HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

HAPdroid

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2012

Autor(en):	Dominik Spengler, Remo Egli
Betreuer:	Prof. Eduard Glatz
Projektpartner:	ETH Zürich
Experte:	Roberto Pajetta
Gegenleser:	Prof. Dr. Andreas Rinkel

Erklärung der Selbstständigkeit

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe,
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Rapperswil, den 15. Juni 2012

Remo Egli

Dominik Spengler

Danksagung

An erster Stelle möchten wir uns bei Prof. Eduard Glatz bedanken, der diese Bachelorarbeit betreut und begleitet hat. Wir schätzen es sehr, dass er sich die Zeit genommen und uns mit vielen guten Ideen und konstruktivem Feedback unterstützt hat. Wenn wir uns in den Konzepten oder dem Code etwas “verrannt” hatten, konnte er uns mit seinem Erfahrungsschatz und grossen Fachwissen stets zurück auf die richtige Spur bringen.

Ebenfalls bedanken möchten wir uns bei Marco Pfiffner und Mathias Fasser, welche in ihrer Bachelorarbeit auch AndEngine für die graphische Oberfläche nutzen. Dieser Umstand, der damit verbundene Austausch und ihre Hilfsbereitschaft brachte uns in einigen Fragen bezüglich AndEngine schneller weiter und ersparte uns entsprechend Zeit.

Zu guter Letzt möchten wir unseren Freunden und Familien danken, welche uns stets unterstützten, auch wenn wir aufgrund der Arbeit weniger Zeit für sie hatten. Wenn es nötig war motivierten sie uns und wenn wir uns in einem Problem verrannt hatten, verhalfen sie uns mit einem kühlen Bier zur nötigen Ablenkung, um das Problem darauf frischen Mutes zu lösen.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Aufgabenstellung zur Bachelorarbeit FS 2012

„Network Traffic Inspector auf Android"

Gruppe: Remo Egli / Dominik Spengler

Ausgangssituation

HAPviewer (host application profile viewer) ist ein Open-Source Tool, das an der ETH Zürich für die verhaltensbasierte Analyse des Netzwerkverkehrs einzelner Rechner entwickelt wurde. Es benutzt eine graphenbasierte Darstellung, die sich an das Berkeley Socketmodell anlehnt, sowie Summarisierungs- und Filtertechniken, die eine kompakte Darstellung von hunderten oder gar tausenden von Netzwerkverbindungen ermöglicht [1, 2] .
Eine interessante Anwendung dieses Programms besteht darin, den Netzwerkverkehr des eigenen Rechners in einfach verständlicher Form dem Benutzer zu visualisieren, was ihm erlaubt z.B. verdächtige von harmlosen Verbindungen zu unterscheiden. Zurzeit ist dies auf die Betriebssysteme Linux und Unix begrenzt, wodurch HAPviewer nur für eine Minderheit der Rechnerbenutzer verwendbar ist.

Aufgabe

Das Ziel dieser Arbeit besteht darin HAPviewer auf Android zu portieren und mit zusätzlichen Informationen zu bestücken, die einem Benutzer helfen zu verstehen, welcher Art der Netzwerkverkehr ist, den sein Android-Gerät erzeugt. Diese Arbeit beinhaltet insbesondere eine sorgfältige Neugestaltung der grafischen Bedienoberfläche, die eine Verwendung auf einem Smartphone optimal unterstützt. Das GUI ist in Java zu entwickeln und über die JNI an den bestehenden C++ Code anzubinden.

Frühere Arbeiten

In [3] wurde ein Plugin für NFsen [4] realisiert, das die Visualisierungsidee des HAPviewers in ein mächtiges web-basiertes Monitoringsystem integriert. In [5] entstand die v2.0, die HAPviewer mit einer klickbaren Grafik und IPv6-Kompatibilität erweitert. Schliesslich in [6] wurde eine Zeitfensterfunktionalität sowie Algorithmen zur Unterrdückung wiederholter Flows und eine Geolokalisierung realisiert.

Berichtsgestaltung

Der Bericht ist gemäss [7] zu gestalten.

Referenzen

[1] Glatz. E.: Visualizing Host Traffic through Graphs, In Proceedings of the 7th International Symposium on Visualization of Cyber Security (VizSec'10), September 2010, pp. 58-63.

[2] Glatz. E.: HAPviewer - Host Application Profile Viewer, Source Forge Projekt, http://http://hapviewer.sourceforge.net/

[3] Schneider, R., Hügli, S.: Integration des HAPviewers in das NfSen Framework, Studienarbeit, HSR - Hochschule für Technik, Rapperswil, 2010.

[4] Haag, P.: NfSen - Netflow Sensor, SourceForge Projekt, URL: http://nfsen.sourceforge.net/

[5] Schneider, R., Hügli, S.: HAPviewer v2.0, Bachelorarbeit, HSR - Hochschule für Technik, Rapperswil, 2011.

[6] Raphael Blatter, Extending HAPviewer: Time Window, Flow Classification, and Geolocation, master's thesis, university of Zurich, 2011.

[7] Glatz, E., „Vorgaben zur Berichtserstellung", Ausgabe des 18. September 2011.

Termine

20. Feb. 2012	Arbeitsbeginn
15. Jun. 2012	Abgabe des Berichts an den Betreuer (bis 12:00)

Weitere Termine siehe Terminangaben auf dem HSR-Web (intern).

Betreuung

Betreuer: Prof. Eduard Glatz, Email: eglatz@hsr.ch
Während der Durchführung der Arbeit findet nach Möglichkeit regelmässig jede Woche eine Besprechung mit dem Betreuer statt. Dazu werden entsprechende Termine bei Arbeitsbeginn festgelegt.

Rapperswil, den 23. Feb. 2012

Eduard Glatz

Abstract

HAPviewer ist ein Tool zur Visualisierung von Netzwerkverkehr in Form eines Graphen. In dieser Bachelorarbeit galt es, die bestehende Software auf das Mobile-Betriebssystem Android zu portieren und den damit verbundenen Gegebenheiten anzupassen.

Aus HAPviewer ist die Kernkomponente in Form der C++ Library “libhapviz”, sowie die benötigte Library “Boost”, für das Zielsystem kompiliert worden. Damit die Library über JNI von unserer Android App “HAPdroid” genutzt werden konnte, wurde die gesamte Schnittstelle auf die Verwendung von Streams umgeschrieben, was die Flexibilität der Library bezüglich der Interprozess-Kommunikation (IPC) erhöht.

Für das Erfassen des Netzwerkverkehrs wurde “libpcap” verwendet, welche aufgrund der benötigten Root-Benutzerrechte über eine zusätzliche Schnittstelle “RootTools” angesteuert wird. RootTools stellt sicher, dass dem Programm alle benötigten Berechtigungen auf dem bereits “gerooteten” System gegeben werden.

Die erfassten Daten werden von der Service-Komponente unserer App aufbereitet und “libhapviz” zu Verarbeitung übergeben, wonach der Service die Ergebnisse in einer Graph-Datenstruktur abgelegt, welche mittels “jgrapht” erstellt wurde. Das grafische Benutzeroberfläche, umgesetzt mit “AndEngine”, kann die Datenstruktur dann mittels einer neuartigen, interaktiven Graphlet-Visualisierung darstellen. Aufgrund der Einschränkungen bezüglich Bildschirmgrösse auf Mobilgeräten, kann die Ansicht des Graphlets mittels Scrolling- und Pinch-To-Zoom-Methoden verändert werden.

Die App ist in ihrer Konzeption und Umsetzung auf Android noch einzigartig, weil die Datenquelle nur mit Betriebssystem-nahen Mitteln erreicht werden kann und die Daten dann komplett von der App, durch eine vielschichtige Komponenten-Architektur, verarbeitet und auch angezeigt werden.

Management Summary

1 Ausgangslage

HAPviewer (host application profile viewer) ist ein Open-Source Tool, das an der ETH Zürich für die verhaltensbasierte Analyse des Netzwerkverkehrs einzelner Rechner entwickelt wurde. Es benutzt eine Graphen basierte Darstellung, die sich an das Berkeley Socketmodell anlehnt, sowie Summarisierungs- und Filtertechniken, die eine kompakte Darstellung von hunderten oder gar tausenden von Netzwerkverbindungen ermöglicht [1].

Eine interessante Anwendung dieses Programms besteht darin, den Netzwerkverkehr des eigenen Rechners in einfach verständlicher Form dem Benutzer zu visualisieren, was ihm erlaubt z.B. verdächtige von harmlosen Verbindungen zu unterscheiden. Zurzeit ist dies auf die Betriebssysteme Linux und Unix begrenzt, wodurch HAPviewer nur für eine Minderheit der Rechnerbenutzer verwendbar ist. [Gla12]

2 Vorgehen, Technologie

Im Rahmen der Arbeit ist das HAPviewer Tool und die dazugehörige Programmbibliothek auf die Mobil-Plattform Android portiert, angepasst und erweitert worden. Insbesondere im Bereich der Inter-Prozess Kommunikation (IPC) wurden einige Verbesserungen implementiert, sodass zwischen den verschiedenen Applikations-Teilen und Schichten ein effizienter Datenaustausch möglich ist. Das Team hat die graphische Oberfläche überarbeitet und mit einem neu-entwickelten, interaktiven Graphlet versehen. Die Verwendung von AndEngine, einer Programmbibliothek für grafikintensive 2D-Spiele auf Android, ermöglicht es, die Ansicht des Graphlets mit gängigen Touch-Eingabemethoden, wie Scrollen und Pinch-to-Zoom, dynamisch zu verändern, um durch die Visualisierung zu navigieren. Die Symbolik und Farbcodierung wurde soweit wie möglich übernommen, um HAPviewer Benutzern trotz dem neuartigen Bedienkonzept den Einstieg zu vereinfachen. Bei der gesamten Entwicklung wurde darauf geachtet, den Anforderungen für Applikationen auf Mobilgeräten gerecht zu werden, speziell im Hinblick auf Performance und Benutzerfreundlichkeit.

1 http://hapviewer.sourceforge.net/

3 Ergebnisse

Die Portierung der HAPviewer Programmbibliothek verlangte vom Team eine eingehende Analyse des bestehenden Codes und erforderte aufwändige Änderungen. Parallel dazu wurde die Benutzeroberfläche entwickelt, welche aufgrund der nicht vorhandenen Dokumentation von AndEngine ebenfalls mehr Aufwand generierte als erwartet. Der starke Fokus auf eine solide Aufarbeitung der Basiskomponenten erwies sich jedoch als wegweisend, da es dem Team einen tiefen Einblick in die bestehende HAPviewer Applikation sowie in AndEngine ermöglichte. In Folge dessen konnten in der Schlussphase doch noch zusätzliche Features implementiert werden, welche das Bild für eine gute, erste Version einer ausbaufähigen App abrunden.

Struktur der Arbeit

Um dem Leser die Inhalte möglichst in einem logischen Zusammenhang zu präsentieren, haben wir uns entschieden die Arbeit aufzuteilen. Sie besteht nun aus einem technischen und einem projektspezifischen Teil, wobei Zusatzinhalte, welche nur einen partiellen Zusammenhang zu den beiden Teilen haben, im Anhang zu finden sind.

Der technische Bericht bietet, nach einer Einleitung in die Thematik, die Inhalte aufgeteilt nach den Kategorien Analyse, Design und Implementation. Die Ergebnisse der Arbeit werden dann in den Schlussfolgerungen, ebenfalls im technischen Teil, zusammengefasst und diskutiert. Mögliche Erweiterungen und Ideen für die weitere Entwicklung finden sich ebenfalls darin.

Im Projekt-Bericht ist der Projektplan enthalten, welcher den Verlauf des Projekts beschreibt. Ebenfalls im Projekt-Bericht befindet sich eine Beschreibung der verwendeten Entwicklungssoftware, sowie ein allgemeiner Projektrückblick und die jeweils persönlichen Erfahrungsberichte der Teammitglieder.

Weitere Informationen wie eine Anleitung zum Kompilieren der Arbeit finden sich im Anhang.

Inhaltsverzeichnis

Teil I

Technischer Bericht

1 Einleitung

Dieses Kapitel beschreibt die Probleme, welche es in der Arbeit zu lösen gilt, die Mittel die dafür zur Verfügung stehen und erläutert das Umfeld der Arbeit.

1.1 Problemstellung

Wie aus der Aufgabenstellung zu entnehmen ist, geht es darum eine bestehende Applikation, welche auf gängigen Computern eingesetzt werden kann, auf eine Mobile-Plattform zu portieren. Bereits dieser rudimentären Darstellung lässt sich entnehmen, dass es einige Probleme aufgrund der unterschiedlichen Plattform zu lösen gibt.

- Weniger leistungsfähige Hardware
  Obwohl moderne Mobilgeräte mittlerweile über eine beachtliche Leistung verfügen, kann diese nicht mit einem vollwertigen Computer mithalten. Die beschränkte Leistung der Hardware betrifft unser Projekt vor allem im Bezug auf:
  - CPU
    Je nach Menge der zu verarbeitenden Daten kann dies durchaus eine Rolle spielen. Wenn Netzwerkverkehr über längere Zeit aufgezeichnet wird, müssen mehr Daten reduziert werden. Ausserdem kann eine aufwändige Visualisierung das Gerät stark verlangsamen, da aktuelle Mobilgeräte erst in seltenen Fällen über eine separate GPU verfügen, welche der CPU die Berechnungen der graphischen Ausgabe abnehmen.
  - Arbeitsspeicher
    Um die Daten reduzieren, filtern und summarisieren zu können, muss ein nennenswerter Teil davon während der ganzen Berechnungsdauer im Arbeitsspeicher gehalten werden. Übersteigt die Datenmenge die zur Verfügung stehenden Ressourcen, können die Berechnungen schlecht bis gar nicht durchgeführt werden.
  - Datenspeicher
    Zeichnet man den Netzwerkverkehr vollständig auf, so sammeln sich über kurze Zeit unzählige Megabytes Daten an. Sollte das Mobil-Gerät nicht über eine entsprechend grossen Speicher verfügen, wäre dieser in Kürze aufgebraucht.
- Andere Ein- & Ausgabemethoden
  Mit dem Formfaktor ändert sich auch die Bedienung des Geräts. Dieser Tatsache müssen wir im speziellen Rechnung tragen, da diese Faktoren nicht einfach durch den Kauf eines leistungsfähigeren Geräts verbessert werden können.

  - Dateneingabe & Bedienung
    Aufgrund der geringen Grösse der meisten Mobil-Geräte sollten umständliche Eingaben von Daten oder mühsame Bedienabläufe vermieden werden. Sie kosten mehr Zeit als beispielsweise auf einer vollwertigen Tastatur und verschlechtern die Benutzerfreundlichkeit.
  - Ausgabe & Darstellung
    Die wohl einschneidendste Änderung durch die Portierung ist die geringe Bildschirmgrösse der Mobilgeräte. Die graphische Benutzeroberfläche kann zwar in Anlehnung an die bestehenden Konzepte wie z.B. Farbcodierung entwickelt, aber kaum übernommen werden.

- Andere Konzepte & Anwendungsbereiche
  HAPviewer kann sämtlichen Netzwerkverkehr verarbeiten, welcher dem Programm über Dateiimport oder die Netzwerkschnittstellen zugeführt wird und Graphlets für alle beteiligten Kommunikationspartner erstellen, wenn nur genügend Daten vorhanden sind. Da Mobil-Geräte jedoch hauptsächlich über drahtlose Schnittstellen kommunizieren, wird das Aufzeichnen von Netzwerkverkehr stark eingeschränkt, auf jenen Verkehr des eigenen Geräts. Somit kann eine der wichtigen Stärken von HAPviewer auf Android nicht genutzt werden, was die Frage aufwirft, welches den die konkreten Anwendungsbereiche für die zu portierte Software sein soll.

1.2 Bisherige Arbeiten

Zum HAPviewer Projekt von Professor Glatz gibt es bereits Arbeiten die im Rahmen einer Semester-, Bachelor- oder Master-Arbeit entstanden sind. Hier ein kurzer Überblick der dort behandelten Inhalte.

1.2.1 “Visualizing Host Traffic through Graphs”

[Gla10] Dies ist die ursprüngliche Arbeit, welche die Konzepte der ersten HAPviewer-Version erläutert. Professor Glatz beschreibt in diesem Whitepaper die Ideen, wie Netzwerkdaten verarbeitet und mittels eines Graphen möglichst verständlich dargestellt werden können. Nahezu alle Ideen und Konzepte aus dem Dokument sind essentiell wichtig für das Verständnis der aktuellen Arbeit. Wir gehen davon aus, dass der Leser die Inhalte des Dokuments gelesen und verstanden hat, weshalb wir die dort enthaltenen Ideen und Konzepte in der aktuellen Arbeit nicht nochmals wiedergeben werden.

1.2.2 “Integration des HAPviewers in das NfSen Framework”

[**?**] Um die HAPviewer-Software einem grösseren Publikum zugänglich zu machen, ist in einer Semesterarbeit an der HSR eine Schnittstelle zum NfSen Framework entwickelt worden. In dieser Arbeit wird erstmals beabsichtigt, das Programm nicht nur als eigenständige Version zu entwickeln, sondern zusätzlich über eine Library in andere Produkte zu integrieren. Dies betrifft die aktuelle Arbeit insofern, als dass auch wir nur die Kernkomponenten ohne GUI-Logik übernehmen können, ergo eine Schnittstelle benötigen.

1.2.3 “HAPviewer v2.0”

[**?**] Die Fortsetzung der Semesterarbeit wurde von den selben Studenten in Form einer Bachelorarbeit realisiert. “HAPviewer v2.0” fokussiert sich vor allem auf die Unterstützung des IPv6-Protokolls, die Erweiterung des GUI um interaktive Elemente und der Verbesserung des Codes sowie der verwendeten, externen Komponenten. Aus diesem Dokument greifen wir hauptsächlich die Bedienkonzepte auf und mussten uns zwangsläufig mit dem neu eingeführten Build-System vertraut machen.

1.3 Konkurrenz

Von der Analyse bestehender Android Applikationen im Themenbereich der Arbeit erhofften wir uns in der Einarbeitungsphase erste Einblicke wie unser Produkt schlussendlich aussehen könnte. Wir waren jedoch etwas enttäuscht, dass wir, auch bis zum Ende der Arbeit, keine Apps gefunden haben, welche auch nur ansatzweise ähnlich in Funktionalität oder Komplexität sind. Gefunden haben wir vor allem Apps welche statistische Auswertungen vom Datenverkehr des Gerätes anfertigen und diese dann entsprechend visualisieren. Aufgrund der Vielzahl solcher Apps verzichten wir auf eine Auflistung. Die einzige nennenswerte App heisst “Connection Tracker” [1] und stellt einen Teil der Informationen dar, welche auch wir benötigen (z.B. source und remote IP, dazugehörige App, etc.), da wir jedoch kein öffentliches Repository dazu gefunden haben, half uns diese Entdeckung nicht weiter.

Wesentlich weniger ergiebig war die Suche nach Apps, welche Netzwerkverkehr aufzeichnen oder in irgend einer Form verarbeiten. Im Android Market (neu “Google Play Store”) sind wir auf folgende Programme gestossen, welche im wesentlichen Sniffer-Programme sind: “NetSpector Sniffer/Ad Blocker”, “Packet Sniffer”, “Shark for Root”. Da das Aufzeichnen des Netzwerkverkehrs nicht einfach ein Feature sondern eigentlich eine grundlegende Anforderung/Voraussetzung für unsere App ist, haben wir uns nicht weiter mit diesen Programmen auseinander gesetzt, da auch zu keinem genannten App der Quellcode zugänglich war.

Mit der Absicht, Beispiele für den Zugriff auf die Netzwerkschnittstellen unter Android zu finden sind wir noch auf die Apps “Fing - Network Tools” und “Firebind” gestossen, welches im wesentlichen Netzwerk-Scanner sind, jedoch ebenfalls nicht Open-Source.

Um uns den Einstieg in die Verwendung von JNI etwas zu vereinfachen haben wir den Quellcode der App “OS Monitor” [2] [3] analysiert. Dieses Programm ist nicht nur sehr gut geschrieben, sondern bietet auch den besten Umfang an Features für ein Systemmonitoring-Tool, welche wir in der ganzen Suche gefunden haben.

1 https://play.google.com/store/apps/details?id=com.borgshell.connectiontrackerfree
2 http://code.google.com/p/android-os-monitor/
3 https://play.google.com/store/apps/details?id=com.eolwral.osmonitor

1.4 Verwendete Technologien

1.4.1 Android

Wir haben uns für Android Froyo (2.2) als Entwicklungsversion entschieden. Diese Wahl wurde aufgrund der Verteilung der Android Versionen zu beginn der Arbeit getroffen.

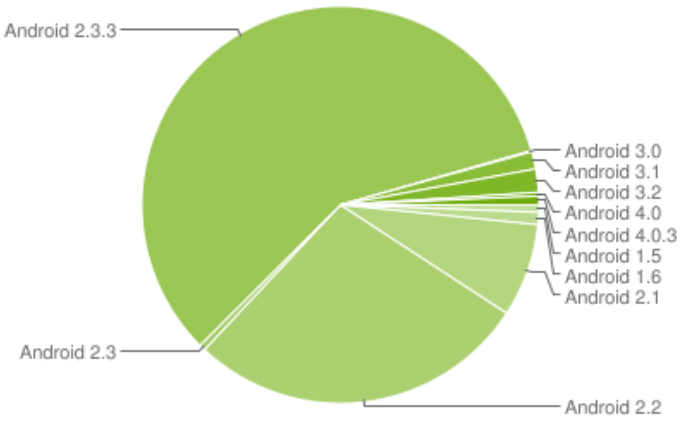

Abbildung 1.1: Android Versionsverteilung (Stand 01.02.2012)

1.4.2 libpcap

Pcap ist eine Programmierschnittstelle (API) welche es Programmen erlaubt Netzwerkverkehr mitzuschneiden. Für unsere Applikation haben wir uns entschieden eine eigene, reduzierte

Platform	Codename	API Level	Verteilung
Android 1.5	Cupcake	3	0.6%
Android 1.6	Donut	4	1.0%
Android 2.1	Eclair	7	7.6%
Android 2.2	Froyo	8	27.8%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.5%
Android 2.3.3 - Android 2.3.7	Gingerbread	10	58.1%
Android 3.0	Honeycomb	11	0.1%
Android 3.1	Honeycomb	12	1.4%
Android 3.2	Honeycomb	13	1.9%
Android 4.0 - Android 4.0.2	Ice Cream Sandwich	14	0.3%
Android 4.0.3	Ice Cream Sandwich	15	0.7%

Tabelle 1.1: Android Versionsverteilung (Stand 01.02.2012)

Version von tcpdump[1] zu implementieren, siehe Analyse. Für diesen Zweck nutzen wir libpcap für den Zugriff auf die Netzwerkschnittstellen. Die pcap Bibliothek bietet sich durch die stetige Entwicklung und Verwendung in bekannten Open Source Projekten wie zum Beispiel tcpdump, Wireshark, Nmap und Snort für diesen Verwendungszweck an.

1.4.3 Boost

Boost [2] (englisch Boost C++ Libraries) ist eine freie C++-Bibliothek bestehend aus einer Vielzahl von portablen Unterbibliotheken. Die Unterbibliotheken dienen unterschiedlichsten Aufgaben von Algorithmen auf Graphen über Metaprogrammierung bis hin zu Speicherverwaltung. [Wik12] Libhapviz nutzt Boost, womit dies eine externe Abhängigkeit für uns darstellt.

1.4.4 AndEngine

AndEngine [3] ist eine freie 2D OpenGL Spiele-Engine für Android.

1.4.5 JGraphT

JGraphT [4] ist eine Library von Datenstrukturen und Algorithmen für Graphen.

1.4.6 roottools

RootTools [5] stellt Android-Entwicklern eine Schnittstelle für das Ansteuern von Applikationen, welche Root-Rechte benötigen, zur Verfügung.

1.4.7 aFileChooser

Afilechooser [6] ist eine Schnittstelle für Android zur Dateiauswahl.

1 http://www.tcpdump.org/
2 http://www.boost.org/
3 http://www.andengine.org/
4 http://jgrapht.org/
5 http://code.google.com/p/roottools/
6 http://code.google.com/p/afilechooser/

2 Anforderungsspezifikation

2.1 Funktionale Anforderungen (User Stories)

Die Funktionalen Anforderungen werden in Form von User Stories, wie sie in Scrum verwendet werden, beschrieben. Die Beschreibung beinhaltet eine detailliertere Zielformulierung, den beabsichtigten Zweck/Hintergrund, sowie eine Prioritäten-Skala von 1 (unwichtig) bis 9 (kritisch) und einer entsprechenden Aufwandsschätzung.

“Als Benutzer kann ich Netzwerkverkehr anschauen”

Ziel: Der Benutzer kann den Netzwerkverkehr des Mobilgerätes in einer einfachen Textausgabe lesen und die Ausgabe mit einem Start/Stop-Button beeinflussen.

Zweck: Diese wichtige Anforderung soll sicherstellen, dass die Applikation den Netzwerkverkehr auslesen und in einfacher Form durch alle Software- und Betriebsystem-Schichten übergeben kann, da diese Fähigkeit grundlegend für den weiteren Verlauf der Entwicklung ist. Die Anzeige der Daten ist noch sekundär, soll jedoch möglichst früh implementiert werden, um den Entwicklungsfortschritt sichtbar zu machen.

Priorität: 9

Aufwandsschätzung: 20h

“Als Benutzer kann ich Netzwerkverkehr im Hintergrund aufzeichnen”

Ziel: Der Benutzer kann die Aufzeichnung des Netzwerkverkehrs starten und dass Programm-Anzeige verlassen, ohne dass die Aufzeichnung unterbrochen wird. Er kann dann zur Anzeige zurückkehren und die Aufzeichnung abschliessen.

