone sales soared in fourth quarter of 2011 with 47 percent growth*. http://www.pressebox.com/attachment/447933/Mobile+Devices+4Q11+FINAL+EMEA+version.pdf. Aufgerufen: 2016-09-26.

[15] J. Gosling u. a. *The Java Language Specification*. Pearson Education, 2014.

[16] H. Guihot u. a. *Pro Android Apps Performance Optimization*. Bd. 1. Springer, 2012.

[17] J. Haucap und U. Heimeshoff. "Google, Facebook, Amazon, eBay: Is the Internet driving competition or market monopolization?" In: *International Economics and Economic Policy* 11.1 (2014), S. 49–61. ISSN: 1612-4812. DOI: 10.1007/s10368-013-0247-6. URL: http://dx.doi.org/10.1007/s10368-013-0247-6.

[18] Y. Huang u. a. "Moby: A mobile benchmark suite for architectural simulators". In: *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. 2014, S. 45–54. DOI: 10.1109/ISPASS.2014.6844460.

[19] B. Jacob u. a. "SystemTap: instrumenting the Linux kernel for analyzing performance and functional problems". In: *IBM Redbook* (2008).

[20] J.-M. Kim und J.-S. Kim. "AndroBench: Benchmarking the Storage Performance of Android-Based Mobile Devices". In: *Frontiers in Computer Education*. Hrsg. von S. Sambath und E. Zhu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, S. 667–674. ISBN: 978-3-642-27552-4. DOI: 10.1007/978-3-642-27552-4_89. URL: http://dx.doi.org/10.1007/978-3-642-27552-4_89.

[21] M. H. Krieger u. a. "Urban Tomography". In: *Journal of Urban Technology* 17.2 (2010), S. 21–36. DOI: 10.1080/10630732.2010.515087. eprint: http://dx.doi.org/10.1080/10630732.2010.515087. URL: http://dx.doi.org/10.1080/10630732.2010.515087.

[22] B. aDam LeVenthaL. "Flash storage memory". In: *Communications of the ACM* 51.7 (2008), S. 47–51.

[23] C.-M. Lin u. a. “Benchmark dalvik and native code for android system”. In: *Innovations in Bio-inspired Computing and Applications (IBICA), 2011 Second International Conference on*. IEEE. 2011, S. 320–323.

[24] T. Lindholm u. a. *The Java Virtual Machine Specification: Java SE 8 Edition*. Pearson Education, 2014.

[25] M. Mouly und M.-B. Pautet. *The GSM System for Mobile Communications*. Telecom Publishing, 1992. ISBN: 0945592159.

[26] M. L. Murphy. *The busy coder's guide to Android development*. United States: CommonsWare, 2008., 2008.

[27] A. Patel. “CyanogenMod as an Aftermarket Firmware Replacement for Android”. In: *International Journal of Advanced Research in Computer and Communication Engineering* 5.2 (2016), S. 413–415.

[28] S. Phillips. “A brief history of Facebook”. In: *the Guardian* 25.7 (2007).

[29] V. Prasad u. a. “Locating system problems using dynamic instrumentation”. In: *2005 Ottawa Linux Symposium*. Citeseer. 2005, S. 49–64.

[30] F. Qian u. a. “Characterizing Radio Resource Allocation for 3G Networks”. In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. Melbourne, Australia: ACM, 2010, S. 137–150. ISBN: 978-1-4503-0483-2. DOI: 10.1145/1879141.1879159. URL: http://doi.acm.org/10.1145/1879141.1879159.

[31] M.-R. Ra u. a. “Energy-delay Tradeoffs in Smartphone Applications”. In: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. MobiSys '10. San Francisco, California, USA: ACM, 2010, S. 255–270. ISBN: 978-1-60558-985-5. DOI: 10.1145/1814433.1814459. URL: http://doi.acm.org/10.1145/1814433.1814459.

[32] B. Roch. “Monolithic kernel vs. Microkernel”. In: *TU Wien* (2004).

[33] R. Romans. “Red Hat Enterprise Linux 5.4 SystemTap Language Reference”. In: (2009).

[34] A. Shiravi u. a. “Toward developing a systematic approach to generate benchmark datasets for intrusion detection”. In: *Computers & Security* 31.3 (2012), S. 357–374.

[35] J. L. Stortz. *Audio power management*. US Patent 6,108,426. 2000.

[36] A. Traeger und E. Zadok. “How to cheat at benchmarking”. In: *USENIX FAST Birds of a feather session* (2009).

[37] R. P. Weicker. “Dhrystone: A Synthetic Systems Programming Benchmark”. In: *Commun. ACM* 27.10 (Okt. 1984), S. 1013–1030. ISSN: 0001-0782. DOI: 10.1145/358274.358283. URL: http://doi.acm.org/10.1145/358274.358283.

[38] C. Yoon u. a. “Appscope: Application energy metering framework for android smartphone using kernel activity monitoring”. In: *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*. 2012, S. 387–400.

Abbildungsverzeichnis

Tabellenverzeichnis

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem Titel

_____

_____

_____

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift