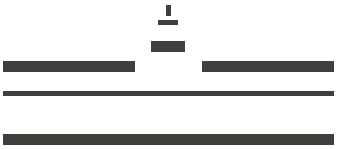
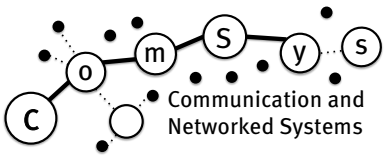


Westfälische Wilhelms-Universität Münster



Communication and Networked Systems

Masterarbeit

Migration des Routing-Frameworks DES-SERT auf Android und Evaluation unter Erzeugung mobiler Ad-hoc-Netze

Denis Klaus Küchemann

Matr. 412644

Betreuer: Prof. Dr. rer. nat. Mesut Güneş
Betreuender Assistent: M.Sc. Michael Frey

Institut für Informatik, Universität Münster, Deutschland

12. Dezember 2016

Danksagung

An dieser Stelle möchte ich meiner Freundin Annika Eiken und besonders meiner Großmutter Krista Maria Charlotte Paula Küchemann danken, dass sie mich während des Anfertigens der Arbeit in jeder Lebenslage unterstützt haben.

Meinem Vater Klaus Küchemann danke ich im Speziellen für sein Engagement beim Korrekturlesen meiner Arbeit.

Herr Prof. Dr. Mesut Güneş hat mich fachlich durch seine Lehrangebote sehr geprägt und mir dadurch eine hervorragende Basis für meine weitere Entwicklung geschaffen. Dafür möchte ich mich bedanken.

Herrn Prof. Dr Markus Müller-Olm möchte ich dafür danken, dass er stets die Zeit fand, mich fachlich sowie organisatorisch zu unterstützen.

Mein besonderer Dank gilt Dipl.-Inf. (FH), M.Sc. Michael Frey von der FU Berlin für die zu jedem Zeitpunkt fachlich exzellente und persönlich motivierende Betreuung meiner Arbeit.

Zusammenfassung

Zusammenfassung

Dezentralisierte Netze wie Ad-hoc-Netze benötigen spezielle Routing-Algorithmen, um eine effiziente Kommunikation innerhalb des Netzwerks zu gewährleisten. Das Routing-Framework DES-SERT ermöglicht eine Einbindung von Routing-Daemons über den User-Space in einem Underlay-Netz („Schicht 2,5“). Die in der DES-Arbeitsgruppe entwickelte Applikation für Android 1.5+ funktioniert nicht mehr einwandfrei unter Android-Versionen > 4.1+. Die Applikation wird im Rahmen dieser Arbeit für Android-Version 4.1+ und CyanogenMod 11+ aktualisiert. Es werden der Applikationsquellcode sowie die benötigten Kompilierskripts der C-Bibliotheken aktualisiert. Innerhalb dieser Skripts werden die Bibliotheken und die für DES-SERT angepassten Routing-Daemons für die ARMv7-Architektur kompiliert. Darüber hinaus wird eine Evaluation von DES-SERT unter Android vorgenommen.

Abstract

Decentralized networks such as ad-hoc networks require special routing algorithms to maintain an efficient data transfer within the network. The routing framework DES-SERT implements routing daemons via an underlay network (“Layer 2.5”) in the user space. The DES-SERT Android application developed by the DES workgroup does not work on Android versions 4.1+. As part of this thesis the application will be updated to Android version 4.1+ and CyanogenMod 11+. The update procedure consists of updating the application source code and the required C-library compile scripts. These scripts compile the libraries and the DES-daemons for the ARMv7 architecture. Furthermore the performance of DES-SERT on Android will be evaluated.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

KAPITEL 1

Einleitung

In westlichen Breitengraden ist eine unvernetzte Welt mittlerweile undenkbar. Mobile vernetzte Endgeräte wie Smartphones und Tablet-Computer können sensible persönliche Informationen enthalten. Die Möglichkeiten, die sich dadurch bieten, birgen aber vielerlei Sicherheitsrisiken. Edward Snowden konnte der Welt eindrucksvoll zeigen, dass auch als sicher erachtete Kryptographiestandards nicht ausreichend vor dem Missbrauch persönlicher Daten schützen können. Dies könnte ein Grund sein, ein autonomes (mobiles) Netzwerk (temporär) anzulegen. Zum Beispiel ist es denkbar, dass eine Security-Firma für 15 Minuten ein kontrolliert administriertes, abgeschottetes Funknetz aufspannt, um einen Geldtransport zu sichern. Ebenso könnten ein Hacker-Angriff, wie zum Beispiel der kürzlich massenhaft aufgetretene Totalausfall der Telekom-Router [1], ein größerer Strom-, ein Telefonleitungs-, oder ein Backbone-Ausfall auftreten. Durch den temporären Einsatz eines aufgespannten Ad-hoc-Netzes könnten solche Ausfälle überbrückt werden, um zum Beispiel in einem Notfall die Feuerwehr zu verständigen. Smartphones oder Tablet-Computer sind vorerst nicht von Stromausfällen betroffen, da sie per Akku betrieben werden können und meist zumindest für einen kurzen zu überbrückenden Zeitraum genügend Strom und Leistungsfähigkeit besitzen. Das gemeine Verständnis für Netzwerktechnologien ist meist gering; mit welchen zugrundeliegenden Technologien oder Protokollen wir täglich kommunizieren, wird kaum hinterfragt. Eine Analyse der Netztechnologien könnte zu der Idee anregen, selbst ein Netz aufzuspannen. Mit entsprechender Software könnte es sogar möglich sein, ohne SIM-Karte innerhalb eines solchen Netzes zu telefonieren [2]. Eine weitere Erforschung von Netzwerktechnologien unter Anbetracht oben genannter Aspekte sollte im Fokus der universitären Forschungsarbeit Bestand haben.

1.1 Technische Einleitung

In drahtlosen lokalen Netzen (WLANs) sind jeweils nur Endgeräte drahtlos mit der Netzinfrastruktur verbunden, während innerhalb der Netzinfrastruktur eine Verbindung via Kabel erfolgen kann. Drahtlose Multi-Hop-Netze können auch drahtlose Verbindungen innerhalb des Multi-Hop-Netzes beinhalten. Dabei bezeichnet ein Hop den Übergang von einem Knoten zum nächsten. In vermaschten Netzen sind Knotenpunkte mit mindestens einem weite-

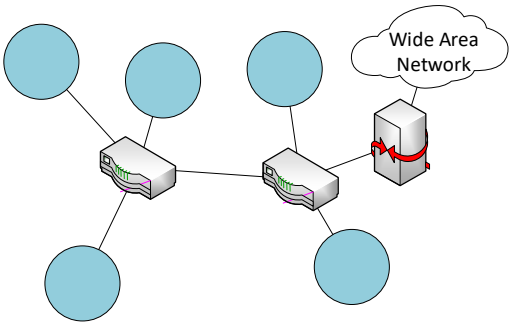


Abbildung 1.1: WMN mit zwei stationären Mesh-Routern und Gateway

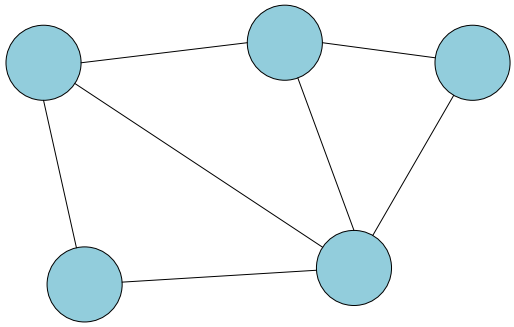


Abbildung 1.2: MANET ohne Netzinfrastruktur

ren Knotenpunkt aus dem Netz verbunden.

Ad-hoc-Netze sind vermaschte Funknetze, deren Topologie sich stetig ändert und die sich daher selbstständig neu konfigurieren können müssen. Sie sind nicht von einer vorhandenen Infrastruktur abhängig. Es existieren verschiedene Typen von Ad-hoc-Netzen. Insbesondere wichtig für diese Arbeit sind mobile Ad-hoc-Netze (englisch: mobile ad hoc network, MANET) [3] und drahtlose Multi-Hop-Mesh-Netze (englisch: multi hop wireless mesh network, MWN oder WMN) [4]. In MANETS befinden sich typischerweise viele autonome mobile Systeme, während die Netzinfrastruktur eines WMNs im Gegensatz dazu stationär ist. Die Endgeräte eines WMNs können jedoch mobil sein (zum Beispiel ein Laptop in einem Funknetz der Universität). Abbildung 1.1 zeigt ein typisches WMN im Vergleich zu einem typischen MANET auf Abbildung 1.2.

Innerhalb des Distributed Embedded Systems-Testbeds (DES-Testbed) [5] werden drahtlose Multi-Hop-Netzwerke erforscht. Das DES-Simple and Extensible Routing-Framework for Testbeds (DES-SERT) wird eingesetzt, um verschiedene Arten von Routing-Daemons in MANETs und WMNs zu implementieren, ohne für jeden Daemon ein Kernel-Modul erstellen zu müssen. Weiterhin existiert eine Applikation für ältere Versionen von Android OS (1.5+), welche die Installation der DES-SERT-Bibliothek sowie der DES-Routing-Protokolle (synonym: Routing-Daemons oder DES-Daemons) auf Android-Geräten ermöglicht. Für die Benutzung der Applikation müssen Root-Befehle auf dem Gerät ausgeführt werden können. Die DES-Daemons lassen sich über die Applikation grafisch konfigurieren und ausführen.

1.2 Motivation

Seit Version 1.5 erfuhr Android einige sicherheitskritische Betriebssystem-Updates. Auf aktuelleren Versionen von Android (4.1+), die über 95 % der benutzten Android-Versionen ausmachen [6], funktioniert die Applikation nicht mehr. Das liegt nicht nur an direkten Sicherheits-Updates. Viele frühere API-Methoden sind in neuen Android-Versionen nicht mehr vorhanden oder sind durch Änderungen am Design (grafisch und technisch), insbesondere ab Android 6.0, veraltet und können nicht mehr verwendet werden.

Einige der Protokolle, die von den DES-Daemons verwendet werden, basieren auf Freifunk-

Protokollen. Die Freifunk-Community stellt öffentliche und nicht-kommerzielle drahtlose Netze zur Verfügung, die von Privatpersonen verwaltet werden. Oft wird auch die Internetverbindung eines Teilnehmers mit dem restlichen Netz geteilt. Freifunk-Router haben die angepasste Firmware OpenWrt installiert, welche es ermöglicht, die Freifunk-Protokolle zu nutzen. Die Freifunk-Community hat steigende Teilnehmerzahlen und die verwendeten Routing-Protokolle werden frequent weiterentwickelt, um deren Leistung zu maximieren.

Die DES-SERT-Android-Applikation hat bereits einige Jahre kein Update mehr erfahren. Die Leistungsstärke eines dezentralen Netzes mit stetigen Topologieveränderungen hängt stark von den verwendeten Routing-Protokollen ab. Eine stetige Neuevaluation der Algorithmen ist somit wichtig für die Performance eines Protokolls. Eine Direktanalyse in mobilen Netzen via Smartphone oder Tabletcomputer ist eine naheliegende Möglichkeit, um auf den Faktor Mobilität einzugehen. Dahingehend erscheint ein Update von DES-SERT für neuere Versionen von Android sinnvoll.

1.3 Struktur der Arbeit

Die Arbeit ist wie folgt aufgebaut. In Kapitel 2 werden die DES-Daemons sowie deren verwendete Routing-Technologien untersucht. Überdies werden Architektur und Abhängigkeiten von DES-SERT erklärt. Kapitel 3 analysiert den Status aktueller Versionen von Android und die Veränderungen gegenüber den Android-Versionen, für die DES-SERT ursprünglich portiert wurde. Dadurch wird wiederum eine Analyse der Maßnahmen für eine Aktualisierung von DES-SERT ermöglicht. Anschließend werden in Kapitel 4 Details zur Implementierung des Updates angeführt. Kapitel 5 bewertet die Leistung der neuen Version der Applikation. Abschließend wird in Kapitel 6 ein Resümee gezogen und ein Ausblick gegeben.

KAPITEL 2

Grundlagen

Im Folgenden wird ein Überblick über die Funktionalität von DES-SERT gegeben. Eine kurze Einführung in Routing-Technologien und -Protokolle erklärt die wichtigsten Begriffe, die für ein Verständnis der Funktionalität der DES-Daemons notwendig sind. Überdies werden einige der mithilfe von DES-SERT implementierten Daemons vorgestellt.

2.1 Routing

Routing besteht aus dem Berechnen eines Pfads und dem Verbreiten der berechneten Informationen. Während der Routing-Algorithmus die Berechnung vornimmt, verbreitet das Routing-Protokoll die Informationen. Einfachheitshalber werden die Begriffe Routing-Protokoll und Routing-Algorithmus synonym verwendet.

2.1.1 Wegwahltechniken

Pfade (synonym: Wege oder Routen) zwischen den Knotenpunkten eines Netzes können statisch angegeben oder dynamisch berechnet werden. Die wichtigsten Wegwahltechniken sind Source-, Hop-by-Hop- und Spanning-Tree-Routing.

- Static- beziehungsweise **Fixed-Routing** bezeichnet die manuelle Einrichtung von Wegen. Es existiert jeweils nur ein einziger statischer Pfad zu einem System. Daher ist Fixed-Routing ungeeignet für MANETs, in denen sich die Netztopologie ändern kann. In Kontrast dazu steht Dynamic- beziehungsweise **Adaptive-Routing**, bei dem Pfade berechnet werden und bei Knotenausfällen eine Neukonfiguration der Topologie vorgenommen wird.
- Beim **Source-Routing** [7] bestimmt der Absender eines Pakets die einzelnen Knoten eines Wegs, den das Paket benutzen soll. Source-Routing kann statisch oder adaptiv implementiert werden.
- **Hop-by-Hop-Routing** bezeichnet eine Wegwahltechnik, bei der ein Knoten jeweils nur den nächsten Hop zu einem Zielknoten kennt. Dies ist die wichtigste Wegwahltechnik für MANETs und WMNs.

- **Spanning-Tree-Routing** [8] wird über das adaptive Spannbaumprotokoll (englisch: spanning tree protocol, STP) realisiert. Im Gegensatz zum Source-Routing werden die Pfadinformationen nicht beim Absender gespeichert, sondern bei Brücken oder Switches berechnet. Das Netz wird in eine Baumtopologie überführt und lässt somit keine Routing-Schleifen zu. Routing-Schleifen entstehen durch Routing-Fehler und verursachen eine Schleife in der Wegwahl beim Ansteuern eines Knotens. Abbildung 2.1 verdeutlicht eine Routing-Schleife anhand eines Beispiels.

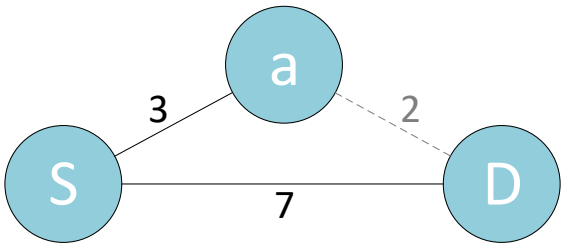


Abbildung 2.1: Knoten S möchte Knoten D über Knoten a erreichen. Knoten a verfügt jedoch über keine Verbindung mehr zu Knoten D, was Knoten S jedoch nicht weiß. Knoten a versucht nun wieder über Knoten S zu Knoten D zu gelangen. Dieser Vorgang wiederholt sich unendlich oft und wird Routing-Schleife genannt.

2.1.2 Topologiebasierte Routing-Verfahren in Ad-hoc-Netzen

In topologiebasierten Routing-Verfahren werden Nachbarknoten zur Informationsgewinnung benutzt. Anhand dessen können Pfade erstellt und eine Wegwahl getroffen werden [9]. Routing-Protokolle lassen sich grob einem der folgenden Verfahren zuordnen.

- **Proaktive** Verfahren („table-driven“) erstellen Pfade mit allen Knoten im Netz, auch wenn diese nicht benutzt werden. Um die Routen jederzeit aktuell zu halten, werden dauerhaft Informationen zu den Routen, sogenannte Updates, im Netz verteilt. Dadurch entsteht ein konstanter Netz-Overhead.
- **Reaktive** Verfahren („on-demand“) hingegen erstellen erst Pfade, sobald ein Knoten Daten empfängt. Dadurch kann es zu hohem Netzwerk-Overhead (Latenzzeiten durch Datenaustausch-Spikes) kommen, wenn ein neuer Knoten Daten verschickt. Allerdings ist die Netzwerklast gering, wenn kein Datenaustausch stattfindet.
- **Hybride** Verfahren kombinieren die Vorteile von proaktiven und reaktiven Verfahren. Typischerweise werden Routing-Zonen verwendet. Innerhalb der Zonen werden proaktiv Routen gesucht. Außerhalb der Zonen werden Routen reaktiv erstellt. Abbildung 2.2 modelliert exemplarisch zwei Routing-Zonen.

Innerhalb dieser drei Verfahren besitzen die Knotenpunkte jeweils dieselben Funktionen, sie werden daher flache Verfahren genannt.

- **Hierarchische** Verfahren [10, 11] kategorisieren Knoten abhängig von Ort und Zugehörigkeit. Details von weiter entfernten Teilen der Netztopologie werden verschleiert, indem Knotenpunkte, ähnlich dem Hybrid-Verfahren, in Regionen (autonome Systeme, AS) unterteilt werden. Einem Knoten sind jeweils nur Knoten innerhalb seines

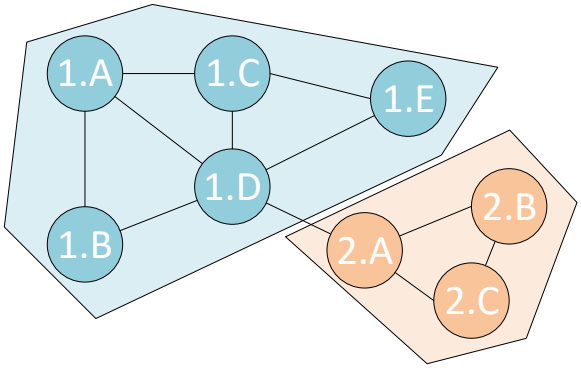


Abbildung 2.2: Die Knotenpunkte 1.X und 2.X befinden sich jeweils in einer Routing-Zone, in der proaktives Routing stattfindet. Am Rand der Zonen (Hop 1.D-2.A) wird reaktives Routing vorgenommen.

AS bekannt. Routing-Tabellen werden dadurch kleiner und das Routing vereinfacht. Jedes AS kann ein anderes Routing-Protokoll zum Einsatz kommen lassen. Abbildung 2.3 zeigt eine Topologie eines hierarchischen Verfahrens.

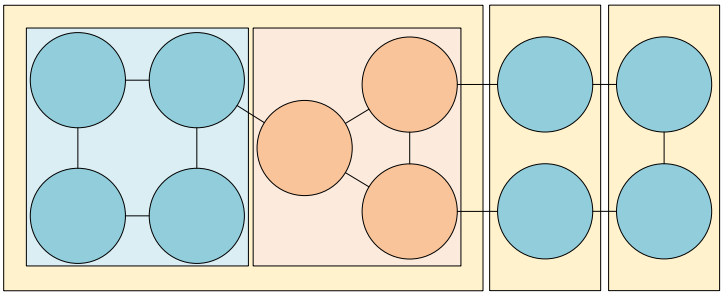


Abbildung 2.3: Topologie mit drei AS. Im ersten AS wird ein Hybrid-Verfahren benutzt.

2.1.3 Klassen von Routing-Protokollen

Zwischen AS werden Daten mit Exterior-Gateway-Protokollen (EGP) ausgetauscht (Inter-Domain-Routing). Interior-Gateway-Protokolle (IGP) hingegen tauschen Daten innerhalb der AS aus (Intra-Domain-Routing).

IGPs können weiter klassifiziert werden. Es existieren zwei Basistypen von IGPs:

- **Distanzvektor-Routing** benutzt den Bellman-Ford-Algorithmus. Ein Router empfängt und verschickt periodisch via Broadcast die komplette Routing-Tabelle. Da einem Router nur seine Nachbarn bekannt sind und die Tabellen nur die Informationen der Nachbar-Router enthalten, kann es zu Routing-Schleifen kommen.
- Protokolle, die **Link-State-Routing** [12] verwenden, benutzen den Dijkstra-Algorithmus zur Pfadberechnung. Router innerhalb eines AS besitzen sämtliche Informationen über die Netztopologie. Bei Ausfällen kann jeder Knoten innerhalb des AS neue Pfade bestimmen. Informationen werden über Multicast inkrementell (als Update) im

Netzwerk verbreitet. Dadurch wird zwar das Netz entlastet, das jeweilige System muss jedoch mehr Rechenleistung aufwenden.

- **Hybrid-Routing** [13, 14, 15] ist ein Begriff, der Protokolle bezeichnet, die der Klasse der Link-State- beziehungsweise Distanzvektorprotokolle angehören. Diese Protokolle besitzen jedoch einige Merkmale der jeweils anderen Protokollklasse. Protokolle, die bei ihrer Markteinführung als Hybride gekennzeichnet wurden, werden teilweise heute als erweiterte oder Hybrid-Distanzvektor- beziehungsweise Link-State-Protokolle bezeichnet.

Distanzvektor-Protokolle sind für kleinere Netze geeignet [16], skalieren aber nicht gut in größeren Netzen. Durch ungünstige Entscheidungen bei der Wegwahl und eine langsamere Verbreitung der Topologieinformationen bei Distanzvektorprotokollen, wird üblicherweise Link-State-Routing bevorzugt. Ein Protokoll kann durch die Art der Informationsspeicherung klassifiziert werden. Protokolle, die kein Link-State-Verfahren benutzen, deren Knoten jedoch Informationen über das gesamte Netz besitzen, fallen eher in die Link-State-Klasse. Sie können dann als „teils Link-State“ klassifiziert werden, respektive als „teils Distanzvektor“, sofern ein Knoten nur Informationen über seine Nachbarknoten besitzt.