Zweck: Diese User Story dient dem Zweck eine gute Architektur der Kern-Komponenten möglichst zu Beginn der Arbeit zu finden, zu klären ob ein Android Service genutzt werden soll/kann und es sinnvoll ist.

Priorität: 7

Aufwandsschätzung: 13h

“Als Benutzer kann ich Netzwerkverkehr nach Kategorien unterscheiden”

Ziel: Der Benutzer sieht nicht mehr nur die einfache Textausgabe, wie in der zuvor beschriebenen User Story, sondern eine geparste Ausgabe der Daten, nach den fünf Kategorien, bzw. Felder des Berkeley Socket-Modells.

Zweck: Um die Daten später in einem Graphlet nutzen zu können, müssen diese erst geparst werden, sodass unsere Applikation die Daten entsprechend einordnen kann.

Priorität: 8

Aufwandsschätzung: 13h

“Als Benutzer kann ich Netzwerkverkehr als Graph anzeigen”

Ziel: Die Daten werden für den Benutzer nun erstmals in einer graphischen Darstellung angezeigt, wenn auch noch in statischer Form.

Zweck: Um diese Anforderung zu erfüllen, muss evaluiert werden, welche Möglichkeiten für die grafische Oberfläche zur Auswahl stehen, welche Programmbibliotheken, welche Aufbau- und Darstellungsmethoden. Es gilt die Richtung für die Entwicklung des restlichen GUIs auszuarbeiten und letztendlich zu bestimmen.

Priorität: 7

Aufwandsschätzung: 13h

“Als Benutzer kann ich sinnvoll durch den Netzwerk Graph navigieren”

Ziel: Der Benutzer kann durch den Graphen navigieren und die Ansicht seinen Bedürfnissen anpassen, sollten nicht alle Informationen gleichzeitig auf dem Bildschirm ersichtlich sein.

Zweck: Wir nehmen an, dass bei grösserer Datenmenge der Graph nicht mehr vollständig auf dem Bildschirm des Mobil-Gerätes angezeigt werden kann. Aus diesem Grund gilt es ein Konzept zu entwickeln und umzusetzen, welches es dem Benutzer ermöglicht, die Ansicht des Graphen so zu ändern, dass alle für ihn relevanten Informationen ersichtlich sind.

Priorität: 7

Aufwandsschätzung: 20h

“Als Benutzer kann ich Netzwerkverkehr abspeichern”

Ziel: Der Benutzer kann die aufgezeichneten Daten für spätere Verwendung abspeichern.

Zweck: Zusammen mit der folgenden User Story soll eine Export/Import-Funktion implementiert werden.

Priorität: 2

Aufwandsschätzung: 8h

“Als Benutzer kann ich gespeicherten Datenverkehr öffnen”

Ziel: Der Benutzer kann die zuvor gespeicherten Daten wieder öffnen und das Graphlet erneut anzeigen lassen.

Zweck: Diese User Story dient nicht nur dem Ziel eine vollständige Export/Import-Funktion anzubieten, sondern soll zudem ermöglichen, auch Daten welche mit dem HAPviewer erfasst wurden auf dem Mobil-Gerät anzuzeigen.

Priorität: 2

Aufwandsschätzung: 8h

“Als Benutzer kann ich Netzwerkverkehr in Transaktions-Darstellung anschauen”

Ziel: Der Benutzer kann die einzelnen Transaktionen, die eindeutigen Pfade von links nach rechts durch den Graph, einzeln in einer Textansicht ausgeben lassen.

Zweck: Dies ist ein Zwischenschritt welcher benötigt wird, um die später beschriebene User Story “Als Benutzer kann ich den Netzwerkverkehr nach Applikation sortieren” erfüllen zu können. Diese benötigt einen Mechanismus, der die aufgezeichneten Daten mit derjenigen Applikation verknüpfen kann, welche diese generiert oder empfangen hat.

Priorität: 8

Aufwandsschätzung: 20h

“Als Benutzer kann ich Netzwerkverkehr nach der dafür verantwortlichen Applikation sortieren”

Ziel: Das Graphlet zeigt dem Benutzer auch die involvierten Applikationen an, welche den jeweiligen Netzwerkverkehr generiert oder empfangen haben.

Zweck: Die zusätzlichen Informationen werden zum bestehenden Graphlet hinzugefügt, um für die Android-Version des HAPviewer eine Funktion mit sinnvollem Mehrwert zu bieten.

Priorität: 7

Aufwandsschätzung: 40h

2.2 Nicht-Funktionale Anforderungen

2.2.1 Usability

- Lesbarkeit
  Die Beschriftungen und Informationen, welche im Graphlet dargestellt werden, sollen lesbar sein.
- Intuitive Navigation
  Zur Navigation im Graphlet sollen bereits verbreitete Methoden verwendet werden, wie z.B. Scrolling per Touch-Eingabe.
- Informationen zu Hintergrund-Prozessen
  Kann die Benutzeroberfläche während längeren Berechnungen keine Informationen anzeigen, soll dem Benutzer zum Verständnis eine entsprechende Warte-Animation gezeigt werden.
- Abbruch von langen Hintergrund-Berechnungen
  Berechnungen welche länger als 5 Sekunden dauern sollen abgebrochen werden können.

2.2.2 Zuverlässigkeit

- Korrektheit der Daten
  Die Angaben im Graphlet, wie auch das Graphlet selbst sollen den aufgezeichneten Netzwerkverkehr korrekt darstellen.

2.2.3 Performance

- GUI Responsiveness
  Die Benutzeroberfläche soll bei Graphlets mit bis zu 60 Knoten oder 60 Edges keine wahrnehmbaren Verzögerungen aufweisen.
- Sinnvolle Verwendung der Ressourcen (RAM & Storage)
  Die Programmierung der App soll möglichst Ressourcen-schonend konzipiert sein.

2.2.4 Wartbarkeit

- Konfigurierbarkeit
  Fixe Parameter, welche während der Entwicklung festgelegt werden, sollen an möglichst zentraler Stelle im Quellcode geändert werden können. (z.B. Schriftgrösse, Qualität der Objekt-Renderings, etc.)

2.2.5 Randbedingungen

- libhapviz
  Die Library von HAPviewer soll so eingebunden werden, dass die Schnittstelle möglichst einfach zu handhaben ist.
- Android (siehe Analyse)
- OpenSource
  Die Software soll als OpenSource Projekt entwickelt werden.

3 Analyse

Die Analyse der Anforderungen ergab eine Aufteilung in die in der folgenden Grafik dargestellten Schritte.

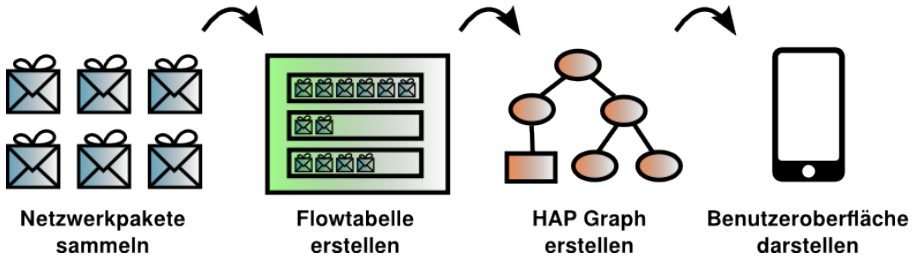

Abbildung 3.1: Übersicht der analysierten Schritte

In diesem Kapitel wird jeder einzelne Schritt kurz beschrieben und analysiert.

3.1 Netzwerkpakete sammeln

Grundanforderung des Projektes war es, den Netzwerk Verkehr des Android Gerätes erfassen zu können. Android bietet aber keine offizielle Schnittstelle für diese Anforderung im SDK da dies eine mögliche Sicherheitslücke ist und in der Regel auch nicht benötigt wird. Nach kurzer Recherche fanden wir heraus dass tcpdump [1] und libpcap [2] für Android existiert. Damit konnten wir auf bewährte Programme mit gutem Community-Support zurückgreifen.

tcpdump ist ein kleines Kommandozeilentool welches für die Darstellung des Netzwerkverkehrs benutzt wird.

libpcap ist eine Bibliothek für das Mitschneiden des Netzwerk Datenverkehres. Eine Vielzahl populärer Netzwerktools, wie auch tcpdump, nutzt libpcap für die Sammlung der Netzwerkdaten.

Ebenfalls existiert mit Jpcap [3] eine Java Portierung von libpcap.

Es existieren also die folgenden Möglichkeiten um den Netzwerkdatenverkehr mit zuschneiden: tcpdump als externe ausführbare Datei nutzen, libpcap oder jpcap als externe Bibliothek

1 http://www.tcpdump.org/tcpdump_man.html
2 http://www.tcpdump.org/pcap3_man.html
3 http://jpcap.sourceforge.net/

nutzen und die gesammelten Daten zur weiteren Verarbeitung an die Kernapplikation weiterzugeben.

Bei dem Versuch Jpcap für das Mitschneiden zu nutzen stellte sich allerdings ein Berechtigungsproblem.

3.1.1 Berechtigung

Der Zugriff auf die Netzwerkdaten eines Android Gerätes standardmässig nur über die offiziell verfügbaren API Schnittstellen möglich. Diese erlauben aber kein Applikationsübergreifendes mitschneiden des Netzwerkverkehrs. Um alle Netzwerkpakete mitschneiden zu können muss somit direkt auf die Netzwerkschnittstelle zugegriffen werden. Die UID unter dem der Applikationsprozess läuft besitzt aber nicht genügend Rechte um auf die Netzwerkschnittstellen zuzugreifen. Um dennoch alle Pakete mitschneiden zu können werden also erweiterte Berechtigungen auf das Telefon benötigt.

Klassisch gibt es in Linux, somit auch in Android, drei Möglichkeiten um die Berechtigung zu erweitern:

Setuid: Durch das setuid Attribut wird einer ausführbare Datei erlaubt, die Rechte des Besitzers der Datei anzunehmen. Die Datei bleibt ausführbar gemäss den Zugriffsrechten des Dateisystems wird aber an bestimmten Programmstellen mit den Rechten des Besitzers der Datei ausgeführt.

Separater Prozess: Mittels eines separaten Prozesses, welcher als Superuser läuft, kann auf die notwendige Ressource zugegriffen werden und die erhaltenen Informationen mittels IPC dem Grundprozess zur Verfügung stellen.

Berechtigung erweitern: Durch die Erweiterung der Berechtigung für die Ressource wird es der Applikation ermöglicht auf die Netzwerkschnittstelle zuzugreifen.

Idealerweise implementiert man einen separaten Prozess welcher zusätzlich mittels setuid nur in den absolut benötigten Programmstellen als privilegierter Benutzer läuft.

Für unser Programm haben wir uns für einen separaten Prozess ohne setuid entschieden. Ein separates Programm ermöglicht es auch die gesammelten Netzwerkdaten mit weiteren Informationen, wie zum Beispiel welcher Prozess die Verbindung aufgebaut hat, zu erweitern. Da das C-Programm ausser der Netzwerk-Datensammlung und mögliche Erweiterung dieser gesammelten Daten keine weiteren Aufgaben hat, würde die zusätzliche Komplexität des Ansatzes mittels setuid nur minimal mehr Sicherheit bieten. Für das Programm nutzen wir libpcap als Bibliothek welche das Mitschneiden der Pakete deutlich vereinfacht.

Es wäre war ebenfalls möglich tcpdump als externen Prozess zu starten und die benötigten Netzwerkdaten so mit zuschneiden. Bei diesem Ansatz können die Pakete aber kaum mit den Informationen zu den Ursprungsprozessen bestückt werden. Zusätzlich werden bei tcpdump für unsere Zwecke zu viele Informationen mitgeschnitten und führt zu der Problematik der Interprozesskommunikation.

3.1.2 Interprozesskommunikation

Weil jedes empfangene Paket Prozess-übergreifend weitergegeben werden muss muss ein stetiger Informationsfluss zwischen dem Paket-capture Prozess und der Flowtabelle gewährleistet werden. Klassisch existieren für die kontinuierliche Übergabe von Daten zwischen verschiedenen Prozessen folgende Möglichkeiten:

Datei: Dies ist die einfachste Art der Informationsübertragung zwischen verschiedenen Prozessen. Es wird vom ersten Prozess eine Datei erstellt welche vom zweiten Prozess gelesen wird.

Pipe: Bei einer Pipe wird der output eines Prozesses an den input eines anderen Prozesses weitergeleitet. So wird ein asynchroner Kommunikationsmechanismus zur Verfügung gestellt. Eine Pipe ist änonymünd ist an den Lebenszyklus des erstellenden Prozesses gebunden.

Named Pipe: Eine Named Pipe stellt eine Erweiterung der Pipe dar. Sie unterscheidet sich von der klassischen Pipe dadurch dass eine Named Pipe einen Namen besitzt und somit nicht anonym ist. Dadurch ist eine Named Pipe nicht an den Lebenszyklus des erstellenden Prozesses gebunden und muss manuell gelöscht werden wenn sie nicht mehr benötigt wird.

Shared Memory: Bei dieser Methode wird ein Bereich im Speicher für mehrere Prozesse zur Verfügung gestellt. Es ist der schnellste aber auch der komplexeste Ansatz.

Socket: Ähnlich zur Named Pipe stellt ein Socket einen asynchronen Kommunikationsmechanismus zur Verfügung.

Da es sich in unserem Fall um einen kontinuierlichen Datenfluss handelt macht die Kommunikation über eine Datei kaum Sinn. Pipe, Named Pipe und Socket können in Java alle mittels InputStream genutzt werden und stellen somit aus Applikationssicht keinen Unterschied dar.

Android bietet mit LocalServerSocket [1] bereits eine Klasse welche für Interprozesskommunikation gemacht ist und auch über native code angesprochen werden kann. Für die Anbindung der HAPviz Bibliothek haben wir uns für die Kommunikation über einen lokalen Server Socket entschieden während die für die ausführbare Datei eine Art Pipe über die Kommandozielen Ausgabe verwendet wird. Für genauere Informationen zu den Designentscheiden wird auf Design verwiesen.

3.2 Flowtabelle erstellen

Ein Netzwerkflow repräsentiert eine Sequenz von Netzwerkpaketen von einem Ursprungsrechner zu einem Zielrechner. Ein Netzwerkflow wird durch die folgenden fünf Werte identifiziert:

1 http://developer.android.com/reference/android/net/LocalServerSocket.html

- Quell IP Adresse
- Layer 4 Protokoll
- Quell Port
- Ziel Port
- Ziel IP Adresse

So werden Netzwerk Pakete welche zur selben logischen Netzwerkverbindung gehören zu einem Flow gruppiert.

Im Netzwerk Flow Bereich gibt es bereits einige bisherige Arbeiten. Leider sind bisherige frei verfügbare Netzwerkflow Generierung Tools nicht auf eine stetige Generierung der Flowdaten ausgelegt, sondern nehmen eine Capture Datei als input. Dies ist nicht Ideal für unsere Applikation da einerseits das Mitschneiden und zwischenspeichern des gesamten Datenverkehrs einen unnötigen Overhead generiert, andererseits die aktuellen Daten möglicherweise von Interesse für den Benutzer sind. Ebenfalls ist es für die weitere Entwicklung der Arbeit äusserst nützlich einfachen Zugriff auf die zugrunde liegenden Flows zu haben. So lässt sich beispielsweise ein Kontextmenü der einzelnen Nodes des Graphen deutlich einfacher Implementieren.

Folglich haben wir uns bei der Applikation entschieden eine Flowtabelle zu führen welche stetig mit den aktuellen Daten angepasst wird.

3.3 HAP Graph erstellen

Teil des HAPviewer ist eine Bibliothek namens libhapviz. Zu Anfang haben wir geplant diese Bibliothek für die Generierung des HAP Graphen zu verwenden und somit die bestehende Logik wiederzuverwenden. Einige Probleme führten jedoch dazu dass wir diese Bibliothek nicht verwenden konnten.

3.3.1 Schnittstelle

So wie die libhapviz Bibliothek zur Version 2.0 besteht ist die Schnittstelle äusserst unflexibel und für unsere Zwecke unbrauchbar. Die Schnittstelle definiert eine Klasse mit folgender Signatur:

Codelisting 3.1: libhapviz Schnittstelle

```
class CInterface {
  private:
    CImport * flowImport; ///< Ref to data for HOST list model
    ChpgData * hpgData; ///< Data for HPG model
    prefs_t prefs; ///< Preferences settings
```

```
  public:
    CInterface();
    ~CInterface();

    // Assign powers of two for OR-ing of option values
    enum summarize_flags_t {
      summarize_client_roles = 1, summarize_multi_client_roles = 2,
          summarize_server_roles = 4, summarize_p2p_roles = 8,
          summarize_all = (1 + 2 + 4 + 8)
    };

    enum filter_flags_t {
      filter_biflows = 1, filter_uniflows = 2, filter_tcp = 4, filter_udp =
           8, filter_icmp = 16, filter_other = 32
    };

    bool get_graphlet(std::string in_filename, std::string & outfile, std::
        string IP_str, summarize_flags_t summarize_flags, filter_flags_t
        filter_flags,
          const std::set<uint32_t> & desum_role_nums);
    bool get_hpg_file(std::string in_filename, std::string & outfile,
        IPv6_addr localIP, int host_count);

  private:
    bool handle_get_graphlet(std::string & in_filename, std::string &
        hpg_filename, std::string & dot_filename, std::string IP_str);
    bool handle_hpg_import(std::string & in_filename, std::string &
        out_filename);
    bool handle_binary_import(std::string & in_filename, std::string &
        out_filename, IPv6_addr localIP, int host_count);
    CSummaryNodeInfos* nodeInfos; ///< Storage for nodeid filter (needed by
         HAP4NfSen)
    desummarizedRoles desum_role_nums; ///< Desummarized role number
};
```

In der Bibliothek werden zwei öffentliche Methoden definiert, welche eine Eingabedatei, eine Ausgabedatei und weitere Filter und Summierungsoptionen als Argumente nehmen. Dies ist denkbar schlecht weil es einerseits sehr unflexibel ist, andererseits lediglich graphviz .dot Dateien und ein eigenes .hpg Format unterstützt werden. Ein Graph-Format ist für den Anwendungsbereich der Applikation aber sehr ungünstig, da die Informationen zu den Ursprünglichen Flows ebenfalls von grossem Interesse sind. Bei Verwendung der Bibliothek müsste für jede noch so kleine Änderung die Berechnung des gesamten Graphen von neuem durchgeführt werden und das resultierende Graph-Format müsste erneut analysiert werden. Es macht aus Performancegründen wenig Sinn, dass für jede erneute Generierung des HAP Graphen alle Informationen in eine Datei gespeichert und gelesen, in ein Graph-Format umgewandelt, erneut in eine Datei gespeichert und gelesen, das Graph-Format wieder analysiert werden müssen.

Die Zwischenspeichern in Dateien kann durch die Nutzung von C++ streams umgangen werden. Allerdings lösen die C++ streams nicht das Problem des Kontextverlustes.

3.3.2 Kontextverlust

Durch die Generierung des HAP Graphen in ein Graph-Format wird zwangsläufig der Kontext der Knoten zu den benutzten Netzwerkflows verloren gegangen. Ohne diese Informationen ist der Anwendungsbereich einer mobilen Applikation stark eingeschränkt. So zum Beispiel lassen sich nur über Umwege Flow spezifische Daten in einem Kontextmenü eines Knoten darstellen. Ebenfalls lassen sich so die einzelnen Knoten kaum der Applikation zuordnen welche die Verbindung aufgebaut hat. Man stelle sich folgende Situation vor:

Start des Live Capture: Der Benutzer startet das Live Capture auf dem Android Gerät

Zugriff auf Facebook mit Browser: Der Benutzer greift auf www.facebook.com mit dem Browser des Gerätes zu.

Zugriff auf Facebook mit Applikation: Der Benutzer greift auf www.facebook.com mit einer Applikation zu.

Innerhalb eines Graphen machen diese beiden Verbindungsaufbauten keinen Unterschied, jedoch sind sie für den Benutzer von Unterschiedlichem Interesse. So zum Beispiel wird der Zugriff auf Facebook mit der offiziellen Applikation als harmlos und gewünscht angesehen, aus einer anderen Applikation aber als ungewollt. Gemäss unserer Analyse und auch aus eigener Erfahrung ist aber genau dies Information von grossem Interesse für den Benutzer.

3.3.3 Transaktionen

Um die Anforderungen der Unterscheidung nach Applikationen zu erfüllen und dennoch die Aufgabenstellung zu erfüllen reichen die Informationen der Summarisierungen der Knoten des HAP Graphen aus. Diese Informationen können auf verschiedene Weisen gewonnen werden:

Graph Darstellung aufbereiten: Eine Möglichkeit ist es mittels der libhapviz die gesammelten Flowdaten in eine Graph Repräsentation umzuwandeln und diese in unserer Applikation aufzubereiten, sprich die Knoten den einzelnen Flows zuzuordnen. Diese Herangehensweise verschleiert allerdings das Problem des Kontextverlustes lediglich, denn immer wenn die Kontextinformationen gewünscht sind, muss die komplette Flowliste durchgegangen werden. Ebenfalls lässt sich mit dieser Variante das dynamische Ein- und Ausblenden einzelner Applikationen nur mit beträchtlichem Rechenaufwand und Komplexität realisieren. Der Vorteil dieser Variante ist aber dass der Bestehende Quellcode nicht stark angepasst werden muss.

Pfade des Graphen ausgeben: Eine andere Möglichkeit wäre es die Bibliothek soweit anzupassen dass nur die benötigten Summarisierungen in einem angemessenen, leicht zu

bearbeitendem Format ausgegeben werden. Da ein einzelner Pfad des Graphen, bis auf die summarisierten Knoten, einem Flow entsprechen, wird das Rückschliessen auf die Ursprünglichen Flows stark vereinfacht. Der grosse Nachteil dieser Herangehensweise ist, dass grosse Anpassungen am bestehendem Quellcode vorgenommen werden müssen.

Nachbau HAPviewer in Java: Anstelle der Nutzung der libhapviz bleibt immer noch eine eigen Implementation der bestehenden Logik. Dies wäre offensichtlich mit einem beträchtlichen Mehraufwand verbunden. Allerdings umgeht man damit die Abhängigkeit an externen Code und kann die Datenstrukturen auf den Anwendungsbereich optimieren.

Aus Überlegungen der Effizienz haben wir die Herangehensweise des Aufbereiten der Graph Darstellung nicht weiter verfolgt. Damit nicht die gesamte Logik des HAPviewer auf Java portieren zu müssen, werden wir nur die nötigsten Summarisierungen aus der HAPviewer Bibliothek mit den Pfaden des Graphen verwenden. Die Pfade des Graphen werden im weiteren Bericht als ”Transaktionen"bezeichnet werden. Der erwähnte Nachteil der grossen Anpassungen am HAPviewer Quellcode wird aber durch die gewonnene Flexibilität und Effizienz in Kauf genommen.

Zur Klärung des Begriffes der Transaktion, folgt ein kleines Beispiel. Zum Beispiel beinhaltet die folgende Grafik vier Transaktionen.

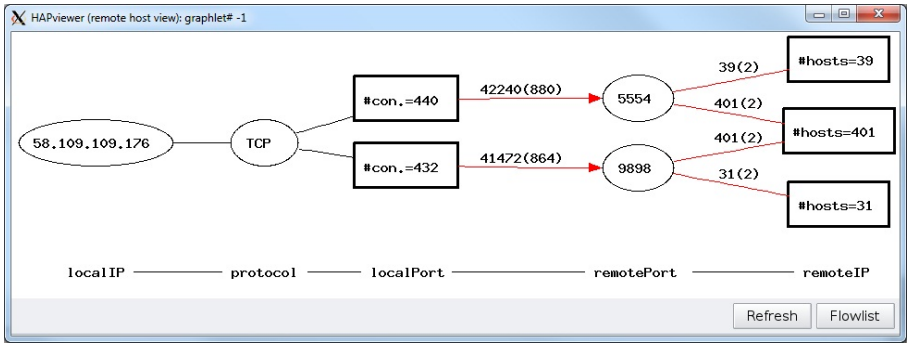

Abbildung 3.2: HAPviewer Beispielgraph

Die Transaktionen sind:

- 58.109.109.176, TCP, 440 lokale Ports, 5554 Zielport, 39 Zielrechner
- 58.109.109.176, TCP, 440 lokale Ports, 5554 Zielport, 401 Zielrechner
- 58.109.109.176, TCP, 432 lokale Ports, 9898 Zielport, 401 Zielrechner
- 58.109.109.176, TCP, 432 lokale Ports, 9898 Zielport, 31 Zielrechner

Auf diese Weise werden die wichtigen Informationen der Summarisierung beibehalten und das Rückschliessen auf die Ursprünglichen Flows ist durch das Nutzen einer eigenen Flowta-

belle ebenfalls nicht zu komplex. So wird ein akzeptabler Kompromiss zwischen Flexibilität und Aufwand erreicht.

3.4 Benutzeroberfläche darstellen

3.4.1 Anwendungsfälle

Um die Anforderungen an die grafische Benutzeroberfläche besser beurteilen zu können, muss erst bekannt sein, welches die wichtigsten Anwendungsfälle für die Applikation sind. Welche Daten sind interessant für den Benutzer? Wie viele Details Benötigt er?
Um diese Fragen zu beantworten haben wir ein Brainstorming mit fünf Mitstudenten durchgeführt. Die Ergebnisse aus dem Brainstorming und aus eigenen Überlegungen möchten wir hier zusammenfassen.

Kernfunktionen

Die Kernfunktionen beschreiben Anwendungsfälle, welche im Rahmen des HAPviewer als Analyse-Werkzeug sinnvoll genutzt werden können und im Zuge der Bachelorarbeit behandelt werden sollten.

