Forschungsprojekt II

Datenübertragung in nicht-zusammenhängenden Netzwerken
Anwendungen und Neighbor Discovery

Eingereicht von
Mathias Lenz, 521129

1. April 2012

HTW Berlin
Fachbereich 4
Angewandte Informatik
Betreuer: Prof. Dr. Jürgen Sieck

Inhaltsverzeichnis

1 Einleitung

1.1 Szenario

Der Hintergrund dieser Forschungsarbeit ist die Entwicklung eines Systems das Sensordaten, beispielsweise Luftfeuchtigkeit und Temperatur, in einem urbanen Umfeld aufzeichnet. Diese Sensordaten sollen opportunistisch eingesammelt werden, indem Personen mit einem Mobilgerät an den Messstationen vorbeigehen. Die Daten sollen mit möglichst wenig Interaktion vom Benutzer, idealerweise direkt im Vorbeilaufen, heruntergeladen und gespeichert werden. Anschließend sollen diese Daten an einen verarbeitenden Zielknoten, also beispielsweise einen Server weitergereicht werden. Dieses Weiterreichen sollte idealerweise auch zwischen verschiedenen Mobilgeräten möglich sein, sodass ein Nutzer der selbst keinen direkten Kontakt zum Zielknoten hat die Daten ebenso interaktionsfrei wie beim einsammeln weiterreichen kann.

1.2 Zielstellung

Das vorliegende Szenario bietet Berührungspunkte zu einer Vielzahl aktueller Forschungsthemen, insbesondere aus dem Bereich von Ad-hoc- und Sensornetzwerken. In den letzten Jahren entwickelte sich der Begriff der Delay Tolerant Networks, die einerseits mit Ad-hoc-Netzwerke zu tun haben, ihre Grundlage aber primär in der Forschung zu Interplanetaren Netzen finden. In dieser Forschungsarbeit soll speziell auf Technologien, Verfahren und Projekte im Bereich der Delay Tolerant Networks eingegangen werden und eine Auswahl an Möglichkeiten getroffen werden, diese für das vorliegende Szenario anzuwenden.

Ein besonderes Augenmerk wird hierbei auf das Thema *Discovery* gelegt, das die Suche nach Netzwerkknoten beschreibt. Ausserdem werden eine Reihe von Projekten beschrieben, die entweder direkt als Delay Tolerant Network gelten können, beim Aufbau solcher Netzwerke behilflich sind oder zur Evaluation, Simulation oder Emulation solcher Netzwerke dienen.

2 Delay Tolerant Networks

Das vorliegende Szenario ist ein Beispiel für eine mögliche Anwendung von Delay Tolerant Networks[1]. Hierbei handelt es sich um Netzwerke, die gegen Verbindungsabbrüche, hohe Latenzen oder das zeitweise Fehlen durchgängiger Routen zum Ziel resistent bzw. tolerant sein sollen. Derartige Störungen können beispielsweise durch Mobilität der Netzwerkteilnehmer oder Ausfälle von Hardware auftreten. Das Grundprinzip dieser Netzwerke ist „Store and Forward“, also das zwischenspeichern von Datensätzen zur späteren Weiterleitung[FSB+07]. In diesem Kapitel soll ein Überblick über DTNs und Forschungsthemen in diesem Bereich gegeben werden.

2.1 Überblick

2.1.1 Routingverfahren

Routingverfahren in DTNs unterscheiden sich in vielen Aspekten. Alle Routingverfahren zeigen dabei bestimmte Eigenschaften, die wichtigsten dabei sind die Latenz, also die Zeit die ein Paket durchschnittlich zum Ziel braucht und die Übertragungswahrscheinlichkeit. Letzteres ist besonders wichtig, da in einem DTN zumeist nicht garantiert werden kann, dass Datenpakete ihr Ziel tatsächlich erreichen.