2.1.4 Routing-Metriken

Für die Berechnung optimaler Pfade werden Metriken verwendet, die entscheiden, ob ein Pfad benutzt werden soll oder nicht. Der Routing-Algorithmus eines Routing-Protokolls erstellt eine Metrik anhand verschiedener Parameter. Oft verwendete Parameter zur Berechnung sind Anzahl der Hops, Geschwindigkeit, Qualität, Kapazität, Bandbreite, Latenz und MTU der Pfade und Verbindungen (Links) [17].

Die gängigsten Metriken beziehungsweise Basiskomponenten einer Metrik zur Einschätzung der Link-Qualität, sind die Paketverlustrate (englisch: Packet-Loss-Rate, PLR) und der Signal-zu-Rausch-Abstand (englisch: Signal-to-Noise-Ratio, SNR) [18]. Die PLR gibt den Paketverlust pro Zeit oder pro Anzahl verschickter Pakete wieder. Die SNR gibt das Verhältnis von Signalstärke zur Rauschamplitude eines Links an.

Es existieren einige Metriken, die typischerweise in MANETs verwendet werden:

- Der **Hop-Count** [19], der die Anzahl von Hops zwischen zwei Knotenpunkten bezeichnet, kann eine eigenständig Metrik sein. Allerdings können längere Routen mit hoher Qualität oft die bessere Wahl gegenüber kürzeren Routen mit niedriger Qualität sein. Dadurch ist der Hop-Count meist nur ein Teil anderer Metriken.
- Der Expected-Transmission-Count (**ETX**) [19] ist eine Metrik, welche die Link-Qualität mit dem PLR-Ansatz berechnet. Es wird die Anzahl erwarteter Paket-Übertragungen der verschiedenen Links aufsummiert. Zu einer erfolgreichen Übertragung gehört auch der Empfang des Bestätigungspakets (englisch: Acknowledgment-Packet, ACK) beim Absender des Sonden-Pakets (Probe). Bei Verlust der Probe-Pakete oder des ACK-Pakets werden neue Probe-Pakete verschickt und der ETX erhöht sich. Die Summe von ETX des ACKs und ETX des Probe-Pakets ergeben den ETX des Pfads. Die Berechnung von ETX erfolgt über die Wahrscheinlichkeit der erfolgreichen Auslieferung eines Pakets (d_f) und der Wahrscheinlichkeit, dass das ACK empfangen (d_r)

wird:

$$\mathrm{ETX} = \frac{1}{d_f \cdot d_r}$$

Umso höher dieser Wert ist, desto schlechter ist die Qualität der Route. Allerdings wird die Last eines Links nicht beachtet. Probe-Pakete beinhalten außerdem weniger Daten als das Durchschnittspaket, was das Ergebnis nach unten abweichen lassen kann. In Abbildung 2.4 wird gezeigt, wie ETX im Vergleich zum Hop-Count einen Pfad wählt.

ETX-ADD [20] ist eine additive Metrik, die auf ETX basiert. Der ETX-Wert wird jedoch aktuell gehalten, während ein Pfad durchlaufen wird.

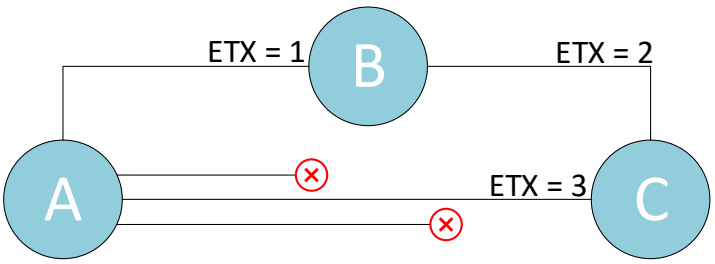


Abbildung 2.4: Um von Knoten A zu Knoten C zu gelangen, gibt es die Routen A-C und A-B-C. Der Hop-Count wählt den direkten Weg von A zu C. ETX wählt den stabileren Pfad über Knoten B, da auf der Route von A zu C durchschnittlich nur jedes dritte Paket ankommt und das Paket erneut verschickt werden muss (ETX = 3). Die Route A-B-C weist eine höhere Qualität auf, da dort kein Paketverlust auftritt (ETX = 2).

- Die Expected-Transmission-Time (**ETT**) [19] ist eine Erweiterung für ETX, um zusätzlich die Link-Datenrate mit in die Metrik einzubeziehen. ETT gibt die Zeit an, die ein Paket für eine Übertragung benötigt. Sie wird folgendermaßen berechnet:

$$\mathrm{ETT} = \mathrm{ETX} \cdot \frac{\varnothing\text{Paketgröße}}{\text{Link-Bandbreite}}$$

2.2 DES-SERT-Routing-Daemons

Es wurden bereits einige Routing-Protokolle über DES-SERT implementiert. Im Folgenden werden die Daemons klassifiziert und eine Einführung in deren Funktionsweise gegeben.

2.2.1 OLSR

Optimized Link State Routing (OLSR) ist ein proaktives Link-State-Routing-Protokoll für MANETs. Die Wegwahl erfolgt über Hop-by-Hop-Routing. OLSR kann wahlweise den Hop-Count, ETX, ETX-ADD oder PLR als Metrik benutzen. ETT ist nicht standardmäßig implementiert. OLSR ist ein IP-Routing-Protokoll. Die Routing-Tabellen bestehen demnach aus IP-Adressen (beim MAC-Routing hingegen aus MAC-Adressen).

Die Funktionsweise von OLSR besteht aus drei Teilen:

- In regelmäßigen Zeitintervallen werden HELLO-Nachrichten von allen Knoten verschickt, um Nachbarn und Links zu ermitteln. Die Nachrichten enthalten alle ermittelten Nachbarknoten und Links, mitsamt deren Status. Der Status eines Links kann asymmetrisch oder symmetrisch sein. Wenn die Nachricht des Knotens gelesen und noch nicht beantwortet wurde, ist der Status asymmetrisch. Falls die Nachricht auch beantwortet wurde, ist der Status symmetrisch [21]. Abbildung 2.5 illustriert den Austausch von HELLO-Nachrichten.

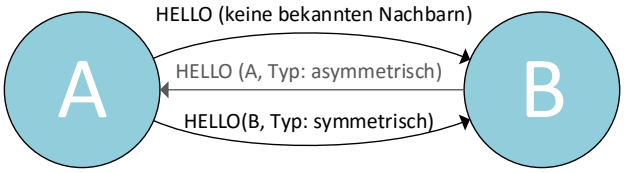


Abbildung 2.5: Vereinfachte Darstellung des Versands von HELLO-Nachrichten zwischen zwei Knotenpunkten A und B.

- Multipoint-Relays (MPR) werden zur Senkung von doppelten Broadcast-Übertragungen eingesetzt. Nur über MPRs erfolgt ein Nachrichten-Broadcast. Die Netztopologie entscheidet nach folgender Regel, welche Knoten als MPR eingesetzt werden: Für alle 2-Hop-Nachbarn n muss ein MPR m existieren, sodass n durch m zu erreichen ist. Abbildung 2.6 zeigt eine Verteilung von MPRs in einer Beispieltopologie.

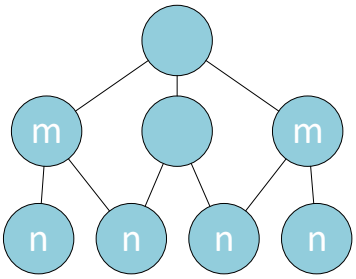


Abbildung 2.6: Der mittlere Kindknoten der Wurzel wird nicht als MPR festgelegt und sendet keine Broadcast-Nachrichten an seine Kindknoten weiter. Diese empfangen die Broadcast-Nachrichten von den bereits als MPR festgelegten Knoten.

- MPRs [22] sind die einzigen Knoten, die Link-State-Nachrichten generieren, was gegenüber dem ursprünglichen Link-State eine Optimierung ist. Bei Paketverlust hingegen ist dies gleichzeitig nachteilig. Eine weitere Optimierung ist die Reduzierung der Größe der Link-State-Nachrichten. Sie enthalten nur Advertisements (Bekanntgabe erreichbarer Links) von MPRs. Link-State-Nachrichten werden in OLSR auch Topology-Control-Nachrichten (TC-Nachrichten) genannt.

2.2.2 B.A.T.M.A.N.

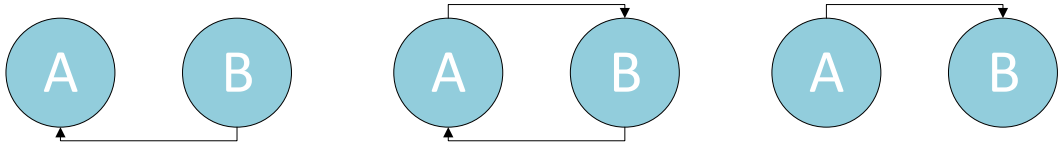
Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N.) [23, 24, 25] wurde entwickelt, nachdem OLSR auch nach vielen Verbesserungen und Entfernung der problematischen Optimierungen enttäuschende Ergebnisse erzielte. Um den hohen Zeitaufwand, der für die Berechnung der Pfade benötigt wird zu verringern und um Routing-Schleifen zu vermeiden, wird im B.A.T.M.A.N.-Routing-Algorithmus, ähnlich wie in Distanzvektorprotokollen, darauf verzichtet, in jedem Knoten die gesamte Netztopologie zu speichern. Informationen über die Qualität eines erreichbaren Links werden proaktiv verwaltet, jedoch ohne typischen Overhead von pur-proaktiven Protokollen.

B.A.T.M.A.N. ist speziell für MANETs entwickelt und folgt dem Hop-by-Hop-Routing-Ansatz. Ähnlich den Hello-Nachrichten werden in regelmäßigen Zeitabständen Originator-Messages (OGM) verschickt, die über die Existenz des verschickenden Knotens berichten. Empfänger der OGMs schicken diese über Broadcast an Knoten außerhalb der Reichweite des Absenders der OGM. Empfänger der Broadcast-Nachrichten speichern Informationen über den Nachbarknoten, von dem die Nachricht verschickt wurde.

OGMs dienen auch der Bestimmung der Link-Qualität. Die für B.A.T.M.A.N. entwickelte TQ-Metrik [24] besteht aus der Übertragungsqualität (englisch: transmission quality, TQ) und der Empfangsqualität (englisch: receive quality, RQ). Die TQ gibt die Wahrscheinlichkeit für ein erfolgreich versendetes Paket an einen Nachbarknoten an; die RQ die Wahrscheinlichkeit über den Empfang von Nachrichten eines Nachbarknotens. Insbesondere anhand der TQ kann ein Knoten bestimmen, über welchen Nachbarknoten ein Paket-Forward erfolgen soll. Die TQ wird berechnet, indem ein Nachbarknoten B die OGM eines Knotens A über einen Broadcast zurücksendet. Man erhält dadurch die Echoqualität (englisch: echo quality, EQ). Die EQ ist das Produkt aus TQ und RQ. Dadurch lässt sich die folgende Formel aufstellen.

$$TQ = \frac{EQ}{RQ}$$

Abbildung 2.7 veranschaulicht den Vorgang der Berechnung der TQ von Knoten A.



1 Knoten A erfährt RQ durch Empfang einer Nachricht von Knoten B.

2 Knoten A erfährt EQ durch OGM-Rebroadcast von Knoten B.

3 Knoten A berechnet TQ zu Knoten B nach gegebener Formel.

Abbildung 2.7: Vorgang der Berechnung der TQ von Knoten A zu Knoten B.

Eine weitere Besonderheit von B.A.T.M.A.N. ist, dass neuere Versionen im Gegensatz zum üblichen IP-Routing MAC-Routing verwenden. Dadurch wird es möglich, zum Beispiel DHCP, IPX, IPv6 oder IPv4 zu verwenden [26].

2.2.3 ARA

Ant Routing Algorithm (ARA) [27, 28] ist ein Routing-Protokoll, welches auf der Idee kollektiver Intelligenz (Swarm Intelligence) beruht. Eine Anwendung ist das Ant-based- beziehungsweise Pheromone-Routing. Ameisen markieren bei Weggabelungen den von ihnen benutzten Weg mit Pheromonen. Die markierten Wege locken weitere Ameisen an, welche ebenso den Weg mit Pheromonen markieren. Pheromone verlieren nach einiger Zeit an Wirkung. Bei Weggabelungen wird vorerst ein zufälliger Pfad gewählt und mit dem Pheromon markiert. Auf kürzeren Pfaden steigt die Pheromonkonzentration schneller. Letztendlich wird durch den Verfall der Pheromonkonzentration auf alternativen Pfaden nur noch der kürzeste Pfad gewählt. Durch eine variable Konzentration der Pheromone kann auf Änderungen im Netzwerk, wie zum Beispiel die Überlastung oder den Ausfall eines Knotens, schnell reagiert werden. Abbildung 2.8 verdeutlicht das Konzept des Ant-based-Routings.

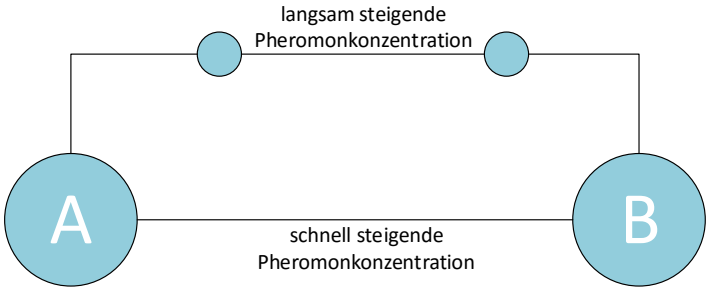


Abbildung 2.8: Der kürzere Pfad von Knoten A nach Knoten B wird öfter beziehungsweise schneller markiert. Das gilt ebenso für die Rückrichtung. Dadurch wird nach einiger Zeit nur noch der untere Pfad benutzt. Die Route A-B kann dabei kürzer sein, oder weniger Netzwerklast besitzen.

Der Routing-Algorithmus von ARA besteht aus den Phasen Pfaderstellung, Pfadverwaltung und Fehlerbehandlung:

- Forward- und Backward-Ants (FANT und BANT) dienen der Erstellung von Pfaden durch Pheromonmarkierungen. Eine FANT läuft vom Start- zum Zielknoten, eine BANT vom Ziel- zum Startknoten. FANT-Pakete werden durch eine eindeutige ID unterschieden; Duplikate werden verworfen. Sie werden durch einen Broadcast an Nachbarknoten (Relay) verschickt, die diese wiederum per Broadcast an weitere Nachbarn schicken. Einträge in den Routing-Tabellen werden beim erstmaligem Eintreffen einer FANT erstellt. Ein Eintrag beinhaltet den Startknoten, den Nachbarknoten, von dem die FANT stammt und den Wert der Pheromonmarkierung. Der Zielknoten erstellt beim Empfangen einer FANT eine BANT und schickt diese mit gleicher Funktionsweise an den Startknoten zurück. Die Pfaderstellung ist beendet, sobald die BANT beim Startknoten eintrifft. Pfade werden nach dem ersten Fehlschlag verworfen [29].
- Wenn Datenpakete einen Pfad benutzen, werden die abnehmenden Pheromonwerte eines Pfads wieder erhöht und die Routen somit verbessert. Für eine Vermeidung von Routing-Schleifen werden Paketduplikate verworfen. Das Paket wird an den Absender

zurückgeschickt und der Link dort deaktiviert. Links werden deaktiviert, indem der Pheromonwert auf 0 gesetzt wird.

- Ebenso werden Links deaktiviert, sobald ein Knoten ausfällt. In der Routing-Tabelle wird dann nach alternativen Links gesucht. Sollte keine alternative Route gefunden werden, wird ein Backtracking gestartet. Sollte der Startknoten ebenfalls keine alternativen Pfade haben, wird wieder in der ersten Phase begonnen und neue Routen erstellt. Abbildung 2.9 illustriert diesen Vorgang.

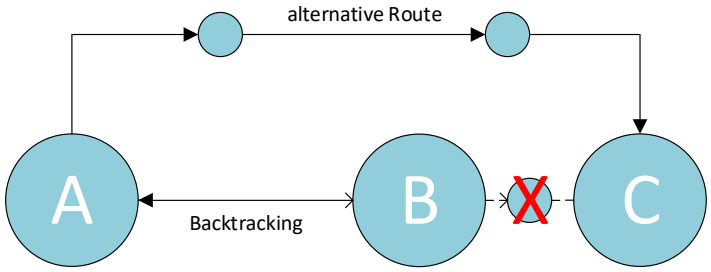


Abbildung 2.9: Es soll ein Paket von Knoten A zu Knoten C gesendet werden, aber die Route B-C ist ausgefallen. Knoten A, der Vorgänger von Knoten B, verschickt das Paket dann über eine alternative Route.

2.2.4 AODV

Ad-hoc On-demand Distance Vector Routing (AODV) [30] ist ein reaktives Distanzvektor-Routing-Protokoll. Eine Wegwahl für ein Paket findet pro Knoten statt. Routing-Schleifen werden durch Sequenznummern in den Paketen verhindert. Die Sequenznummern geben Auskunft darüber, wie aktuell eine Information ist und ob ein Paket verworfen werden kann. AODV benutzt den Hop-Count als Metrik für eine Routenerstellung. Es werden vier Typen von Kontrollnachrichten für den AODV-Algorithmus definiert.

- Die Route-Request-Message (**RREQ**) ist eine Broadcast-Nachricht, die eine Route zu einem Knoten erstellt. Eine Route wird erstellt, wenn die RREQ den Zielknoten oder einen Link mit einer aktuellen Route zum Zielknoten erreicht.
- Verfügbar wird der Pfad erst, wenn die Route-Reply-Message (**RREP**) am Zielknoten oder einem aktuellen Link zum Zielknoten ankommt. RREPs werden per Unicast vom Zielknoten zurück zum Startknoten geschickt. Der Unicast wird durch das Cachen der RREQ-Route zum Zielknoten ermöglicht.
- Bei Setzen des A-Bits einer RREP wird eine Bestätigung der RREP erwartet (**RREP-ACK**).
- Die Route-Error-Message (**RERR**) wird an Nachbarknoten verschickt, sobald ein Knoten feststellt, dass einer der Nachbarknoten ausgefallen ist.

Abbildung 2.10 zeigt, wie AODV eine Route erstellt.

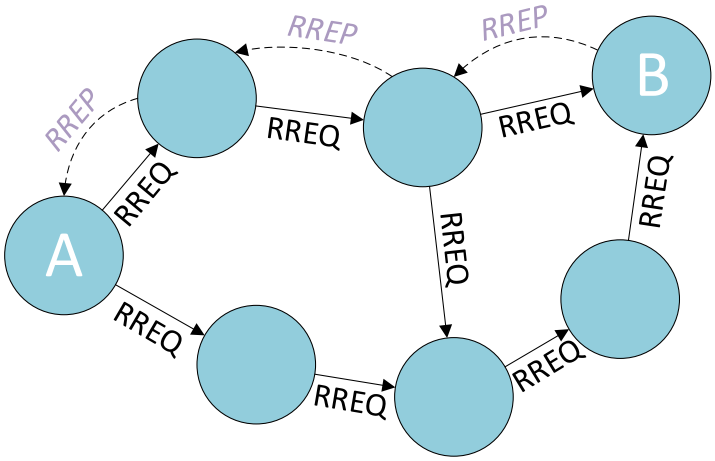


Abbildung 2.10: Knoten A möchte eine Verbindung zu Knoten B herstellen und verschickt eine Broadcast-RREQ. Nach Ankunft der RREQ bei Knoten B verschickt dieser eine RREP per Unicast über den kürzesten Weg zurück zu Knoten A. Die Route wird daraufhin zur Verfügung gestellt.

2.2.5 DSR

Dynamic Source Routing (DSR) [31, 32, 33] ist ein reaktives Routing-Protokoll. Wie der Name vermuten lässt, benutzt DSR adaptiv implementiertes Source-Routing. Dadurch ist eine Speicherung von Routing-Tabellen in den Knoten nicht nötig. Routen werden bei jedem Versand eines Pakets neu erstellt. Optional ist allerdings auch Hop-by-Hop-Routing möglich. Routing-Schleifen werden vermieden, indem die erwartete Time-To-Live (TTL) überprüft wird. Die TTL ist ein Paketfeld, welches sich pro Hop um 1 verringert. Erreicht die TTL den Wert 0, wird das Paket verworfen. Ist die erwartete TTL größer als die tatsächliche TTL, wird eine Fehlernachricht an den Absender verschickt. Das nach RFC 4728 [32] spezifizierte DSR benutzt den Hop-Count als Metrik. DES-Derivate bieten darüber hinaus eine Unterstützung für weitere Metriken, darunter auch verschiedene Formen von ETX. Der DSR-Algorithmus besteht aus der Suche nach Routen und deren Verwaltung.

- Bei der **Routensuche** werden keinerlei periodische Pakete verschickt. Dadurch ist der Overhead minimal. Solange bei Topologieänderungen keine Routen verändert werden, skaliert DSR bis zu einem Overhead von null. Die Suche nach Routen ist dem AODV-Algorithmus ähnlich und erfolgt über das Versenden von Broadcast-RREQs und Unicast-RREPs. Knoten legen dabei gelernte Routen im Cache ab. Dadurch wird bei erneutem Versenden eines Pakets die Routensuche beschleunigt. Auf Abbildung 2.11 wird gezeigt, wie der Vorgang der Routensuche durch die Benutzung eines Cache verbessert wird.
- Die **Verwaltung von Routen** besteht aus mehreren Teilen.

  RERRs werden verschickt, sobald eine Route ausfällt. Die RERR wird per Piggyback in einer neuen RREQ verschickt, damit Zwischenknoten ihre Routen aktualisieren können. Dieser Vorgang nennt sich *RERR-Spreading*.