Analyse von Kommunikationsmustern In einem ersten Schritt ist davon auszugehen, dass der Benutzer nicht genau weiss, nach was er sucht. Es scheint eine grobe Analyse des ganzen Netzwerkverkehrs des Mobiltelefons sinnvoll und dies sollte besonders mit einem Fokus auf die Muster in der Kommunikation gemacht werden können. So kann der Benutzer nun nach einem ersten Überblick feststellen, welches die häufigsten Kommunikationsmuster sind, er sieht aber auch, wenn Anomalien oder seltene Datenflüsse auftreten.
Bezüglich der Muster interessieren vermutlich vor allem quantitative Angaben sowie die wichtigsten Netzwerk-Eckdaten gemäss dem Berkley-Socketmodell. Dieser Anwendungsfall deckt sich ausserdem mit jenem des HAP-Graphlets, da dieses genau die genannten Informationen visualisiert.

Überwachung/Analyse einer bestimmten Applikation Basierend auf dem erstgenannten Anwendungsfall ist davon auszugehen, dass der Benutzer mitunter gewisse Applikationen einer Detailanalyse unterziehen möchte um herauszufinden, wie sich die jeweilige App verhält. Dies würde auf Seite von HAPdroid notwendig machen, dass der Netzwerkverkehr eindeutig einer bestimmten App zugewiesen werden kann. Ausserdem wäre ein spezieller Steuerungsmechanismus für die Erfassung des Netzwerkverkehrs von Vorteil, welche selbstständig überprüfen kann, in welchem Zustand (Gestartet, Gestoppt, etc.) sich die zu überwachende Applikation befindet.
Bei diesem Anwendungsfall interessieren die quantitativen Angaben zu den Netzwerkdaten und im Speziellen vermehrt Detailinformationen zu den jeweiligen Endknoten der Systeme (siehe “Weitere Funktionen”).

Datenverkehr einer bestimmten Zeitspanne auswerten Auf jeder Granularitätsstufe sollte es möglich sein, den zu analysierenden Netzwerkverkehr nach Zeit einzugrenzen. Um dem Benutzer dies zu erleichtern, müsste ersichtlich sein, zu welchem Zeitpunkt wie viel Verkehr angefallen ist. Die folgende Skizze veranschaulicht, wie ein entsprechender Benutzer-Dialog ausschauen könnte.

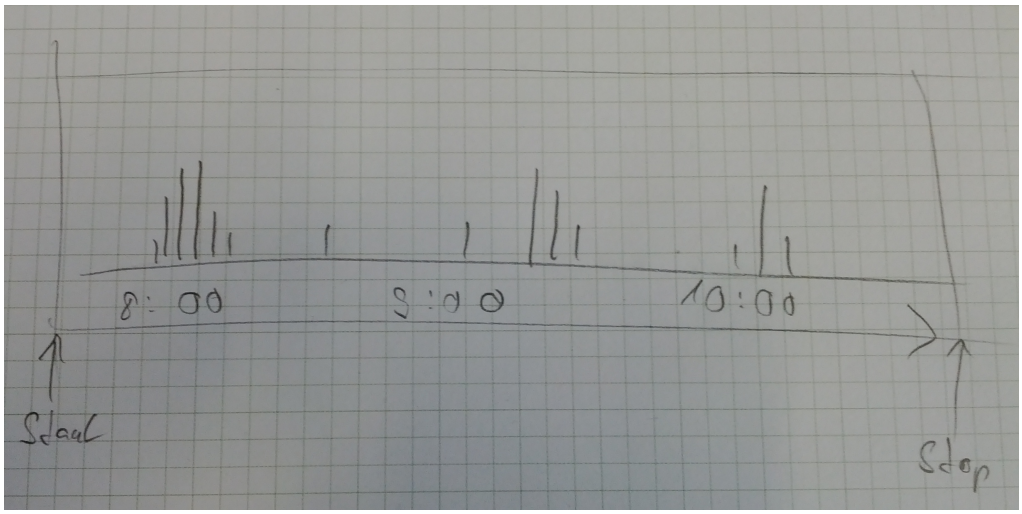

Abbildung 3.3: GUI Konzept: Skizze mit Visualisierung des Zeitauswahl-Diagramms

Weitere Funktionen

Die folgenden Funktionen und Anwendungsfälle wären ebenfalls von Nutzen und sollten nach Möglichkeit in die App integriert werden, jedoch weisen diese nicht die Wichtigkeit der Kernfunktionen auf.

Detailanalyse des Datenverkehrs allgemein oder einer bestimmten Applikation Um die logische Abfolge der Netzwerkverkehrsanalyse abzuschliessen, sollte es dem Benutzer auch möglich sein, den Inhalt der Datenpakete, welche während der Verwendung der App aufgenommen wurden, einzusehen und entsprechend zu analysieren. Da dieser Anwendungsfall bereits den Rahmen der ursprünglichen Aufgabenstellung sprengt müsste es möglich sein, eine Art Schnittstelle zu bieten, dass sich z.B. die Daten exportieren liessen, damit sie in einem anderen Sniffer-Programm analysiert werden könnten (z.B. Wireshark).

Statistik Informationen Als ergänzendes Feature für die App wäre die Darstellung von statistischen Informationen des Netzwerkverkehrs sinnvoll und naheliegend. Da es jedoch bereits eine Vielzahl entsprechender Apps im Android Market gibt, ist die Implementation dieser Funktion weniger wichtig.

Netzwerkinformationen über das Gerät Der Benutzer sollte für die Analyse der Kommunikationsdaten auch mit den lokalen Einstellungen seines Gerätes vertraut sein. Deshalb sollte es möglich sein, die Einstellungen und Informationen auf eine einfache Art, in übersichtlicher Darstellung, abzurufen.

Lookups: DNS, GeoIP, Service Banner Neben den Informationen zum eigenen Gerät, sollte es auch möglich sein, zusätzliche Informationen über die Kommunikationspartner abzurufen. Dies könnte zum Beispiel folgende Details beinhalten:

- DNS(-Reverse)-Lookup
  Es sollte möglich sein, die IP in einen entsprechenden DNS-Namen umwandeln zu lassen. Dies gibt dem Benutzer einen ersten Eindruck über seinen Kommunikationspartner, insbesondere, wenn es sich um einen bekannten Namen wie z.B. Google oder Facebook handelt.
- GeoIP-Lookup
  Als Erweiterung zu den DNS-Informationen könnte man ebenfalls noch entsprechende GeoIP-Details abrufen und dem Benutzer zur Verfügung stellen.
- Service Banner Lookup
  Möglicherweise will der Benutzer nicht nur weitere Informationen über den Host, sondern auch über die Services welche dieser anbietet. Zu diesem Zweck liese sich eine Abfrage für die Banner-Informationen (z.B. bei HTTP oder SMTP) implementieren.

Future Features

Unter “Future Features” möchten wir noch einige nennenswerte Ideen festhalten, welche wir jedoch noch nicht überprüft haben, hinsichtlich Nutzen und Bedürfnis beim Benutzer.

3G Protokol Unterstützung Da die Zielplattform Android von mobilen Geräten genutzt wird, wäre eine entsprechende Unterstützung der Mobil-Kommunikationsprotokolle mitunter interessant. Da diese Protokolle jedoch nicht für das HAP-Graphlet verwendet werden, ist eine Implementation nur bedingt sinnvoll und es müsste wahrscheinliche eine andere Darstellungsform gefunden werden.

Firewall Regel-Generierung Nach eingängiger Analyse des Kommunikationsverhaltens seines Gerätes stellt der Benutzer mitunter fest, dass es unerwünschte Verbindungen von oder zu seinem Gerät gibt. Nun wäre es von Vorteil, wenn, basierend auf den visualisierten Flow-Daten, entsprechende Firewall-Regeln generiert werden könnten und diese dann gleich im Gerät konfiguriert und aktiviert würden. Dieses Feature würde eine gute Integration unserer App in das System voraussetzen, dafür aber eine sehr benutzerfreundliche Variante zu Konfiguration der internen Firewall bieten. Der Anwendungsfall sollte jedoch aus Zeit- und Komplexitätsgründen erst für eine spätere Version der App in Betracht gezogen werden.

Flow Notification Mitunter gibt es seltene Kommunikationsmuster, welche man untersuchen möchte, oder es besteht das Problem, dass man nicht weiss, wer oder was diese Muster auslöst. Zu diesem Zweck könnte der Benutzer eine Benachrichtigung konfigurieren, welche ausgelöst wird, sobald der im Hintergrund aufgezeichnete Datenverkehr eines dieser Muster aufweist. Leider können wir nur Annahmen über den Nutzen eines solchen Features machen, da es dies in dieser Form noch nichts für Mobile-Geräte gibt. Ähnliche Funktionen werden jedoch mittlerweile von Antivirus-Lösungen mit integrierter Firewall angeboten, jedoch mit dem Zusatz, dass der Nutzer aufgrund der Benachrichtigung sich entscheiden kann, ob er die Verbindung zulassen will oder nicht.

3.5 Android

In diesem Abschnitt wird kurz auf einige Android Konzepte eingegangen welche für das Verständnis des weiteren Berichtes, besonders des Design Abschnittes, nützlich sind.

Für weitere Informationen zu Android wird auf die offizielle Dokumentation [1] verwiesen.

3.5.1 Activity

Eine Activity stellt die Komponente einer Android Applikation dar mit welcher der Benutzer interagiert. Jede Activity besitzt ein ”Window"welches für die Darstellung der Benutzeroberfläche verantwortlich ist und Benutzereingaben entgegen nimmt. Eine Android Anwendung besteht in der Regel aus mehreren, lose miteinander gekoppelten Activities welche mit Intents aufgerufen werden. Immer wenn eine Activity gestartet wird, wird die vorher sichtbare Activity gestoppt und ihr Status vom System auf einem Stack gespeichert. Über verschiedene Methoden-hooks kann das Verhalten der Activity gesteuert werden. [Goo12a] Die folgende Grafik veranschaulicht den Activity Lebenszyklus.

1 http://developer.android.com/guide/topics/fundamentals.html

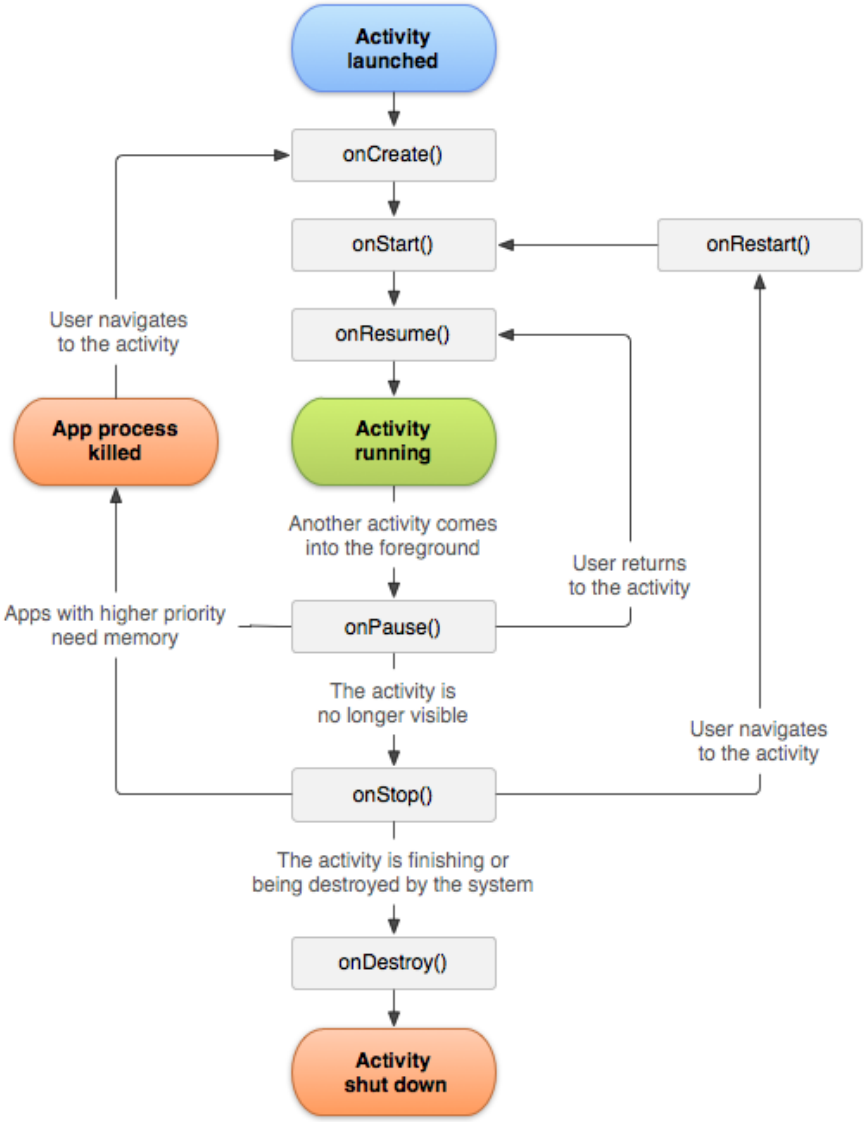

Abbildung 3.4: Android Activity Lebenszyklus

3.5.2 Service

Um in Android eine Aktion im Hintergrund durchführen zu können ist die Implementation eines Service notwendig. Dabei wird zwischen zwei unterschiedlichen Arten von Services unterschieden: [Goo12d]

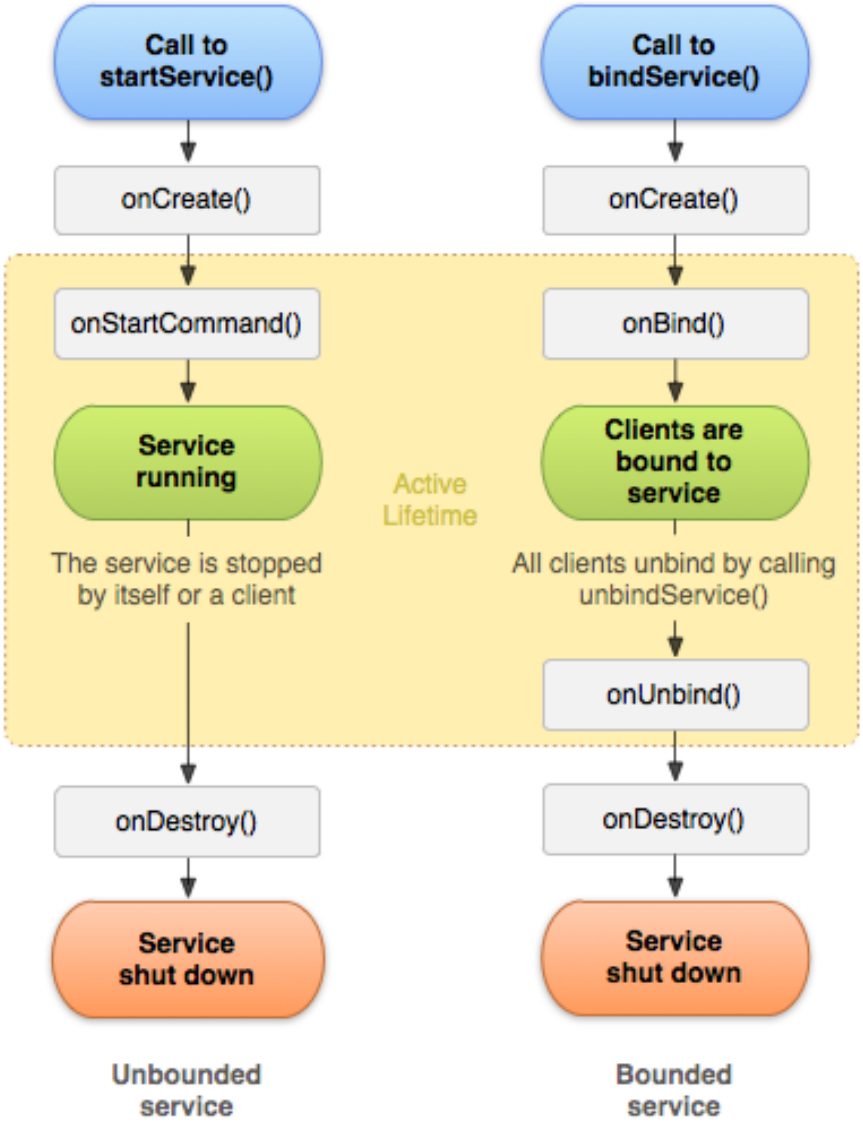

Abbildung 3.5: Android Service Lebenszyklus

Started Service: Ein gestarteter Service beschreibt einen Service welcher unbestimmte Zeit im Hintergrund laufen kann, selbst wenn die aufrufende Applikation bereits geschlossen wurde. Dieser Service stoppt sich selbst nach Beendigung der Aktion.

Bound Service: Ein gebundener Service ist an die Applikation gebunden und beschreibt so ein Client-Server Modell. Diese Art Service existiert nur so lange wie auch Applikationskomponenten an ihn gebunden sind. Es können aber auch mehrere Komponenten an

den Service gebunden sein.

Diese beiden Arten eines Services schliessen sich gegenseitig nicht aus. Dies kam uns sehr entgegen, da es für dieses Projekt nötig ist dass ein Client-Server Modell zwischen Applikation und Service existiert sowie dass der Service nach Beendigung der Applikation weiter seine Arbeit macht.

Für dieses Projekt haben wir uns eine Kombination der beiden oben genannten Services entschieden, da sowohl die Kommunikation zwischen Applikation und Service als auch die Möglichkeit im Hintergrund zu laufen nötig war um die Anforderungen zu erfüllen.

3.5.3 Handler

Ein Handler ermöglicht es Nachrichten und Runnables an einen Thread zu senden. Ein Handler wird an den Thread und die Message Queue des Erstellers gebunden und arbeitet die erhaltenen Nachrichten sequenziell ab. Es existieren grundsätzlich zwei Einsatzgebiete von Handler: [Goo12c]

- Nachrichten und Runnables zu einem späteren Zeitpunkt ausführen
- Aktionen zur Ausführung an einen anderen Thread schicken

Das Konzept des Handlers eignet sich hervorragend zur Implementation eines Threadübergreifenden Callbacks. Das folgende Beispiel veranschaulicht den Einsatz eines Handlers.

Codelisting 3.2: Beispiel eines Handlers

```
public class HAPdroidGraphletActivity extends LayoutGameActivity implements
    IOnSceneTouchListener, IScrollDetectorListener,
    IPinchZoomDetectorListener {

  private TextView mTxtStart;
  private TextView mTxtEnd;
  private Graphlet mGraphlet;
  private HAPdroidService mService;

  private Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
      switch (msg.what) {
      case HAPdroidService.SEND_NETWORK_FLOW:
        break;
      case HAPdroidService.GENERATE_GRAPHLET:
        mProgressDialog.dismiss();
        generateGraphlet();
        break;
      }
    }
  };
```

```
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mTxtStart = (TextView) findViewById(R.id.text_starttime);
    mTxtEnd = (TextView) findViewById(R.id.text_endtime);
    mService.setCallbackHandler(mHandler);
  }

  private void generateGraphlet() {
    Log.d(LOG_TAG,                        );
    mGraphlet.update(mService.getGraphlet());
    mTxtStart.setText(mService.getStartTime().toString());
    mTxtEnd.setText(mService.getEndTime().toString());
  }
  /*
   * der Rest wurde aus Gründen der Übersichtlichkeit weggelassen
   */
}

public class HAPdroidService extends Service {
  /**
   * Message identifier for generating the graphlet. This is mainly
   * used once all the transactions are received.
   */
  public static final int GENERATE_GRAPHLET = 5;

  private void finishGettingTransactions() {
    if (mCallbackHandler != null)
      mCallbackHandler.sendEmptyMessage(GENERATE_GRAPHLET);
  }
  /*
   * der Rest wurde aus Gründen der Übersichtlichkeit weggelassen
   */
}
```

3.5.4 Applikationsprozesse

Android benutzt etablierte Linux Mechanismen zur Abgrenzung der Applikationen. So besitzt jede Applikation eine eigene Benutzer ID. Jede Applikation läuft in einer eigenen VM Instanz welche unter der Anwendungsspezifischen Benutzer ID als selbstständiger Prozess ausgeführt wird. Werden innerhalb einer Applikation weitere Prozesse gestartet, so werden diese ebenfalls mit derselben Benutzer ID ausgeführt wie der Hauptprozess. So kann das System für alle Dateien Anwendungsspezifische Dateisystemrechte setzen damit nur die Applikation welche die Dateien erstellt hat Zugriff auf diese hat. [Goo12b]

4 Design

In diesem Abschnitt geben wir eine generelle Übersicht über das Design und die Überlegungen hinter den einzelnen Entscheidungen unserer Applikation. Dabei wird als erstes eine Einführung in die Architektur der Applikation gegeben. Die Struktur der weiteren Kapitel dieses Abschnittes folgt den logischen Schritten welche nötig sind, um von den Netzwerkdaten zu der Darstellung der Pakete als HAP graphlet zu gelangen. Diese lassen sich wie folgt unterteilen:

Netzwerkpakete sammeln: Als erstes ist es nötig die Netzwerkdaten zu sammeln. Dies geschieht über einen separaten Prozess.

Flowtabelle erstellen: Die gesammelten Netzwerkdaten werden in einem zweiten Schritt in Flows umgewandelt und in einer Flowtabelle gespeichert.

Transaktionen generieren: Um die in der Analyse zu Flowtabelle erstellen genannten Probleme zu lösen nutzen wir die bestehende Bibliothek nicht um einen Graphen zu generieren, sondern um so genannte Transaktionen zu generieren.

Graph Datenstruktur aufbauen: Nachdem die einzelnen Pfade des Graphen mit den Summen-Knoten erstellt sind, kann eine geeignete Datenstruktur für den Graphen aufgebaut werden. Für diesen Schritt nutzen wir die externe Bibliothek jGrapht.

Graphlet darstellen: Ist der Graph erstellt, wird in einem letzten Schritt das Graphlet dargestellt. Um die gewünschte Flexibilität und dynamische Interaktion zu erreichen greifen wir dabei auf die für die Entwicklung von Spielen bekannte AndEngine.

4.1 Architektur

Die Grundarchitektur lässt sich grob in die folgende Komponenten unterteilen.

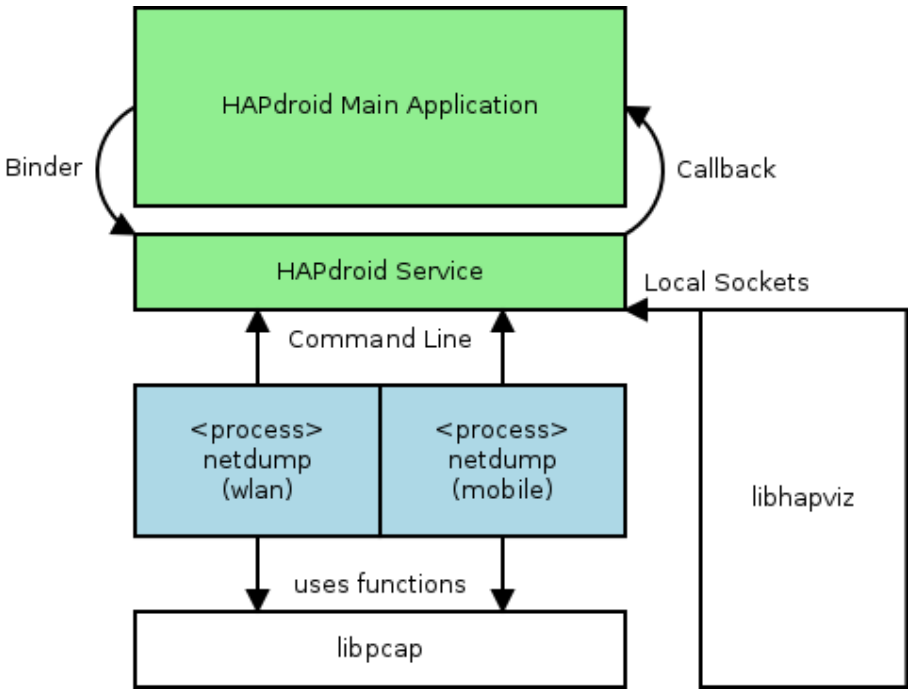

Abbildung 4.1: Architektur Übersicht

HAPdroid Main Application: Dies stellt hauptsächlich die Benutzeroberfläche und die Behandlung der Benutzereingaben dar. Es beinhaltet alle Android Aspekte der Applikation ohne Service, sowie Splash-Screen, Darstellung des Graphen und Eingaben für Dateiimport. Die Kommunikation mit dem Service geschieht über ein Binder Interface des Android Framework.

HAPdroid Service: Die Service Klasse ermöglicht das Ausführen der Applikation im Hintergrund. Der Service kümmert sich zusätzlich um die Kommunikation der einzelnen Komponenten. So startet der Service die ausführbare Datei, nimmt die gewonnenen Paketdaten entgegen, wandelt diese um und gibt sie an die Flowtabelle weiter. Mittels der Schnittstelle zur libhapviz Bibliothek geschieht hier die Aufbereitung der HAP Graph Daten aus der Flowtabelle. Zusätzlich bietet der Service eine Schnittstelle für das Speichern und Laden der Netzwerkdaten. Die Kommunikation mit der HAPdroid Main Application geschieht mittels eines Callback Handers.

libhapviz: Hierbei handelt es sich um die Bibliothek des HAPviewer. Sie wird zur Erstellung des HAP Graphen genutzt. Es waren allerdings starke Anpassungen notwendig um sie für unsere Zwecke nutzen zu können. Da es sich um eine externe Bibliothek handelt wird

sie in diesem Bericht nicht ausführlich behandelt sondern lediglich die Anpassungen welche für das Erstellen der Transaktionen nötig waren.

netdump: Bei netdump handelt es sich um eine kleine Ausführbare Datei welche den Netzwerkdatenverkehr mitschneidet und an den Service weitergibt. Dabei wird für jedes Netzwerkinterface (WLAN und mobile Daten) ein eigener Prozess verwendet. Die Netzwerkdaten können dabei mit weiteren Informationen, wie zum Beispiel der Ursprungsprozess einer Verbindung, bestückt werden. Die Übergabe der Information an den Service wird mittels eine InputStreams welcher die Kommandozeilenausgabe liest erreicht.

libpcap: Für das Mitschneiden des Netzwerkverkehrs wird die externe Bibliothek libpcap verwendet. Für weitere Informationen zu libpcap wird auf die offizielle Homepage verwiesen[1].