Die meisten Protokolle unterscheiden sich in der Art und der Menge der Informationen, die sie zum routen der Datenpakete verwenden. Neben statischen Routen können so beispielsweise folgende Verfahren verwendet werden:

Epidemic Routing ist eines der einfachsten Verfahren, in dem prinzipiell bei jedem erfolgreichen Kontakt alle Datenpakete versendet werden. Dieses Verfahren benötigt somit keinerlei zusätzliche Informationen über das Netzwerk und hat potenziell eine minimale Latenz und eine maximale Übertragungswahrscheinlichkeit. Der größte Nachteil ist die hohe Belastung des Netzwerks und der einzelnen Knoten, zum zwischenspeichern der Daten.[VB00]

Spray Routing ist ein Verfahren, das mehrere Varianten besitzt. Diese haben alle die Eigenschaft, dass sie Datenpakete zunächst in der *Spray*-Phase an eine bestimmte, festgelegte Anzahl an weiteren Knoten versenden. Anschließend werden die Datenpakete von den Netzwerkknoten gehalten, bis sie das Ziel erreichen („Spray and Wait“) oder nach bestimmten Regeln an Knoten weitergeleitet, die eine höhere Chance haben das Ziel zu erreichen („Spray and Focus“). Zu jedem Zeitpunkt existiert somit nur eine feste Anzahl an Kopien eines Datenpakets im Netzwerk.[SPR08]

[1] DTN; auch „Disruption Tolerant Networks“

Social Routing bezeichnet eine ganze Reihe an Protokollen, die Informationen über Sozialität ausnutzen. Hierzu gehören insbesondere Informationen zu bisherigen Kontakten. Manche Protokolle nutzen auch Informationen zu Peer-Groups. Sie leiten beispielsweise an Netzwerkteilnehmer weiter, die mehr Kontakt zu anderen Gruppen oder besonders viele Kontakte innerhalb der eigenen Gruppe haben.[HCY10]

Es existiert eine Vielzahl weiterer Protokolle, die jedoch in dieser Forschungsarbeit nicht primär behandelt werden.

2.1.2 Discovery

Das Erkennen anderer Netzwerkknoten ist der erste Schritt in der Initialisierung eines opportunistischen Netzwerks. Im Idealfall sucht ein im Netzwerk teilnehmendes Gerät ständig nach anderen Knoten. Eine durchgehend laufende Suche nach Nachbarn birgt jedoch zwei Probleme:

- Die Nutzung der Schnittstelle benötigt Leistung, die insbesondere in mobilen Umgebungen nur sehr begrenzt vorhanden ist. Man sollte somit versuchen, die tatsächliche Zeit in der die Suche durchgeführt wird verringern, damit das Mobilgerät eine längere Laufzeit hat

- Unter Umständen kann eine Schnittstelle nicht oder nur sehr ineffizient gleichzeitig Datenübertragung und eine Suche nach anderen Geräten durchführen. In manchen Fällen können auch zwei gleichzeitig suchende Geräte sich nicht gegenseitig sehen.

Um Energie zu sparen kann ein Gerät in einen Schlafmodus gehen, in dem es weder auf Anfragen von aussen hört, noch eigene Daten aussendet. Diesen Modus muss das Gerät im Allgemeinen mit dem Anfragen von anderen Geräten und dem Empfangen von Anfragen abstimmen.

Um also die genannten Probleme zu umgehen gibt es eine Reihe an Techniken. Eine der grundlegendsten Möglichkeiten ist, die Zeit in der ein Gerät in einem bestimmten Zustand (Senden oder Empfangen) ist zufällig zu machen. Ist dies nicht der Fall, und die Zeiten sind deterministisch, so kann es passieren das sich Geräte zueinander „synchronisieren“ indem sie zu jeder Zeit im gleichen Zustand sind. Dies führt dazu das sie niemals in der Lage sind sich gegenseitig zu finden[SBT00]. Dieses Problem tritt nur in *symmetrischer* Discovery auf, in der zwei Geräte die gleiche Rolle einnehmen. In *asymmetrischer* Discovery ist ein Gerät für die Anfrage nach Nachbarn zuständig, während das andere Gerät nur auf diese Anfragen hört. Asymmetrische Discovery ist entsprechend für ein Ad-hoc-Netzwerk nicht gut geeignet, da alle Netzwerktteilnehmer gleichgestellt sind[RK03].