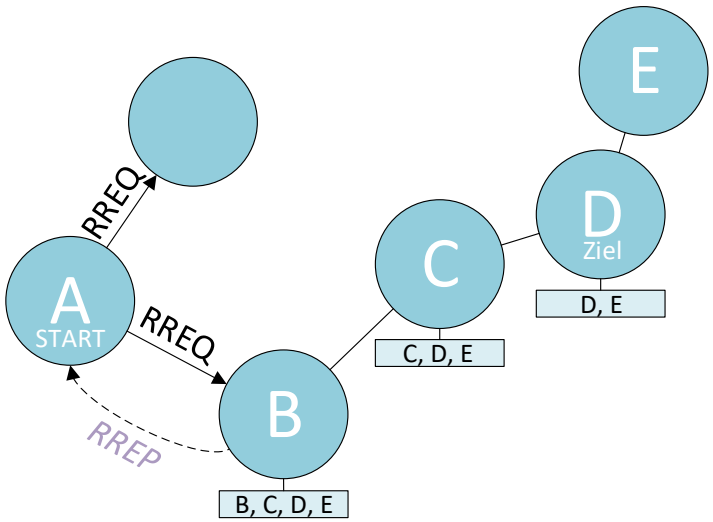


Abbildung 2.11: Knoten A versendet ein Broadcast-RREQ, um eine Route zu Knoten D zu suchen. Da bereits eine Route zu Knoten D im Cache von Knoten B liegt, schickt dieser direkt ein RREP zu Knoten A zurück.

Route-Reply-Storms (Routenkollisionen) treten auf, wenn ein Knoten mehrere RREPs gleichzeitig erhält. Sie werden verhindert, indem RREPs verzögert verschickt werden. Die Länge der Verzögerung ist zufällig.

Das *Hop-Limit* ist ein Paketfeld und gibt an, wann ein Paket verworfen wird. Es wird über das TTL-Feld eines Pakets implementiert, da die Funktionsweise dieselbe ist. Sollte nach einer bestimmten Zeit keine Route gefunden worden sein, wird erneut eine RREQ, jedoch ohne Hop-Limit, versendet.

Packet-Salvaging bedeutet, dass im Fall eines Link-Ausfalls eine Route innerhalb eines Pakets verändert wird, um den Zielknoten zu erreichen. In dem Fall wird eine RERR verschickt. Falls keine alternative Route zur Verfügung steht, wird das Paket verworfen. Packet-Salvaging ist zur Vermeidung von Routing-Schleifen pro Paket nur einmal möglich und wird über ein Salvage-Feld innerhalb des Pakets verwaltet.

Es können *Gratuitous-Messages* innerhalb einer RREP an den Absender eines Pakets verschickt werden, wenn ein Knoten bemerkt, dass ein Zwischenknoten auf einer Route nicht mehr benötigt wird. Dieses Feature nennt sich *Automatic Route Shortening* und wird auf Abbildung 2.12 veranschaulicht.

Tabelle A.1 zeigt eine Gegenüberstellung der Daemons in Tabellenform.

2.3 DES-SERT

Das Routing-Framework DES-SERT [5] wurde für die Forschung an Routing-Protokollen entwickelt und wird hauptsächlich im DES-Testbed, einer Experimentierumgebung für drahtlose Multi-Hop-Netzwerke, verwendet. In Testbeds wird echte Hardware benutzt, statt diese lediglich zu emulieren. DES-SERT erlaubt die Ausführung der Daemons unter Linux

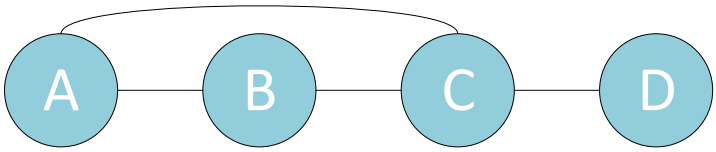


Abbildung 2.12: Knoten A möchte Pakete zu Knoten D schicken. Knoten A bekommt von Knoten C mitgeteilt, dass kein Umweg mehr über Knoten B gemacht werden muss. Pakete können direkt die Route A-C-D benutzen.

im User-Space mit der Einrichtung eines Underlay-Netzes. Underlay-Netze sind Netze, die im ISO/OSI-Modell eine niedrigere Schicht besitzen, als das darüberliegende Netz. Obwohl einige der vorgestellten DES-Daemons IP-Routing benutzen, erfolgt mit der Benutzung des Underlays ein Routing über MAC-Adressen. Das vom Autor sogenannte „Layer-2.5-Framework“ [34] wurde für eine einfachere beziehungsweise überhaupt mögliche Implementierung von reaktiven Routing-Protokollen entworfen. Durch die Einbindung der Daemons über den User-Space muss nicht für jeden Daemon ein Kernel-Modul erstellt werden. Dadurch können die Routing-Algorithmen der Daemons unabhängig vom Kernel untereinander verglichen und ausgewertet werden. DES-SERT übernimmt anstelle der Daemons die Einrichtung der nötigen Mechanismen im User- und Kernel-Space. Im folgenden Abschnitt werden diese Mechanismen vorgestellt.

2.3.1 Architektur und Abhängigkeiten

Die Einbindung eines Daemons im User-Space erfolgt über die Benutzung der DES-SERT-Bibliothek [5, 34, 35] libdessert. Neben libdessert benötigt DES-SERT die Bibliotheken libdessert-extra, libpcap und libcli sowie die Standardbibliotheken libpthread und stdlib. Diese Bibiliotheken sind in der Programmiersprache C verfasst.

Die Packet-Capture-Bibliothek **libpcap** erlaubt es, den Verkehr in einem Netzwerk abzufangen.

Mit der Bibiliothek **libcli** wird ein Cisco-ähnliches Internetwork Operating System Command-Line-Interface (Cisco IOS CLI) implementiert. Cisco IOS ist ein Betriebssystem, welches auf vielen Cisco-Routern und -Switches installiert ist. Über das CLI kann es gesteuert werden. Innerhalb von DES-SERT wird eine Laufzeitparametrisierung der Daemons via Telnet ermöglicht.

Libdessert stellt einem Daemon folgende Funktionen zur Verfügung, die benötigt werden, um ein Routing via User-Space zu ermöglichen. Der lokale *Empfang und Versand von Paketen* erfolgt über ein virtuelles TUN/TAP-Netzwerkgerät, welches den Paketempfang und -versand für User-Space-Programme ermöglicht. Ein TUN-Gerät liest und schreibt IP-Pakete, ein TAP-Gerät Ethernet-Frames. Das TUN/TAP-Gerät wird synonym auch Sys-Interface genannt. Das Sys-Interface dient als Zugangspunkt für den Netzwerk-Stack des Betriebssystems. Vom und ins Netzwerk verschickte Pakete in Form von DES-SERT-Nachrichten werden via libpcap vom Netzwerk-Interface (synonym: Mesh-Interface) empfangen. Die DES-SERT-Architektur ist in Abbildung 2.13 illustriert.

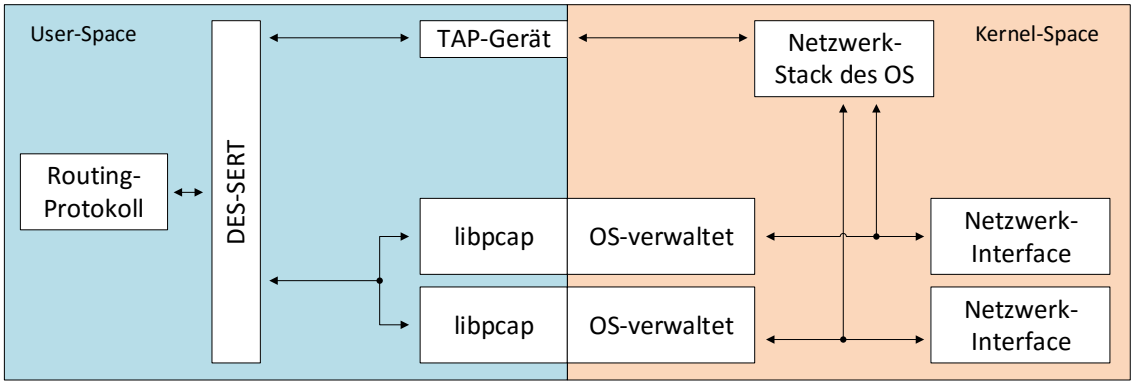


Abbildung 2.13: Architektur von DES-SERT

DES-SERT-Nachrichten werden in Form eines Ethernet-Frames verschickt. Neben Ethernet-, DES-SERT-Header und Payload bestehen die Nachrichten aus sogenannten DES-SERT-Extensions. Extensions beinhalten zusätzliche Informationen für das Routing-Protokoll, wie zum Beispiel Latenzzeiten, die von DES-SERT *(de)serialisiert* werden und per Hop *dynamisch modifiziert oder hinzugefügt* werden können („Piggyback-Daten“). Da jede Extension einen Overhead von 16 Bit verursacht, sollten Strukturen anstelle von einzelnen Werten benutzt werden [36]. In Abbildung 2.14 wird der Aufbau einer DES-SERT-Nachricht [34] gezeigt.

DES-SERT unterstützt außerdem das Simple Network Management Protocol (*SNMP*) und eine Schnittstelle für eine Netzwerkdiagnose [34]. Das von DES-SERT implementierte Logging-System ist *syslog* nachempfunden und kann wie `printf` benutzt werden.

Die Bibliothek **libdessert-extra** [37] stellt CLI-Callbacks zur Verfügung. Der Daemon wird dadurch mit Sys- und Mesh-Interfaces versorgt, sofern er nicht zur Kompilierzeit, sondern anhand einer Konfigurationsdatei eingerichtet wird.

2.3.2 Aufbau eines Daemons

Für eine theoretische Implementierung eines Beispiel-Daemons wird Bezug auf einige weitere Features von DES-SERT [34] genommen.

Für die verschiedenen Module eines Routing-Algorithmus, wie zum Beispiel Schleifenerkennung oder das Nachschlagen der Routing-Tabelle, unterstützt DES-SERT *Process-Pipelining*. Eine Implementierung der verschiedenen Daemon-Module kann klar aufgeteilt werden und somit unabhängig voneinander erfolgen. Es existiert je eine Pipeline für die Pakete des Sys-Interface und des Mesh-Interface. Über den Prioritätswert der für die Pipeline registrierten Callback-Funktionen kann die Reihenfolge der Verarbeitung verändert werden. Pakete mit höherem Prioritätswert werden später verarbeitet. DES-SERT übernimmt die Speicherverwaltung eintreffender Pakete. Werden Extensions benutzt, muss aber angegeben werden, dass der Puffer vergrößert werden soll.

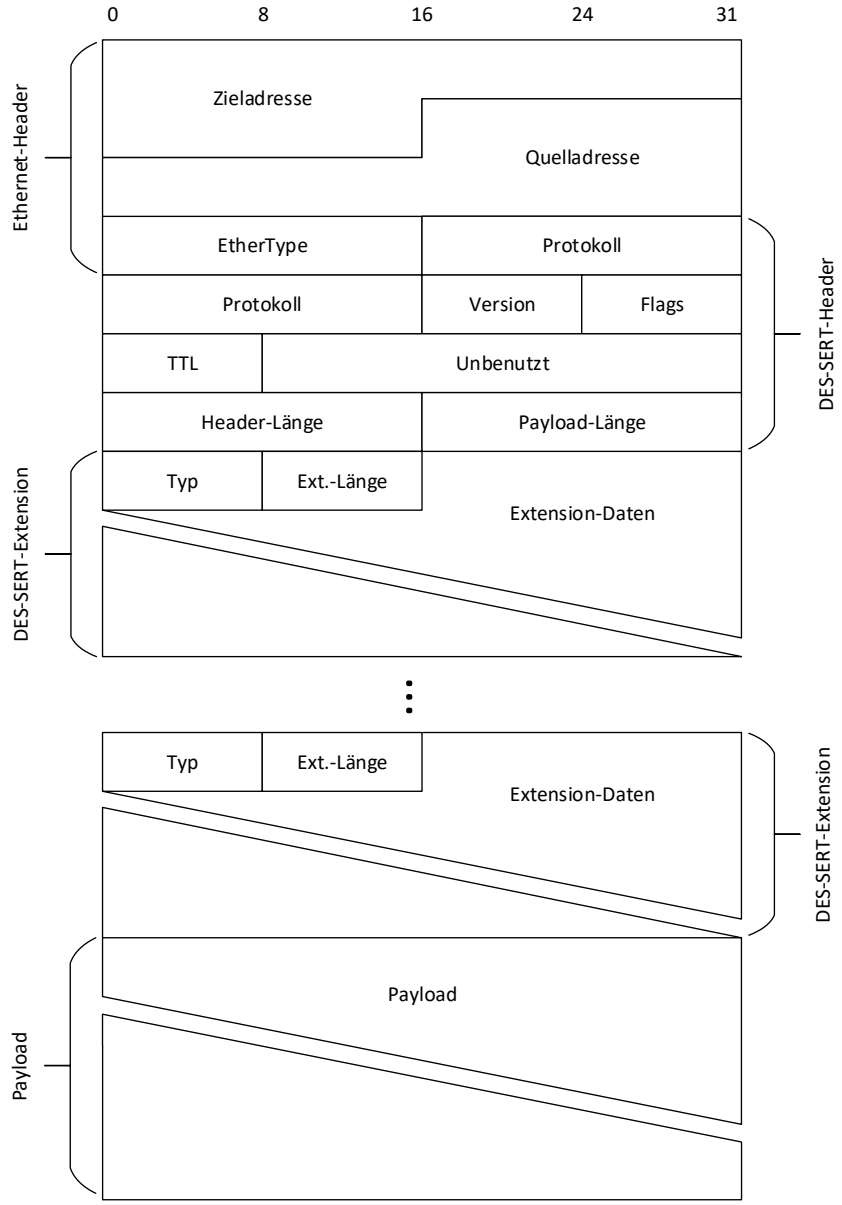


Abbildung 2.14: Aufbau einer Nachricht in DES-SERT

Sollte ein Paket über mehrere Callback-Funktionen hinweg verarbeitet werden, kann explizit ein *Prozesspuffer* angefordert werden, der den Callback-Funktionen zur Verfügung gestellt wird. Im Prozesspuffer sind bereits einige Flags gesetzt, die zum Beispiel Auskunft über die Herkunft des Pakets geben. Außerdem wird 1KB Puffer für zusätzliche Daten zur Verfügung gestellt.

Es kann ausgewählt werden, welcher Typ Sys-Interface benutzt werden soll (TUN oder TAP). Die Benutzung eines TUN-Interface setzt ein benutzerdefiniertes Mapping von Schicht 2 zu Schicht 3 voraus. TUN-Pakete können indizieren, ob ein IPv4- oder ein IPv6-Paket emp-

fangen wurde. Felder mit MAC-Adressen werden ignoriert. Sie werden auf 00:00:00:00:00:00 gesetzt. Bei beiden Sys-Interface-Typen werden empfangene Frames der Pipeline mit einem Zeiger auf den Ethernet-Frame hinzugefügt.

Im Gegensatz zum Sys-Interface kann es mehrere Mesh-Interfaces geben. Sie verschicken und empfangen DES-SERT-Nachrichten statt Ethernet-Frames. Der Versand einer Nachricht an ein Mesh-Interface kann im Unicast oder im Broadcast erfolgen.

Um Routing-Tabellen aktuell zu halten, werden periodische Tasks benötigt. DES-SERT kann über die Registrierung von Callback-Funktionen eine sichere Implementierung gewährleisten. Die Callback-Funktionen können entweder regelmäßig oder um eine bestimmte Zeitspanne versetzt ausgeführt werden.

KAPITEL 3

Analyse

Die bisher aktuellste Version der DES-SERT-Applikation ist für die Benutzung auf Android-Geräten mit Android-Version 1.5+ konzipiert wurden. Diese Version wird ab jetzt DES-SERT-App 1.0 genannt. Der Aktualisierungsvorgang ermöglicht die Lauffähigkeit der Applikation auf Android 4.1+, ab jetzt DES-SERT-App 2.0 genannt. Eine Analyse der Abhängigkeiten von DES-SERT, der DES-Daemons und der Applikation in Bezug auf die Eigenschaften von Android 4.1+ zeigt die notwendigen Änderungen an DES-SERT-App 1.0 auf. Optionale Verbesserungen, welche die Applikation leistungsfähiger machen, jedoch keinen direkten Einfluss auf die Funktionalität haben, werden in die Analyse miteinbezogen.

3.1 Android OS

Android ist ein Linux-Derivat, welches speziell für tragbare Geräte wie zum Beispiel Smartphones und Tablets angepasst wurde. Die Unterschiede zu einem typischen Linux sind jedoch immens. Diese Unterschiede können eine Portierung von Standard-Linux-Programmen auf die Android-Plattform erschweren.

Android verzichtet auf alle GNU-Komponenten [38]. Mit der Applikation BusyBox können Standardbefehle (zum Beispiel: su oder chmod) nachträglich installiert werden. Allerdings funktioniert BusyBox nur auf Android-Geräten, auf denen Root-Rechte freigeschaltet worden sind. Die Benutzung von Software mit der GNU-General-Public-License (GNU GPL oder GPL) ist unter Stock-Android nicht möglich.

Die von Android benutzte C-Bibliothek Bionic [39] ist eine verkleinerte Version von glibc (GNU C Library). Da glibc unter der GPL lizenziert ist, darf sie nicht in Android verwendet werden. Bionic ist nicht vollständig zu glibc kompatibel.

Android unterstützt nicht den klassischen Multiuser-Ansatz [40], bei dem sich ein Benutzer beim Betriebssystem anmeldet. Stattdessen bekommt jede Applikation eine eigene Benutzer-ID (englisch: user ID, UID) zugewiesen. Somit hat jede Applikation nur Zugriff auf ihre eigenen Daten. Mit Zuweisung einer Gruppen-ID (group ID, GID) kann eine Applikation aber zum Beispiel auf die gesamte SD-Karte zugreifen.

Android-Applikationen werden in einem Java-Dialekt geschrieben. Es ist jedoch möglich,

C-Programmcode mithilfe des Android-Native-Development Kits (NDK) einzubinden. Die Applikationen werden alle in einer eigenen virtuellen Maschine (VM) gestartet. Ab Android 5.0 wird die Android-Runtime (ART) [41] verwendet, welche die Dalvik-VM ersetzt. C-Code wird außerhalb der VM ausgeführt [42, 43]. Die Benutzung von nativem Code kann Portierungen vereinfachen und die Applikations-Performance verbessern.

3.1.1 Ad-hoc-Unterstützung

Ad-hoc-Netze werden nicht nativ von Android unterstützt. Frühere Stock-Android-Versionen (zum Beispiel Android 2.1) mit Root-Rechten konnten nach [44] noch mit einfachen Methoden für eine Benutzung von Ad-hoc-Netzen konfiguriert werden. In aktuelleren Versionen von Android (4.0+) ist nach [45], [2] und [46] sowie eigenen Tests eine derartige Konfiguration nicht mehr funktional oder äußerst kompliziert: Durch die vollständige Entfernung der Ad-hoc-Unterstützung, müsste ein tiefer Eingriff in den Android-Kernel erfolgen. Eine Unterstützung von Ad-hoc-Netzen mit den angestrebten Android-Versionen (4.1+) kann jedoch recht einfach über die Verwendung von CyanogenMod erzielt werden. CyanogenMod ist ein Android-Derivat und kann als Alternative zu einem Stock-Android installiert werden.

3.1.2 ARM-Instruktionssatz-Architektur

ARM-Mikroprozessoren gehören zu den meist verbreiteten Chips für eingebettete und mobile Systeme. Über 85 % aller mobilen Systeme weltweit besitzen einen ARM-Chip [47]. Die ARM-Architektur unterscheidet sich von der x86- und x64-Architektur unter anderem durch ihren Instruktionssatz. Desktop-Systeme verfügen über Prozessoren mit einem erweiterten Instruktionssatz (englisch: complex instruction set computer, CISC). ARM-Prozessoren folgen dem Ansatz eines effizienten Energiehaushalts und besitzen einen reduzierten Instruktionssatz (englisch: reduced instruction set computer, RISC). Während ARM-Prozessoren pur dem RISC-Ansatz folgen, kann die Architektur eines x86-Prozessors als Hybrid aus CISC und RISC beschrieben werden, welche CISC-Befehle zu RISC-Befehlen übersetzt [48].

Aktuelle Android-Geräte setzen meist ARMv7-Chips ein. Ältere Android-Geräte können jedoch auch ARMv5- und ARMv6-Chips, neuere ARMv8-Chips verbaut haben. Die Versionen sind jeweils abwärtskompatibel. ARMv8 benutzt 64-Bit-Register, kann jedoch über den AArch32-Modus ARMv7-Code ausführen [49].

3.2 Sicherheits-Features von Android

Die DES-SERT-Applikation benötigt Zugriff auf einige Schnittstellen und Features des Betriebssystems, die aus Sicherheitsaspekten nicht direkt aus der Applikation zugänglich sind. Stock-Android muss vorher so angepasst werden, dass die Applikation die DES-Daemons starten und verwalten kann. Darüber hinaus müssen bei aktuellen Android-Versionen ausführbare Dateien auf eine bestimmte Art und Weise kompiliert werden.

3.2.1 Android-Root-Rechte

Root-Rechte ermöglichen die Benutzung typischer Linux-Befehle wie „`su`“. Dadurch werden andere Befehle von Android zugelassen, die sonst blockiert würden.

Durch die angepasste Nutzung von UIDs und GIDs ist es nicht möglich, sich bei Android als Benutzer mit Root-Rechten anzumelden (siehe Kapitel 3.1). Android-Geräte müssen abhängig von der jeweilig verbauten Hardware des Geräts „gerootet“ werden.

DES-SERT benötigt die Root-Befehle „`su`“, „`sh`“, „`ln`“, „`kill`“ und „`chmod`“ für das Starten und Verwalten der Daemons.