4.2 Netzwerkpakete sammeln

Die Netzwerkpakete werden mittels eines separaten Prozesses gesammelt. Dabei wird für jede einzelne Netzwerkschnittstelle ein eigener Prozess benutzt. Der Service dient dabei als Managementklasse der einzelnen Prozesse. Um das in der Analyse besprochene Problem der Berechtigung zu lösen, werden die Prozesse vom Service mit Root Rechten gestartet.

4.2.1 netdump

Für das Sammeln der Pakete nutzt der netdump Prozess die libpcap Bibliothek. Libpcap bietet eine High-Level Schnittstelle um Netzwerkpakete zu sammeln. Dabei werden Funktionen für das Öffnen von Netzwerkschnittstellen und Pcap Dateien bereit gestellt.

Zusätzlich zu der Funktionalität des Sammelns der Netzwerkpakete kann hier die Zuweisung der Applikationsprozesse zu den einzelnen Paketen durchgeführt werden. Es ist nötig diese Zuweisung möglichst früh zu machen, da es zu einem späteren Zeitpunkt kaum mehr möglich ist nachzuvollziehen welcher Prozess welches Paket generiert hat.

Applikationsinformationen

Um die PID des Prozesses zu einer bestehenden Netzwerkverbindung zu finden gibt es in Linux verschiedene Möglichkeiten:

netstat ist ein kleines Kommandozeilen Programm zur Darstellung von aktiven Netzwerkverbindungen, der Routingtabelle und weiteren Netzwerkstatistiken. Es lassen sich auch die PID/Programm Namen der geöffneten Netzwerksockets anzeigen.

1 http://www.tcpdump.org/

lsof gibt eine Liste aller geöffneten Filedescriptoren aus. Diese Liste beinhaltet geöffnete Sockets. Der grosse Nachteil für unseren Anwendungszweck ist, das durch die grössere Informationsmenge das Programm nicht so schnell wie netstat ist und deutlich mehr Informationen überprüft werden müssen.

/**proc**/**net** beinhaltet Pseudodateien welche Zugriff auf Informationen des Linux Netzwerkstacks ermöglichen. Hier sind viele nützliche Daten zur Netzwerknutzung zugänglich, unter anderem auch die TCP/UDP Sockettabellen. Dateien des proc Dateisystems sollten in der Regel aber nicht für Programme verwendet werden, da die Struktur der Dateien nicht statisch ist und durchaus mit Kernel-updates ändern kann. Aus diesem Grund ist diese Lösung nicht sonderlich ideal für unsere Zwecke.

Zwar gibt es noch weitere Kommandozeilen-Tools für Linux, allerdings sind die oben genannten Programme die bekanntesten und auch am weitesten verbreiteten.

Generell erweist sich das Zuweisen der PID zu der Netzwerkverbindung als nicht so einfach wie durch den Linux Kernel als Bestandteil von Android erwartet und erhofft. Dies liegt daran, dass die Version von netstat welche Android beinhaltet, die PID zu den Netzwerkverbindungen nicht anzeigt. Somit bleiben als Möglichkeiten nur noch der Web über /proc/net oder die Portierung des Codes von netsat auf einem Desktop Linux. Wünschenswert wäre die Portierung von netstat als Bestandteil von netdump. Dieser Lösungsansatz ist allerdings wieder mit einem nicht erwartetem Mehraufwand verbunden.

In der jetzigen Form verfügt netdump aber noch über keine weitere Funktionen ausser dem Sammeln und Ausgeben der Netzwerkpakete.

4.2.2 RootTools

Der Lebenszyklus des netdump Executables wird vom Service verwaltet. Damit netdump mit genügend Berechtigungen für den Zugriff auf die Netzwerkschnittstellen ausgeführt wird, wird die externe Bibliothek RootTools[1] genutzt.

RootTools bietet Hilfsfunktionen für Root Zugriff auf Android Geräte. Die Bibliothek erleichtert das Überprüfen von vorhandenen ausführbaren Dateien und Installieren eigener Executables. Ebenfalls bietet RootTools eine vereinfachte Möglichkeit Log Nachrichten nur im Debug Modus zu schreiben und eine Callback Klasse für das zeilenweise Verarbeiten von Kommandozeilenausgaben.

So können dank der RootTools Bibliothek stetig die gesammelten Pakete analysiert und in der Flowtabelle gespeichert werden.

1 http://code.google.com/p/roottools/

4.3 Flowtabelle erstellen

Zu der Flowtablle werden stetig neue Netzwerkpakete hinzugefügt. Damit die Korrektheit der Flows sichergestellt ist, muss die hinzufüge Funktion folgende Schritte implementieren:

Bestehende Flows überprüfen: Als erstes muss überprüft werden ob das Paket einem bereits existierendem Flow zugeordnet werden kann

Flowwerte anpassen: Sofern bereits ein Flow für das Paket existiert, muss Paketanzahl, Dauer, Richtung und Flowgrösse mit den Informationen des Paketes angepasst werden

Neuen Flow erstellen: Wenn noch kein Flow zu dem Paket existiert, wird ein neuer Flow mit den Werten des Paketes erstellt.

Für die Richtung der Flows ist es wichtig zu wissen welche IP Adressen das Android Gerät besitzt. Weil Android Geräte in der Regel zwei Netzwerkschnittstellen besitzen, Mobil und Wireless, muss beim erstellen eines neuen Flows darauf geachtet werden dass die Pakete auf alle lokalen IP Adressen überprüft werden.

4.4 Transaktionen generieren

Ist die Flowtabelle erstellt, können die gesammelten Daten an die libhapviz Bibliothek übergeben werden. Die Bibliothek akzeptiert folgende Dateiformate:

pcap wird von tcpdump, Wireshark und vielen anderen unterstützt. Pcap ist daher sehr verbreitet, da mit libpcap eine eine flexible, offene und funktionsreiche Bibliothek existiert. Ein Nachteil von pcap für unsere Applikation ist, dass alle Informationen der Pakete gespeichert werden und somit einen verhältnismässig grossen Speicherbedarf haben.

nfdump wird von NfDump und NfSen als Dateiformat benutzt.

ipfix ist ein Standard der Internet Engineering Task Force (IETF) welches den Export von IP Flow Informationen spezifiziert.

cflow ist ein eigene proprietäres Flow format von HAPviewer(proprietary binary flow format employed by HAPviewer)

Wir haben uns für das cflow Dateiformat, genauer das cflow4 Dateiformat, für die Übergabe von der Flowtabelle zur Bibliothek entschieden. Diese Entscheidung wurde aufgrund der äusserst kompakten Informationen getroffen und weil das Format effizient importiert werden kann. Cflow enthält alle für uns nötigen Daten mit sehr geringen Overhead.

4.4.1 libhapviz

C++ streams

Für die Ausgabe der Transaktionen sind einige Anpassungen am HAPviewer source code notwendig. In einem ersten Schritt muss die Importierschnittstelle auf C++ streams umgeschrieben werden um nicht den unnötigen Umweg über temporäre Dateien nehmen zu müssen. Dazu wurde in der CImport Klasse und in den Importfiltern eine neue Funktion namens *read_stream* definiert und die bestehenden *read_file* Funktionen umgeschrieben damit diese auf *read_stream* verweisen.

4.1 zeigt die Änderung an der CImport Schnittstelle und 4.2 zeigt die Änderungen an der Importfilter Schnittstelle.

Codelisting 4.1: Änderungen CImport Schnittstelle

```
/**
 *  \class CImport
 *  \brief Import of binary data from gzipped cflow_t binary files or from
     pcap binary files
 */
class CImport {

  // Functions to import flows
  public:
    static bool acceptForImport(const std::string & in_filename);
    static bool acceptForExport(const std::string & out_filename);
    void read_file(const IPv6_addr & local_net = IPv6_addr(), const
        IPv6_addr & netmask = IPv6_addr());
    // added function read_stream
    void read_stream(std::istream & in_stream, unsigned int flowcount,
        const IPv6_addr & local_net = IPv6_addr(), const IPv6_addr &
        netmask = IPv6_addr());
    void write_file(std::string out_filename, const CFlowList & flowlist,
        bool appendIfExisting);
    void write_file(std::string out_filename, const Subflowlist &
        subflowlist, bool appendIfExisting);
    static std::string getFormatName(std::string & in_filename);
    static std::vector<std::string> getAllFormatNames();
    static std::vector<std::string> getAllHumanReadablePatterns();
    static std::ostream & printAllTypeNames(std::ostream & os);
    static std::string getFormatNamesAsString();
    static unsigned int initInputfilters();

    // rest ommited
}
```

Codelisting 4.2: Angepasste Importfilter Schnittstelle

```
/**
```

```
 *  \class  GFilter
 *  \brief  GFilter is an pure virtual class which can be implemented by
     various in-/outputfilter
 */
class GFilter {
  public:
    GFilter(std::string formatName =                    , std::string
        humanReadablePattern =                           ,
          std::string regexPattern =           );
    virtual ~GFilter(){};
    // general methods
    std::string getFormatName() const;
    std::string getHumanReadablePattern() const;
    virtual bool acceptFilename(std::string in_filename) const;

    // import methods
    // replaces function read_file
    virtual void read_stream(std::istream & in_stream, CFlowList & flowlist
        , const IPv6_addr & local_net, const IPv6_addr & netmask, unsigned
        int flowcount, bool append) const=0;
    virtual bool acceptFileForReading(std::string in_filename) const=0;

    // export methods
    virtual bool acceptFileForWriting(std::string in_filename) const;
    // replaces function write_file
    virtual void write_stream(const std::ostream & out_stream, const
        Subflowlist subflowlist, bool appendIfExisting = true) const;

  protected:
    typedef HashKeyIPv6_5T flowHashKey;
    typedef hash_map<HashKeyIPv6_5T, cflow_t *, HashFunction<HashKeyIPv6_5T
        > , HashFunction<HashKeyIPv6_5T> > flowHashMap;

    std::string formatName; ///< Name of this format (e.g. pcap, cflow,
        nfdump)
    std::string humanReadablePattern; ///< A human "readable" pattern for
        the fileextension (e.g. *.pcap)
    std::string regexPattern; ///< A regex pattern for the file extension (
        e.g. .*\\.pcap$)
};
```

Zum Schluss wird noch der Konstruktor der CGraphlet Klasse so angepasst dass ein *std::ostream* anstelle eines Dateinamen übergeben werden kann.

Transaktionen

Der HAPviewer speichert den Graphen intern über die Kanten, welche in verschiedenen HashMaps abgelegt sind. Es existieren dabei die folgenden HashMaps:

Codelisting 4.3: HAPviewer Graph HashMaps

```
// *** Use individual hash maps for each rank type

// Derive a fully annotated graphlet in form of a k-partite graph (k=5)
// Use k-1 hash tables to capture edges between the k partitions
// Additionally, one more hash map is used to keep track of unique host
     numbers
// Partitions are: localIP, prot, localPort, remotePort, remoteIP
graphletHashMap * hm_localIp_prot; // localIP--prot

// Local port number is not shared between protocols and hosts
// (we have one hosts only, but might have several protocols)
// -> store protocol enum code together with port number (use enum
    values as protocol code)
graphletHashMap * hm_prot_localPort_11; // prot--localPort
graphletHashMap * hm_prot_localPort_1n; // prot--localPort

// Remote port numbers are not shared between remote hosts and
    protocols
// (we might have several protocols and several remoteIPs)
// -> store protocol enum code and remote host number together with
    port number
graphletHashMap * hm_localPort_remotePort_11; // localPort--remotePort
graphletHashMap * hm_localPort_remotePort_n1; // localPort--remotePort
graphletHashMap * hm_localPort_remotePort_1n; // localPort--remotePort
graphletHashMap * hm_localPort_remotePort_nn; // localPort--remotePort
// Additional has map needed for edge annotations
graphletHashMap * hm_localPort_remotePortE; // localPort--remotePort (
    extension)

// Each remote port (enhanced with protocol enum code and host number)
    is associated
// with exactly one remote host. To save space we do not identify
    remote host by its IP address,
// but with a consecutively allocated host number (host code).
// -> store unique remote Ips together with a host code (starting at 0;
     up to (2**14)-1)
graphletHashMap * hm_hnum_remoteIp; // Auxiliary hash map for unique
    remote host numbers

// Each remote port per protocol is associated with exactly one
    remoteIp.
graphletHashMap * hm_remotePort_remoteIp_11; // remotePort--remoteIP
graphletHashMap * hm_remotePort_remoteIp_n1; // remotePort--remoteIP
graphletHashMap * hm_remotePort_remoteIp_1n; // remotePort--remoteIP
graphletHashMap * hm_remotePort_remoteIp_nn; // remotePort--remoteIP
// Additional has map needed for edge annotations
graphletHashMap * hm_remotePort_remoteIpE; // remotePort--remoteIP (
    extension)
graphletHashMap::iterator iterIpProt, iterProtEport, iterEport2,
    iterEport3, iterHnumIp, iterEportIp;
```

Es existiert mindestens eine HashMap für jede mögliche Verbindung zwischen den Partitionen basierend auf den Schlüsselattributen des Berkeley socket Modells. Zusätzlich zu den HashMaps mit den eins zu eins Verbindungen, angegeben mit dem _11 Suffix, existieren weitere HashMaps mit den möglichen Kombinationen der Summarisierten Knoten, angegeben mit _1n, _n1, _nn Suffixen.

Um die Transaktionen zu Generieren wird eine neue Funktion namens *write_transactions* für die CGraphlet Klasse definiert. Die Idee zur Generierung der Transaktionen liegt im Sequenziellen durchlaufen der HashMaps. Dies resultiert in mehrere ineinander verschachtelte Schleifen. Die Funktionsweise kann auf die folgende Weise beschrieben werden.

Codelisting 4.4: Generation Transactions Pseudocode

```
Für jede Kante k_srcIp_proto in hm_localIp_prot
  Für jede Kante k_proto_localPort in hm_prot_localPort
    wenn k_proto_localPort.Protokoll gleich k_srcIp_proto.Protokoll
    Für jede Kante k_localPort_remotePort in hm_localPort_remotePort
      wenn k_localPort_remotePort.localPort gleich k_proto_localPort.
          localPort
      Für jede Kante k_remotePort_dstIp in hm_remotePort_dstIp
        wenn k_remotePort_dstIp.remotePort gleich k_localPort_remotePort.
            remotePort
        Gib Transaktion für srcIp, Protokoll, localPort, remotePort, dstIp
            aus
```

Zusätzlich muss noch auf die verschiedenen HashMaps für die Summarisierten Knoten geachtet werden. Diese Herangehensweise ist durch die verschachtelten Schleifen äusserst ineffizient bei einem grossen Graphen. Allerdings ist es die nächstliegendste Art und Weise Transaktionen auszugeben.

Nun da alle Informationen zu den einzelnen Knoten des Graphen vorhanden sind ist es noch notwendig diese Informationen an den Service weiterzuleiten.

4.4.2 Kommunikation Service

Da die Transaktionen in der Funktion *write_transactions* an den *std::ostream*, welcher dem CGraphlet Konstruktor übergeben wurde, geschrieben werden, sind wir äusserst flexibel in der Übergabe der Informationen.

Bei den Transaktionen handelt es sich um einen Stream von Daten. Diese werden sinnvollerweise in Java über einen InputStream abgebildet damit ein stetiges Abarbeiten der Informationen innerhalb von Java erreicht werden kann. Ein weiterer Vorteil der Nutzung eines InputStreams ist, dass nicht unnötig viel Speicher für alle Transaktionsdaten initialisiert werden muss. Diese Daten werden später in Transaktionsobjekte umgewandelt und würden somit erhöhten Speicherbedarf generieren.

Theoretisch wäre es möglich einen Java InputStream mit einen C++ output Stream zu verbinden und so eine Art Pipe zum Informationsaustausch zu erhalten. Dies wäre die sauberste Lösung für die Kommunikation mit dem Service. Da wir noch keine Erfahrung mit einer solchen

Implementation haben, konnten wir allfällig auftauchende Probleme sowie Zeitaufwand nicht abschätzen. Deshalb entschieden wir uns für die Implementation eines lokalen Server Socket. Eine solche Lösung wurde bereits in einer früheren Phase des Projektes implementiert und getestet.

4.5 Graph Datenstruktur aufbauen

Für die Graph Datenstruktur haben wir uns entschieden die bestehende Bibliothek jGrapht[1] zu nutzen. JGrapht ist eine freie Graphen Bibliothek in Java. Die Bibliothek zeichnet sich durch eine Vielzahl von Algorithmen, guter Dokumentation und Typensicherheit aus. Dadurch konnte die gesamte Komplexität der Graphen Datenstruktur umgangen werden.

4.5.1 jGrapht

Für die Implementation wurde auf einen Simplegraph zurückgegriffen. Diese Wahl wurde getroffen die SimpleGraph Klasse die minimalen Anforderungen an den HAP Graphen erfüllt. Eine SimpleGraph Instanz erlaubt nur eine Kante zwischen zwei Knoten und gestattet keine Schleifen.

4.6 Graphlet darstellen

4.6.1 Kommunikation Service

Für die Kommunikation von Service und Applikation stellt das Android Framework verschiedenste Mechanismen bereit. Der einfachste bereitgestellte Weg für die Methodenaufrufe des Services aus der Applikation ist mittels eines Binders [2].

Da der Service die Empfangenen Flows der Applikation bereitstellen muss, ist ein Callback Mechanismus notwendig welcher idealerweise über einen "Pushäusgehend von dem Service die Daten an die Applikation liefert. Wir haben uns dafür entschieden einen Handler [3] als Callback zu implementieren, da dieser genau unsere Anforderungen erfüllt.

4.6.2 Konzepte

Den Netzwerkverkehr in Form eines HAP-Graphen darzustellen bringt den Vorteil eine grosse Menge an Daten übersichtlich und dennoch verständlich darzustellen. Für die Anzeige eines solchen Graphen auf kleinen Bildschirmen bedeutet dies, dass die Anordnung der Inhalte bereits optimiert wurde und nicht mehr merklich verbessert werden kann. Das Hauptproblem

1 http://jgrapht.org/
2 http://developer.android.com/reference/android/os/IBinder.html
3 http://developer.android.com/reference/android/os/Handler.html

liegt also darin, grosse Graphen, für welche der Anzeigebereich nicht ausreicht, so darzustellen, dass der Kontext der Informationen, für die Interpretation ausreichend, erhalten bleibt.

Wärend der Analyse- und Designphase sind verschiedene Ideen entstanden, welche hier kurz erläutert werden. Für alle Ideen gelten die gleichen Symbol- und Farbbedeutungen wie beim HAPviewer, Erweiterungen werden entsprechend beschrieben. Dies soll bestehenden HAPviewer Benutzern den Einstige in HAPdroid vereinfachen und die gemeinsame Basis der Software hervorheben.

Adapted-Zoom-Graphlet

Das Kernkonzept dieser Idee besteht darin, ein interaktives Graphlet zu programmieren, welches je nach Zoom-Stufe nur die dafür relevanten Informationen anzeigt. Dieses Konzept lässt sich auch bei Google Maps beobachten, wo zusätzliche Details eingeblendet werden, wenn der Zoom-Faktor vergrössert wird.

In der kleinsten Zoom-Stufe, mit der grössten Übersicht, geht es darum, dass der Benutzer mögliche Muster und Rollen erkennen kann, wie sie im Whitepaper von Herr Glatz beschrieben sind (siehe folgende Abbildung unten). [Gla10]

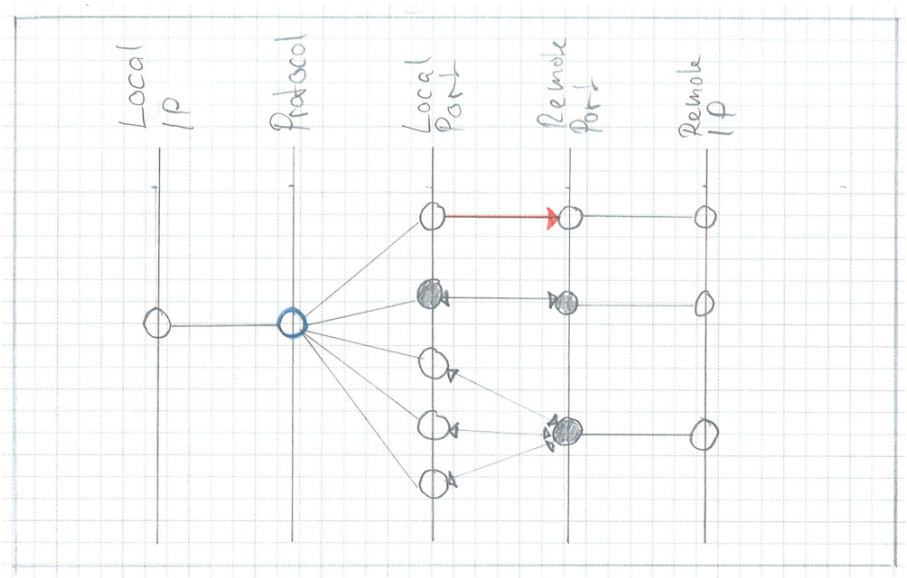

Abbildung 4.2: Skizze der Adapted-Zoom Idee in der Pattern-Ansicht

In der mittleren Zoom-Stufe entspricht das Graphlet dem bestehenden Konzept aus HAPviewer. Die Bereiche des Graphlets, welche nicht in den Bildschirmausschnitt passen, können mittels vertikalem Scrolling in den sichtbaren Bereich verschoben werden. Bei diesem Vorgang werden alle Knoten in der selben Spalte nur vertikal verschoben, während die anderen Spalten an ihrer Position verbleiben.

In der höchsten Zoom-Stufe, der Detail-Ansicht, sind nicht mehr alle Spalten im sichtbaren

Bereich. Der Benutzer konzentriert sich auf einen Teilbereich des Graphen und richtet die gewünschten Knoten im sichtbaren Bereich aus. Alternativ könnte eine Funktion implementiert werden, welche mit einem Klick den gewünschten Knoten in seiner Spalte vertikal zentriert und farblich hervorhebt (blau umrandete Knoten in der folgenden Abbildung). Die Knoten könnten aufgrund ihrer Grösse weitere Informationen anzeigen, wenn dies erwünscht ist (z.B. bei “remote IP” könnte zusätzlich der dazugehörige Domain-Name angezeigt werden).

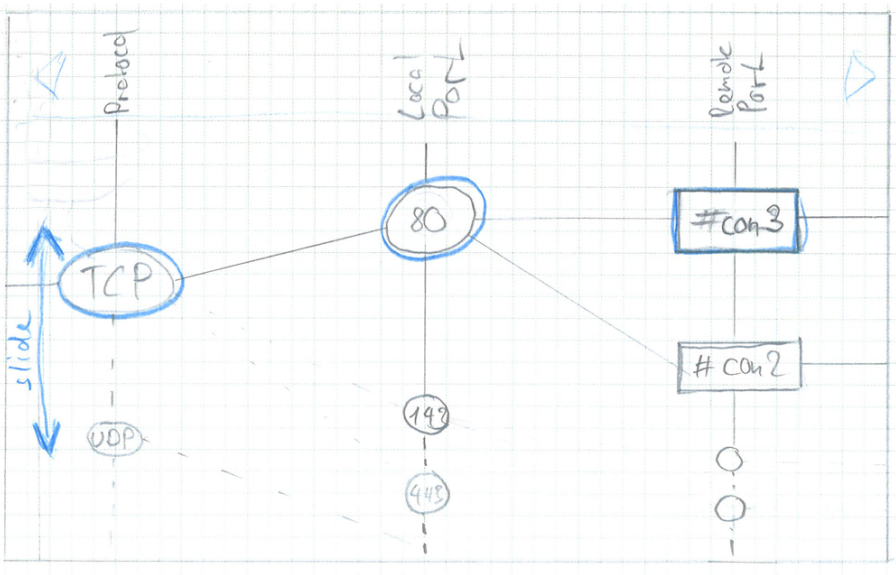

Abbildung 4.3: Skizze der Adapted-Zoom Idee in der Detail-Ansicht

Um den Fokus auf die hervorgehobenen Knoten zu verstärken, könnte ein Fischauge-Effekt in einer Art genutzt werden, dass die Knoten immer kleiner werden, je näher sie sich am Bildschirmrand befinden. Dies hätte zudem den Vorteil, dass mehr dieser kleineren Knoten im sichtbaren Bereich verbleiben könnten.

Vorteile :

- Das Graphlet bietet mit den unterschiedlichen Zoom-Stufen flexible Darstellungsmöglichkeiten.
- Die Visualisierung wäre innovativ.

Nachteile :

- In der höchsten Zoom-Stufe kann der Kontext zum 5-Tupel des Socket Modells verloren gehen.
- Der mögliche Kontextverlust wäre nur mit verbesserter Lesbarkeit oder höherem Informationsgehalt gerechtfertigt.
- Die Bedienung des Scrollings und der Zoom-Funktion kann, wenn nicht verständlich, zur Verwirrung führen.