2.2 Relevante und verwandte Projekte

Es existieren eine Reihe an (Forschungs-)Projekten die Delay-Tolerant-Network-Techniken nutzen. Einige von diesen sollen an dieser Stelle kurz vorgestellt werden. Die vorgestellten

Projekte sind bereits zumeist für mobile Anwendungen entwickelt und zeigen daher nur einen Ausschnitt der Möglichkeiten, die DTN-Verfahren bieten. Nichtsdestotrotz zeigt auch diese Auswahl bereits eine ganze Reihe unterschiedlicher Techniken und Möglichkeiten.

2.2.1 Haggle

Haggle[2] ist ein Projekt das es seinen Nutzern ermöglichen soll verschiedene, besonders aber multimediale Inhalte zu verteilen. Haggle operiert dabei nach dem Publish/Subscribe-Modell, indem Nutzer Interesse an bestimmten Inhalten über Schlüsselwörter anmelden. Die Inhalte werden opportunistisch verbreitet, sobald Nutzer in Reichweite zueinander kommen. Haggle unterstützt sowohl Bluetooth- als auch WiFi-Kommunikation. Es existieren Implementierungen für Windows, Linux, Mac OS X und für die Mobilplattformen Windows Mobile und Android.[NGR09]

Haggle wird seit 2006 an der Universität Uppsala entwickelt. Mit Hilfe von Haggle wurden beispielsweise Anwendungen zur Verteilung von Fotos, aber auch zum Versand von E-Mails über das opportunistische Netzwerk entwickelt. Haggle hat die besondere Eigenschaft, dass die „Interessen"-Metadaten im Netzwerk genauso behandelt werden wie andere zu verteilende Daten. Hierdurch ist es möglich, die Interessen von Netzwerkknoten zu erfahren, mit denen man bisher selbst noch keinen direkten Kontakt hatte. Alle Daten, also die Nutzdaten und Knotendaten, werden in einem Relationsgraphen gehalten. Über diesen Graphen können mit Hilfe der bekannten Metadaten Datenobjekte, und damit auch Netzwerkknoten, miteinander verglichen und in Rangfolge gebracht werden. Die direkte Verteilung an einen interessierten Knoten wird als *interest forwarding* bezeichnet. Verteilt ein Knoten dagegen an weitere Knoten, die für sich kein Interesse an den Daten haben, wird dies als *delegate forwarding* bezeichnet. Beim delegieren wird dabei eine Metrik genutzt, die bestimmt ob das Verteilen an diesen Knoten die Wahrscheinlichkeit erhöht, dass die Daten an einen interessierten Knoten gelangen.

Haggle nutzt keine klassischen Adressierungs- und Namensschemata. Die von einem Knoten definierten Interessen bestimmen, welche Daten er erhält und seine Position im Relationsgraphen ermöglicht eine Lokalisierung. Diese beiden Mechanismen können als Ersatz für Namen und Adressen verwendet werden.

2.2.2 Bytewalla und DTN2

Das Bytewalla-Projekt wurde 2009 an der KTH Stockholm als Studienprojekt begonnen[3]. Ziel war es, eine DTN-Infrastruktur aufzubauen die es ländlichen Gegenden in Afrika ermöglicht mittels Androidbasierten Mobiltelefonen zu kommunizieren. Konkret sollte es möglich sein, dass Dorfbewohner bestimmte Daten mit sich tragen und zu einem späteren Zeitpunkt in einer Stadt abliefern. Dieses Ziel spiegelt die Idee von *Data Mules* wieder. Data Mules sind Netzwerkknoten, die sich zumeist in regelmäßigen Abständen zwischen

[2] http://haggleproject.org/
[3] http://www.tslab.ssvl.kth.se/csd/projects/092106/

Teilen des Netzwerks bewegen und somit eine Übertragung zwischen diesen ansonsten nicht verbundenen Punkten ermöglichen.