3.2.2 SELinux

Seit Android-Version 4.3 wird Security Enhanced Linux (SELinux) [50] implementiert, welches eine zusätzliche Zugriffskontrolle (Mandatory Access Control, MAC) durchsetzt. Der Zugriff auf sensible Daten und bestimmte Betriebssystem-Features (zum Beispiel die Öffnungen von Ports) soll eingeschränkt werden, um Gerät und Betriebssystem zu schützen. Insbesondere Root-Applikationen erhalten mit aktiviertem SELinux keinen uneingeschränkten Zugriff auf alle Funktionen des Betriebssystems mehr.

SELinux besteht aus drei Modi und verwaltet eine Vielzahl von Richtlinien, die in Domänen unterteilt sind [51]. Welche Richtlinien in welchem Modus aktiv sind, hängt vom Hersteller des Geräts und der dafür angepassten Android-Version ab. Im strikten Modus werden die aktivierten Richtlinien durchgesetzt, während im moderaten Modus keine Durchsetzung der Richtlinien erfolgt. Im Gegensatz zum Deaktiviert-Modus wird im moderaten Modus jedoch geloggt, welche Richtlinien von einer Applikation übergangen wurden. Ab Android-Version 5.0 ist der Modus standardmäßig auf strikt eingestellt. Die DES-SERT-Applikation ist daher nicht unmittelbar auf einem Stock-Android-Gerät einwandfrei lauffähig, unabhängig davon, ob das Gerät über Root-Rechte verfügt.

Es existiert ein Programm (*sepolicy-inject*), dass es ermöglicht, einzelne Richtlinien auf dem Gerät zu ändern. Das für DES-SERT angepasste Skript ist allerdings nicht mehr aktuell und reicht nicht aus, um DES-SERT uneingeschränkt lauffähig zu machen.

CyanogenMod und SELinux

CyanogenMod (CM) verfügt ebenfalls seit seiner Implementierung von Android 4.3 [52] über SELinux. Je nach Gerät und den davon abhängigen Eigenschaften der Installationsdateien, ist SELinux üblicherweise entweder als moderat oder strikt voreingestellt. Gegenüber Stock-Android müssen die voreingestellten Richtlinien nicht zwangsweise die gleichen sein. Es besteht die Möglichkeit, dass eine Applikation, die im strikten Modus auf einem Stock-Android nicht lauffähig ist, im strikten Modus auf CM funktioniert. Dies gilt insbesondere für CM 12.1, nicht jedoch für CM 13.0 (siehe auch Tabelle A.2).

Optimierungsmöglichkeiten

Um Applikationen auszuführen, die aufgrund eingestellter Richtlinien im strikten Modus nicht ordnungsgemäß funktionieren [50], genügt es in den moderaten Modus zu wechseln. Allerdings ist das Betriebssystem dadurch auch einfacher angreifbar. SELinux verfügt über die Möglichkeit, einzelne Prozesse in den moderaten Modus zu versetzen und das restliche System im strikten Modus zu belassen. Darüber hinaus können für jeden Prozess die betroffenen Richtlinien festgestellt und einzeln (de)aktiviert werden. Allerdings ist dies sehr zeitaufwendig, da sehr viele Richtlinien existieren und eventuell nicht bekannt ist, welche Richtlinien von der Applikation benutzt werden und welche Richtlinien vom Betriebssystem voreingestellt sind. Ein Abgleich der Richtlinien des strikten Modus unter CM, unter dem die Applikation funktioniert und des strikten Modus unter Stock-Android, unter dem die Applikation nicht funktioniert, gibt eine Liste der relevanten Richtlinien aus, die zusätzlich erlaubt werden müssen. In einem günstigen Fall ist diese Liste kurz genug, um sie einzeln auf die tatsächliche Relevanz für die Applikation zu überprüfen. Im ungünstigsten Fall beinhaltet der strikte Modus unter CM keine oder wenige aktive Richtlinien und erfüllt demnach keinen Zweck. Das sepolicy-inject-Skript könnte bei Bekanntheit der für DES-SERT relevanten Richtlinien aktualisiert werden und die Richtlinien auf den Geräten anwenden.

Ab Android 6.0+ (API 23) muss darauf geachtet werden, dass Android-Permissions nicht bei der Installation, sondern zur Laufzeit abgefragt werden [53]. Android-Permissions werden in der Applikation definiert und bestimmen, welche Zugriffe ausgeführt werden dürfen. Dazu gehört zum Beispiel die Benutzung des Internets. Für eine Anpassung an Android 6.0+ müssen zusätzliche Maßnahmen ergriffen werden. Zum Beispiel muss für das Erstellen einer Datei auf externem Speicher oder den Zugriff auf diese Datei nicht nur die Android-Permission angegeben sein, sondern im jeweiligen relevanten Code-Abschnitt die Berechtigung auch konkret überprüft werden. Über den moderaten SELinux-Modus kann eine solche Überprüfung aber umgangen werden.

3.2.3 Speicher-Layout-Randomisierung

Android implementiert ab Version 4.1 Speicher-Layout-Randomisierung (englisch: address space layout randomization, ASLR) [54]. Stack, Heap, Basis-Programmdatei und Bibliotheken ausführbarer Dateien bekommen zufällige Positionen im Speicher zugeteilt. Dadurch werden Angriffe in Form von Stack- oder Pufferüberläufen auf das System erschwert.

Eine mit ASLR kompilierte Datei wird „positionsunabhängige ausführbare Datei“ (englisch: position independent executable, PIE) genannt. Der kompilierte Code wird „positionsunabhängiger Code“ genannt (englisch: position independent code, PIC).

PIEs können auf Android nicht vor Version 4.1 verwendet werden. Ab Android-Version 4.1 hingegen müssen Programmdateien mit ALSR kompiliert werden. Durch diese Änderung an Android wurde entschieden, DES-SERT-App 2.0 für Android 4.1+ zu aktualisieren.

Optimierungsmöglichkeiten

Es existieren zwei Alternativen, um PIC [55] zu generieren. Der GNU C-Compiler (gcc) verwendet für Bibliotheken die Compiler-Flags *-fpic* und *-fPIC*. Programmdateien können darüber hinaus auch *-fpie* und *-fPIE* verwenden. Die Varianten -fpic und -fpie generieren gegenüber -fPIC und -fPIE kleineren Code. Einige Plattformen unterstützen nur eine begrenzte Größe der globalen Offset-Tabelle. Wird diese Größe überstiegen, müssen die Pendants -fPIC und -fPIE verwendet werden. Diese definieren kein Limit. AArch64 hat zum Beispiel ein Limit von 28 Kibibyte (KiB).

3.3 DES-SERT-Bibliotheken

Die in C geschriebenen Bibliotheken werden mit gcc für die ARM-Architektur kompiliert. Üblicherweise passiert dies auf einem Linux-Host. Wird auf eine andere Architektur kompiliert, spricht man von Cross-Compiling. Beim Cross-Compiling müssen dem Compiler zusätzliche Informationen via Compiler-Flags mitgeteilt werden.

3.3.1 Cross-Compiling der Bibliotheken und Kompilier-Skripts

DES-SERT wird bisher für ARMv5 mit gcc Version 4.4.9 kompiliert. Ein in Perl verfasstes Skript vereinfacht die wiederholten Aufrufe des gcc. Innerhalb eines Bash-Skripts wird das Perl-Skript angewandt, um alle benötigten Bibliotheken zu kompilieren.

Die benutzte Android-Programmierschnittstelle (englisch: application programming interface, API) namens android-platform, unterscheidet sich bisher folgendermaßen innerhalb der beiden Skripts: Während das Bash-Skript Version 3 benutzt, wird Version 4 im Perl-Skript angesteuert. API-Version 3 ist für Android 1.5, API-Version 4 für Android 1.6 konzipiert.

Optimierungsmöglichkeiten

- Das Bash-Skript muss auf aktuelle Versionen der Bibliotheken angepasst werden. Internetverweise (Hyperlinks) müssen aktualisiert werden und, da zum Beispiel libdessert nun über GitHub bezogen werden kann, die Installationsroutinen angeglichen werden.
- Die Ziel-APIs des Perl- und des Bash-Skripts unterscheiden sich. Eine Vereinheitlichung trägt zur Übersicht bei und erhöht das Verständnis der Kompilier-Skripts. Dies ließe sich zum Beispiel über eine einzige exportierte Variable umsetzen, die im Bash-Skript definiert und vom Perl-Skript abgefragt wird.
- Die beiden Skripts könnten um Funktionen erweitert werden, die es ermöglichen, die Daemons herunterzuladen und zu kompilieren. Ebenso könnte ein aktualisiertes sepolicy-inject-Skript (siehe Kapitel 3.2.2) mitkompiliert werden.
- Durch Nutzung einer aktuelleren API, die eine verbesserte Unterstützung von Pthreads bietet, muss die Bibliothek **pthreadex**, die erweiterte Mutex-Funktionen zur Verfügung stellt, nicht mehr verwendet werden. Ab API-Version 9 ist sie redundant. Ebenso wird ab API 9 Die Nutzung von IPv6 ermöglicht. Die bisher manuell vorgenommene

Einbindung einer IPv6-Unterstützung ist nun nicht mehr erforderlich. Zusätzliche Bibliotheken für das Verwenden regulärer Ausdrücke müssen ab API 16 ebenfalls nicht mehr berücksichtigt werden.

- Da DES-SERT für Android-Geräte mit Android 4.1+ portiert wird, werden hauptsächlich Geräte mit ARMv7-Chips angesteuert. ARMv7 beinhaltet drei Profile: Applikationsprofil A, Echtzeitprofil (real-time-profile) R und Mikro-Controller-Profil M. Relevant für den Kompiliervorgang von DES-SERT ist das das ARMv7-A-Profil, welches für Mikroprozessoren der ARM Cortex-A Familie angepasst wurde. Durch Rückwärtskompatibilität der ARM-Prozessoren kann Code, der für ARMv5 kompiliert wurde, auch auf ARMv7-Chips ausgeführt werden. Jedoch kann eine höhere Leistung durch Nutzung der ARMv7-Features [56] erreicht werden. Überdies ließe sich eine Unterstützung für mehrere Architekturen in DES-SERT-App 2.0 integrieren, um auch die ARMv8-64-Bit-Mikroprozessoren optimal nutzen zu können, statt den AArch32-Modus zu benutzen.

3.3.2 Installation der Bibliotheken

Die Ordnerstruktur der Applikationsentwicklungsumgebung stellt unter anderem den *assets*-, den *jni*- und den *jniLibs*-Ordner zur Verfügung. Der jni-Ordner enthält nativen Code, während sich die kompilierten C-Bibliotheken bisher im assets-Ordner befinden und von dort aus auf Android installiert werden können. Der jniLibs-Ordner wird bisher nicht verwendet, kann aber für die Unterstützung mehrerer Architekturen verwendet werden.

Anders als unter Android 5.1, wird unter Android 6.0 nur eine Version der Bibliothek installiert. Die Applikation kann dann unter Umständen nach einem Bibliotheksnamen suchen, der nicht vorhanden ist (siehe Abbildungen 3.1 und 3.2).

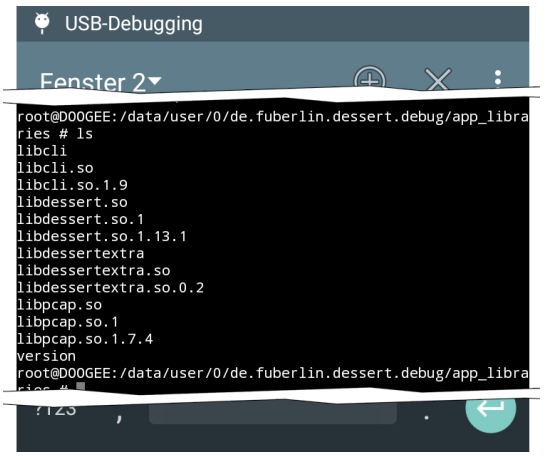


Abbildung 3.1: Bibliotheksinstallation unter Android 5.1

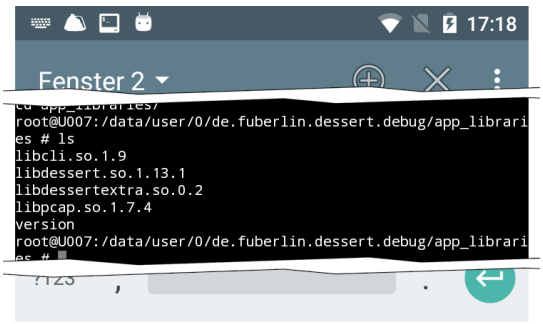


Abbildung 3.2: Bibliotheksinstallation unter Android 6.0

Optimierungsmöglichkeiten

Die Unterstützung mehrerer Architekturen wird üblicherweise mithilfe des jniLibs-Ordners gehandhabt, der statt des assets-Ordners in aktuellen Applikationen verwendet wird. Dazu werden Unterordner für die jeweilige Architektur angelegt, die Kopien der Bibliotheken enthalten, die für diese Architektur kompiliert wurden.

Allerdings müssten die Installationsmethoden geändert oder neu verfasst werden. Der Aufwand, welcher dazu nötig wäre, könnte dadurch gerechtfertigt werden, dass unter Stock-Android 6.0 die benötigten Bibliotheken nicht wie unter älteren Android-Versionen installiert werden. Die Bibliotheksinstallation wurde allerdings nicht mit einer CM-Version, die Android 6.0-Features besitzt (CM 13), getestet. Vor einer Anpassung der Installationsroutinen sollte überprüft werden, wie eine Installation unter CM 13 erfolgt.

3.4 Änderungen der Android-API

Neben der hinzugefügten Funktionen der API (siehe Kapitel 3.3.1) wurden auch einige bereits vorhandene Methoden umbenannt oder durch andere Schnittstellen ersetzt oder erweitert.

3.4.1 Notifications

Benachrichtigungen (Notifications) können vom Betriebssystem oder von einer Applikation erstellt werden. Sie werden in der obersten Leiste (Notification Bar) der GUI von Android angezeigt. Zum Beispiel zeigen im Hintergrund ausgeführte E-Mail-Programme oft über eine Notification an, dass eine E-Mail eingetroffen ist.

Das Notification-System, dass DES-SERT bisher benutzt, verwendet veraltete Methoden. Zwar werden Notifications erstellt und angezeigt, jedoch nur bis Android 5.x. Ab Android 6.0 werden diese Methoden nicht mehr unterstützt und können nicht mehr verwendet werden.

3.4.2 GUI

Darüber hinaus bieten aktuelle Android-APIs mehrere Möglichkeiten, das Handling der Applikation zu verbessern. DES-SERT benutzt bisher auch eine veraltete Ausführung von Tabs.

Optimierungsmöglichkeiten

- Eine Aktualisierung des Notification-Systems von DES-SERT ist nicht zwingend nötig für einen Betrieb unter Android 4 / 5. Da aber mit einer Zunahme von Geräten mit installiertem Android 6+ zu rechnen ist, lohnt eine Aktualisierung, sodass DES-SERT auf möglichst vielen Android-Versionen lauffähig ist. Notification-Icons werden ab Android-Version 5.0 allerdings nur in der Farbe weiß voll unterstützt [57] und sollten dementsprechend aktualisiert werden.

- Alle benutzten Bilder (auch die Notification-Icons) können in mehreren Auflösungen zur Verfügung gestellt werden. Abhängig von der eingestellten Auflösung des Displays des benutzten Geräts wird das Bild mit der entsprechenden Auflösung gewählt. Im Ressourcen-Ordner der Applikation kann statt einem drawable-Ordner eine Unterteilung in die Ordner drawable-ldpi (120dpi) bis analog drawable-xxxhdpi (640dpi) vorgenommen werden, um verschiedene Display-Auflösungen zu unterstützen. Bisher ist eine solche Unterteilung nicht erfolgt.
- Moderne Applikationen bieten oft die Möglichkeit, über „Drawer-Menus“ Applikationseinstellungen von kontextabhängigen Einstellungen zu kapseln. Drawer-Menus werden über die linke Seite der Applikation in die GUI gezogen („geswiped“) oder über eine Schaltfläche am linken oberen Rand der Applikation aktiviert. Im Gegensatz dazu werden kontextabhängige Menüs am rechten oberen Rand der Applikation oder über den teils am Gerät vorhandenen Menü-Knopf geöffnet. Die Integration eines Drawer-Menus setzt, abgesehen von einer komplizierten Implementierung, das Laden der umfangreichen support.v4-Bibliothek voraus. Es lässt sich allerdings ein Drawer-Menu simulieren, welches einfacher zu implementieren ist und keine zusätzlichen Bibliotheksaufrufe benötigt. Eine Swipe-Funktion kann mit einem simulierten Drawer-Menu nicht umgesetzt werden, sodass der Name Drawer-Menu faktisch nicht mehr korrekt ist.
- Die Swipe-Funktion kann allerdings genutzt werden, um ein schnelleres Umschalten der Tabs zu ermöglichen, so wie es oft in modernen Applikationen auch der Fall ist. Dazu muss lediglich eine aktualisierte Version des Tab-Frameworks der Android-API benutzt werden.

3.5 Android-Applikation

Die Android-Applikation (DES-SERT-App 1.0) portiert DES-SERT mit einer grafischen Benutzeroberfläche (englisch: graphical user interface, GUI) auf Android.

3.5.1 Aufbau der DES-SERT-App 1.0

Ein Tab-System teilt die GUI in die Bereiche Repository („*manage*“), Daemon-Übersicht („*installed*“) und Daemon-Steuerung („*running*“).

- Durch Aktivieren des **Repository**-Tab werden die auf einem Webserver abgelegten Daemon-Archive gelistet. Die in den Optionen von DES-SERT angegebene Web-Server-Adresse muss auf einen Ordner zeigen, der die index.xml-Steuerdatei enthält. Die Steuerdatei muss fünf Attribute enthalten, damit der Repository-Tab einen auf dem Webserver vorhandenen Daemon anzeigen kann. Dazu gehören der Name, die Version und der Dateipfad des Daemons, sowie die Applikations-Version und die DES-SERT-Version. Quellcode 3.1 zeigt beispielhaft eine index.xml mit einem Eintrag für den Daemon des-batman, der für DES-SERT 2.0 angepasst wurde. Stimmen die dort angegebene und die benutzte Applikationsversion nicht überein, wird im Repository-Tab der Name des Daemons durchgestrichen. Dennoch kann er von dort aus installiert werden.

```
1        <RepositoryIndex>
2        <Entry name="des-batman" version="2.0" applicationVersion="
             2.0.0-2.*.*" libraryVersion="1" path="des-batman-2.0.zip"/>
3        </RepositoryIndex>
```

Quellcode 3.1: Beispiel einer index.xml mit einem Eintrag

Ebenso ist eine lokale Installation von Daemons möglich. Dazu müssen sich die Daemon-Archive im Wurzel-Ordner der SD-Karte befinden. Die Installation eines Daemons ist nur möglich, sofern er innerhalb eines Zip-Archivs ausgeliefert wird, welches aus sechs Dateien besteht. Tabelle 3.1 zeigt eine Auflistung der Dateien, die sich in einem Daemon-Archiv befinden.

Datei	Zweck
launcher.xml	Platziert GUI-Steuerelemente im Daemon-Launch-Tab für Daemon-Startoptionen
config.template	Standardwerte für GUI-Steuerelemente im Daemon-Launch-Tab
manager.xml	Platziert GUI-Steuerelemente im „running“-Tab zur Laufzeitparametrisierung des Daemons
daemon	Ausführbarer Daemon
icon.png	Icon des Daemons
daemon.properties	Beinhaltet Informationen über den Daemon wie dessen Name und Versionsnummer

Tabelle 3.1: Dateien in einem Daemon-Archiv

- Die **Daemon-Übersicht** listet lediglich installierte Daemons. Von dort aus lassen sich die Daemons über ein für jeden Daemon separates Startmenü, den „Daemon-Launch-Tab“, starten. Der Aufbau des Daemon-Launch-Tab wird über die Datei launcher.xml gesteuert. Die Default-Startoptionen lassen sich in dem Daemon-Launch-Tab anpassen. Ist ein Daemon gestartet, lassen sich bis zum Beenden des Daemons keine weiteren Daemons starten.
- Ist ein Daemon gestartet worden, können in der **Daemon-Steuerung** die Laufzeitparameter des Daemons verändert werden. Über das Tab-sensitive Kontextmenü lassen sich Daemon und der dafür nötige Telnet-Dienst beenden. Custom-Befehle an den Daemon können über die Applikation oder über Telnet eingeben werden. Die GUI zeigt meist nicht alle Möglichkeiten, mit denen ein Daemon konfiguriert werden kann. Der Daemon ist in der Lage, direkt über libcli (siehe Kapitel 2.3.1) Custom-Befehle entgegenzunehmen. Eine Liste aller für den Daemon zur Verfügung stehenden Custom-Befehle lässt sich üblicherweise über den Custom-Befehl „`help`“ abrufen.

Überdies existiert ein separates Optionsmenü mit voreingestellten Standardwerten. Neben Einstellungen für System- und Mesh-Interface können dort Repository-URL, CLI-Port und die Pfade zu den benötigten Root-Befehlen (siehe Kapitel 3.2.1) konfiguriert werden.

Die Bibliotheken libpcap, libcli, libdessert und libdessert-extra werden als dynamische Bibliotheken (englisch: shared libraries) in der DES-SERT-Applikation benutzt. Bisher werden diese Bibliotheken für ARMv5 kompiliert.

Optimierungsmöglichkeiten

Während der generelle Aufbau der DES-SERT-App 1.0 beibehalten werden kann, besteht die Möglichkeit, die Tab-sensitiven Menüs zu entlasten und die Bedienung an moderne Standards anzupassen. Somit kann redundanter Code vermieden und die Wartung vereinfacht werden. Optionsmenü, „About“-Programminformation und Exit-Befehl könnten in ein applikationsübergreifendes Menü ausgelagert werden (siehe Kapitel 3.4.2).