Select-Reduce-Graphlet

Der mögliche Kontextverlust im Adapted-Zoom-Graphlet brachte uns auf die Idee, anstelle der Zoom-Funktion mit Filtern zu arbeiten. Das Select-Reduce-Graphlet entspricht im wesentlichen dem Adapted-Zoom-Graphlet auf mittlerer Zoom-Stufe, ebenfalls mit dem beschriebenen, vertikalen Scrolling. Das spezielle daran ist die zusätzliche Filterfunktion, welche sehr einfach angewendet werden kann. Der Filter wird aktiviert, indem man den gewünschten Knoten entweder per Touch-And-Drag in den blau markierten Filter-Bereich zieht, oder den Knoten per Touch-And-Hold direkt auswählt. Ist in einer Spalte ein Knoten als Filter gewählt, werden alle anderen Knoten, die nicht den Kriterien entsprechen, ausgeblendet. Auf diese Weise kann zwar pro Spalte nur jeweils ein Filterkriterium gesetzt werden, jedoch würde einer Erweiterung zur Verbesserung dieser Funktionalität nichts im Wege stehen. Die Deaktivierung der Filter funktioniert auf die selbe Weise, wie sie gesetzt wurden, mit dem Unterschied, dass beim Touch-And-Drag der Knoten aus dem Filter-Bereich heraus gezogen werden muss.

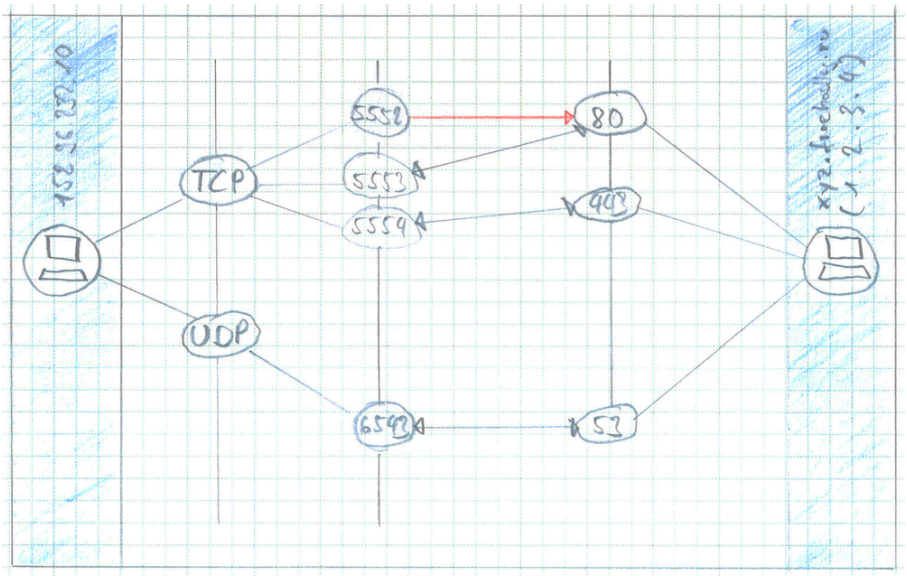

Abbildung 4.4: Skizze der Select-Reduce Idee mit aktiviertem Filter auf “remote IP”

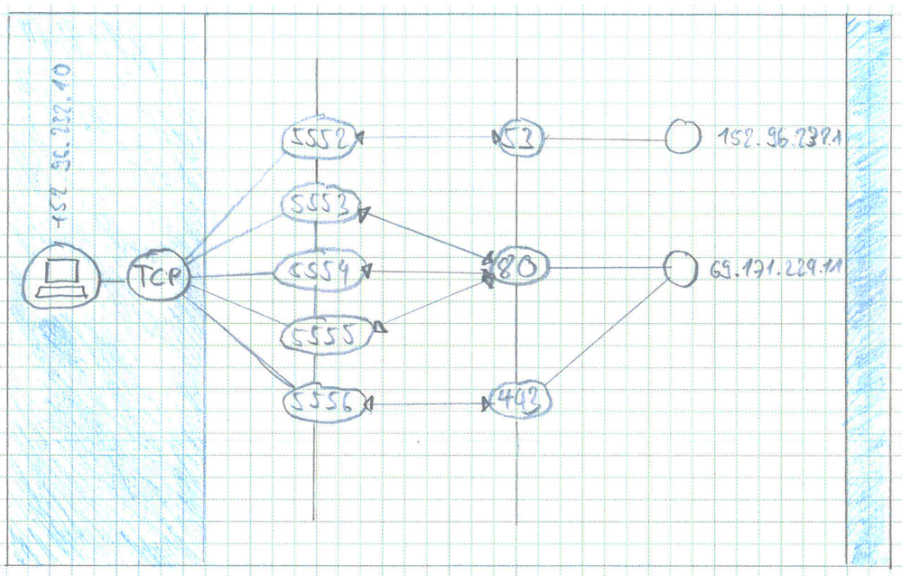

Abbildung 4.5: Skizze der Select-Reduce Idee mit aktiviertem Filter auf “remote IP” und “TCP”

Vorteile :

- Der Benutzer kann auswählen, welche Daten ihn speziell interessieren und diese isoliert betrachten.
- Die Visualisierung wäre innovativ.
- Die Applikation würde hohe Anforderungen an die Datenaufbereitung erfüllen.
- Der Kontext zum 5-Tupel des Socket Modells bleibt zu jeder Zeit erhalten.

Nachteile :

- Die Entwicklung der Datenaufbereitung ist möglicherweise komplex, je nachdem, wie die Filterfunktion umgesetzt wird.
- Das Aktivieren eines Filters würde vermutlich eine Neuberechnung des (Teil-)Graphen erfordern, was auf einem Mobilgerät zu Performance-Problemen führen könnte.
- Das Handling der TouchEvents müsste sehr ausgeklügelt sein, sodass nicht beim Scrollen fälschlicher weise Filter aktiviert würden.

Value-Graphlet

Die beiden zuvor beschriebenen Konzepte schiessen etwas über das Ziel hinaus, da im Hinblick auf die Arbeit und die zur Verfügung stehende Zeit, der Fokus zuerst auf die Entwicklung einer stabilen Benutzeroberfläche gelegt werden sollte. Danach können immer noch weitere Features für das Filtern oder spezielle Anzeigemodi entwickelt werden.

Der Anspruch, ein realitätsnäheres Konzept zu kreieren, bracht uns auf die Idee für das Value-Graphlet. Offensichtlich muss jedes Konzept eine Variante vorsehen, auch Graphen darzustellen die grösser als der Anzeigebereich sind, weshalb die bereits vorgestellte Idee mit dem vertikalen Scrolling auch für diese Variante gilt. Abgesehen davon entspricht das Graphlet beinahe seinem Pendant im HAPviewer und soll lediglich durch weitere Informationen aufgewertet werden.

Wenn nicht mittels einer Reduzierung der Daten, oder Vergrösserung von Teilbereichen des Graphlets, die Darstellung von Zusatzinformationen ermöglicht werden kann, dann müssen diese anderweitig abgerufen werden können. Zu diesem Zweck sieht das Value-Graphlet vor, für die interessanten GUI-Elemente eine Art Tooltips zur Verfügung zu stellen. Mit einem einfachen Klick auf das gewünschte Element, werden vorhandene Zusatzinformationen in einem Tooltip-Balloon neben dem Element angezeigt und können mit einem weiteren Klick wieder ausgeblendet werden.

Der eigentliche Mehrwert, oder “Value”, dieses Konzeptes besteht darin, dass anstelle des Feldes “local IP” die, für den jeweils dargestellten Netzwerkverkehr, verantwortliche Android-App angezeigt wird. Ausserdem lässt das vereinfachte GUI-Konzept Raum für Erweiterungen im Bereich der Summarisierung. Wie in der folgenden Abbildung angedeutet, könnte zusätzliche Logik für die Summarisierung nach DNS-Wildcards oder die Port-Bereiche “Well-known”, “Registered” und “Dynamic” implementiert werden.

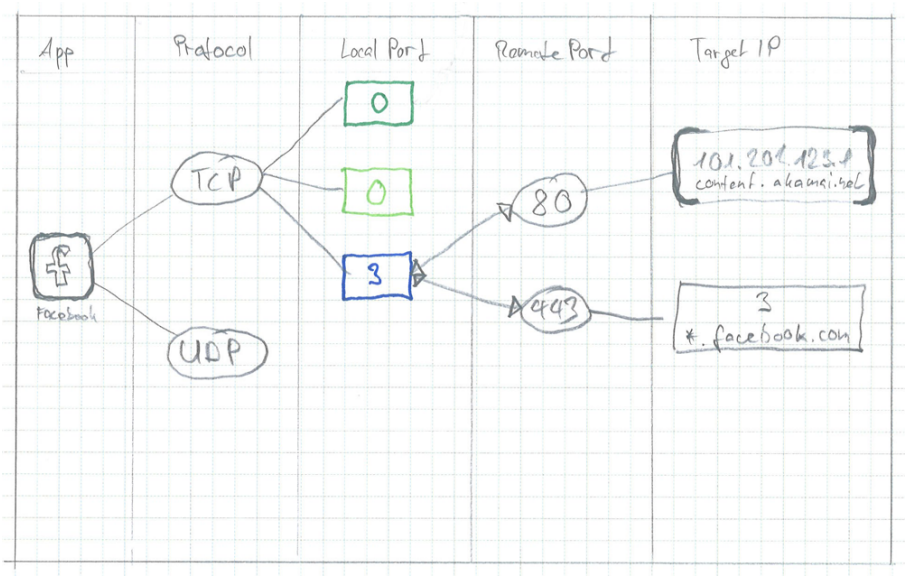

Abbildung 4.6: Skizze der Value-Graphlet Idee mit zusätzlichen Summarisierungs-Möglichkeiten (keine korrektes Graphlet)

Vorteile :

- Die Visualisierung wäre ausbaufähig.

- Der Kontext zum 5-Tupel des Socket Modells bleibt zu jeder Zeit erhalten.
- Die Visualisierung wäre übersichtlich.
- Zusatzinformationen könnten bei Bedarf angezeigt werden.
- Mit der Spalte für Android Apps kann der Informationsgehalt des Graphlets verbessert werden.
- Es werden keine komplexe Touch-Eingabe-Konzepte verwendet.

Nachteile :

- Die Daten, welche Android App den Netzwerkverkehr generiert hat, müssen mit den entsprechenden Daten zuverlässig und korrekt verknüpft werden.
- Die Visualisierung wäre zwar neuartig aber nicht speziell innovativ.

5 Implementation

In diesem Abschnitt werden die einzelnen Implementationsschritte und die Änderungen an der bestehenden Bibliothek beschrieben. Die Struktur folgt dem logischen Ablauf der Applikation, welcher sich auch in unserer Paketstruktur widerspiegelt.

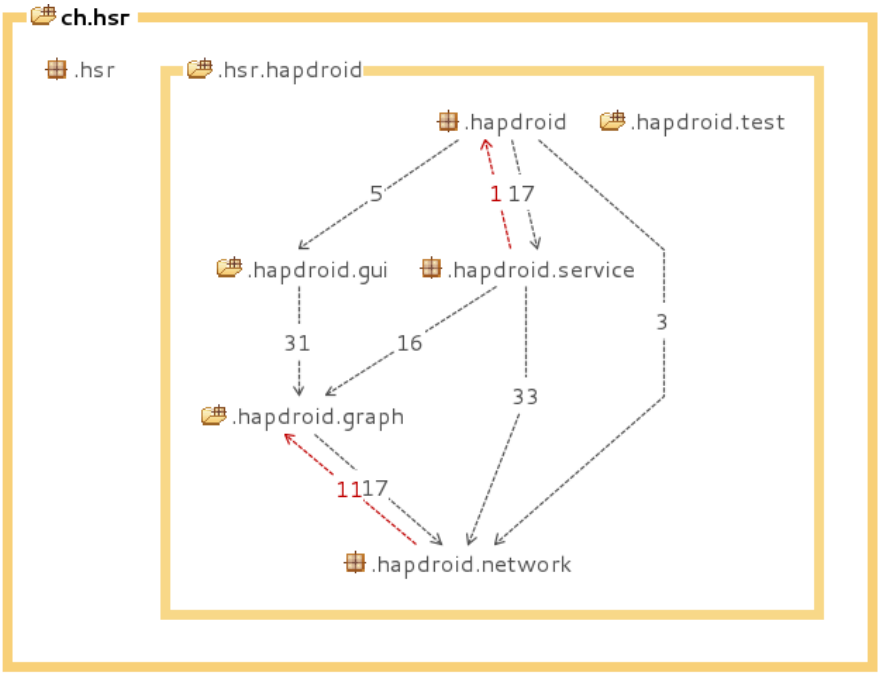

Abbildung 5.1: Pakete Übersicht

Für genauere Informationen zu einzelnen Klassen wird auf die Doxygen Dokumentation verwiesen.

hapdroid: Dieses Paket enthält die Grundkomponenten der Applikation. Hier sind die Activities enthalten. Die primäre Verantwortlichkeit dieses Paketes ist das entgegennehmen von Benutzereingaben wie zum Beispiel Start, Stopp des Netzwerk-Capture, importieren von pcap und cflow4 Dateien.

hapdroid.service: Hier ist die Service Klasse und die JNI Schnittstelle zur libhapviz Bibliothek enthalten.

hapdroid.network: Das Network Paket beinhaltet die Packet- und Flow-Klassen sowie die Flowtabelle.

hapdroid.graph: In diesem Paket ist die Transaction-Klasse und weitere Klassen enthalten welche für den Aufbau der internen Graphen-Datenstruktur benötigt werden.

hapdroid.gui: Alle Klassen und Komponenten der Benutzeroberfläche befinden sich in diesem Paket.

Zusätzlich zu den Java Paketen existieren mit *netdump* und *libhapviz* zwei Komponenten welche als native Code in die Applikation eingebunden wurden. Diese werden in den Abschnitten Netdump Paketlooper Callback Funktion und CaptureSource genauer behandelt.

5.1 Grundkomponenten

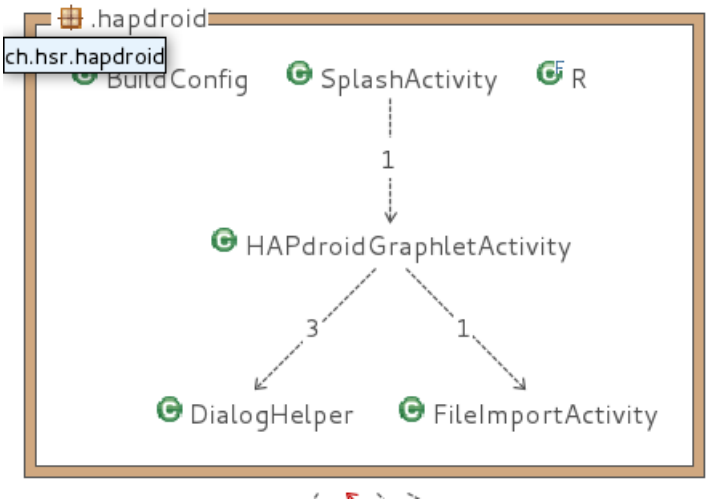

Abbildung 5.2: ch.hsr.hapdroid Paketübersicht

Im ch.hsr.hapdroid Paket befinden sich die Generellen Klassen welche keinem anderen Paket zugeordnet werden konnten.

5.1.1 SplashActivity

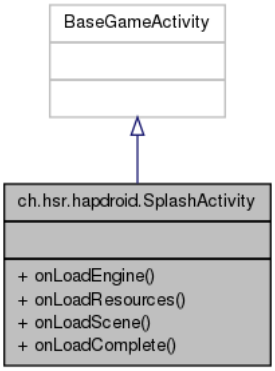

Abbildung 5.3: SplashActivity UML Diagramm

5.1.2 HAPdroidGraphletActivity

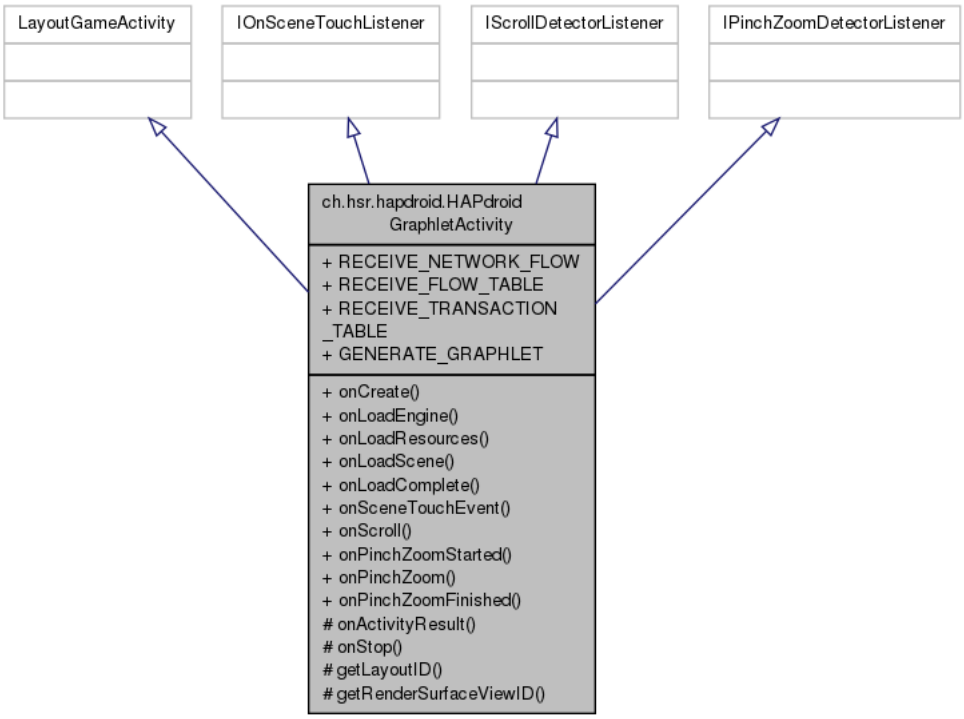

Abbildung 5.4: HAPdroidGraphletActivity UML Diagramm

Die HAPdroidGraphletActivity Klasse kümmert sich um die Benutzereingaben und die Darstellung des HAPGraphlet. Sie gibt die Benutzereingaben wenn nötig an den DialogHelper weiter

oder startet die HAPdroidGraphletActivity.

5.1.3 FileImportActivity

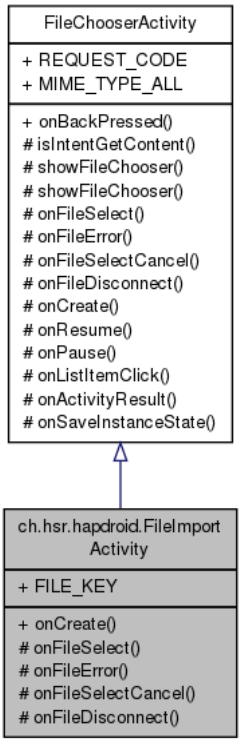

Abbildung 5.5: FileImportActivity UML Diagramm

Die FileImportActivity kümmert sich um die Auswahl von Dateien. Sie nutzt aFileChooser als externe Bibliothek.

5.1.4 DialogHelper

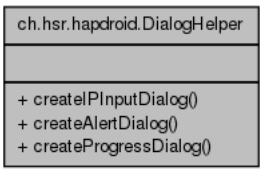

Abbildung 5.6: DialogHelper UML Diagramm

Der DialogHelper ist eine Klasse welche nur über statische Methoden zur Erstellung von Dialogen verfügt. Die Dialoge wurden ausgelagert um nicht die SplashActivity unnötig aufzublasen.

5.2 HAPdroid Service

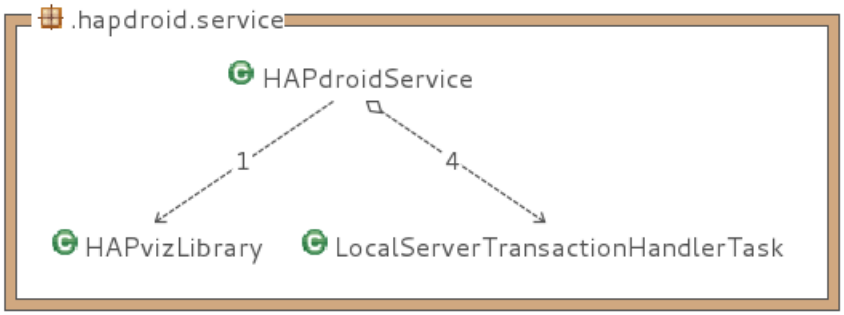

Abbildung 5.7: ch.hsr.hapdroid.service Paketübersicht

Im ch.hsr.hapdroid.service Paket befindet sich die HAPdroidService Klasse und weitere Klassen welche den Service in seiner Ausführung unterstützen.

5.2.1 HAPdroidService

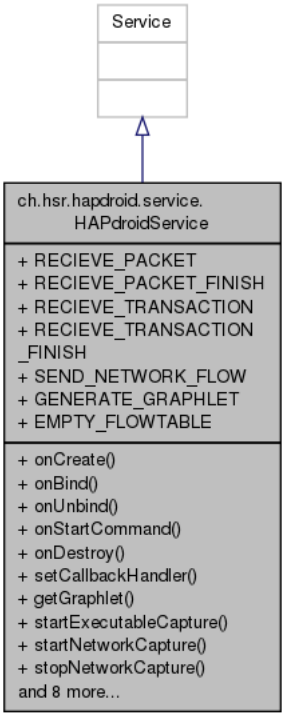

Abbildung 5.8: HAPdroidService UML Diagramm

Der HAPdroidService ermöglicht die Ausführung der Applikation im Hintergrund. Dies ist wichtig dafür, dass die Prozesse auch verwaltet werden können ohne dass die Applikation im Vordergrund läuft.

Der Service ist die generelle Verwaltungsklasse der Applikation. So nimmt er die Daten der Netdump Paketlooper Callback Funktion Prozesse entgegen, wandelt sie in Pakete um und füllt diese in die Flowtabelle ab. Zusätzlich verwaltet der Service den Lebenszyklus der netdump Prozesse und koordiniert die Bibliotheksaufrufe. Ebenfalls werden die Daten der Bibliothek vom Service an den **??** weitergeleitet.

5.2.2 HAPvizLibrary

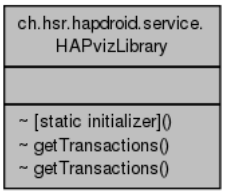

Abbildung 5.9: HAPvizLibrary UML Diagramm

Die HAPvizLibrary Klasse stellt die Java Schnittstelle zu den JNI Bibliotheksaufrufen dar. Selbst besitzt diese Klasse keine Aufgabe.

5.2.3 LocalServerTransactionHandlerTask

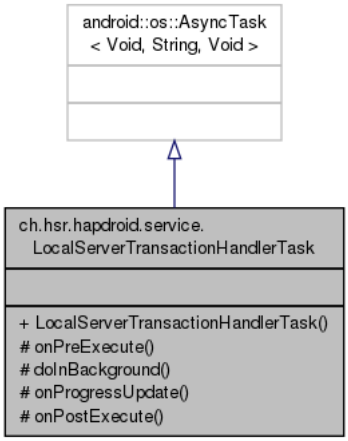

Abbildung 5.10: LocalServerTransactionHandlerTask UML Diagramm

Der LocalServerTransactionHandlerTask wird vom HAPdroidService zum asynchronen Empfangen der Informationen der Bibliothek benutzt. Er nutzt dabei Android Handler als Kommuni-

kationsmechanismus.

5.3 Netzwerkpakete sammeln

Für die Aufgabe des Sammeln der Netzwerkpakete wurde eine eigene kleine Ausführbare Datei geschrieben. Für den Zugriff auf die Netzwerkschnittstelle wird libpcap als externe Bibliothek genutzt. Libpcap bietet die Möglichkeit verschieden Netzwerkschnittstellen sowie auch .pcap Dateien für das Lesen von Netzwerkpaketen zu öffnen. Zur Nutzung von libpcap für das Paketsammeln sind die folgenden Schritte notwendig:

1. Öffnen einer Netzwerkschnittstelle mit *pcap_open_live* oder

   Öffnen einer .pcap Datei mit *pcap_open_offline*

2. Optionales kompilieren und setzen von Filter-Ausdrücken.

   Dazu ist es nötig die Ausdrücke mit *pcap_compile* zu kompilieren und mit *pcap_setfilter* zu setzen.

3. eine Looper Callback Funktion registrieren welche für jedes Paket aufgerufen wird.

   Dies wir mit dem Aufrunf von *pcap_loop* erreicht.

Die Callback Funktion von netdump durchläuft folgende Schritte:

1. IP Header Grösse bestimmen
2. Protokoll Header Grösse bestimmen
3. IP Payload Grösse bestimmen
4. Ausgabe von Protokoll, Quell-IP, Quellport, Ziel-IP, Zielport, IP ToS Feld, IP Header Grösse, IP Payload Grösse und Zeitstempel

Die Paketdaten werden auf einer Linie ausgegeben mit einem ':' als Trennzeichen für die einzelnen Elemente. Diese Art der Ausgabe wurde gewählt da es äusserst kompakt und leicht in die einzelnen Elemente aufzuteilen ist.

Der Quellcode der Callback Funktion *got_packet* ist nachfolgend noch angegeben.