Die Implementierung basiert stark auf der Bundle-Referenzimplementierung *DTN2* der DTNRG[4]. Konkret wurde versucht, den in C++ geschrieben Code in Java umzuschreiben und damit auf die Android-Plattform zu portieren. Dies hat zur Folge, dass sich auch die Gesamtarchitektur von DTN2 in Bytewalla wiederspiegelt. Das Projekt wurde in weiteren Studienarbeiten weitergeführt, sodass es inzwischen 5 Versionen von Bytewalla gibt.

Bytewalla I ist die zuvor genannte erste Implementierung. Sie portierte die grundlegende Funktionalität des Bundle-Protokolls. Als Proof-of-Concept für die Implementierung wurde zusätzlich eine E-Mail-Integration programmiert die es ermöglichte, E-Mails in Bundle zu konvertieren und im Netzwerk zu verteilen.

Bytewalla II ist im Zuge einer Abschlussarbeit entstanden und hatte das Ziel, Sicherheitsfunktionalitäten in Bytewalla zu integrieren. Die Arbeit bezog sich primär auf Sicherheits- und Datenschutzaspekte und implementierte das Bundle Security Protocol[FWSL11, Dom10].

Bytewalla III war ein weiteres Gruppenprojekt[5] das verschiedene neue Funktionalitäten in Bytewalla integriert. Hierzu gehören die Implementierung des PRoPHET-Routingprotokolls[LDGD11], IP-basierte Neighbor Discovery[BE10] und eine Reihe an Werkzeugen.

Bytewalla IV beschäftigte sich primär mit der Verbesserung des Routings und baute auf der PRoPHET-Implementierung von Bytewalla III auf, indem es einen Warteschlangenmechanismus für diese implementierte. Zusätzlich wurde eine SMTP-Anwendung umgesetzt.[Hog11]

Bytewalla V ist die aktuellste Iteration von Bytewalla. In diesem Projekt wurden vor allem Werkzeuge entwickelt die um die eigentliche Android-Anwendung herum genutzt werden sollten. Hierzu gehören einige Anwendungen, die eine Integration von DTNs und WSNs[6] unterstützen sollen. Ausserdem war geplant, eine Integration von Bytewalla II und IV durchzuführen, da die Sicherheitsfunktionen nicht mit der Nutzung des PRoPHET-Routings kompatibel waren. Diese Integration wurde jedoch nicht fertiggestellt. Im Endeffekt ist durch dieses Projekt keine Änderung an der Bytewalla-Anwendung durchgeführt wurden.

Bytewalla ist eine der existierenden Implementierungen des Bundle-Protokolls in der Java-Programmiersprache. Tatsächlich gibt es eine Reihe davon, die jedoch seit Jahren nicht mehr gepflegt werden. Bytewalla wurde allerdings auch von der Android-spezifischen Funktionalität getrennt und für die reguläre Java VM portiert[7].

[4] Delay Tolerant Network Research Group; http://www.dtnrg.org/wiki
[5] http://www.tslab.ssvl.kth.se/csd/projects/1031352/
[6] Wireless Sensor Networks
[7] https://wiki.umiacs.umd.edu/VirtualMeshTest/index.php/JavaDTN

Es existieren viele weitere Implementierungen des Bundle-Protokolls. Die meisten sind für besondere Umgebungen, beispielsweise Embedded Systems konzipiert. Ein Beispiel hierfür ist Postellation[8], das besonders leichtgewichtig und einfach einzurichten sein soll. Eine weitere, verbreitete Implementierung ist IBR-DTN[9] welches ebenfalls für Embedded Systems entwickelt wurde. Ein in Ruby entwickelter Bundle-Agent namens RDTN[10] ist ebenfalls erhältlich.