3.5.2 Zugrundeliegendes Programmdesign

Quellcode-Ordnerstruktur

Die Quellcode-Ordnerstruktur von DES-SERT besteht aus dem *dessert-app-gradle*-Ordner, der (abhängig von Plattform, Compiler, Entwicklungsumgebung und Art des Projektimports) Metainformationen zum Build-Vorgang enthält und dem *app*-Ordner. Im darunterliegenden *app/app*-Ordner befindet sich eine Datei mit den wichtigsten Einstellungen der Applikation (zum Beispiel Ziel-API, benutzte Abhängigkeiten oder mögliche Build-Varianten wie „release“ oder „debug“) Darüber hinaus beinhaltet der *build*-Ordner weitere Metainformationen zum Build-Vorgang und der src/main-Ordner sämtlichen Code und Ressourcen der Applikation selbst.

Standardmäßig befinden sich im *assets*-Ordner die (dynamischen) Bibliotheken, im *res*-Ordner unter anderem sämtliche Grafiken, XML-Layout-Dateien, Text-Strings für die Unterstützung verschiedener Sprachen und die *AndroidManifest.xml*-Datei. Diese Datei beinhaltet Metainformationen zur Applikation (zum Beispiel die interne Versionsnummer, die von der App genutzten Android-Permissions oder die verwendeten Dienste der Applikation). Der Programm-Code befindet sich in den Ordnern *jni* und *java*. Ersterer beinhaltet den nativen C-Code, der java-Ordner sämtlichen Java-Code, welcher im nächsten Unterkapitel näher analysiert wird. Abbildung 3.3 verdeutlicht die Quellcode-Ordnerstruktur von DES-SERT.

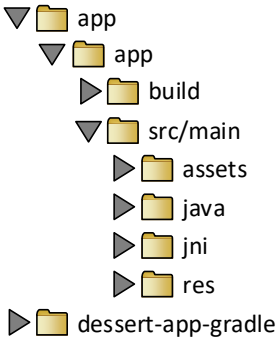


Abbildung 3.3: Vereinfachte Quellcode-Ordnerstruktur der DES-SERT-Applikation (nur Ordner und keine Dateien abgebildet)

Code-Struktur

Der java-Ordner ist in weitere Unterordner (Packages) eingeteilt, welche die Klassen von DES-SERT enthalten. Abbildung 3.4, welche einem UML-Modell nachempfunden ist, zeigt den Aufbau der Packages ohne test- und debug-Packages.

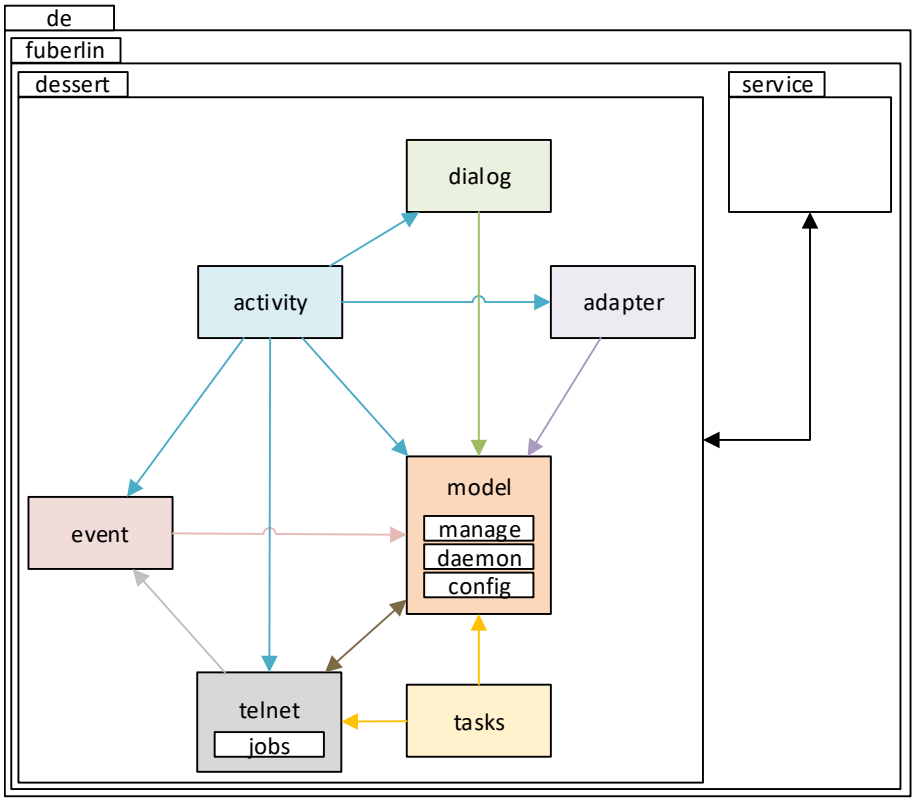


Abbildung 3.4: Vereinfachte Package-Modellierung von DES-SERT (ohne Klassen und ohne test- sowie debug-Packages)

Das service-Package beinhaltet sämtliche Dienste, die von DES-SERT benutzt werden. Dazu zählt der *NotificationService*, der die Benachrichtigungen verwaltet (siehe Kapitel 3.4.1).

Das dessert-Package beinhaltet die Klasse *DessertApplication*, in welcher globale Variablen und Methoden implementiert sind. Helfermethoden sind durch eine *Utils*-Klasse implementiert. Außerdem teilt sich das Package in folgende Sub-Packages.

- Das **model**-Package besteht aus zwei Klassen, die Methoden für die Applikations- und die Bibliotheksversion enthalten. Sie können Versionen untereinander vergleichen und Kompatibilität mit der Benutzung einer Versionsmaske überprüfen. Darüber hinaus existieren drei Sub-Packages im model-Package.

  Das model.config-Package enthält die Klassen für die einzelnen Elemente, die nach dem Einlesen einer Daemon-Konfigurationsdatei für den Aufbau des DES-SERT-Daemon-Konfigurationdatentyps benötigt werden.

  Im Package model.daemon werden sämtliche Informationen über einen Daemon gesammelt und verarbeitet. Dazu gehört zum Beispiel die Daemon-Version, der Name

des Daemons oder bei aktivem Daemon die Prozess-ID (PID). Abhängig vom aktiven Tab werden die Informationen verschiedener Datentypen angezeigt.

Das model.manage-Package beinhaltet Klassen, die Datentypen für die Verwaltung aktiver Daemons und das Erstellen von Telnet-Befehlen zur Verfügung stellen.

- Die Klassen im **telnet**-Package können Telnet-Befehle (CLI-Commands oder nur „Commands“) erstellen, mit denen die Daemons über CLI bedient und konfiguriert werden können. Der Telnet-Scheduler verwaltet die sequentielle Abarbeitung von Telnet-Jobs. Im Sub-Package telnet.jobs werden „Jobs“ für den Telnet-Scheduler erstellt. Ein Job enthält eine fixe Anzahl von auszuführenden Commands.
- Im **event**-Package sind nur Java-Interfaces mit Methodendeklarationen enthalten. Die Interfaces sind für die Aktualisierung der GUI zuständig, sofern gewisse Programmzustände erreicht werden beziehungsweise ein *Event* ausgelöst wird. Wenn zum Beispiel ein Daemon gestartet ist, soll die Schaltfläche zum Starten eines Daemons deaktiviert werden. Beim Beenden eines Daemons soll diese Schaltfläche wieder aktiviert werden.
- Im **dialog**-Package werden sämtliche Dialoge verwaltet. DES-SERT erstellt hier lediglich die Eigenschaften der „About“-Programminformationen.
- Im **activity**-Package befinden sich die Klassen, welche die Funktionalität der GUI steuern.
- Das **tasks**-Package beinhaltet eine Klasse für Datei- und Ordneroperationen wie zum Beispiel „chmod“-Operationen, das Parsen der Daemon-Konfigurationsdateien und die Installation der dynamischen Bibliotheken (siehe Kapitel 3.3.2). Aufrufe nativer WiFi-Tethering-Funktionen, welche über die dynamische Bibliothek *libnativetask.so* implementiert sind, erfolgen über eine weitere tasks-Klasse. Ebenso befindet sich eine Klasse für die XML-Verarbeitung im tasks-Package. Dort wird zum Beispiel das Parsing der GUI-Dateien der Daemon-Archive (siehe Kapitel 3.5.1) integriert.
- Adapter versorgen die GUI mit Daten (siehe Abbildung 3.5). Das **adapter**-Package besitzt einen Adapter, um die geparste Konfiguration eines laufenden Daemons zu laden (im „running“-Tab) und einen für die Auswahl der installierten Daemons (im „installed“-Tab). Die benötigten Daten werden aus dem jeweiligen Daemon-Archiv bezogen.

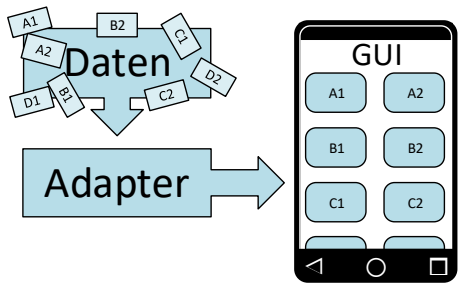


Abbildung 3.5: Konzept von Adaptern in Android

Optimierungsmöglichkeiten

Das Programmdesign kann beibehalten werden. Im Detail können jedoch viele der einzelnen Klassen mithilfe aktueller API-Funktionalitäten verbessert werden. Dadurch können zum Beispiel erhöhte Leistung und Sicherheit, sowie eine Verschlankung des Quellcodes erreicht werden. Darüber hinaus besteht die erwähnte Möglichkeit, die Ordnerstruktur der dynamischen Bibliotheken anzupassen (siehe Kapitel 3.3.2). Je nach Einstellung der benutzen Entwicklungsumgebung und Quellcode-Analyse-Tools, können mehrere Hundert bis zu mehreren Tausend geringfügige und kritische Schwachstellen erkannt werden.

KAPITEL 4

Implementierung

Die im Kapitel Analyse untersuchten Optionen zur Optimierung der DES-SERT-App 1.0 dienen als Entwurf für eine Implementierung der DES-SERT-App 2.0, die Android 4.1+ unterstützt. Im folgenden Kapitel werden die Implementierungsdetails der aktualisierten Applikation vorgestellt.

4.1 Aktualisierung der Bibliotheken und Kompilier-Skripts

Das Bash- und das Perl-Skript (siehe Kapitel 3.3.1) sind Bestandteil des Kompiliervorgangs der Bibliotheken. Aktualisierungen der Skripts und der Bibliotheken werden hier daher gemeinsam aufgeführt.

- Durch die Kompilier-Skripts der Bibliotheken, die für Android-Version 1.5 und 1.6 angepasst wurden, entstand eine unnötige Komplexität, die nun beseitigt wurde. Beide Skripts werden nun einheitlich für API 16 (Android 4.1) kompiliert. DES-SERT benötigt allerdings die nicht in API 16 vorhandene C-Header-Datei „*if_bonding.h*“, über die mehrere Ethernet-Schnittstellen in einem Ethernet-Kanal gebündelt werden können. Da ein aktuelles NDK im Installations-Skript heruntergeladen und benutzt wird, wird die Header-Datei aus API 21 benutzt, in der die sie erstmalig unterstützt wird. Die Implementierungen von IPv6, der erweiterten Mutex-Funktionen und der regulären Ausdrücke konnten entfernt werden. In aktuellen Android-APIs sind diese bereits enthalten.
- Neben dem NDK werden auch sämtliche andere Dateien durch das Installationsskript heruntergeladen. Es wurden alle Hyperlinks aktualisiert, sodass aktuelle Bibliotheksdateien zum Kompilieren benutzt werden und libdessert von einem öffentlichen GitHub-Repository (siehe Kapitel A.4), anstelle eines passwortgeschützten SVN-Servers, bezogen werden kann. Im Gegensatz zu den aktualisierten Bibliotheken libdessert, libcli und libpcap wird noch eine alte Version der libdessert-extra-Bibliothek benutzt. Sie ist als einzige Bibliothek noch für ARMv5 kompiliert.
- Die Kompilier-Skripts wurden um die Funktion erweitert, einige der für die DES-SERT-App 2.0 portierten Daemons zu kompilieren.

4.2 Aktualisierung der Daemons

Durch die Notwendigkeit, ALSR-Programmdateien zu benutzen (siehe Kapitel 3.2.3), müssen die für DES-SERT-App 2.0 portierten Daemons angepasst werden. Es wurde bereits über die vorhandenen Skripts (siehe Kapitel 3.3.1) mit *-fpic* kompiliert. Es wurde die Compiler-Flag *-fpie* in den jeweiligen Daemon-Makefiles hinzugefügt, sodass die Daemons auf Android 4.1+ lauffähig sind. Die Daemons wurden daher mit neuen Versionsnummern versehen, die nicht mehr kompatibel mit einer DES-SERT-Version kleiner 2.0 sind.

Da auch eine aktualisierte Bibliothek von libcli mit veränderter CLI-Command-Funktionssignatur benutzt wird, wurden die für die DES-SERT-App 2.0 portierten Daemons dementsprechend angepasst. Aktuelle libcli-Versionen sehen vor, dass ein CLI-Command konstant („*const*“) ist. Quellcode 4.1 zeigt die typische Funktionssignatur der CLI-Commands am Beispiel von *cli_run_command*.

```
1    int cli_run_command(struct cli_def *cli, const char *command);
```

Quellcode 4.1: Beispiel der Funktionssignatur des CLI-Commands *cli_run_command* aus libcli.h mit konstantem Parameter char *command

Für die DES-SERT-App 2.0 wurden die Daemons DES-B.A.T.M.A.N., DES-ARA, DES-OLSR und der Test-Daemon DES-HELLO portiert. Allerdings wurden in den GUI-Dateien von DES-HELLO keine Start- oder Laufzeitoptionen definiert.

4.3 Aktualisierung der Applikation

Viele Änderungen an der Applikation sind geringfügig und werden nicht im Detail erläutert. Im folgenden werden Implementationsdetails zu den relevanten Änderungen aufgeführt.

4.3.1 Aktualisierung der Bibliotheksinstallationsroutinen

Da auf Android 6.0+ die Installation der Bibliotheken nicht wie gewünscht erfolgte (siehe Kapitel 3.3.2), wurden die Installationsroutinen angepasst. Da verschiedene Namen für dieselbe Datei verwendet wurden, bestehen mehrere Möglichkeiten einer Implementation. Man könnte bereits installierte Bibliotheken kopieren und mit anderem Namen im gleichen Ordner einfügen. Einfacher erschien es, die Installation zu wiederholen. Es wird über eine vorher definierten Menge an Versionsnummern, die die Namen der Bibliothek wie gewünscht verändern, iteriert. Die bisherige Menge bestand lediglich aus der Versionsnummer, welche in einer Bibliothekskonfigurationsdatei angegeben wurde. Zum Beispiel ist die für libdessert angegebene Version „1.13.1“. Der Menge wurde nun eine *majorVersion* hinzugefügt, sodass neben *libdessert.so.1.13.1* auch *libdessert.so.1* installiert wird. Die *minorVersion* „.13.1“ wird entfernt. Die Aktualisierung hat keinen Einfluss auf die Installation auf früheren Android-Versionen.

4.3.2 SELinux und AndroidManifest-Permissions

Eine genaue Untersuchung der von DES-SERT verwendeten SELinux-Richtlinien (siehe Kapitel 3.2.2) steht noch aus. Bisher muss der moderate Modus unter Stock-Android für eine einwandfreie Ausführung der DES-SERT-App 2.0 benutzt werden.

Ein konkretes Beispiel für Stock-Android 6.0 mit striktem SELinux-Modus: Trotz Angabe der Android-Permission „*WRITE_EXTERNAL_STORAGE*“ in der AndroidManifest.xml (siehe Kapitel 3.5.2), entsteht standardmäßig der Fehler „*EACCES (Permission denied)*“ beim Schreiben oder Öffnen einer Prozess-ID-Datei für den jeweiligen Daemon.

4.3.3 Notification-System

Das Notification-System (siehe Kapitel 3.4.1) wurde angepasst, um auch Android 6.0+ zu unterstützen. Die von DES-SERT generierten Notifications unterstützen jetzt mehrere Zeilen und informieren darüber, welcher Daemon aktiv ist. Es war darüber hinaus nötig, neue Icons zur Verfügung zu stellen (siehe Kapitel 3.4.2).

4.3.4 GUI

In DES-SERT-App 2.0 verwendete Icons

Neben neuen einfarbigen Icons für Notifications (siehe Abbildung 4.1) wurden auch Icons für verschiedene Auflösungen hinzugefügt. Unterstützt werden alle Auflösungen von ldpi bis xxxhdpi und tvdpi.

Auslagerung von Applikationsoptionen

Gegenüber den Optionen (siehe Optimierungsmöglichkeiten in Kapitel 3.5.1), die vom jeweils aktiven Tab abhängig sind, wurden die Menüpunkte *Applikation beenden* („Exit“), *Einstellungen* („Preferences“) und *About*-Menü („About“) in ein separates Menü ausgelagert.

Das Menü wurde so gestaltet, dass es einem modernen Drawer-Menü ähnelt. Dadurch ist ein Import der großen support.v4-Bibliothek nicht nötig. Statt des Drawer-Menus wird ein einfach zu implementierender Dialog benutzt. Dieser kann nicht wie ein Drawer-Menu in die Applikation gezogen werden. Jedoch lässt sich das Optionsmenü, wie auch bei Drawer-Menus üblich, über die linke obere Ecke der Applikation aktivieren. Dort befindet sich ein DES-SERT- mitsamt typischem Drawer-Menü-Icon (siehe Abbildung 4.1), um grafisch zu verdeutlichen, dass diese Funktion implementiert wurde.

Tab-spezifische Optionen und Swipe-Feature

Der „manage“-Tab bot bereits vorher die Möglichkeit, Daemons aus Daemon-Archiven von der SD-Karte zu installieren (siehe Kapitel 3.5.1). Sollte die Option gewählt werden, wird eine eindeutigere Fehlermeldung ausgegeben, sofern sich keine Daemon-Archive im Wurzel-

Ordner der SD-Karte befinden. Vorher konnte die genaue Funktionalität der Option nur dem Quellcode des Programms entnommen werden.

Im „running“-Tab existierte bisher eine *Telnet*-Option. Die bereits über DES-SERT aufgebaute Telnet-Verbindung soll bei Aktivierung der Option beendet werden und eine entsprechende installierte Applikation geöffnet werden. Ohne diese Telnet-Verbindung kann der Daemon nicht mehr über die DES-SERT-App 2.0 konfiguriert oder der Status eingesehen werden. Dadurch kann ebenso nicht mehr über DES-SERT festgestellt werden, ob ein Daemon aktiv ist oder nicht. Eine entsprechende Telnet-Applikation wird durch Aktivieren der Telnet-Option nicht geöffnet. Aus diesem Grund wurde die Telnet-Option ersatzlos entfernt. Die Steuerung des Daemons erfolgt weiterhin direkt über die DES-SERT-App 2.0 mit der Option *Custom command*.

Die Tab-Übersicht kann nun durch wischen („swipen“) geändert werden. Die Applikation fühlt sich somit moderner an und lässt sich besser bedienen. Zusätzlich kann auch über ein Antippen der jeweilige Tab erreicht werden.

Vergleich zu alter GUI

Abbildung 4.1 zeigt die DES-SERT-App 2.0-GUI im Detail. Das linke Icon der Notification-Bar ist das aktualisierte Notification-Icon der DES-SERT-App 2.0. Darunter befindet sich das typische Drawer-Menu-Icon mitsamt DES-SERT-Icon.

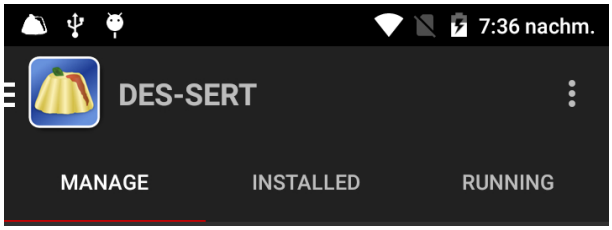


Abbildung 4.1: Oberer Teil der DES-SERT-App 2.0-GUI mit Notification-Bar

Abbildung 4.2 zeigt die DES-SERT-App 1.0 zum Vergleich. Das geöffnete Repository-Tab verdeutlicht, dass des-ara 0.6 nicht kompatibel mit der Applikation ist.

4.4 Programmfehler

4.4.1 Charakteristische Fehler-Codes beim Starten eines Daemons

Es existieren typische Bash-Fehler-Codes; allerdings sind diese abhängig von der ausgeführten Applikation [58]. Der Sinn eines zurückgegebenen Fehler-Codes muss demnach erst interpretiert werden. Durch empirische Studien können einige der Fehler(-Codes), die beim Starten eines Daemons auftreten können, beschrieben werden. Dabei ist zu beachten, dass ein Right-Shift (RSH) von 8 Bits nötig ist, um einen typischen Bash-Fehler-Code zu erhalten. Nach [59] wird neben dem eigentlichen Fehler-Code zusätzlich der Grund in einem niedrigeren Byte ausgegeben. Der höherwertige Byte enthält den Fehler-Code Die Tabellen 4.1 und 4.2 listen bekannte Bash-Fehler-Codes, die in DES-SERT vorkommen können.