Codelisting 5.1: Netdump Paketlooper Callback Funktion

```
void got_packet(u_char *args, const struct pcap_pkthdr *header,
    const u_char *packet) {

  /* declare pointers to packet headers */
  const struct sniff_ethernet *ethernet; /* The ethernet header [1] */
  const struct sniff_ip *ip; /* The IP header */
  const struct sniff_transp *transp; /* The transport protocol header */
  const struct sniff_tcp *tcp; /* The TCP protocol header */

  const struct timeval ts = (struct timeval) (header->ts);
```

```
int size_ip;
int size_transp;
int size_payload;

/* define ethernet header */
ethernet = (struct sniff_ethernet*) (packet);

/* define/compute ip header offset */
ip = (struct sniff_ip*) (packet + SIZE_ETHERNET);
size_ip = IP_HL(ip) * 4;
if (size_ip < 20) {
  fprintf(stderr,                                          , size_ip);
}

u_char ip_p = ip->ip_p;
u_char ip_tos = ip->ip_tos;

size_transp = 0;
switch(ip->ip_p) {
  case IPPROTO_TCP:
    tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
    size_transp = TH_OFF(tcp)*4;
    break;
  case IPPROTO_UDP:
    size_transp = 8;
    break;
  case IPPROTO_ICMP:
    size_transp = 8;
    break;
  case IPPROTO_IP:
    break;
  default:
    printf(                    );
    return;
}

transp = (struct sniff_transp*) (packet + SIZE_ETHERNET + size_ip);

/* get payload size */
int size_header = size_ip + size_transp;
size_payload = ntohs(ip->ip_len) - size_header;

/*
 * Print packet data.
 */
fdprintf(fd,          , dev, ip_p);
fdprintf(fd,         , inet_ntoa(ip->ip_src), ntohs(transp->h_sport));
fdprintf(fd,       );
fdprintf(fd,         , inet_ntoa(ip->ip_dst), ntohs(transp->h_dport));
fdprintf(fd,                          , ip_tos, size_header, size_payload,
    ts.tv_sec,
```

```
        ts.tv_usec);

    return;
}
```

5.4 Netzwerkdaten verarbeiten

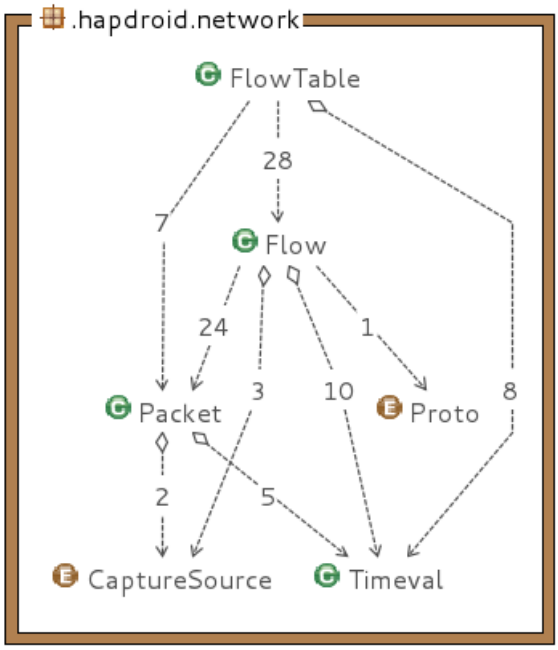

Abbildung 5.11: ch.hsr.hapdroid.network Paketübersicht

5.4.1 Packet

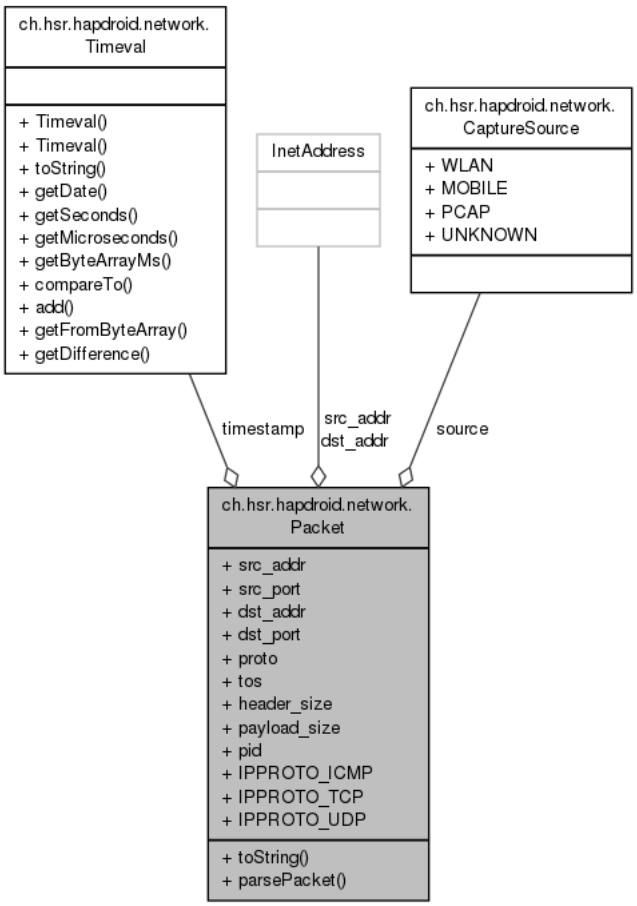

Abbildung 5.12: Packet UML Diagramm

Beim Packet handelt es sich um eine simple Datenklasse welche ein Netzwerkpaket darstellt. Diese Klasse enthält eine statische Methode für das Parsen der Informationen die der HAPdroidService von den netdump Prozessen erhält.

5.4.2 Proto

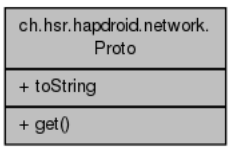

Abbildung 5.13: Proto UML Diagramm

Proto ist ein Enum welcher das Mapping der IP Protokollnummern auf ihre Textuelle Repräsentation erleichtert.

5.4.3 Timeval

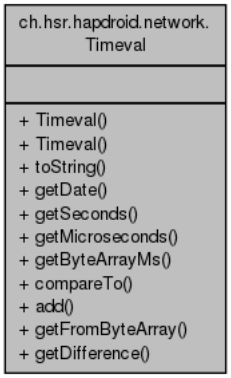

Abbildung 5.14: Timeval UML Diagramm

Diese Klasse stellt eine Java Repräsentation der Unix time dar. Zusätzlich verfügt sie über Hilfsfunktionen für das lesen und schreiben der Bit-Repräsentationen wie sie vom cflow Format benötigt werden. Für die Berechnungen der Flowdauer besitzt sie ebenfalls *getDifference* und *add* Funktionen.

5.4.4 Flow

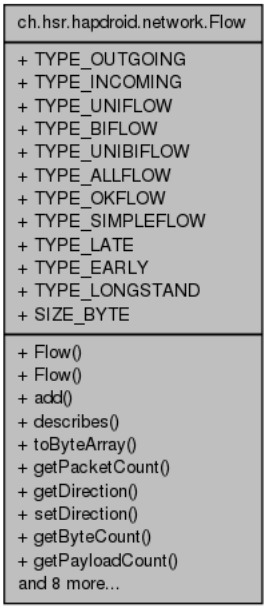

Abbildung 5.15: Packet UML Diagramm

Die Flow Klasse beinhaltet alle wichtigen Informationen eines Flows. Sie bietet Methoden für das Hinzufügen von Paketen und kann mittels *describes* überprüfen ob ein Paket durch den Flow beschrieben wird. Auch sind Methoden zum lesen und schreiben im cflow Format enthalten.

5.4.5 FlowTable

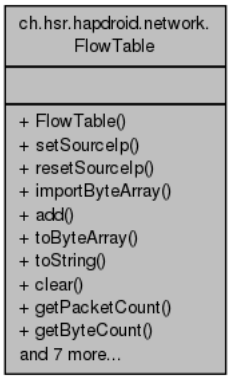

Abbildung 5.16: FlowTable UML Diagramm

Die FlowTable Klasse kümmert sich um das Hinzufügen von Paketen zu den Flows. Wird ein neues Netzwerkpaket empfangen, überprüft die FlowTable Klasse ob schon ein Flow existiert welcher das Paket beschreibt und fügt das Empfangene Paket dem Flow hinzu. Existiert noch kein solcher Flow, wird mit den Informationen des Paketes ein neuer Flow erstellt.

Die FlowTable hat auch Kenntnis über die lokalen IP Adressen des Gerätes und kann so die Richtung der Flows korrekt setzten.

5.4.6 CaptureSource

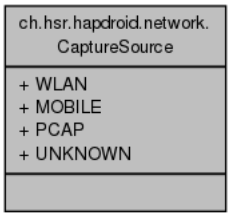

Abbildung 5.17: CaptureSource UML Diagramm

Der CaptureSource Enum ist ein Identifikator dafür von welcher Quelle ein bestimmtes Paket empfangen wurde. Dies ist für die Applikation wie sie im Moment implementiert ist nicht wichtig, schadet aber bestimmt nicht.

5.5 Libhapviz Bibliothek

Für die Portierung der HAPviewer Bibliothek wurde zu Beginn versucht nur die benötigten Quelldateien einzubinden. So konnte ein Grossteil der Dateien ignoriert werden. Besonders die verschiedenen Importfilter und die Dateien mit Code für die Benutzeroberfläche wurden stark reduziert. Als Importfilter sind für uns nur noch die *gfilter_cflow* Dateien von Bedeutung.

Nach der Reduktion der Dateien kann die Aufgabe der Anpassung der Bibliothek kann in die folgenden Schritte aufgeteilt werden:

Kompilierung: Der erste Schritt war die erfolgreiche Kompilierung der Bibliothek.

Umschreiben auf C++ streams: Wie im Design in libhapviz beschrieben wurde die Bibliothek auf C++ streams umgeschrieben um die Flexibilität der Schnittstelle zu steigern.

Transaktionen generieren: Der letzte und auch aufwändigste Schritt war es die Transaktionen zu generieren.

Generell wurde versucht die Änderungen auf ein Minimum zu beschränken.

5.5.1 Kompilierung

Obwohl Android auf Linux aufbaut und der HAPviewer Code für Linux geschrieben wurde, konnte der Quellcodes nicht ohne weiteres kompiliert werden. Zwei wesentliche Punkte erschwerten die Portierung:

Build System: HAPviewer nutzt CMake als Build System während Android ein eigenes Build System nutzt.

Boost Abhängigkeiten: HAPviewer nutzt verschiedene Module der Boost Bibliothek. Für diese existiert noch keine Portierung für Android.

Build System

Android nutzt ein eigenes Build System über .mk Dateien. Dabei handelt es sich grob um ein kleines GNU Makefile Fragment. Die Syntax erlaubt es Quelldateien in Module zu gruppieren. Module sind können static oder shared Bibliotheken sein, aber auch Executables. Für weitere Informationen zu den Android Makefiles wird auf die offizielle Dokumentation, welche Bestandteil des NDK ist, verwiesen.

Die Standard Android C++ System Laufzeit Bibliothek bietet keine Unterstützung für Exceptions und Laufzeit Typeninformationen sowie nur einen minimalen Satz der STL Bibliotheken. Weil HAPviewer aber Exceptions nutzt musste auf eine andere Laufzeit Bibliothek zurückgegriffen werden. Glücklicherweise existiert mit *gnustl* eine von Android unterstützte C++ Laufzeit welche auch Exceptions Unterstützung bietet.

Das Build System für sich erschwerte die Portierung nicht stark, das Zusammenspiel mit den Boost Abhängigkeiten war aber nicht sehr naheliegend. Ziel war es ein eigenes Android Makefile für die verwendeten Boost Module zu erstellen. So kann die Boost Bibliothek auf einfache Art und Weise aktualisiert werden. Dazu musste der Quellcode jedes Moduls identifiziert und in einem eigenen Android Makefile angegeben werden.

Um die Bibliothek erfolgreich kompilieren zu können musste zusätzlich die gimport_config, welche in HAPviewer von CMake genutzt wird, angepasst werden. Da wir lediglich cflow4 Dateiimport unterstützen, ist die neue gimport_config.h Datei äusserst minimal:

Codelisting 5.2: gimportconfig.h der HAPdroid Bibliothek

```
#ifndef GIMPORT_CONFIG_H
#define GIMPORT_CONFIG_H

#include
#include

std::vector<GFilter *> CImport::inputfilters;

unsigned int CImport::initInputfilters()
{
  inputfilters.push_back(new GFilter_cflow4);

  return inputfilters.size();
}

#endif /* GIMPORT_CONFIG_H */
```

Boost Abhängigkeiten

Für die von uns benutzten Dateien des HAPviewer werden die zwei Boost Module iostreams und regex benötigt. Diese sind als externe Module über Android Makefiles definiert. Damit diese Module sich aber auch kompilieren lassen muss Boost noch angepasst werden. Nämlich ist die endian Konfiguration noch nicht für Android ausgelegt. Dazu existiert ein Patch im Repository im Ordner *jni/external/patches*:

```
--- old/boost/detail/endian.hpp 2012-06-14 01:30:13.259839531 +0200
+++ boost/boost/detail/endian.hpp 2012-06-14 01:37:57.459818832 +0200
@@ -31,7 +31,7 @@
 // GNU libc offers the helpful header <endian.h> which defines
 // __BYTE_ORDER

-#if defined (__GLIBC__)
+#if defined (__GLIBC__) || defined(ANDROID)
 # include <endian.h>
 # if (__BYTE_ORDER == __LITTLE_ENDIAN)
 #  define BOOST_LITTLE_ENDIAN
```

5.5.2 Umschreiben auf C++ streams

Die nötigen Änderungen an den Header Dateien für die C++ streams Unterstützung sind wurden bereits in libhapviz besprochen. Durch die Änderung an der CImport Schnittstelle muss jeder verwendete Importfilter angepasst werden. Da wir lediglich cflow4 als Importfilter verwenden, halten sich diese Änderungen noch in Grenzen.

Boost Abhängigkeiten

Für die von uns benutzten Dateien des HAPviewer werden die zwei Boost Module iostreams und regex benötigt. Diese sind als externe Module über Android Makefiles definiert. Damit diese Module sich aber auch kompilieren lassen muss Boost noch angepasst werden. Nämlich ist die endian Konfiguration noch nicht für Android ausgelegt. Dazu existiert ein Patch im Repository im Ordner *jni/external/patches*:

Codelisting 5.3: Boost endian Patch für Android

```
--- old/boost/detail/endian.hpp 2012-06-14 01:30:13.259839531 +0200
+++ boost/boost/detail/endian.hpp 2012-06-14 01:37:57.459818832 +0200
@@ -31,7 +31,7 @@
 // GNU libc offers the helpful header <endian.h> which defines
 // __BYTE_ORDER

-#if defined (__GLIBC__)
+#if defined (__GLIBC__) || defined(ANDROID)
 # include <endian.h>
 # if (__BYTE_ORDER == __LITTLE_ENDIAN)
 #  define BOOST_LITTLE_ENDIAN
```

5.5.3 Umschreiben auf C++ streams

Die nötigen Änderungen an den Header Dateien für die C++ streams Unterstützung sind wurden bereits in libhapviz besprochen. Durch die Änderung an der CImport Schnittstelle muss jeder verwendete Importfilter angepasst werden. Da wir lediglich cflow4 als Importfilter verwenden, halten sich diese Änderungen noch in Grenzen. Sie lassen sich in folgende Schritte zusammenfassen:

Änderung der Methodensignatur: Die Methodensignatur wurde von *read_file* zu *read_stream* geändert. Neu wird zusätzlich die Anzahl an Flows übergeben. Dies ist etwas Unglücklich und wäre bei einem Stream eigentlich auch zu vermeiden. Es wurde aber so implementiert, da diese Kenntnis im bestehenden Code verwendet wird und so möglichst nahe am bestehenden Code geblieben werden konnte.

Entfernung des Dateiöffnungscode: In der alten Version wurde die Datei innerhalb der Methode geöffnet und wieder geschlossen. Dieser Code konnte komplett entfernt werden

Der Einfachheit halber wurde aller cflow6 Code entfernt. Ebenfalls das der Code welcher sich mit dem Schreiben einer Datei befasst wurde entfernt.

Um die Korrektheit des Refactorings zu gewährleisten wurde zum Schluss der HAPviewer Quellcode mit dem geänderte Code ersetzt und erfolgreich getestet.

5.5.4 Transaktionen generieren

Um die Transaktionen generieren zu können wurden Anpassungen an den ggraph.cpp und gimport.cpp Dateien gemacht. Die Änderungen an den Header Dateien wurden bereits in **??** im Design Abschnitt besprochen.

CImport Anpassungen

Für das Schreiben der Transaktionen wurde versucht möglichst viel Quellcode der *cflow2hpg* Funktion zu verwenden. Diese Funktion erstellt aus der Liste von Flows eine Graphen Repräsentation im proprietären HPG Format. Hier geschieht die Erstellung der Summen-Knoten. Die Funktion ist im Quellcode wie folgt beschrieben.

Codelisting 5.4: Beschreibung der cflow2hpg Funktion

```
/**
 *  Transform flow data (from active_flowlist) of a single local hosts into
      host profile graphlet data.
 *
 *  If configured then transformation also includes role summarization (
     configurable per role type).
 *
 *  Precondition: the flow data stored in flowlist has to be sorted such
     that all flows belonging to the
 *  same localIP are grouped together. This is essential to separate
     graphlets from each other.
 *
 *  Result: output file containing graphlet database (= collection of
     graphlets described in binary form)
 *
 *  Overview:
 *
 *  (I) Initialize
 *  (II) Role Identification
 *    - for all flows identify candidates for client and server role
 *    - prune client and server candidates involved in roles not meeting
     requirements
 *    - for all flows identify candidates for p2p flows
 *    - from all p2p candidate flows identify candidate p2p roles
 *    - prune p2p candidates involved in roles not meeting requirements
 *    - rate roles to be able to resolve conflicts in step III
 *    - create sub-roles, used for desummarization
 *    - convert multi summary node desummarization to role desummarization
```

```
 *  (III) Transform flows & roles to binary "hpg" graphlets. For each
    localIP do
 *    - Firstly, detect and resolve role conflicts
 *    - Add all not summarized flows to graphlet
 *    - Then, add client/server/p2p role summaries to graphlet(node
    desummarization is done here)
 *    - Finally, create binary output data from edge information collected
 *
 *    \exception std::string Errorstring
 *
 */
```

Für das Erstellen der Transaktionen ist es nötig den Schritt drei, genauer der finale Schritt der Erstellung der binären Ausgabe, zu ersetzen. Zu diesem Zweck wurde eine neue Funktion *prepare_graphlet* definiert welche die Schritte eins, zwei und Teile des dritten Schrittes durchführt. Die Funktion *cflow2hpg* ruft die neue Funktion *prepare_graphlet* auf und erstellt danach die binäre Ausgabe des HPG Formates wie bisher im HAPviewer Quellcode.

Codelisting 5.5: Neue cflow2hpg Funktion

```
/**
 *  Create binary "hpg" output data from edge information collected in
 *  prepae_graphlet.
 *
 *  \exception std::string Errorstring
 *
 */
void CImport::cflow2hpg() {
  std::ofstream outfs;
  try {
    util::open_outfile(outfs, hpg_filename);
  } catch (string & e) {
    stringstream error;
    error <<                                  << hpg_filename;
    throw error.str();
  }

  CRoleMembership roleMembership; // Manages groups of hosts having same
      role membership set
  CGraphlet * graphlet = new CGraphlet(outfs, roleMembership);

  prepare_graphlet(graphlet, roleMembership);

  // Finalize current graphlet and prepare for a next one
  // ============================================================
  // Next flow belongs to new localIP: thus, finalize current host graphlet
      .
  //
  graphlet->finalize_graphlet(0);

  if (hap4nfsen) {
```

```
      nodeInfos = graphlet->nodeInfos;
    }

    outfs.close();
    delete graphlet;
  }
```

Die bisherige *read_file* Methode der CImport Klasse iteriert durch alle verfügbaren Importfilter und leitet den Methodenaufruf weiter. Diese wird umbenannt auf *read_stream* und eine neue *read_file* Methode wird definiert welche einen std::ifstream initialisiert und an *read_stream* weitergibt.

CGraphlet Anpassungen

Bei der CGraphlet Klasse wurde eine neue Methode *write_transactions* definiert. Diese verhaltet sich in der Logik wie in C++ streams beschrieben.

5.6 HAP Graph

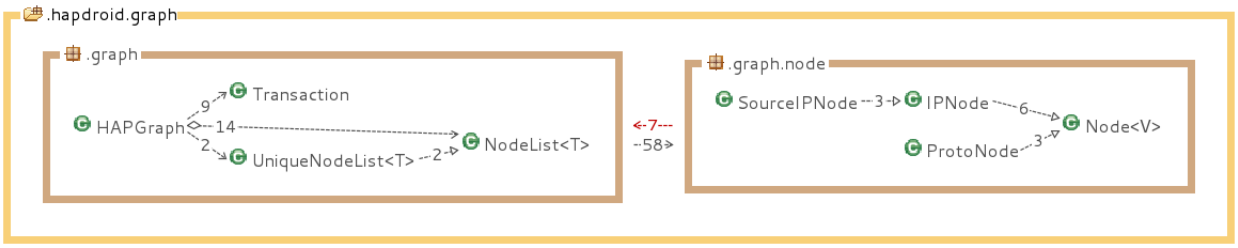

Abbildung 5.18: ch.hsr.hapdroid.graph Paketübersicht

5.6.1 HAPGraph

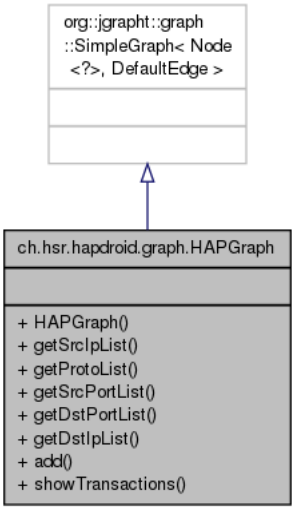

Abbildung 5.19: HAPGraph UML Diagramm

Die HAPGraph Klasse stellt die interne Graphen-Repräsentation dar. Sie enthält eine Methode zum Hinzufügen von Transaktionen. Für jede Partition des HAPGraphlet wird intern eine Liste geführt. Dies erleichtert den Aufbau der Benutzeroberfläche.

Bei den internen Listen handelt es sich um Instanzen der NodeList Klasse. Für die Quell-IP und Protokoll Partitionen wird dabei eine UniqueNodeList Instanz verwendet.

5.6.2 Transaction

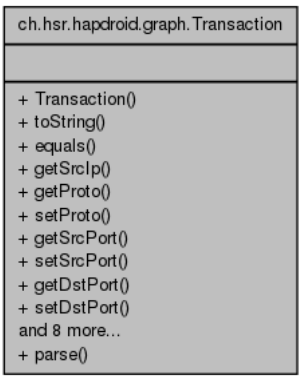

Abbildung 5.20: Transaction UML Diagramm

Die Transaction Klasse widerspiegelt die in der Analyse beschriebenen Kontextverlust.

5.6.3 NodeList

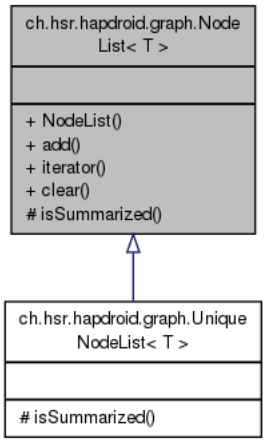

Abbildung 5.21: NodeList UML Diagramm

Die NodeList beinhaltet eine Liste von Nodes. Für summarisierte Nodes verhaltet sich die Klasse wie ein Set, für nicht summarisierte Nodes hingegen wie eine List.

5.6.4 UniqueNodeList

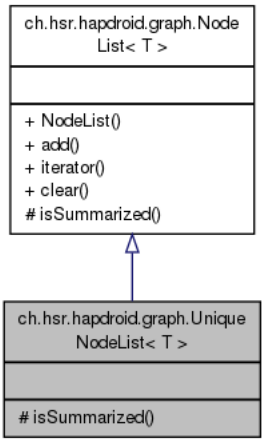

Abbildung 5.22: UniqueNodeList UML Diagramm

Die UniqueNodeList unterscheidet sich von der NodeList insofern dass ein Knoten exakt einmal vorkommen kann, auch wenn dieser kein Summen-Knoten ist. Dieses Verhalten wird für die Quell-IP und Protokoll Partitionen benötigt.

5.6.5 Node

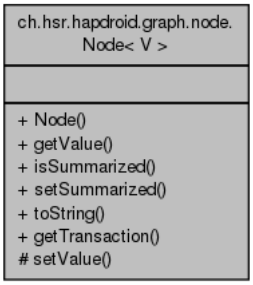

Abbildung 5.23: Node UML Diagramm

Die Node Klasse repräsentiert einen Knoten im HAPGraphlet. Sie besitzt die Möglichkeit als summarisiert gekennzeichnet zu werden. Für die Möglichkeit zusätzliche Informationen abzurufen, beispielsweise über ein Kontextmenü, enthält sie eine Referenz auf die Transaktion zu welcher die Node Instanz gehört.

5.7 Benutzeroberfläche

5.7.1 Aufbau

Die GUI-Elemente werden von AndEngine in einer Art Tree gehalten, sodass jedes Element, dass gezeichnet werden soll ein Parent-Objekt besitzen muss. An der Wurzel dieser Tree-Struktur befindet sich die Scene Klasse, in unserem Fall heisst diese Graphlet. Die Schwierigkeit bestand nun aus dem Netzwerkverkehr, welche aus Datensicht in einem Graphen hinterlegt sind (HAPgraph), in GUI-Elemente umzuwandeln und in einer Tree-Datenstruktur abzubilden. Dafür wurde ein Grundgerüst erstellt, welches einen möglichst effizienten und Ressourcen-schonenden Aufbau der Darstellung ermöglichen soll.