2.2.3 Store and Forward Transport

Das *Store and Forward Transport*[11]-Protokoll ist eine Alternative Möglichkeit zur Datenübertragung für Netzwerke, in denen oft Routenunterbrechungen durch Mobilität der Teilnehmer auftreten[HBMP06]. Es ist als Erweiterung klassischer Ad-hoc-Routingprotokolle (wie AODV[12] und DSR[13]) entwickelt und definiert zwei zusätzliche Schichten zwischen der Anwendungs- und der Netzwerkschicht:

Hop-by-Hop Layer für die Übertragung zwischen einzelnen Knoten. Diese Schicht ist entsprechend auf jedem Knoten des Netzwerks aktiv. Sie sorgt für eine fragmentierte Übertragung von einigen wenigen IP-Paketen. Diese Schicht sorgt für Datenfluss- und Auslastungskontrolle zwischen den Hops des Netzwerks.

End-to-End Layer für die Übertragung von Start- zu Zielknoten. Diese Schicht operiert entsprechend nur auf diesen beiden Knoten und sorgt für globale Auslastungskontrolle und für zuverlässige Übertragung. In diesem Sinne stellt es, wie TCP, einen zuverlässigen Datenstrom zur Anwendungsschicht bereit.

Flusskontrolle wird in SaFT durch *ACKs*[14] auf dem Hop-by-Hop-Layer durchgeführt, diese ist also lokal und jeweils zwischen zwei Geräten auf der Route zum Ziel. Im Falle einer Routenunterbrechung würden klassische Ad-hoc-Protokolle die gesamte Route invalidieren, da keine Ende-zu-Ende-Verbindung über diese Route durchgeführt werden kann[MM04]. Da SaFT auch Hop-by-Hop-Verbindungen nutzt ist dieses Verhalten kontraproduktiv und wird durch Caching von Routen verhindert, sodass für eine bestimmte Zeit die Weiterleitung auf einer Route weiterhin möglich ist, auch wenn ein späterer Hop nich mehr verfügbar ist.

2.2.4 Weitere Projekte

Viele weitere Projekte existieren im DTN-Forschungsbereich. Manche dieser Projekte werden für sehr spezifische Szenarien genutzt, andere ähneln den beiden zuvor genannten in vielen Punkten. Die meisten der im folgenden genannten Projekte nutzt eigens entwickelte Software. Grund dafür ist oft, dass viele der Projekte bereits vor der

[8] http://postellation.viagenie.ca/
[9] http://www.ibr.cs.tu-bs.de/projects/ibr-dtn/
[10] https://github.com/jgre/rdtn
[11] SaFT
[12] Ad-hoc On-demand Distance Vector
[13] Dynamic Source Routing
[14] Acknowledgements, dt.: *Empfangsbestätigung*

tatsächlichen Standardisierung des Bundle-Protokolls und der Entwicklung der DTN2-Referenzimplementierung gestartet sind.

EMMA (Environmental Monitoring in Metropolitan Areas) [15] ist ein Projekt der Technischen Universität Braunschweig. Ziel ist es, Schadstoffe in der Luft zu überwachen. Diese Überwachung soll mit Hilfe von Fahrzeugen des Öffentlichen Personennahverkehrs durchgeführt werden.

DieselNET ist Teil des DOME[16]-Testbeds der University of Massachusetts. Das Projekt beinhaltet 35 Busse des ÖPNV, die mit eigens entwickelten „Diesel Bricks“ ausgerüstet sind. Diese besitzen ein WiFi-Modul das zusätzlich als Access Point für Reisende dient. Ausserdem werden über dieses Modul weitere Access Points aus der Nähe (bspw. anderer Busse) gescannt. Über eine weitere 900MHz-Funkschnittstelle höherer Reichweite werden aufgezeichnete Daten zur Auswertung versendet.

Serval Mesh hat das Ziel, eine Infrastrukturlose Kommunikation zwischen mobilen Geräten zu ermöglichen[17]. Hierzu wurde eine auf Android-Smartphones lauffähige Software entwickelt, mit der kabellose Mesh-Netzwerke auf WiFi-Basis erstellt werden können. Aktuell arbeitet das Projekt an Anwendungen, die diese Netzwerke ausnutzen, darunter MeshMS, eine SMS-Anwendung und Anwendungen zum Austausch allgemeiner Daten und Dateien über das Netzwerk.