Abbildung 4.2: DES-SERT-App 1.0-GUI mit Repository

Fehler-Code	Generelle Bedeutung	Bedeutung in DES-SERT
1 = RSH(256, 8)	Sammlung allgemeiner Fehler	Datei nicht gefunden; fehlendes Compiler-Argument „-fpie“
2 = RSH(512, 8)	Fehlbenutzung eines Bash-Befehls	Für falsche Architektur kompiliert; Aufruf einer Bibliothek, aber ausführbare Datei erwartet
11 = RSH(2816, 8)	keine Bedeutung	Zugriff auf nicht vorhandene oder fehlerhafte (System)dateien
126 = RSH(32256, 8)	Aufgerufener Befehl kann nicht ausgeführt werden	Unbekannt
127 = RSH(32512, 8)	Befehl nicht gefunden	Keine Root-Rechte
128 = RSH(32768, 8)	Ungültiges Argument für exit-Befehl	Unbekannt

Tabelle 4.1: Beim Starten eines Daemons auftretende Fehler-Codes I

128+n = RSH(32768 + 256 · n, 8)	Fehler-Signal "n"	Siehe Fehler-Code 132
132 = RSH(33792, 8)	n = 4 = „illegal instruction“	Typischerweise für falsche Architektur kompiliert
255 = RSH(65280, 8)	Argument von exit-Befehl nicht im Intervall von $[0, 255]$	Unbekannt

Tabelle 4.2: Beim Starten eines Daemons auftretende Fehler-Codes II

4.4.2 Fehler bei Daemon-Implementationen

Einige der Daemons sind noch nicht oder nur beschränkt in der DES-SERT-App 2.0 lauffähig. Die Portierung von DES-ARA auf Android ist fehlgeschlagen. Der Daemon lässt sich nicht starten. B.A.T.M.A.N. hingegen lässt sich starten. Befehle können jedoch größtenteils nicht über die GUI, sondern nur über Custom commands eingegeben werden.

KAPITEL 5

Evaluation

In diesem Kapitel wird die Funktionalität und Leistung von DES-SERT und den DES-Daemons untersucht. Dazu wurden die DES-Daemons OLSR und B.A.T.M.A.N. auf Android portiert. Diese werden aus der DES-SERT-App 2.0 gestartet.

5.1 Testumgebung

5.1.1 Testgeräte und Betriebssysteme

Für die Evaluation stehen folgende Geräte zur Verfügung. Den Geräten wird einfachheitshalber und, da doppelte Geräte vorhanden sind, jeweils ein übliches Kürzel zugeordnet. Eine detaillierte Liste der Testgeräte kann in Tabelle A.2 eingesehen werden.

- HTC Desire HD: *DHD.1* und *DHD.2*
- Asus Nexus 7 (Grouper): *N7.1*, *N7.2*, *N7.3* und *N7.4*
- Samsung Galaxy S II GT-I9100: *S2*
- Motorola Moto E2 (XT1524): *Moto-E*
- Doogee X5: *X5.1*
- Ulefone U007: *U007*
- HTC Desire Eye: *DEye*

Die Testgeräte DHD.1, DHD.2, S2 und Moto-E unterstützen den MANET-Modus durch installiertes CM (Version 12.1), welches Android 5.1-Features beinhaltet (siehe Kapitel 3.1.1), nur bedingt. Voll unterstützt wird unter CM 12.1 lediglich das Asus Nexus 7 (Grouper) [45]. Dieses wird deshalb hauptsächlich innerhalb der Evaluation benutzt. Auf Testgerät X5.1 ist Stock-Android 5.1 installiert. Auf den Testgeräten U007 und DEye ist Stock-Android 6.0 installiert. Im Gegensatz zu allen anderen Testgeräten wurde das DEye nicht „gerootet“ und kann dementsprechend auch keine Root-Befehle ausführen oder den SELinux-Modus wechseln. Bei der Evaluation entstandene Fehler sind in Tabelle A.3 gelistet.

5.1.2 Verfügbare Funkfrequenzen

Von den MANET-fähigen Testgeräten verfügt lediglich Testgerät S2 über die Möglichkeit, 5 GHz-Frequenzen zu nutzen. Es werden daher die 2,4 GHz-Frequenzen [60] verwendet. In Europa ist die Benutzung 13 verschiedener 2,4 GHz- beziehungsweise IEEE 802.11 b/g/n-Kanäle (mit 100 mW Sendeleistung) im Frequenzbereich 2,400 GHz bis 2,4835 GHz erlaubt. IEEE 802.11 b-Kanäle besitzen eine 22 MHz-Kanalbreite, IEEE 802.11 g/n-Kanäle eine 20 MHz-Kanalbreite und 16,25 MHz pro Träger (siehe Abbildung 5.1). Kanal 1 ist in beiden Fällen ein überlappungsfreier Kanal: Die Frequenzen überlappungsfreier Kanäle überschneiden sich nicht mit denen anderer überlappungsfreier Kanäle. Unter IEEE 802.11 g/n sind die Kanäle 1, 5, 9 und 13 überlappungsfrei.

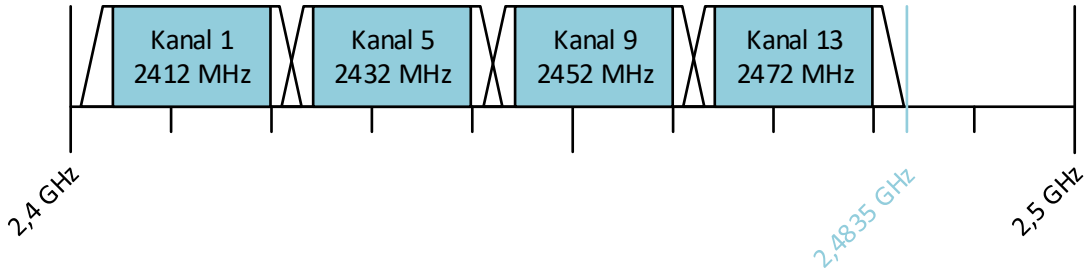


Abbildung 5.1: IEEE 802.11 g/n Kanalfrequenzen

Gegenüber den 5 GHz-Frequenzen besitzen 2,4 GHz-Frequenzen eine größere Reichweite; Störungen durch weitere nahe 2,4 GHz-Netze treten daher in dicht besiedelten Gebieten häufiger auf.

5.2 Versuchsaufbau

Die Daemons OLSR und B.A.T.M.A.N. sollen auf die beiden Versuchsmetriken Latenz und Durchsatz untersucht werden. Mit den Diagnosewerkzeugen `ping` und `iperf` lassen sich Latenzzeiten respektive der Durchsatz feststellen.

Für die Benutzung von (OSI-)Schicht-3-Werkzeugen ist es unerlässlich, die IP-Adresse des Mesh-Interface zu löschen, sodass die Sys-Interfaces miteinander wie gewünscht kommunizieren können. Solange sich die Geräte in der direkten Nachbarschaft befinden, funktionieren zwar Werkzeuge wie `arping`, aber es wird zum Beispiel für eine erhaltene Echo-Anfrage von `ping` auf dem Zielgerät kein Echo erzeugt. Ebenso empfangen die Zielgeräte keinerlei Daten von Schicht-3-Werkzeugen mehr, falls sich die Geräte weiter als einen Hop voneinander entfernt befinden.

Es wird ausschließlich im MANET-Modus mit Geräten, auf denen CM 12.1 installiert ist, evaluiert. Im Access-Point-Modus würden sämtliche Verbindungen über einen Router geregelt. Damit wäre nur eine Sterntopologie beziehungsweise eine Linientopologie aus Router und zwei Endgeräten möglich. Konkret werden für die Leistungsbewertung die Geräte N7.1 bis N7.4 benutzt. Zu beachten ist, dass sich die Geräte nicht im Standby-Modus befinden dürfen, weil die Daemons nur aktiv sind, solange das Display aktiv ist.

Auf den Testgeräten sind neben CM 12.1 und DES-SERT zusätzlich ein Terminal-Emulator, sowie die Netzwerktools `iw` und `iwconfig` installiert. Übliche Systemwerkzeuge wie `ping` sind mit busybox installiert worden; `iperf` hingegen musste manuell installiert werden.

Die Testumgebung befindet sich innerhalb eines dicht besiedelten Wohngebiets. Über einige Testmessungen konnte festgestellt werden, dass keine signifikante Abweichung zu den Messungen in einem Gebiet ohne WLAN-Interferenzen besteht. Die in der Testumgebung installierten Router verfügen über einen Algorithmus, der automatisch die günstigste Frequenz für das jeweilige Netz wählt, um Interferenzen zu minimieren. Die Frequenz des MANETs muss daher üblicherweise nach einer beliebigen aber fixen Wahl nicht mehr angepasst werden.

Durch die Testmessungen konnten auch Probleme bei den Messungen des Durchsatzes festgestellt werden. Im Client-Modus von `iperf` ist es nicht möglich, sich zum `iperf`-Server zu verbinden. Es besteht zwar die Möglichkeit, über den Parameter „`-B`“ eine IP-Adresse zu binden. Jedoch ist es nicht möglich, wie bei `ping` mit dem Parameter „`-I`“ ein Interface zu wählen, über welches die Daten versendet werden. Über ein Binden der IP-Adresse wird nicht der gewünschte Effekt erzielt. In der Konsequenz wird von Durchsatzmessungen vorerst abgesehen.

Als Topologie wird eine Linientopologie, wie in Abbildung 5.2 dargestellt, benutzt. Messungen werden von dem dort beschriebenen Knoten S aus initiiert; `ping`-Anfragen werden von dort aus an die Knoten a, b und D verschickt.

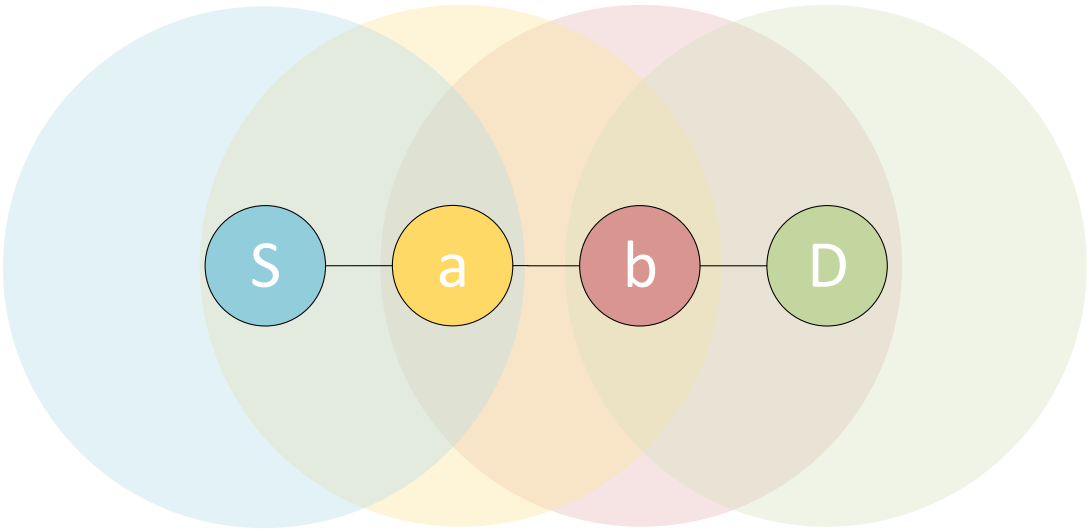


Abbildung 5.2: Die Linientopologie mit dem Startknoten S, den intermediären Knoten a, b und dem Zielknoten D ist so aufgebaut, dass nur die intermediären Knoten zwei direkte Nachbarn besitzen. Start- und Zielknoten haben je nur einen Knoten in ihrer 1-Hop-Nachbarschaft und einen weiteren Knoten in ihrer 2-Hop-Nachbarschaft.

Vor der Analyse der einzelnen Latenzzeiten wird die Funktionalität der DES-SERT-App 2.0 getestet und die Versuchsumgebung empirisch analysiert. Dazu werden auch die Geräte untersucht, die den Ad-hoc-Modus nicht voll unterstützen (siehe auch Tabellen A.2 und A.3).

5.3 Versuchsdurchführung

5.3.1 Empirische Netzwerkstabilität und Akkuverbrauch

Mit den Geräten DHD.1, DHD.2, Moto-E und S2 ist es oft nicht möglich, an einem aufgespannten MANET teilzunehmen. Meist ist dies jedoch auf mindestens einem der Geräte möglich, sodass diese dann an dem aufgespannten Netz teilnehmen können.

Da die genannten Geräte nicht voll unterstützt werden, kann es allerdings zu willkürlichen Ausfällen oder Unterbrechungen der Netzwerk-Verbindungen kommen. Die DES-Daemons können dadurch terminieren. Ebenso ist es möglich, dass Geräte nicht mehr in den Ad-hoc-Modus wechseln und damit auch nicht mehr an einem MANET teilnehmen können. Auch durch die Benutzung der Netzwerk-Programme *iw* und *iwconfig* konnte kein stabiles MANET mit aktiven DES-Daemons auf mehr als zwei nichtunterstützten Geräten erstellt werden (siehe auch Tabelle A.3).

Mit den unterstützten Geräten N7.1 bis N7.4 konnte jedoch ein stabiles Test-MANET erstellt werden. Auch bei Verlassen der Empfangs- beziehungsweise Sendereichweite (circa 19,8 Meter) der anderen Geräte im MANET, bleiben die Daemons aktiv und aktualisieren weiterhin ihre Routing-Tabellen.

In einem Netzwerk mit aktiven DES-Daemons ist ein stark erhöhter Akkuverbrauch festzustellen, unabhängig davon, ob sich die Geräte im MANET- oder im Access-Point-Modus befinden. Bei angeschaltetem Display ist der Akkuverbrauch circa um den Faktor 10 erhöht. Die konkreten Akkulaufzeiten können dabei vom jeweilig benutzten Daemon abhängen.

Wie in Kapitel 5.2 beschrieben, sind keine signifikanten Messungsunterschiede, wie zum Beispiel stark erhöhte Latenzzeiten, zwischen der Testumgebung und einem interferenzlosen Gebiet festzustellen.

5.3.2 Funktionalität der DES-SERT-App 2.0

5.3.3 Ausführbarkeit der Applikation

Die Applikation wurde für Android 4.1+ entwickelt, mit besonderem Fokus auf CM 12.1 beziehungsweise Android 5.1. Die Applikation ist auf jedem der Testgeräte ausführbar und unterstützt somit die Betriebssysteme Android 5.1, Android 5.1.1, Android 6.0, CM 12.1 und CM 13.0. Es wird davon ausgegangen, dass die DES-SERT-App 2.0 auf CM 11+ beziehungsweise den Android Versionen 4.1 bis 4.4.4, sowie zukünftigen CM- und Android-Versionen ebenfalls einwandfrei ausführbar ist. Diese Angaben gelten für frische Installationen (nach einer Betriebssystemaktualisierung) der DES-SERT-App 2.0.

5.3.4 Installation der Bibliotheken

Die Bibliotheksinstallation funktioniert nach Aktualisierung der Installationsroutinen unter Stock-Android 6.0+ wie gewünscht (siehe Kapitel 4.3.1).

5.3.5 Starten der Daemons

Das Starten der Daemons ist auf allen Root-Testgeräten mit Stock-Android 5.1 und 6.0 oder CM 12.1 und CM 13 möglich gewesen. Die Benutzung von Root-Befehlen (siehe Kapitel 3.2.1) ist essentiell. Daher konnte das Testgerät DEye keine Daemons starten. Abhängig vom Gerät musste teils der moderate SELinux-Modus verwendet werden (Siehe Kapitel 3.2.2).

5.3.6 Messungen der Latenz

Die Ergebnisse werden in Boxplots dargestellt. Es werden zusätzlich Ausreißer und extreme Ausreißer analysiert. Dafür werden die Whisker des Boxplots auf das maximal 1,5-fache des Interquartilsabstands (englisch: interquartile range, IQR) skaliert. Werte, die größer als $3 \cdot IQR$ sind, werden als extreme Ausreißer behandelt. Alle Messungen werden sequentiell vorgenommen, sodass sich die Messungen nicht gegenseitig bedingen können.

Ausreißer und verlorene Pakete werden folgendermaßen dargestellt:
- obere Ausreißer
- obere extreme Ausreißer
- Paketverlust

Untere (extreme) Ausreißer werden nicht berücksichtigt, da in keiner der Messungen untere (extreme) Ausreißer festgestellt wurden.

Erwartete Ergebnisse

Unabhängig von Art und Einstellungen des Daemons wird erwartet, dass die Latenzzeit pro Hop logarithmisch zunimmt [61] und der Datendurchsatz logarithmisch [62] abnimmt. Abbildung 5.3 visualisiert diesen Zusammenhang.

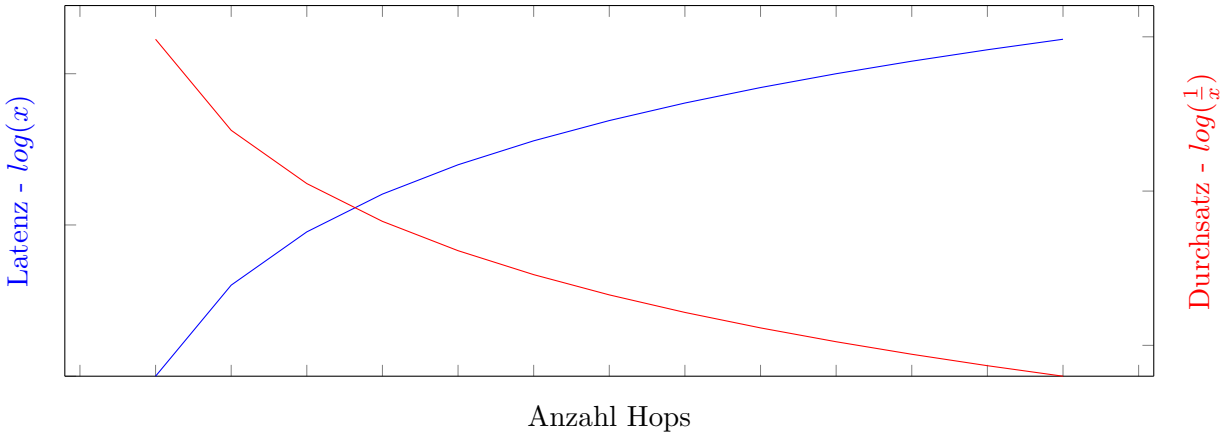


Abbildung 5.3: Abhängig von der Anzahl intermediärer Hops nimmt der Durchsatz ab und die Latenzzeiten nehmen zu.

Konkrete Latenz

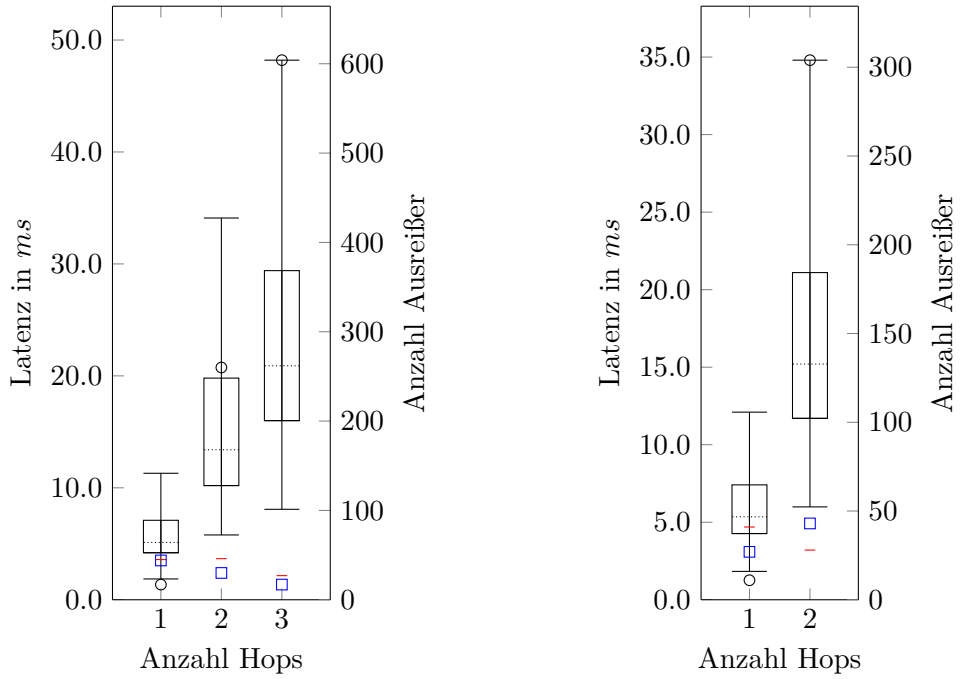


Abbildung 5.4: OLSR mit Standardeinstellungen, Ping mit 64 (links) und 128 Bytes (rechts)

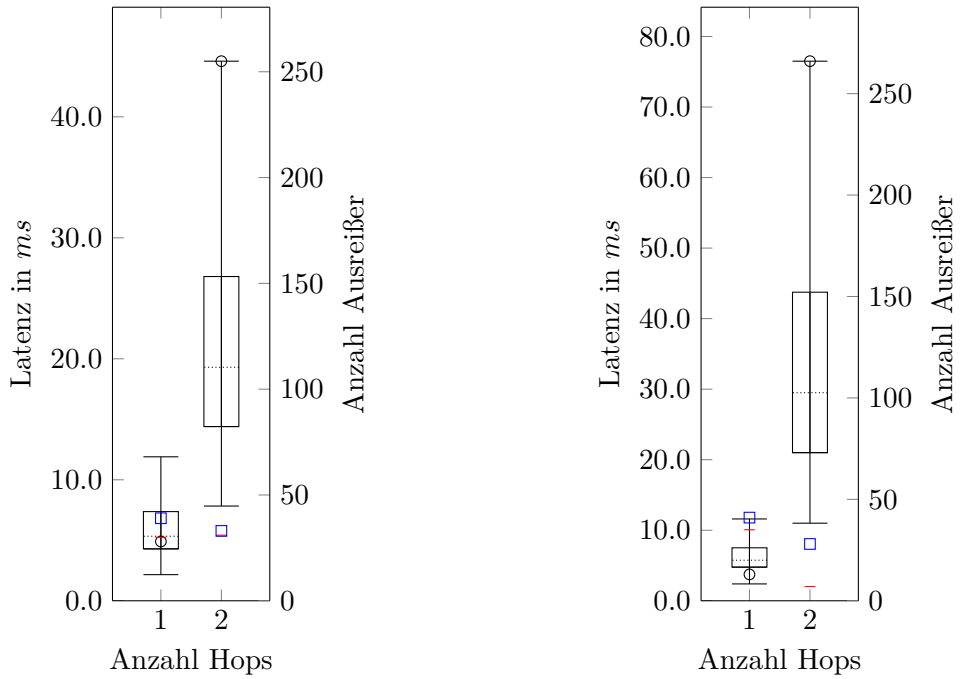


Abbildung 5.5: OLSR mit Standardeinstellungen, Ping mit 256 (links) und 512 Bytes (rechts)