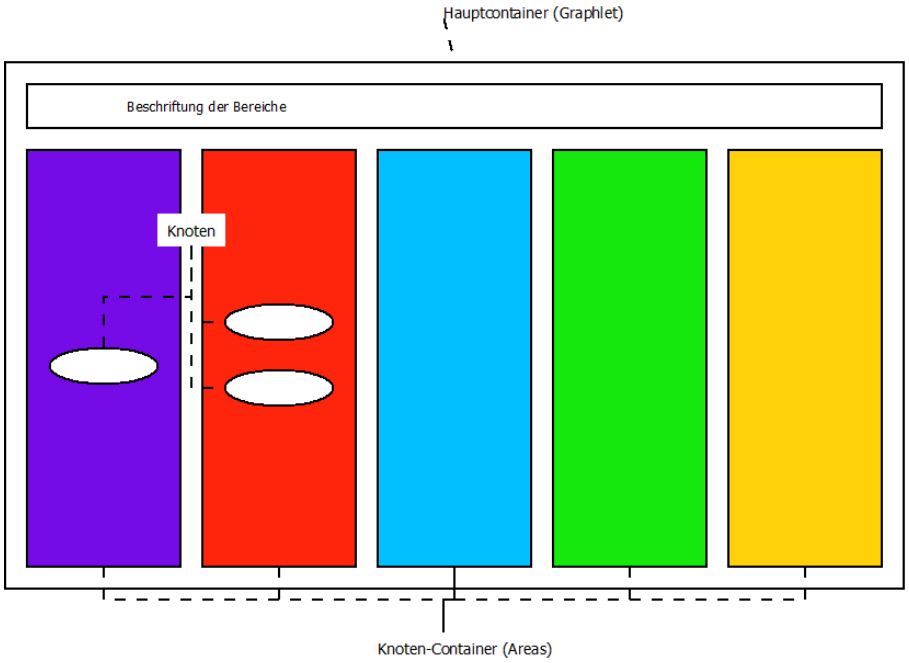

Abbildung 5.24: Grafischer Aufbau der GUI-Elemente (ohne Edges)

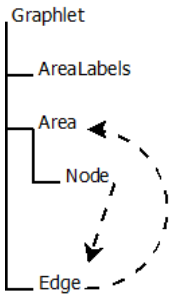

Abbildung 5.25: Logischer Aufbau der GUI-Elemente

Hauptcontainer (Graphlet): Da AndEngine die Scene Klasse bereits als Root-Element des Objekt-Trees vorsieht, haben wir diese Klasse nach unseren Bedürfnissen erweitert. Die so entstandene Graphlet Klasse ist verantwortlich für die Abbildung des HAPgraphen in GUI-Objekte und hält ausserdem die dafür benötigten Container als Child-Objekte.

Diese Container, in Form der Area Klasse, enthalten sämtliche Nodes ihrer Kategorie (z.B. source Port) als Child-Objekte.

Ebenfalls direkt dem Graphlet "angehängt" sind sämtliche Edge-Instanzen. Dies verringert die Tiefe des Objekt-Trees und bringt darum bei der Berechnung der beiden Endpunkte der Edges einen Performance-Gewinn. Grund dafür ist, dass die Koordinaten im Normalfall relativ zum Parent-Objekt sind und die Nodes jedoch die entsprechende Area als Parent haben (Grund dafür siehe Beschreibung der Area). Darum müssen die Node-Koordinaten, welche somit relativ zur Area sind, erst in Scene-Koordinaten umgewandelt werden, um die effektiven Endpunkte der Edges zu erhalten. Bei dieser Berechnung muss AndEngine für jeden benötigten Punkt, von der jeweiligen Objekt-Instanz, den Objekt-Tree bis zu seinem Root-Objekt durchlaufen, um die relativen Koordinaten in absolute umzuwandeln.

Knoten-Container (Area): Die Area Klasse hat die Aufgabe alle GUI-Nodes einer bestimmten Kategorie zu verwalten, entsprechend auszurichten und das vertikale Scrolling der Area, gemäss Design, umzusetzen. Ausserdem hält die Area die Referenzen zu allen Edges, welche eine Verbindung zu einem ihrer Nodes haben. Dies bringt den Vorteil, dass wenn eine bestimmte Area "gescrollt" wird, jeweils bekannt ist, welche Edges ihre Endpunkte neu berechnen müssen. Wäre dies nicht so implementiert, müssten jeweils sämtliche Edges des ganzen Graphlets neu gezeichnet werden, wenn die Position einer einzigen Area verändert wird.

5.7.2 Graphlet

5.7.3 Anpassungen an der Multitouchextension von AndEngine

Bei der Implementierung der Pinch-To-Zoom Funktionalität sind immer wieder "Index out of Bounds"-Exceptions aufgetreten, welche wir nicht auf unseren Programmcode zurückführen konnten. Eine Analyse des Fehlers ergab, dass Android bei Multitouch-Events manchmal im TouchEvent-Objekt nicht wie erwartet mehrere Koordinaten, sondern nur eine einzige enthält. Die Klasse PinchZoomDetector.java der Multitouchextension von AndEngine überprüft nicht, ob wenigstens zwei Koordinaten vorhanden sind, sondern versucht direkt via ArrayIndex auf die beiden erwarteten Koordinaten zuzugreifen.

Das Problem wurde zwar mit einer einfachen Überprüfung der Array-Grösse gelöst, jedoch musste der Code der Multitouchextension angepasst werden. Der so entstandene Patch wurde wurde dem Entwickler entsprechend auf der Projekt-Hosting-Plattform übermittelt. [1]

1 http://code.google.com/p/andenginemultitouchextension/issues/detail?id=3

Codelisting 5.6: Änderungen an der Methode calculatePointerDistance() entsprechend mit - oder + gekennzeichnet.

```
--- a/src/org/anddev/andengine/extension/input/touch/detector/
    PinchZoomDetector.java
+++ b/src/org/anddev/andengine/extension/input/touch/detector/
    PinchZoomDetector.java
@@ -103,10 +103,14 @@
    * Calculate the euclidian distance between the first two fingers.
    */
   private static float calculatePointerDistance(final MotionEvent
       pMotionEvent) {
-    final float x = pMotionEvent.getX(0) - pMotionEvent.getX(1);
-    final float y = pMotionEvent.getY(0) - pMotionEvent.getY(1);
-    return FloatMath.sqrt(x * x + y * y);
- }
+    if(pMotionEvent.getPointerCount()>=2){
+       final float x = pMotionEvent.getX(0) - pMotionEvent.getX(1);
+      final float y = pMotionEvent.getY(0) - pMotionEvent.getY(1);
+      return FloatMath.sqrt(x * x + y * y);
+     }
+     return 0;
+ }
+
```

5.8 Tests

Das Testing unserer Applikation erwies sich als äusserst schwierig. Zum einen ist Android nicht für seine gute Testbarkeit bekannt, zum anderen gab es kaum Möglichkeiten für Unit-Testing da eine Logische Aktion der Applikation viele Zwischenschritte benötigt. Man nehme zum Beispiel das Generieren von cflow Daten. Es handelt sich logisch um eine Aktion, jedoch sind intern folgende Schritte notwendig:

- Laden der .pcap Datei
- Netdump sendet die Pakete an den Service
- Der Service erstellt die Flowtabelle
- Die Flowtabelle wird im cflow Format geschrieben

Beim testen der Umwandlung vom pcap ins cflow Format werden also alle oben genannten Schritte getestet. Diese Tests existieren zwar aber der grosse Nachteil ist, dass es in einem Fehlerfall schwierig ist, zuzuordnen wo der Fehler auftrat.

Dummerweise ist das Testen der Umwandlung vom pcap ins cflow Format die Einzige Operation unserer Applikation die sich gut Testen lässt. Die Testbarkeit der anderen Kernaspekte der Applikation werden im folgenden kurz umschrieben.

Benutzeroberfläche: Eine wichtige Problemstellung in der Benutzeroberfläche wäre die nicht überlappende Anordnung der Knoten des Graphen. Grafische Darstellungen sind aber kaum automatisiert testbar.

Transaktionen: Die Transaktionen entspringen der grafischen Repräsentation des HAPGraphlets. Genau so wie die Benutzeroberfläche sind die Transaktionen aufgrund ihrer grafischen Natur kaum automatisiert testbar.

Aus diesen Gründen konnte die Korrektheit der Transaktionen nicht sichergestellt werden. Zudem ist bei manuellen Tests aufgefallen dass die Summarisierungen Teilweise sehr falsch sind aber es konnte in der verbleibenden Zeit leider keine Erklärung dafür gefunden werden.

6 Schlussfolgerungen

6.1 Zusammenfassung

Ziel der Arbeit war es die Portierung der Software HAPviewer auf das Mobile Betriebszstem Android. Dabei soll es möglich sein den eigenen Datenverkehr mit zuschneiden und anzuzeigen. Als Bestandteil dieser Portierung war die Neugestaltung der Benutzeroberfläche zur verbesserten Darstellung eines grossen Graphen auf einem kleinen Bildschirm.

Die Portierung wurde dadurch erschwert, dass für den Benutzer eines mobilen Gerätes nicht nur Hosts sondern auch Applikationen von Interesse sind. Durch das veränderte Interesse ändert sich auch die Qualifizierung von schädlichem Datenverkehr. Die HAPviewer Software ist aber nicht für diesen Fall ausgelegt. Aus diesem Grund konnte der bestehende Quellcode nicht komplett übernommen werden.

Für die Neugestaltung der Benutzeroberfläche wurde der Ansatz eines dynamischen Graphen und dynamischer desummarisierung der Knoten verfolgt. Dieses Bedienkonzept ist aber mit den Standard Android GUI Elementen kaum umsetzbar und die Benutzeroberfläche wurde deshalb mit AndEngine Entwickelt. AndEngine ist eine, für die Entwicklung von Spielen optimierte, Bibliothek welche viele Möglichkeiten in der Gestaltung der Benutzeroberfläche bietet.

Zahlreiche Probleme in der Portierung und Entwicklung der Benutzeroberfläche verlangsamten die Entwicklung der Applikation aber stark. Aus diesem Grund wurde das Ziel einer lauffähigen Applikation mit Funktionsumfang vergleichbar zu HAPviewer nur Teilweise erreicht. Zum Beispiel konnte die Korrektheit der Daten des Graphen nicht garantiert werden. Ebenfalls wurde die Desummarisierung der Knoten nicht implementiert und Zuordnung der Pakete zu den Ursprünglichen Applikationen ist nicht vorhanden.

6.2 Mögliche Erweiterungen

Es existieren zahlreiche mögliche Erweiterung. Die wichtigsten sind:

Korrektheit der Daten: Die wichtigste Erweiterung ist die Korrektheit des Graphen sicherzustellen.

Zuordnung Applikationen: Eine weitere wichtige Erweiterung ist die Zuordnung der Verbindungen zu der Applikation welche die Verbindung aufgebaut hat.

Desummarisierung: Eine sehr Nützliche Erweiterung ist die dynamische Desummarisierung der Summen-Knoten. So können auf leichte Weise weitere Informationen aus dem Graphen gewonnen werden.

Kollisionsauflösung des Graphen: Wenn sich viele Kanten überlagern, sind die Schriftzüge kaum mehr lesbar. Diese kann man in einer Erweiterung beheben durch eine geeignete Kollisionsauflösung der Kanten im Graphen.

Verbindungen nach Zeitfenster/Applikationen auswählen: Werden sehr viele Verbindungen angezeigt, kann man die Übersichtlichkeit auch mit einem sehr guten Bedienkonzept nicht sicherstellen. Als Lösung dieser Problemstellung kann man die Verbindungen nach Zeitfenster/Applikation einschränken.

Teil II

Projekt Bericht

7 Projektplan

Als Entwicklungsmodell haben wir eine reduzierte Form von SCRUM gewählt. Mittels User Stories wird bei SCRUM der Schwerpunkt auf die Möglichkeiten der Benutzer der zu entwickelnden Applikation gelegt. Dies erschien uns als sehr passendes Modell für eine Android Applikation. Besonders bei Applikationen für mobile Geräte stehen die Möglichkeiten und Aktionen des Benutzers im Zentrum. Da unser Team aber lediglich aus zwei Personen besteht erschien uns die Benutzung aller Aspekte von SCRUM als unnötiger Aufwand und somit unvereinbar mit der angestrebten agilen Entwicklung.

Die Unterschiede unseres Entwicklungsmodell zum klassischen SCRUM lassen sich wie folgt zusammenfassen:

- Die Rollen wurden auf Entwicklungsteam und Customer reduziert
- Kein Daily SCRUM
- Sprint Review und Sprint Planungsmeeting werden zu einem Meeting mit Herrn Glatz zusammengenommen

Somit lässt sich das verwendete Entwicklungsmodell “SCRUM auf das nötigste reduziert” umschreiben.

7.1 Meilensteine

Scrum sieht vor, dass das Ende jedes Sprints einem Meilenstein entspricht, was wir entsprechend berücksichtigt und bei den Projektphasen beschrieben haben. Alle weiteren Meilensteine sind hier aufgelistet.

Datum	Meilenstein
23.2.2012	Kick-Off Meeting mit dem Betreuer
8.6.2012	Abstract an den Betreuer schicken
13.6.2012	Der Betreuer gibt das Abstract dem Sekretariat weiter
15.6.2012	Abgabe der Arbeit an den Betreuer

Tabelle 7.1: Vorgegebene Meilensteine

7.2 Projektphasen

7.2.1 Initiale Konfiguration

Endet am: 1.3.2012

Die ersten zwei Wochen des Projekts wurden zur Konfiguration des Entwicklungs-Servers und für das Aufsetzen der Entwicklungsumgebung genutzt und wurden aus diesem Grund noch nicht zu den Sprints gezählt.

7.2.2 Sprint 1

Endet am: 15.3.2012

User Stories

- Als Benutzer kann ich Netzwerkverkehr anschauen

Retrospektive

Positives:

- Fortschritt der Applikation ist sichtbar
- GUI Konzept existiert
- Bessere Vorstellung der Herangehensweise
- Konstruktive Gespräche mit Herrn Glatz

Negatives:

- Zeiterfassung nicht fortlaufend –> Burndown sieht seltsam aus
- Anfangs Schwierigkeiten den Umfang der Arbeit einzuschätzen
- Wenig Kommunikation bezüglich Stand der Dinge, Arbeitsvorschritt
- User Stories sind tendenziell zu gross

Lessons Learned:

- Fortlaufende Zeiterfassung
- Mehr Zeit bei Sprint Planung für das Aufbrechen der User Stories in Tasks im Team aufwenden

7.2.3 Sprint 2

Endet am: 29.3.2012

User Stories

- Als Benutzer kann ich Netzwerkverkehr im Hintergrund aufzeichnen

Retrospektive

Positives:

- Gute Design-Diskussionen
- Viele neue Erkenntnisse gewonnen
- Die Architektur erwies sich als flexibel genug, um mit der geänderten Ausgangslage (Abhängigkeiten) umzugehen.

Negatives:

- Wenig sichtbarer Output
- Das Ziel bezüglich Prototypen, welcher durch alle Schichten funktioniert, war zu hoch gesteckt.
- Frühzeitige Analyse der Library hätte einigen Aufwand erspart.
- Recherchen benötigten viel Zeit.

Lessons Learned:

- In evtl. vorhandene Vorarbeiten sollte man sich möglichst früh und genau einlesen.

7.2.4 Sprint 3

Endet am: 12.4.2012

User Stories

- Als Benutzer kann ich Netzwerkverkehr nach Kategorien unterscheiden

Retrospektive

Positives:

- Anspruchsvolle Problemstellungen
- Detaillierte Analyse des bestehenden Codes
- Konkrete Lösungsansätze gefunden für:
  - Flow in Transactions umwandeln
  - Einbindung welcher Teile von HAPviewer / libhapviz
  - Abläufe der Benutzerführung

Negatives:

- Wenig sichtbarer Output
- Dokumentation und Administratives etwas vernachlässigt
- Code-Analyse und Reverse-Engineering von HAPviewer war sehr zeitraubend.
- Initialaufwand für die Einbindung von HAPviewer unterschätzt.
- Libhapviz eignet sich nicht als generische Schnittstelle und kann daher nicht verwendet werden.
- Eigene Schnittstelle für HAPviewer kostet zusätzlich Zeit.
- Erschwerte Bedingungen bezüglich GUI-Entwicklung:
  - Analyse von Libgraphviz hat viel Aufwand gekostet, lässt sich aber nicht verwenden.
  - AndEngine ist nicht dokumentiert.
  - AndEngine deckt die Anforderungen nicht komplett ab. Eigene Erweiterungen sind notwendig.

Lessons Learned:

- Frühere Analyse der Schnittstellen (libhapviz)

- Frühere Einarbeitung in Frameworks (AndEngine) um Aufwand besser abschätzen zu können.
- Klare Sprint Zielsetzungen (Tasks) verfassen

7.2.5 Sprint 4

Endet am: 26.4.2012

User Stories

- Als Benutzer kann ich Netzwerkverkehr als Graph anzeigen

Retrospektive

Positives:

- Einblick in AndEngine vertieft
- Library Schnittstelle flexibler da nun Streams verwendet werden können
- Weitere Herangehensweise konnte festgelegt werden (Transactions)

Negatives:

- Erheblicher mehr Aufwand da noch keine Portierungen für libhapviz Abhängigkeiten existieren (Boost)
- Weiterhin langsames Vorankommen mit AndEngine, aufgrund fehlender Dokumentation
- Da Mehraufwand in der Programmierung Dokumentation und Administratives zurückgestellt

Lessons Learned:

- Wechsel von Komponenten/Abhängigkeiten zu späterem Zeitpunkt ist unrealistisch (AndEngine)
- Zusätzliche Abhängigkeiten können wesentlich mehr Aufwand generieren als erst vermutet (z.B. Kompilierung von Boost unter Android)

7.2.6 Sprint 5

Endet am: 10.5.2012

User Stories

- Als Benutzer kann ich Netzwerkverkehr in Transaktions-Darstellung anschauen

Retrospektive

Positives:

- Die Kommunikation der unabhängig voneinander entwickelten Teile funktionierte auf Anhieb
- GUI entwickelt sich zwar langsam aber stetig

Negatives:

- Binäres Dateiformat schreiben mit Java ist umständlich
- Fehler in der Datenverarbeitung (libhapviz)
- Architektur von AndEngine verlangt umständliche Berechnungen
- GUI-Controll (Graphlet) verlangt Implementierung grundlegender Funktionen (onDraw, Scrolling) was zusätzlicher Aufwand bedeutet

Lessons Learned:

- Java eignet sich nur schlecht für das schreiben von proprietären Binärformaten
- Die Architektur von Abhängigkeiten hat Einfluss auf die eigene Architektur
- Implementierung eigener GUI-Controlls benötigt Programmieraufwand (low-level), ab von der eigentlichen Problemdomäne.

7.2.7 Sprint 6

Endet am: 24.5.2012

User Stories

- Als Benutzer kann ich sinnvoll durch den Netzwerk Graph navigieren

Retrospektive

Positives:

- Rücksprachen mit Dozent sehr konstruktiv und pragmatisch
- Programm und Problemdomäne ist sehr interessant und bietet viel Raum zur Weiterentwicklung

Negatives:

- Fokussierte Problemlösung (cflow Format, Dynamische Edges) in der Entwicklung binden Ressourcen und erschweren neben-läufige Aufgaben wie Dokumentation

Lessons Learned:

- Im Verlauf des Projekts können weitere gute Ideen in die Entwicklung einfliessen, welche man zu Beginn noch nicht hatte.
- Nicht alle guten Ideen lassen sich mit gerechtfertigtem Aufwand umsetzten. :-)

7.2.8 Sprint 7

Endet am: 7.6.2012

User Stories

- Als Benutzer kann ich Netzwerkverkehr abspeichern
- Als Benutzer kann ich gespeicherten Datenverkehr öffnen

Retrospektive

Positives:

- Reifegrad der Applikation verbessert sich zusehends
- Erfolgserlebnisse mit Problemlösung (cflow Import, dynamische Nodegrösse)
- Work-around mit Splashscreen zur Erkennung der Auflösung wirkt nicht störend

Negatives:

- Viel Nachholbedarf bei Dokumentation und Administrativem
- Entwicklungsstand würde motivieren weitere Features umzusetzen, jedoch reicht die Zeit nicht aus

Lessons Learned:

- Features zur Verbesserung der Benutzerfreundlichkeit sind oft aufwendig und nicht immer tragbar.

7.2.9 Sprint 8

Endet am: 15.6.2012

User Stories

- Keine (Dokumentation und Bugfixing)

Retrospektive

Positives:

- Endspurt!
- Gutes Teamwork

Negatives:

- Zu wenige Iterationen bei der Dokumentation
- Grosse Müdigkeit

Lessons Learned:

- Konsequentes Nachführen der Dokumentation, sodass am Schluss Zeit für die Überarbeitung bleibt.

7.3 Risikoanalyse

Hier sollen die wichtigsten Risiken aufgelistet und werden. Es wurde mit dem Betreuer vereinbart, dass nur die projektspezifischen Risiken erfasst werden und Allgemeinposten vermieden werden sollten.

Nr.	Risiko	Massnahmen	Aufwand	Eintritts-wahrschein-lichkeit	Ergebnis
1	Mehraufwand wegen Problemen mit Vorarbeiten	Frühes Einarbeiten in die Materie und Vorarbeiten	80h	0.75	60
2	Mehraufwand wegen Problemen mit externen Abhängigkeiten	Gute Evaluation von neuen externen Abhängigkeiten	80h	0.75	60
3	Fehlende Zeit für die Implementierung optionaler Funktionen	Aufwandsschätzungen nach Scrum machen	-	0.5	-

Tabelle 7.2: Risikoanalyse

7.4 Arbeitsaufteilung

In diesem Abschnitt sind die Themen und Arbeiten aufgelistet, für welche ein bestimmtes Teammitglied verantwortlich war. Alle Arbeiten welche nicht explizit erwähnt werden, sind im Team bearbeitet worden. Jedes Teammitglied hat jeweils die Dokumentation über seinen Teil selbst verfasst.

Dominik Spengler :

- Einrichtung und Administration des Projekt-Servers
- Anpassungen an Libhapviz
- Entwicklung des Android Services für die App
- Implementierung der Inter-Prozesskommunikation
- Parsing der Netzwerkdaten
- Import/Export Feature
- Testing

Remo Egli :

- Entwicklung des GUI Konzeptes
- Entwicklung der Graphlet Visualisierung
- Evaluierung der Graph Datenstruktur
- Abbildung der Datenstruktur in GUI-Elemente
- Schreiben der Protokolle
- Schreiben des Projektberichts
- Korrespondenz

8 Projektsetup

8.1 SDK

Für die Entwicklung von Android Apps werden umfassende Entwicklungswerkzeuge bereitgestellt. Das Setup ist deshalb auch sehr detailliert auf der Developer Site von Android[1] beschrieben, weshalb wir uns vollständig an jene Anleitung[2] gehalten haben.
Folgende Versionen wurden für das Projekt eingesetzt:

- Eclipse Indigo, Version 3.7.2[3]
- Android SDK, Revision 16[4]
- Android ADT for Eclipse, Version 16.0.1[5]
- Adnroid NDK, Revision 7b[6]

1 http://developer.android.com
2 http://developer.android.com/sdk/installing.html
3 http://www.eclipse.org/downloads/
4 http://developer.android.com/sdk/index.html
5 http://developer.android.com/sdk/eclipse-adt.html
6 http://developer.android.com/sdk/ndk/index.html

8.2 Server

Ein Server wurde eingerichtet um uns bei der Entwicklung bestmöglich zu unterstützen. Der Server musste drei Aufgabenbereiche abdecken:

- Quellcode Versionsverwaltung
- System zur kontinuierlichen Integration
- Zeiterfassung

8.2.1 Git

Als Versionsverwaltungssystem wurde Git[1] gewählt. Git wurde gewählt aufgrund der Einfachheit branches zu erstellen und wieder zu mergen. Ebenfalls zeigt die Erfahrung dass Git zu deutlich weniger Probleme bei der Arbeit im Team an der selben Codebase führt. Dabei haben wir zwei unabhängige Repositories eingerichtet. Eines für die Dokumentation und eines für den Quellcode.

Bei der Entwicklung am Quellcode mit Git wird der Ansatz der Feature Branches[2] verfolgt. Grundlegend existieren zwei Branches:

master: Der master Branch wird für voll funktionsfähige Releases genutzt. Er wird am Ende jedes Sprints mit dem development Branch gemerged.

development: Im development Branch findet die Entwicklung statt. Dabei wird für jedes neue Feature und jeden Bugfix ein weiterer Branch erstellt.

Dieses Vorgehen führt dazu, dass zu einem bestimmten Zeitpunkt niemals mehr als vier Branches existieren: master, development, feature von Herr Egli, feature von Herr Spengler. Damit beim push auf den Server die Revisions-History komplett erhalten bleibt, wird nach folgender Methodik gearbeitet:

Codelisting 8.1: Git Feature Branch Push

```
$ git pull origin development
$ git checkout -b featureXY development
...
<Arbeit am featureXY>
...
$ git pull origin development
$ git checkout development
$ git merge --no-ff featureXY
...
```

1 http://git-scm.com/
2 http://martinfowler.com/bliki/FeatureBranch.html

```
Updating ea1b82a..05e9557
...
$ git branch -d featureXY
...
Deleted branch featureXY
...
$ git push origin development
```

Nach dieser Vorgehensweise entsteht ein Branch Graph nach folgendem Vorbild:

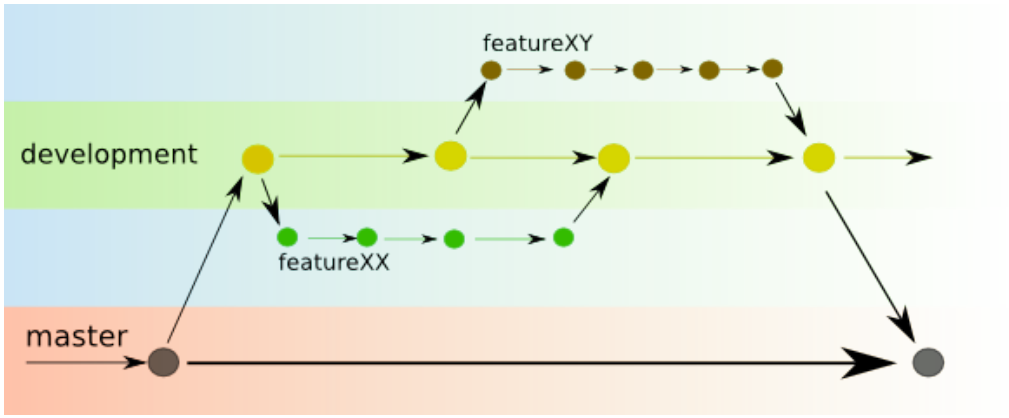

Abbildung 8.1: Git Feature Branch graph

8.2.2 Jenkins

Zur kontinuierlichen Integration wurde Jenkins[1] verwendet. Dabei wurden drei Jobs eingerichtet:

hapdroid dient zum testen und bilden der zu entwickelnden Applikation aus der aktuellen Quelle des master branch des Git Repository.

hapdroid-doc bildet täglich die aktuelle Dokumentation aus dem master branch des Git Repository.

hapdroid-backup dient zur Sicherung der Projekt relevanten Informationen des Servers. Dies besteht aus der Datenbank des Servers und der Redmine installation.