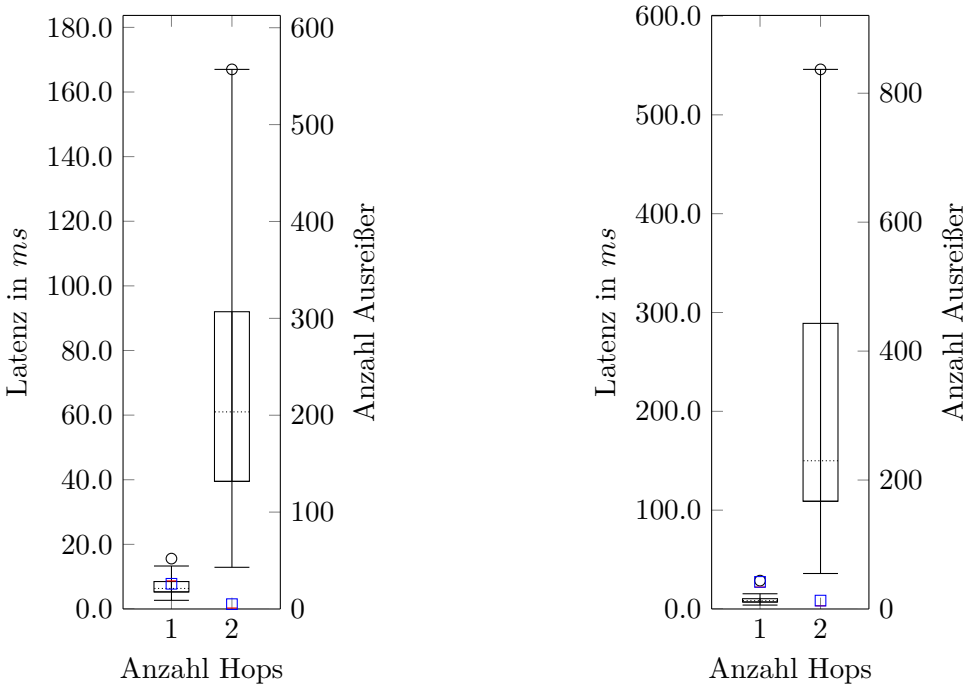


Abbildung 5.6: OLSR mit Standardeinstellungen, Ping mit 1024 (links) und 2048 Bytes (rechts)

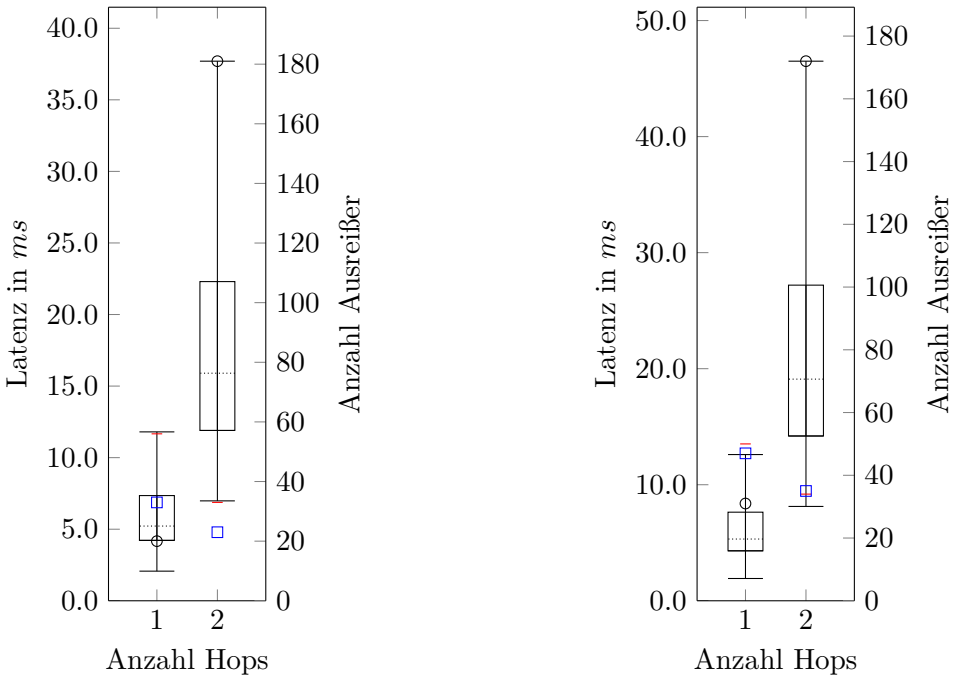


Abbildung 5.7: B.A.T.M.A.N. mit Standardeinstellungen, Ping mit 64 (links) und 128 Bytes (rechts)

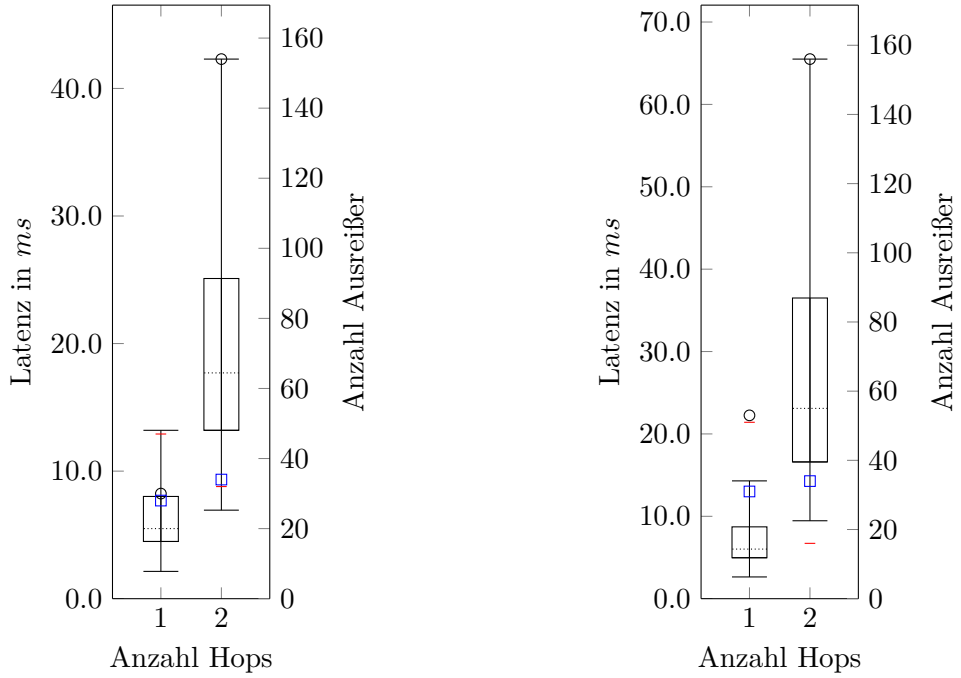


Abbildung 5.8: B.A.T.M.A.N. mit Standardeinstellungen, Ping mit 256 (links) und 512 Bytes (rechts)

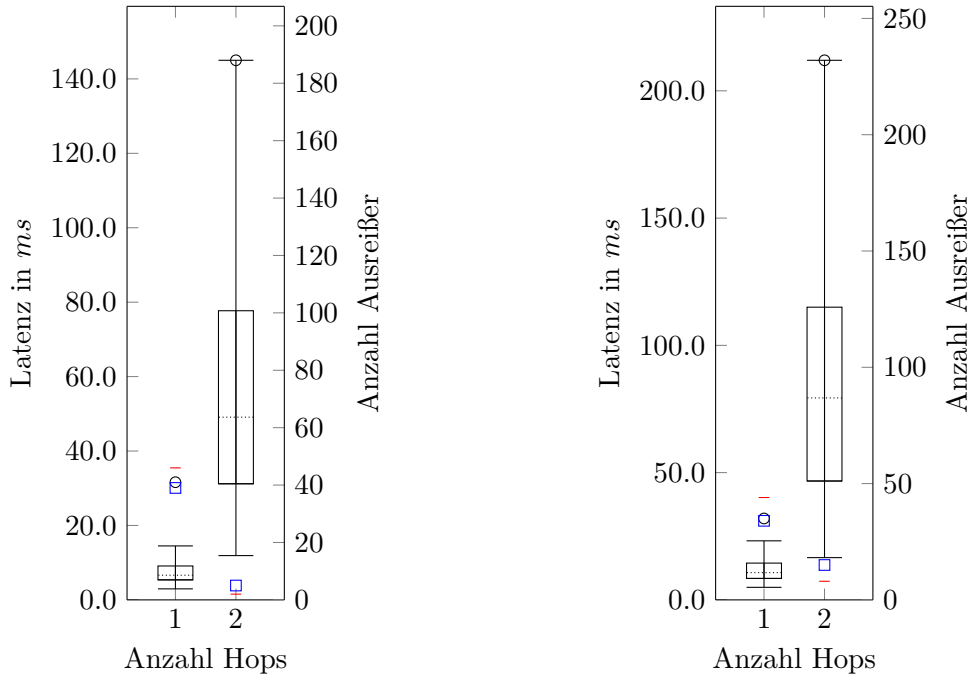


Abbildung 5.9: B.A.T.M.A.N. mit Standardeinstellungen, Ping mit 1024 (links) und 2048 Bytes (rechts)

Nachwort und Ergebnisanalyse

Ursprünglich sollten Messungen ab vier Hops wie in Abbildung 5.3 interpoliert werden. Da bereits ab dem dritten Hop oft direkte Nachbarn temporär oder dauerhaft nicht mehr erkannt wurden oder alternativ der 2-Hop-Nachbar in der direkten Nachbarschaft erkannt wurde, sind lediglich Messungen mit bis zu zwei Hops aussagekräftig, da diese stabil ausgeführt werden konnten. Dadurch kann eine Interpolation jedoch nicht sinnvoll umgesetzt werden. Abbildung 5.4 zeigt eine `ping`-Messung mit 64 Bytes über drei Hops. Bei über 60 % verlorener Pakete können die Messungen noch hinreichend akkurat sein. Jedoch kann durch die beschriebenen Probleme der Nachbarschaftsknoten auch über zwei Hops gemessen worden und die hohe Latenz über die Entfernung zwischen den Knoten zustande gekommen sein. Dieses Problem konnte, trotz vielfacher geografischer Neupositionierung und Einstellen der Sendeleistung (txpower), in der Testumgebung und ebenso in einer Umgebung ohne weitere drahtlose Netzwerke in der Nähe, beobachtet werden. Ähnliches konnte auch in früheren Versionen der DES-SERT-Applikation festgestellt werden [63].

Durch schnell steigende Latenzzeiten und größere Paketverluste kann bei 2-Hop-Messungen die totale Anzahl Ausreißer geringer ausfallen, als bei 1-Hop-Messungen.

Während sich die gemessenen 1-Hop-Latenzen von OLSR kaum zu denen von B.A.T.M.A.N. unterscheiden, kann B.A.T.M.A.N. bessere Ergebnisse mit zwei Hops erreichen. Zudem wurden während der Testphase deutlich weniger Paketverluste festgestellt. Dies gilt für sämtliche 2-Hop-Latenzmessungen.

Die Benutzung von DES-SERT und den DES-Daemons bietet zwar zum Beispiel eine erhöhte Sicherheit im Vergleich zur Benutzung der Daemons als Kernel-Modul, birgt aber auch einige Nachteile [34]. Weitergeleitete Pakete verursachen je zwei Kontextwechsel und damit eine stark erhöhte Systemlast. Paketverluste können durch zusätzlichen Paket-Overhead zunehmen. Ein Abgleich der Messungen mit entsprechenden Kernel-Modulen der Daemons könnte gegebenenfalls Aufschluss über die Ursachen der problematischen Nachbarknotensuche und der hohen Latenzzeiten unter der DES-SERT-App 2.0 geben. Bei der Verwendung von Kernel-Modulen muss jedoch beachtet werden, dass bereits Linux keine geeigneten Schnittstellen für die Einbindung reaktiver Routing-Protokolle zur Verfügung stellt. Es ist daher davon auszugehen, dass Android ebenfalls keine besitzt.

In zukünftigen Tests (siehe auch Kapitel 6.1) ist eine weiter optimierte Testumgebung wünschenswert, um sämtliche äußere Störfaktoren, wie zum Beispiel Interferenzen, zu minimieren. Ein dediziertes Testbed scheint zu diesem Zweck geeignet. Eine genaue Einstellung der Sendeleistung und eine damit einhergehende Messung der Sendereichweite der Geräte könnte zukünftige Experimente optimieren. Dadurch könnten Paketverluste minimiert und die Messqualität maximiert werden. Zusätzlich sind Vergleiche zwischen weiteren Daemons empfehlenswert, um eine genaue Aussage darüber zu treffen, ob DES-SERT und inwiefern ein Daemon Auswirkungen auf das Erkennen von Nachbarknoten hat. Gleichzeitig müsste sichergestellt werden, dass die Ad-hoc-Modus-Implementierung des Betriebssystems funktional ist beziehungsweise die entsprechenden WLAN-Adapter voll unterstützt werden. Dabei kann die Installation weiterer Analysewerkzeuge unterstützen. Eventuell müssten darüber hinaus zusätzlich eigene Werkzeuge entwickelt werden, die speziell auf die Funktion von DES-SERT angepasst sind.

KAPITEL 6

Resümee und Ausblick

Ziel der Arbeit war es, die DES-SERT-Applikation zu aktualisieren und auf aktuellen Android-Versionen (ab Version 4.1) lauffähig zu machen. Dazu mussten die Abhängigkeiten von DES-SERT aktualisiert und die DES-Daemons entsprechend angepasst werden. Zu Evaluationszwecken wurden die beiden Daemons B.A.T.M.A.N. und OLSR auf Android portiert.

Eine Einführung in Routing-Technologien (siehe Kapitel 2) informiert über speziell für MANETs und drahtlose Mesh-Netzwerke erstellte beziehungsweise angepasste Verfahren und Klassifizierungen von Routing-Protokollen und -algorithmen. Wichtige Wegwahltechniken wie das Hop-by-Hop-Routing, das auch oft von den DES-Daemons verwendet wird, werden neben den typischen Ad-hoc-Netz-Routingverfahren erklärt; DES-SERT unterstützt die Verwendung proaktiver und reaktiver Routing-Protokolle gleichermaßen. Eine Klassifizierung der Daemons wurde vorgenommen, indem sie entweder der Klasse Link-State- oder der Klasse Distanzvektor-Routing-Protokoll zugeteilt wurden. Ein Routing-Daemon kann aber mehrere Routing-Metriken wie ETX, ETT, oder den Hop-Count nutzen. Zusätzlich zu speziell für einen Daemon angepasste Technologien, zum Beispiel TQ für B.A.T.M.A.N., kann anhand dieser Informationen verstanden werden, wie ein DES-Daemon funktioniert und mit welchen Verfahren er arbeitet. Zu den unter DES-SERT verwendeten Daemons gehören unter anderem AODV, OLSR, ARA, B.A.T.M.A.N. und DSR.

Darauf aufbauend wurde die Funktionsweise von DES-SERT erklärt (siehe Kapitel 2.3). In der DES-SERT-Architektur werden Daemons im User-Space genutzt. Mithilfe der Abhängigkeiten von DES-SERT im Kernel-Space wird die Infrastruktur zum Ausführen und Verwalten der Daemons zur Verfügung gestellt. Die in C verfassten Abhängigkeiten respektive Bibliotheken libcli, libpcap, libdessert und libdessert-extra werden mithilfe des Android-NDK in die Applikation eingebunden. Ein DES-SERT-Routing-Daemon wird mit folgenden Framework-Features von DES-SERT erstellt. Neben einem oder mehreren Mesh-Interfaces kann gewählt werden, ob ein TUN- oder ein TAP-Gerät als Sys-Interface zum Einsatz kommt. Mesh- und Sys-Interface bekommen je eine Prozesspipeline zugeordnet. Callback-Funktionen können periodisch ausgeführt werden, um zum Beispiel Routing-Tabellen aktuell zu halten. DES-SERT bietet größtenteils ein automatisiertes Speichermanagement.

Eine Aktualisierung der Applikation (siehe Kapitel 4) basiert auf einer Analyse (siehe Kapi-

tel 3) des Programmcodes der ursprünglichen Applikation, der zugehörigen C-Bibliotheken und der DES-Daemons. Die Installationsroutinen wurden aktualisiert, sodass aktuelle Bibliotheken und die entsprechende Android-API 16 als Ziel-API verwendet werden. DES-OLSR, DES-BATMAN und der Testdaemon DES-HELLO werden einfachheitshalber innerhalb des Skripts mitkompiliert. Der Quellcode ist darüber hinaus nun durch die Umstellung auf GitHub komplett öffentlich. Bibliotheken und Daemons werden für ARMv7 kompiliert und benötigen keine externen Bibliotheken mehr für IPv6, reguläre Ausdrücke oder erweiterte Mutex-Funktionen. Mit der Aktualisierung des Notification-Systems und der Bibliotheksinstallation der Applikation unterstützt die DES-SERT-App 2.0 auch Android 6.0+. Die GUI wurde um Swipe-Tabs erweitert und die einzelnen Optionsmenüs wurden in Tab-Menü und Applikations-Menü aufgeteilt. Die nicht funktionierende Telnet-Option aus dem „running"-Tab-Optionsmenü wurde entfernt.Sämtliche Icons wurden nach aktuellen Design-Standards für Android angepasst und für mehrere Auflösungen zur Verfügung gestellt.

Zuletzt wurde eine Leistungsbewertung (siehe Kapitel 5) der DES-SERT-App 2.0 vorgenommen. Anhand des OSI-Schicht-3-Werkzeugs `ping` konnten Latenzzeiten mit verschieden großen Paketen festgestellt werden. Dazu wurde eine Linientopologie benutzt. Dabei wurde sukzessiv mit einem Single-Hop begonnen und mit bis zu insgesamt zwei Knoten beziehungsweise einem Hop getestet. Durch Benutzung des MANET-Modus und WLAN im Allgemeinen ist es außerordentlich schwierig, eine umfangreiche und aussagekräftige Leistungsbewertung zu erstellen. Stabilitätsprobleme des Netzwerks führten dazu, dass lediglich 1-Hop-Messungen und gegebenenfalls 2-Hop-Messungen verwertbare Ergebnisse liefern konnten. Eine angedachte Interpolation der Ergebnisse mit bis zu acht Knoten musste entsprechend verworfen werden.

6.1 Ausblick

Im Rahmen dieser Arbeit wurden OLSR und B.A.T.M.A.N. für die DES-SERT-App 2.0 portiert. Jedoch konnten einige Probleme beim Portierungsvorgang festgestellt werden. ARA funktionierte nicht wie erwartet und B.A.T.M.A.N. verwendet fehlerhafte GUI-CLI-Commands. Eine weitere Analyse der Ursachen für das fehlerhafte Verhalten von ARA und B.A.T.M.A.N. steht noch aus. Eine Portierung weiterer Daemons wie DSR oder AODV für die DES-SERT-App 2.0 scheint angebracht, um weitere Latenzmessungen durchführen zu können. Es ist darüber hinaus allgemein erstrebenswert, eine größere Menge an DES-Daemons für die DES-SERT-App 2.0 zur Verfügung zu stellen, insbesondere da ARA nicht erfolgreich für die DES-SERT-App 2.0 portiert werden konnte. Bei zukünftigen Portierungen der DES-Daemons für die DES-SERT-App 2.0 müssen gegebenenfalls noch die erforderlichen Icon- und GUI-Dateien (siehe Kapitel 3.5.1) erstellt werden.

Die DES-SERT-App 2.0 wurde für ARMv7 und für Android-Version 4.1+ optimiert. Es besteht die Möglichkeit, mit zukünftigen DES-SERT-App-Versionen ebenfalls ARMv8 und damit Geräte mit 64 Bit-Architektur zu unterstützen. Dazu müssten zusätzlich zu den bestehenden ARMv7-Bibliotheken und -Daemons ARMv8-Bibliotheken und -Daemons in die Applikation integriert und die Ordnerstruktur angepasst werden.

Das Einrichten des Ad-hoc-Modus unter CM ist trotz der GUI-Einbindung von CM kompliziert. Ein Ad-hoc-Netz aufzuspannen, ist mit dem Systemtool `iw` oder mit dem äl-

teren `iwconfig` praktischer. Eine mögliche Erweiterung in zukünftigen DES-SERT-App-Versionen ist eine direkte Einbindung der Ad-hoc-Modus-Einrichtung in die DES-SERT-Einstellungen. Da weder `iw` noch `iwconfig` standardmäßig in den busybox-Systemtools enthalten sind, müssten eigene ausführbare Dateien in die Applikation integriert werden. Darüber hinaus könnte in weiteren DES-SERT-App-Versionen ein Mechanismus implementiert werden, welcher dynamisch eine Tap-IP-Adresse vergibt. Auf jedem Gerät ist standardmäßig dieselbe Tap-IP-Adresse eingestellt. In einem zweiten Schritt müsste die IP-Adresse des Mesh-Interface gelöscht werden, sodass die Applikation direkt zu benutzen ist.

Die Evaluation könnte um weitere Metriken und Messungen ausgeweitet werden. Bisher konnte der Durchsatz noch nicht gemessen werden. Dazu müssten Alternativen zu `iperf` genutzt werden. Eine erweiterte Messung der Latenzzeiten mit mehr Hops ist ebenso angebracht. Für eine möglichst aussagekräftige Leistungsanalyse ist ein Testbed ohne WLAN-Interferenzen als Testumgebung wünschenswert (siehe auch Kapitel 5.3.6). Durch Benutzung von MANETs ist allerdings, auch mit einer niedrigen Sendeleistung (txpower) der Geräte, viel Platz für die Positionierung von Nöten. Ein Abschirmen der Geräte könnte die Messungen verfälschen.

Lediglich das Asus Nexus 7 (Grouper) wird für eine Benutzung des Ad-hoc-Modus unter CM 12.1 unterstützt (siehe Kapitel 5.1). Ob eine Unterstützung für neuere CM-Versionen beziehungsweise neuere Geräte geplant ist, ist nicht bekannt [45]. Ebenso ist nicht bekannt, ob CM wie bisher weiterentwickelt wird. Der Entwickler Cyanogen entlässt derzeit viele Mitarbeiter und schließt Büros. CM soll zwar vorerst nicht unmittelbar betroffen sein [64]; es ist jedoch abzuwarten, ob sich diese Einschätzung bewahrheitet. Pull-Requests, um den Ad-hoc-Modus in Stock-Android zu implementieren wurden abgelehnt [45] und die anfänglich in Stock-Android vorhandene Code-Basis für eine Implementierung (siehe Kapitel 3.1.1) des Ad-hoc-Modus komplett entfernt. Ob DES-SERT auf dem Asus Nexus 7 (Grouper) mit CM 13 noch einwandfrei funktioniert, ist unbekannt. Da CM 13 auf Testgerät S2 bereits keinen Ad-hoc-Modus mehr unterstützt und Aktualisierungen des Ad-hoc-Modus bereits vor einiger Zeit schon unregelmäßig erfolgt waren, müsste für neuere Geräte demnach eine manuelle Ad-hoc-Modus-Implementierung der CM-Custom-Roms vorgenommen werden. Es wäre auch denkbar, ein minimales (Linux-)Betriebssystem für Mobilgeräte zu entwickeln, welches den Ad-hoc-Modus unterstützt und für den Einsatz von DES-SERT spezialisiert ist.

Literaturverzeichnis

[1] J. Kuri. Großstörung bei der Telekom: "Schlecht programmierte Schadsoftware" verhinderte schlimmere Folgen. Webseite, 2016. https://www.heise.de/newsticker/meldung/Grossstoerung-bei-der-Telekom-Schlecht-programmierte-Schadsoftware-verhinderte-schlimmere-Folgen-3506909.html; abgerufen am 6. Dezember 2016.

[2] P. Gardner-Stephen. The Serval Project - Reflections Two Years In. Webseite, 2012. http://servalpaul.blogspot.de/2012/07/serval-project-reflections-two-years-in.html; abgerufen am 6. Dezember 2016.

[3] K. I. Lakhtaria. *Technological Advancements and Applications in Mobile Ad-Hoc Networks*. IGI Global, 1. edition, 2012. ISBN: 978-1-4666-0321-9.

[4] E. Glatz. Wireless Mesh Networks: Introduction. Webseite, 2006. http://www.csg.ethz.ch/education/lectures/ATCN/ws06_07/doc/WMN-BasicsWS0607-print.pdf; abgerufen am 22. November 2016.

[5] DES-Workgroup. About DES-SERT. Webseite. http://des-testbed.net/DES-SERT/about; abgerufen am 6. August 2016.

[6] developer.android.com. Dashboards. Webseite, 2016. https://developer.android.com/about/dashboards/index.html; abgerufen am 22. November 2016.

[7] D. B. Johnson und D. A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. Webseite, 1996. http://www.utdallas.edu/~jjue/cs6390/papers/DSR.pdf; abgerufen am 22. November 2016.

[8] Cisco Systems. Spanning Tree Protocol. Webseite, 2008. http://www.cisco.com/c/en/us/td/docs/routers/access/3200/software/wireless/SpanningTree.pdf; abgerufen am 22. November 2016.

[9] K. Plößl. *Mehrseitig sichere Ad-hoc-Vernetzung von Fahrzeugen*. Springer Gabler, 1. edition, 2009. ISBN: 978-3-8349-9993-1.

[10] Z. J. Haas und B. Liang. Hybrid Routing in Ad Hoc Networks with a Dynamic Virtual Backbone. Webseite, 2006. http://www.comm.toronto.edu/~liang/publications/ToWC06.pdf; abgerufen am 22. November 2016.

[11] A. Kuzmanovic. Hierarchical Routing. Webseite, 2007. http://cs.northwestern.edu/~akuzma/classes/CS340-w07/doc/class12.ppt; abgerufen am 22. November 2016.

[12] R. Pohlmann. Vergleich Distance-Vector und Link-State Protokolle. Webseite, 2010. http://www.ralf-pohlmann.de/dok/cisco-ccna/ccna-100414-177-distancevectorvslinkstate.pdf; abgerufen am 22. November 2016.

[13] Cisco. Cisco NX-OS/IOS EIGRP Comparison. Webseite. http://docwiki.cisco.com/wiki/Cisco_NX-OS/IOS_EIGRP_Comparison; abgerufen am 22. November 2016.

[14] Cisco Engineers. Configuration Notes for the Implementation of EIGRP over Frame Relay and Low Speed Links. Webseite, 2005. http://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/13672-12.html; abgerufen am 22. November 2016.

[15] Cisco Engineers. Enhanced Interior Gateway Routing Protocol. Webseite, 2016. http://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/16406-eigrp-toc.html; abgerufen am 22. November 2016.

[16] W. Heaton. Should you use distance vector or link state routing protocols? Webseite, 2000. http://www.techrepublic.com/article/should-you-use-distance-vector-or-link-state-routing-protocols/; abgerufen am 22. November 2016.

[17] R. Baumann, S. Heimlicher, M. Strasser und A. Weibel. A Survey on Routing Metrics. TIK Report 262, Computer Engineering and Networks Laboratory ETH-Zentrum, Switzerland, 2007. http://rainer.baumann.info/public/tik262.pdf; abgerufen am 22. November 2016.

[18] J. Maier. Instrumentelle Analytik. Webseite, 2012. http://dodo.fb06.fh-muenchen.de/maier/analytik/Blaetter/N022_Messunsicherheit_a_BAneu.pdf; abgerufen am 22. November 2016.

[19] J. Guerin, M. Portmann und A. A. Pirzada. Routing Metrics for Multi-Radio Wireless Mesh Networks. Webseite, 2007. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.329.5280&rep=rep1&type=pdf; abgerufen am 22. November 2016.

[20] JP. Vasseur, M. Kim, K. Pister, N. Dejean und D. Barthel. Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks. RFC 6551 (Proposed Standard), March 2012. http://www.ietf.org/rfc/rfc6551.txt; abgerufen am 22. November 2016.

[21] T. Clausen und P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003. http://www.ietf.org/rfc/rfc3626.txt; abgerufen am 22. November 2016.

[22] A. Tønnesen. Mobile Ad-Hoc Networks. Webseite. http://www.olsr.org/docs/wos3-olsr.pdf; abgerufen am 22. November 2016.

[23] E. Wagenrad. The OLSR.ORG story. Webseite. https://www.open-mesh.org/projects/open-mesh/wiki/The-olsr-story; abgerufen am 22. November 2016.

[24] F. Oehlmann. Simulation of the “Better Approach to Mobile Adhoc Networking” Protocol. Webseite, 2011. http://home.in.tum.de/~oehlmann/ba.pdf; abgerufen am 22. November 2016.

[25] A. Neumann, E. Wagenrad und M. Lindner. B.A.T.M.A.N. - Better Approach to Mobile Ad-Hoc Networking. Webseite, 2007. https://events.ccc.de/camp/2007/Fahrplan/events/2039.en.html; abgerufen am 22. November 2016.

[26] S. Wunderlich und M. Lindner. Wireless Kernel Tweaking. Webseite, 2007. https://events.ccc.de/congress/2007/Fahrplan/attachments/1013_wireless_hacking_talk.pdf; abgerufen am 22. November 2016.

[27] I. Kassabalidis, M.A. El-Sharkawi, R.J.Marks II, P. Arabshahi und A.A. Gray. Swarm Intelligence for Routing in Communication Networks. Webseite, 2001. http://staff.washington.edu/paymana/papers/globecom01_2.pdf; abgerufen am 22. November 2016.

[28] M. Güneş, U. Sorges und I. Bouazizi. ARA - The Ant-Colony Based Routing Algorithm for MANETs. Webseite, 2002. ISBN: 0-7695-1680-7, http://staff.washington.edu/paymana/papers/globecom01_2.pdf; abgerufen am 22. November 2016.

[29] P. Janacik, O. Kao und U. Rerrer. A routing approach using swarm-intelligence for resource sharing in wireless ad hoc networks. Webseite, 2004. http://www2.cs.uni-paderborn.de/cs/ag-kao/de/persons/rerrer/pdfpapers/2004_SympoTIC_StigmergyRessourceSharing.pdf; abgerufen am 22. November 2016.

[30] C. Perkins, E. Belding-Royer und S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003. http://www.ietf.org/rfc/rfc3561.txt; abgerufen am 22. November 2016.

[31] DES-Workgroup. Dynamic Source Routing (DSR). Webseite. http://www.des-testbed.net/content/dynamic-source-routing-dsr; abgerufen am 3. August 2016.

[32] D. Johnson, Y. Hu und D. Maltz. The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4. RFC 4728 (Experimental), February 2007. http://www.ietf.org/rfc/rfc4728.txt; abgerufen am 22. November 2016.

[33] B. Awerbuch und A. Mishra. Dynamic Source Routing (DSR) Protocol. Webseite, 2015. http://www.cs.jhu.edu/~cs647/dsr.pdf; abgerufen am 22. November 2016.

[34] P. Schmidt. Entwurf und Implementierung eines Layer-2.5-Frameworks für reaktives Routing. Webseite, 2009. https://www.net.t-labs.tu-berlin.de/papers/S-EIL2FRR-09.pdf; abgerufen am 22. November 2016.

[35] DES-Workgroup. Architecture and concepts of DES-SERT. Webseite. http://www.des-testbed.net/libdessert/doc/0.93/group__Architecture.html; abgerufen am 6. August 2016.

[36] P. Schmidt. DES-SERT - An Extensible Routing-Framework for Testbeds. Webseite, 2012. https://github.com/des-testbed/libdessert/blob/master/README.md; abgerufen am 22. November 2016.

[37] DES-Workgroup. Registering Interfaces. Webseite. http://www.des-testbed.net/libdessert/doc/libdessert-extra/0.1/group__interfaces.html; abgerufen am 6. August 2016.

[38] R. Stallman. Android and Users' Freedom. Webseite, 2011. https://www.gnu.org/philosophy/android-and-users-freedom.en.html; abgerufen am 22. November 2016.

[39] X. Ma. Android OS. Webseite, 2010. https://cseweb.ucsd.edu/classes/fa10/cse120/lectures/CSE120-lecture.pdf; abgerufen am 22. November 2016.

[40] J. J. Drake, Z. Lanier, C. Mulliner, P. O. Fora, S. A. Ridley und G. Wicherski. *Android Hacker's Handbook*. Wiley Publishing, 1st edition, 2014. ISBN: 111860864X, 9781118608647.

[41] R. Cohen und T. Wang. *Android Application Development for the Intel® Platform*. Springer, 2014. ISBN: 978-1-4842-0100-8, http://link.springer.com/content/pdf/10.1007%2F978-1-4842-0100-8.pdf; abgerufen am 22. November 2016.

[42] S. Lee und J. Wook Jeon. Evaluating Performance of Android Platform Using Native C for Embedded Systems. Webseite, 2010. http://anibal.gyte.edu.tr/hebe/AblDrive/69276048/w/Storage/104_2011_1_601_69276048/Downloads/m33.pdf; abgerufen am 22. November 2016.

[43] L. Batyuk, A.-D. Schmidt, H.-G. Schmidt, A. Camtepe und S. Albayrak. Developing and Benchmarking Native Linux Applications on Android. Webseite, 2009. https://pdfs.semanticscholar.org/2ca0/be1a7a500a0e762b2bf9fa0a2c6e084c71aa.pdf; abgerufen am 22. November 2016.

[44] L. Sicard, M. Markovics und G. Manthios. An Ad-hoc Network of Android Phones Using B.A.T.M.A.N. Webseite, 2010. http://docplayer.net/7526964-An-ad-hoc-network-of-android-phones-using-b-a-t-m-a-n.html; abgerufen am 22. November 2016.

[45] B. Randolf. Ad-Hoc (IBSS) mode support for Android 4.2.2 / 4.3 / 4.4 / 5.0. Webseite, 2015. http://www.thinktube.com/android-tech/46-android-wifi-ibss; abgerufen am 22. November 2016.

[46] J. P. Thompson. Is it possible to make the mesh work with a Samsung Galaxy SII Skyrocket SGH-I727? Webseite, 2014. https://groups.google.com/forum/#!topic/serval-project-developers/Ai_tJ-Nc14k; abgerufen am 22. November 2016.

[47] ARM. ARM at a Glance. Webseite, 2016. http://arm.com; abgerufen am 22. November 2016.

[48] G. C. Germoglio. The Intel x86 approach to the RISC model. Webseite, 2006. http://brahms.di.uminho.pt/discip/minf/ac0607/faq-03.pdf; abgerufen am 22. November 2016.

[49] D. Kanter. ARM Goes 64-bit. Webseite, 2012. http://www.realworldtech.com/arm64/2; abgerufen am 22. November 2016.

[50] source.android.com. Security-Enhanced Linux in Android. Webseite, 2016. https://source.android.com/security/selinux/index.html; abgerufen am 22. November 2016.

[51] S. Smalley und C. PeBenito. SELinux Object Classes and Permissions Reference. Webseite, 2013. http://selinuxproject.org/page/ObjectClassesPerms#security; abgerufen am 22. November 2016.

[52] A. Devkota. This Week In CyanogenMod - Special SELinux Edition. Webseite, 2013. http://www.cyanogenmod.org/blog/this-week-in-cm-july-19-13; abgerufen am 22. November 2016.

[53] tldp.org. Requesting Permissions at Run Time. Webseite, 2016. https://developer.android.com/training/permissions/requesting.html; abgerufen am 22. November 2016.

[54] H. Bojinov, D. Boneh, I. Malchev und R. Cannings. Address Space Randomization for Mobile Devices. In *Proceedings of the Fourth ACM Conference on Wireless Network Security*, WiSec '11, pages 127–138, New York, NY, USA, 2011. ACM. ISBN: 978-1-4503-0692-8, http://bojinov.org/professional/wisec2011-mobileaslr-paper.pdf; abgerufen am 22. November 2016.

[55] Inc. Free Software Foundation. 3.16 Options for Code Generation Conventions. Webseite, 2016. https://gcc.gnu.org/onlinedocs/gcc/Code-Gen-Options.html; abgerufen am 22. November 2016.

[56] ARM Infocenter. Migrating a software application from ARMv5 to ARMv7-A/R Application: ARMv5 and ARMv7 compared. Webseite, 2014. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0425/BABFJIIE.html; abgerufen am 22. November 2016.

[57] developer.android.com. Android 5.0 Behavior Changes. Webseite, 2016. https://developer.android.com/about/versions/android-5.0-changes.html; abgerufen am 22. November 2016.

[58] developer.android.com. Requesting Permissions at Run Time. Webseite, 2016. http://tldp.org/LDP/abs/html/exitcodes.html; abgerufen am 22. November 2016.

[59] Benutzer R. Amaral und N. Kitchen. Any benefit in using WEXITSTATUS macro in C over division by 256 on exit() status? Webseite, 2009. http://stackoverflow.com/questions/808541/any-benefit-in-using-wexitstatus-macro-in-c-over-division-by-256-on-exit-statu; abgerufen am 3. Dezember 2016.

[60] Freifunk.net. WLAN-Antennen. Webseite, 2016. https://wiki.freifunk.net/WLAN-Antennen; abgerufen am 22. November 2016.

[61] K. Pentikousis, R. Agüero Calvo, M. García-Arranz und S. Papavassiliou. *Mobile Networks and Management - Second International ICST Conference, MONAMI 2010, Santander, Spain, September 2010, Revised Selected Papers*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, 2011. ISBN: 0-7695-1680-7.

[62] L. Lüssing. batman-adv - Assessment & Lookout. Webseite, 2014. https://media.hamburg.freifunk.net/geekend/2014.ffnordcon/batman-adv-assessment%2Blookout-ffnordcon2014.pdf; abgerufen am 22. November 2016.

[63] R. Baradari. Android basierte Mobilgeräte als Plattform für mobile Mesh-Router, 2010.

[64] A. Spier. Android-Alternative: Cyanogen vor dem Aus, weitere Entlassungen und Standortschließung. Webseite, 2016. https://www.heise.de/newsticker/meldung/Android-Alternative-Cyanogen-vor-dem-Aus-weitere-Entlassungen-und-Standortschliessung-3507117.html; abgerufen am 1. Dezember 2016.

Anhang

A.1 Tabellarischer Vergleich der DES-Daemons

Daemon	Wegwahltechnik	Verfahren	Klasse	Metrik
OLSR	Hop-by-Hop	proaktiv	Link-State	PLR, ETX, ETX-ADD, Hop-Count
B.A.T.M.A.N.	Hop-by-Hop	teils proaktiv	teils Distanzvektor	TQ
ARA	Hop-by-Hop	reaktiv	teils Distanzvektor	Ant-Metrik
AODV	Hop-by-Hop	reaktiv	Distanzvektor	Hop-Count
DSR	Source-Routing (adaptiv), Hop-by-Hop	reaktiv	DSR-Cache	Hop-Count, verschiedene ETX-Derivate, MDSR, SMR, BackupPath 1 / 2

Tabelle A.1: Tabellarischer Vergleich der DES-Daemons

A.2 Testgeräte und Betriebssysteme

Testgerät	Anzahl	API	OS	Android-Version	SELinux
HTC Desire HD	2	22	CyanogenMod 12.1	5.1.1	moderat
Asus Nexus 7 (Grouper)	4	22	CyanogenMod 12.1	5.1.1	strikt
Samsung Galaxy S II GT-I9100	1	22	CyanogenMod 12.1	5.1.1	strikt
Samsung Galaxy S II GT-I9100	1	23	CyanogenMod 13.0	6.0.1	moderat
Motorola Moto E2 (XT1524)	1	22	CyanogenMod 12.1	5.1.1	strikt
Doogee X5	2	22	Android	5.1	moderat
Ulefone U007	1	23	Android	6.0	moderat
HTC Desire Eye	1	23	Android	6.0	moderat

Tabelle A.2: Benutzte Testgeräte und Android-Versionen

Testgerät	Root	DES-SERT-Installation	Daemon-Funktionalität	MANET-Fähigkeit
HTC Desire HD	ja	✓	✓	teils, nicht unterstützt
Asus Nexus 7 (Grouper)	ja	✓	✓	ja, unterstützt
Samsung Galaxy S II GT-I9100	ja	✓	✓	CM 12.1: teils, nicht unterstützt, CM 13.0: nein
Motorola Moto E2 (XT1524)	ja	✓	✓	teils, nicht unterstützt
Doogee X5	ja	✓	✓	nein, kein CM
Ulefone U007	ja	✓	✓	nein, kein CM
HTC Desire Eye	nein	✓	✗	nein, kein CM

Tabelle A.3: Benutzte Testgeräte und Funktionalität

A.3 Entstandene Gerätefehler innerhalb dieser Arbeit

Testgerät	Fehler	Ursache
DHD.1	Root-Befehle: Segmentation Fault	Mehrfaches Remounting von /system/x-bin
DHD.2	WLAN-Adapter-Ausfälle im Ad-hoc-Modus	Nicht unterstütztes Gerät, Alter des Geräts, Software-Fehler
N7.3	Softbrick (repariert)	Fehler bei Installation von CyanogenMod
S2	Aufspannen von Ad-hoc-Netz plötzlich nicht mehr möglich	Nicht unterstütztes Gerät, Alter des Geräts
Moto-E	Ad-hoc-Netz-Zelle wird plötzlich nicht mehr gesetzt	Nicht unterstütztes Gerät, manuelle Konfiguration
X5.2	Hardbrick	Fehler bei Installation von CyanogenMod

Tabelle A.4: Innerhalb des Aktualisierungsvorgangs und der Evaluation sind vereinzelt Gerätefehler entstanden. Es werden die in Kapitel 5.1 erstellten Aliase für eine Unterscheidung mehrfach vorhandener Geräte verwendet.

A.4 Öffentlicher Quellcode

Das GitHub-Repository `https://github.com/Dekue/` beinhaltet sämtlichen verfassten Quellcode der Applikation, der Bibliotheken und der Daemons. Im Repository der DES-SERT-Applikation wurde der Branch *devb* erstellt. Beim Import des *Eclipse*-Projekts in die jetzige Standard-IDE für Android-Projekte *Android-Studio* wurden einige Dateien, zum Beispiel *.txt-Dateien, nicht automatisch mitimportiert. Sie sind zwar für die Funktionalität der Applikation unerheblich, tragen jedoch zur besseren Verständlichkeit des Projekts bei. Bei einem Pull-Request werden sie vorher wieder zum Repository hinzugefügt.

Hiermit versichere ich, dass die vorliegende Arbeit mit dem Titel *Migration des Routing-Frameworks DES-SERT auf Android und Evaluation unter Erzeugung mobiler Ad-hoc-Netze* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommenen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Münster, 9. Dezember 2016

(Denis Klaus Küchemann)

Hier die Hülle

mit der CD/DVD einkleben

Diese CD enthält:

- eine *pdf*-Version der vorliegenden Bachelorarbeit
- die LaTeX- und Grafik-Quelldateien der vorliegenden Bachelorarbeit samt aller verwendeten Skripte
- die Quelldateien der im Rahmen der Bachelorarbeit erstellten Software
- die Websites der verwendeten Internetquellen