1 http://jenkins-ci.org/

8.2.3 Redmine

Für die Zeiterfassung des Projektes haben wir uns für Redmine[1] entschieden. Durch ein zusätzliches SCRUM plugin[2] konnten die Sprints und User Stories direkt in Redmine erfasst werden und dadurch der Arbeitsablauf flüssiger gestaltet werden.

1 http://www.redmine.org/
2 https://github.com/software-project/scrum-pm

9 Projektrückblick

10 Erfahrungsberichte

10.1 Persönlicher Bericht von Dominik Spengler

10.1.1 Generell

Im allgemeinen kann ich sagen dass ich die Thematik des HAPviewers äusserst interessant finde. Besonders die Portierung auf ein Android Gerät Die Thematik der Darstellung des Datenverkehrs mit Graphen empfinde ich als äusserst interessant. Auch die Portierung auf ein Android Gerät interessiert mich persönlich ebenfalls stark. Allerdings ergibt sich bei der Portierung auf ein Mobiles Gerät durch die stärkere Gewichtung der Applikationen gegenüber den Hosts eine andere Ausgangslage. Dies haben wir im Verlauf des Projektes stark gemerkt.

10.1.2 Projektverlauf

Rückblickend betrachtet wäre es aber vielleicht Hilfreich gewesen eine ausführlichere Analyse durchzuführen. Besonders die Lösung mit den Transaktionen welche im weiteren Projektverlauf zu einigen Schwierigkeiten, besonders in den Anpassungen in der Bibliothek führte, wäre vielleicht anders gewählt worden. Dadurch hätte vielleicht einige Probleme verhindert werden können, aber ich bin der Überzeugung dass ich mich bei gleicher Voraussetzung und Wissensstand gleich entschieden hätte.

10.1.3 Planung

Wir haben für unser Projekt SCRUM verwendet. Obwohl ich persönlich ein grosser Befürworter von SCRUM bin, muss ich gestehen, dass für dieses Projekt eine andere Entwicklungsmethode vielleicht besser geeignet gewesen wäre. SCRUM zeichnet sich durch den niedrigen Administrationsaufwand und starke Konzentration auf das Programmieren aus. Dies wird sich damit erkauft dass die einzelnen Lösungsansätze in der Regel nicht bis ins Detail ausgearbeitet werden.

10.1.4 Resultat

Durch viele Probleme in der Anpassung der Bibliothek und in der Benutzeroberfläche ging die Entwicklung der Applikation sehr schleppend voran. Die resultierte Applikation ist deshalb nicht annähernd so ausgereift wie erhofft. Besonders ist die Korrektheit der Daten nicht gewährleistet, was doch etwas den Informatikerstolz verletzt.

Dennach kann ich sagen dass wir viele Erkenntnisse getroffen haben und die entstandene Applikation ist eine gute Grundlage für weitere Entwicklungen. Ebenfalls gestaltete sich die Arbeit mit Remo Egli als besonders angenehm. Aus diesem Standpunkt bin ich zufrieden mit der geleisteten Arbeit.

10.2 Persönlicher Bericht von Remo Egli

10.2.1 Generell

Ich empfand diese Arbeit als sehr anspruchsvoll, da sie sich thematisch durch sehr viele Betriebssystemschichten bewegt. Im Hinblick auf die systemnahen Komponenten bin ich sehr froh, dass mein Team-Partner bereits Erfahrungen in diesem Bereich hatte. Die Thematik interessiert mich zwar, jedoch gehört die C/C++ Programmierung, welche dafür meist unumgänglich ist, überhaupt nicht zu meinen stärken. Die Zusammenarbeit mit Dominik war sehr gut und ich schätzte es sehr, mit jemandem die Arbeit zu machen, der mir in den Diskussionen Paroli bieten konnte.

10.2.2 Projektverlauf

Zu Beginn des Projekts war ich skeptisch, da ich mir unter dem Thema, wie es im AVT-Tool beschrieben worden ist, nicht viel vorstellen konnte. Ich hatte zwar zuvor schon einmal ein Projekt auf der Android-Plattform umgesetzt, jedoch kam ich bei jenem Projekt nicht in Kontakt mit den Kernkomponenten des Systems. Im Verlauf des Projekts konnte ich mich immer mehr für das Thema begeistern.

Ab dem Zeitpunkt, wo wir mit der Programmierung starteten, war die Arbeit wesentlich intensiver als erwartet, da die Materie zunehmend komplexer wurde. Diese Tatsache und weitere Hindernisse, wie z.B. nicht vorhandene Dokumentationen, bargen jedoch nicht selten einiges an Frust-Potential. Im Gegenzug dazu war jedoch das Erfolgserlebnis umso grösser, wenn ein wichtiges Problem gelöst werden konnte.

10.2.3 Planung

In diesem Projekt verwendete ich das erste Mal die SCRUM-Methode. Mein Fazit dazu ist durchwegs positiv, weil wir gerade in diesem Projekt, und auch noch im fortgeschrittenen Verlauf, unerwarteten Mehraufwand bewältigen mussten. Eine “altmodische” Planungsmethode hätte uns erhebliche Schwierigkeiten bereitet, mit SCRUM konnten wir jedoch flexibel unsere Planung anpassen.

10.2.4 Resultat

Ich bin mit dem Ergebnis nur bedingt zufrieden. Wir verloren enorm viel Zeit mit Abhängigkeiten unserer Software, welche öfters zu Problemen auf der Zielplattform und darum zu einigem

Frust führten. Einige der interessanten Ideen, welche zu Beginn besprochen wurden, konnten deshalb nicht in Angriff genommen werden, was mich etwas enttäuscht. Bewusst bin ich mir auch der Tatsache, dass der Projekt-Bericht noch einiges Potential in Umfang und Qualität hätte. Wir hatten uns manchmal etwas stur auf die Entwicklung und die damit verbundene Problemlösung fokussiert, wodurch der Bericht etwas zu kurz kam. Somit wird es für die Bewertung leider nur schwer erkennbar, wie viel Aufwand, wie viele Ideen und Überlegungen in das Programm eingeflossen sind. Ich bin eigentlich sogar etwas stolz auf unser Programm, jedoch trübt die Dokumentation das Gesamtbild leider etwas.

Ich möchte mich an dieser Stelle nochmals bei meinem Team-Kollegen Dominik Spengler und auch bim Betreuer Prof. Eduard Glatz herzlich für die gute Zusammenarbeit bedanken.

Teil III

Anhang

Literaturverzeichnis

[Gla10] GLATZ, Eduard: Visualizing Host Traffic through Graphs (2010)

[Gla12] GLATZ, Eduard: Aufgabenstellung Network Traffic Inspector auf Android, Website (2012), URL https://avt.hsr.ch, zugriff 10. Juni 2012

[Goo12a] GOOGLE: Android Activity Guide (2012), URL http://developer.android.com/guide/topics/fundamentals/activities.html, zugriff 28. Mai 2012

[Goo12b] GOOGLE: Android Application Fundamentals (2012), URL http://developer.android.com/guide/topics/fundamentals.html, zugriff 3. April 2012

[Goo12c] GOOGLE: Android Handler Referenz (2012), URL http://developer.android.com/reference/android/os/Handler.html, zugriff 3. Juni 2012

[Goo12d] GOOGLE: Android Service Guide (2012), URL http://developer.android.com/guide/topics/fundamentals/services.html, zugriff 21. März 2012

[Wik12] WIKIPEDIA: Boost (C++-Bibliothek), Website (2012), URL http://de.wikipedia.org/wiki/Boost_%28C%2B%2B-Bibliothek%29, zugriff 8. Juni 2012

Abbildungsverzeichnis

Tabellenverzeichnis

Codelistingsverzeichnis

A Entwicklungsanleitung

In diesem Abschnitt wird eine Anleitung zum Import des Projektes gegeben. Für den Import des Projektes zur weiteren Entwicklung sind einige Schritte notwendig.

A.1 Voraussetzungen

Folgende Voraussetzungen gelten für die Entwicklung unseres Projektes:

Eclipse: Als Entwicklungsumgebung für Android wird Eclipse genutzt. Es ist zwar möglich eine andere IDE zu verwenden, jedoch wird Eclipse 3.6.2 oder neuer von Google empfohlen.

ADT Eclipse Plugin: Android bietet ein Eclipse Plugin namens Android Development Tools (ADT). Das Plugin erleichtert die Entwicklung von Android Applikationen stark.

Android SDK: Das Android Software Development Kit wird zusätzlich benötigt. Bestandteil des ADT Plugins ist der Android SDK Manager. Dieser erleichtert das Herunterladen der SDK für die jeweiligen API Level. Für unsere Applikation wird API Level 8 (Android 2.2) benötigt.

Android NDK: Damit auch der C++ Code kompiliert werden kann, wird ebenfalls das Native Development Kit benötigt. Das NDK erlaubt es C/C++ Code in Android Applikationen zu integrieren und bietet eine Reihe von stabilen APIs für die Entwicklung von nativem Code.

Für die weitere Entwicklungsanleitung wird davon ausgegangen, dass die oben genannten Tools installiert und korrekt für ein Android Projekt konfiguriert sind.

A.2 Projektimport

Als erstes muss der bestehende Quellcode in Eclipse importiert werden. Dafür empfiehlt sich das EGit[1] Eclipse Plugin. Es ist aber auch möglich den Quellcode vom Dateisystem aus zu importieren.

1 http://www.eclipse.org/egit/

EGit: EGit bietet die Möglichkeit Projekte direkt von einem Git Repository zu importieren. Dazu öffnet man den Importwizard von Eclipse über *File* → *Import*. Hier kann man *Projects from Git* auswählen. Das externe Repository ist git://github.com/xuno/HAPdroid.git. Wichtig beim import mit dem Wizard ist es den neuen Projekt Wizard von Eclipse zu nutzen.

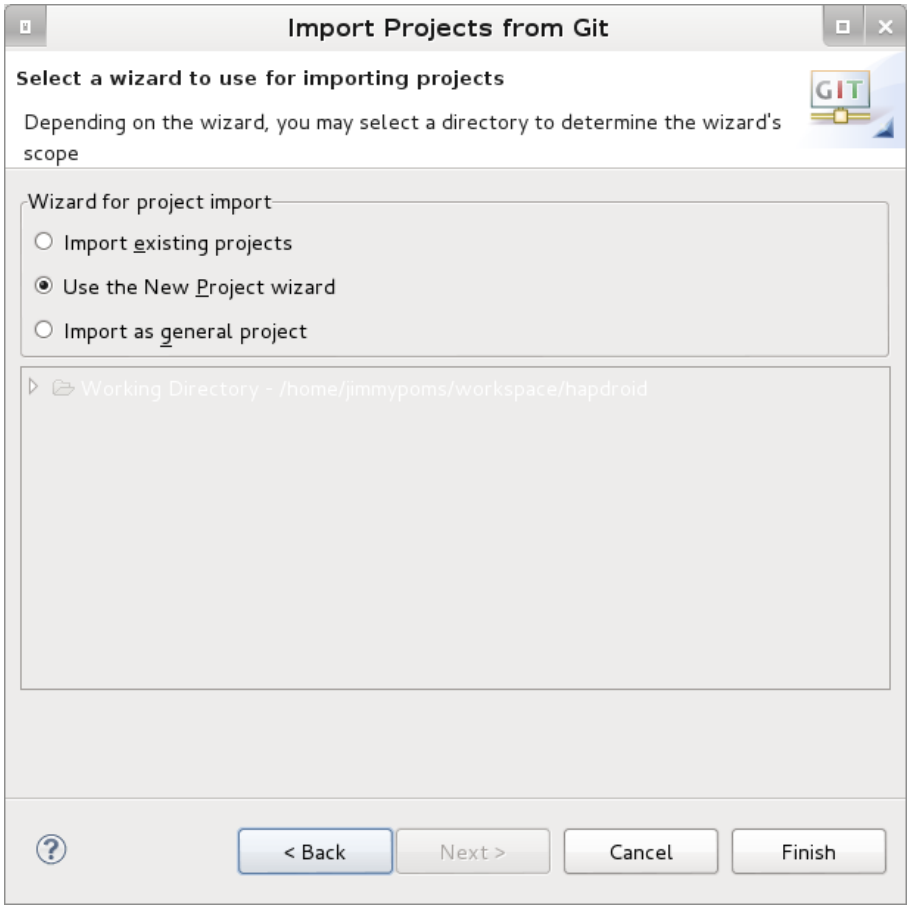

Abbildung A.1: Importwizard des EGit Eclipse Plugin

Die weiteren Schritte sind gleich wie beim Eclipse Import.

Eclipse Import: Die Import Funktionalität des ADT Plugin leider nicht sehr intuitiv. Es muss mittels des Wizards für ein neues Projekt importiert werden. Dazu wählt man *File* → *New* → *Android Project*. Der Wizard bietet die Möglichkeit ein neues Projekt aus bestehendem Code zu erstellen.

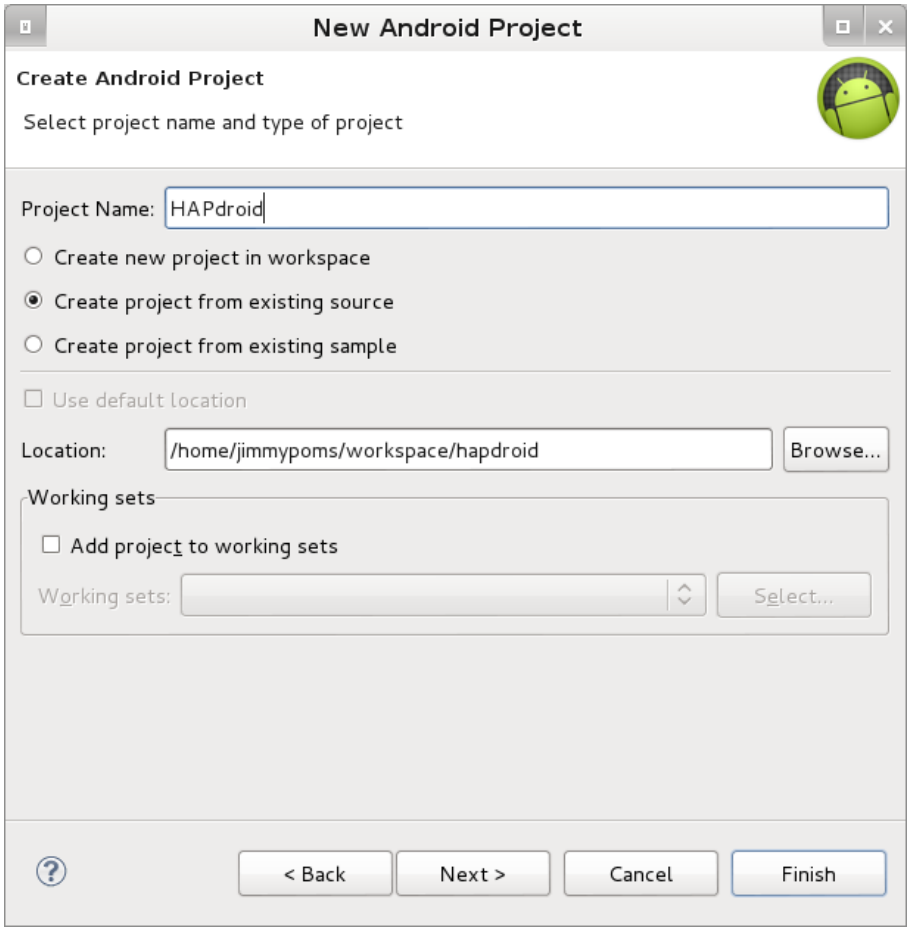

Abbildung A.2: Importwizard des ADT Eclipse Plugin

Der Wizard sollte nun alle korrekten Werte aus der AndroidManifest.xml Datei erhalten.

Ist das Projekt importiert werden als erstes eine Vielzahl von Fehlern mit nicht gefundenen Abhängigkeiten auftauchen. Um diese Abhängigkeiten zu lösen müssen noch die externen Bibliotheken hinzugefügt werden.

A.3 Externe Java Bibliotheken

Das hinzufügen von externen Bibliotheken ist denkbar einfach in Eclipse. Es muss lediglich ein neuer Ordner namens *libs* im Stammordner des Projektes erstellt werden und die .jar Dateien hinzugefügt werden.

Die benötigten Bibliotheken sind die folgenden:

RootTools: http://code.google.com/p/roottools/downloads/list

aFileChooser: http://code.google.com/p/afilechooser/downloads/list

Da aFileChooser die eigenen Drawables benötigt, genügt es nicht lediglich eine .jar Datei einzubinden, sondern es ist nötig aFileChooser als externes Projekt einzubinden. Dazu kann der Quellcode als bestehendes Eclipse Projekt importiert werden. Als letztes muss aFileChooser noch als Bibliothek aus unserem Projekt referenziert werden. Dies geschieht über *Project* → *Properties* → *Android* → *Library* → *Add*.

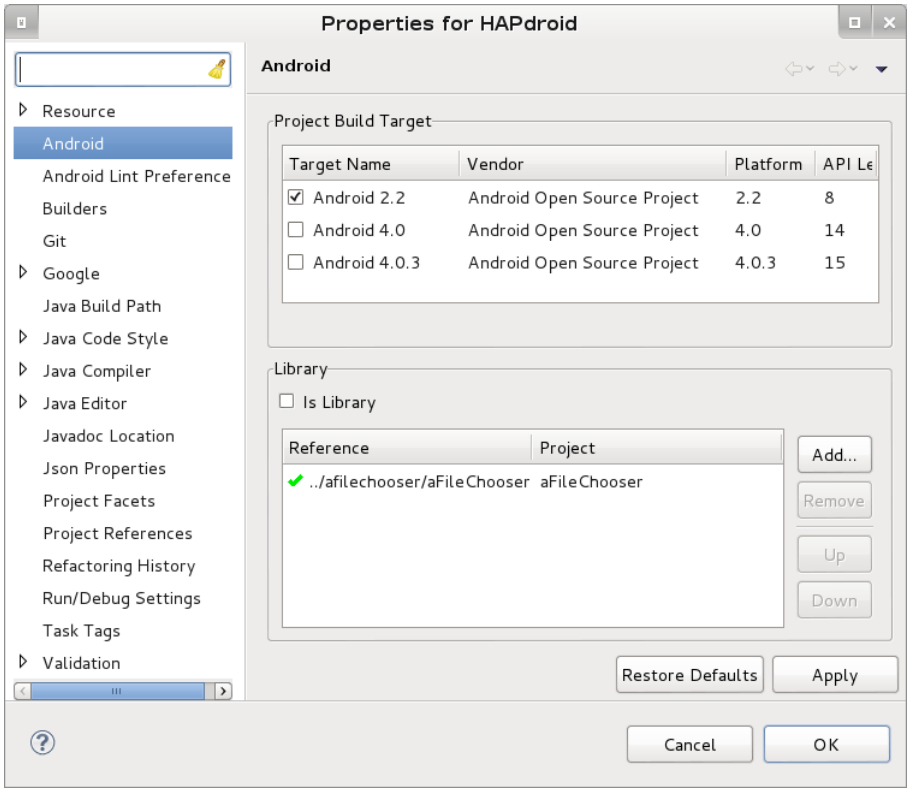

Abbildung A.3: Externe Android Bibliothek ADT Eclipse Plugin

AndEngine: http://wiki.andengine.org/AndEngine_Jars

AndEngine Multitouch Extension: Da kein öffentlich zugängliches .jar für diese Bibliothek existiert und ein kleiner Bug Fix angewandt werden musste, beinhaltet das Repository die .jar Datei.

jGrapht: http://jgrapht.org/

Hier kann man die .jar Datei nicht direkt herunterladen. Die Bibliothek ist im Archiv unter *jgrapht-x.x.x/jgrapht-jdk1.6.jar* enthalten.

Schlussendlich sollte die folgende Ordnerstruktur vorhanden sein.

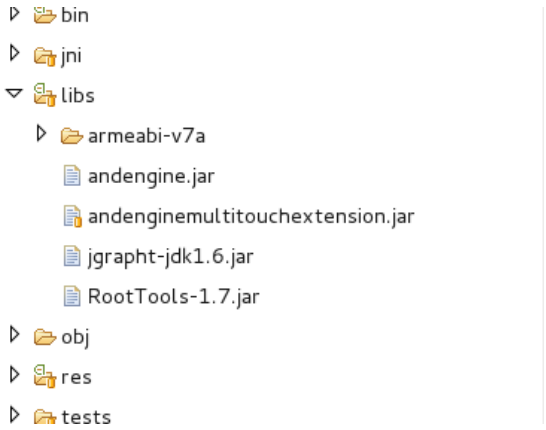

Abbildung A.4: Ordnerstruktur des Java libs Ordner

Wenn nach dem Import noch Fehler bei einigen *@Override* Annotations von Interfaces existieren, liegt das daran dass Eclipse standardmässig Java Compiler Level 1.5 nimmt. Wenn man diesen auf 1.6 umstellt sind auch die letzten Fehler in Java behoben.

Als letztes muss noch das NDK richtig Konfiguriert werden.

A.4 Externe C++ Bibliotheken

Der gesamte C++ Quellcode befindet sich im jni Unterordner des Projektes. Dabei existiert ein Unterordner namens *external* welcher für die externen Bibliotheken gedacht ist und die Android make Dateien enthält.

Es werden zweit externe Bibliotheken benötigt:

libpcap: Für libpcap existiert bereits eine Portierung auf Android. In Android.mk Datei ist bereits der Pfad *external/libpcap* vordefiniert. Um keine Änderungen an den make Dateien vornehmen zu müssen soll der Quellcode auch dort abgelegt werden. Dazu klonen wir das bestehende Repository in diesen Ordner:

git clone git://github.com/android/platform_external_libpcap.git libpcap

Boost: Boost wird als Abhängigkeit von libhapviz benötigt. Genauer gesagt benötigen wir die beiden Module iostreams und regex. Für diese beiden Module sind bereits make Dateien definiert und es muss lediglich noch der Quellcode beschafft werden. Dazu ist es am einfachsten die komplette Bibliothek von der offiziellen Homepage[1] herunterzuladen. Das Archiv beinhaltet einen Ordner mit Versionsnamen, der vordefinierte Pfad in der Android make Datei beinhaltet aber keinen Versionsnamen. Damit keine Änderungen an den bestehenden Andriod.mk Dateien notwendig sind soll der Quellcode in den Ordner

1 http://www.boost.org/

external/boost extrahiert werden, oder ein symbolischer Link auf den Boost Ordner mit Versionsnamen erstellt werden.

Damit Boost auch erfolgreich kompiliert muss noch der im *patches* Ordner enthaltene Patch angewandt werden:

patch -p0 < patches/endian.hpp.patch

Schlussendlich sollte die folgende Ordnerstruktur vorhanden sein.

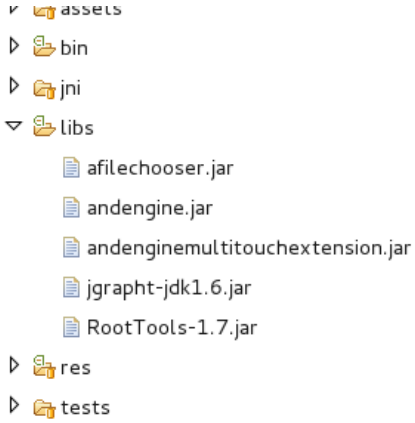

Abbildung A.5: Ordnerstruktur des Java libs Ordner

Jetzt kann mit der Kompilation fortgefahren werden.

A.5 Native Development Kit

Für die Kompilation des nativen Codes bietet das ADT Plugin keine Funktion, es muss also von Kommandozeile selbst durchgeführt werden.

Für diesen Schritt gehen wir davon aus, dass das NDK gemäss der offiziellen Dokumentation bereits installiert ist. Der letzte Schritt der Kompilation reduziert sich damit auf das Ausführen des *ndk-build* Kommandos im Projekt Wurzelordner.

A.6 Installation Executable

Nun sind alle nötigen Abhängigkeiten eingebunden und kompiliert. Der letzte Schritt zum erfolgreichen Ausführen der Android Applikation ist die Installation der netdump Datei.

RootTools erwartet die Datei im *res/raw* Ordner, das NDK kompiliert die Dateien aber in die *libs/armeabi* und *libs/armeabi-v7a* Ordner. Die ausführbare Datei muss also noch kopiert werden. Die beiden generierten Dateien unterscheiden sich in der Zielplattform auf welche sie kompiliert wurden. Zusammenfassend kann man sagen dass armeabi auch ältere Geräte unterstützt, armaebi-v7a aber schneller ist. Die Wahl liegt beim Entwickler.

Nun ist alles bereit für die weitere Entwicklung an HAPdroid.