Crowd ist ein Projekt der Université Pierre et Marie Curie das ähnlich wie Haggle auf dem Publish/Subscribe-Prinzip basiert[18]. Ziel ist es, ein Web 2.0 basiertes Onlineportal zu entwickeln, dass es ermöglicht benutzergenerierte Multimediadaten opportunistisch zu verteilen.

PodNet verfolgt ebenfalls das Ziel, benutzergenerierte Inhalte mobil und opportunistisch zu verteilen[19]. Der grundlegende Zweck ist dabei, Podcasts zu verteilen. Es handelt sich um ein Projekt das in Zusammenarbeit der ETH Zürich und der KTH Stockholm durchgeführt wird.

HTTP-DTN ist ein weiterer Versuch, ein grundlegendes und allgemein verwendbares DTN-Protokoll zu entwickeln. Für das vorgeschlagene Protokoll existiert ein RFC-Draft[WH11]. Dieser schlägt vor, HTTP zu erweitern um nicht zusammenhängende Netze zu unterstützen. Hierzu werden zusätzliche HTTP-Header vorgeschlagen die beispielsweise Routinginformationen tragen. Datenübertragungen geschehen über dedizierte Knoten-zu-Knoten-Verbindungen und die Zusatzinformationen werden verwendet, um opportunistisch weitere solcher Verbindungen aufzubauen, bis das Paket am Ziel ankommt.

[15] http://www.ibr.cs.tu-bs.de/projects/emma/
[16] http://prisms.cs.umass.edu/dome/umassdieselnet
[17] http://www.servalproject.org/
[18] http://anr-crowd.lip6.fr/index.php/Main/HomePage
[19] http://podnet.ee.ethz.ch/

2.3 Evaluation von Protokollen und Verfahren

Die Evaluation verschiedener für DTN entwickelter oder relevanter Verfahren kann auf tatsächlichen Geräten, bspw. in Testbeds wie dem DOME (2.2.4 auf der vorherigen Seite) durchgeführt werden. Der Aufbau eines solchen Testbeds ist jedoch aufwändig und teuer, weswegen in vielen Fällen auf Simulatoren zurückgegriffen werden muss. Zahlreiche Programme wurden speziell für die Simulation von DTNs entwickelt, die Nutzung von allgemeinen Netzwerksimulatoren ist jedoch genauso möglich. In diesem Abschnitt sollen einige Möglichkeiten zur Evaluation, insbesondere Simulatoren, vorgestellt werden.

2.3.1 The ONE Simulator

Die Entwicklung des ONE[20]-Simulators begründet sich auf der Tatsache, dass Routingperformanz in DTNs stark von der Mobilität der Netzwerkknoten und deren Eigenschaften abhängt. Es bietet daher die Möglichkeit, Routingsimulationen mit Mobilitätsmodellen oder Mobilitätsaufzeichnungen durchzuführen. Zusätzlich bietet es ein Rahmenwerk zur Implementierung von Protokollen in der Simulation[KOK09]. Zur Auswertung bietet ONE neben der Aufzeichnung der Ergebnisse in Dateien auch eine grafische Benutzeroberfläche. Zusätzlich gibt es die Möglichkeit, den Simulator an einen DTN2-Daemon anzubinden. Hierzu kann die Ausgabe des Simulators als Eingabe für DTN2 genutzt werden. Es ist somit möglich, Verbindungsaufbau und -abbruch zu simulieren, was beispielsweise in Testbeds Verwendung finden kann. Die zweite Möglichkeit der Anbindung ist der in ONE implementierte *External Convergence Layers* in DTN2 um eine Netzwerkemulation durchzuführen. Durch die Emulation können beliebig viele DTN2-Daemons an den Simulator angebunden werden, die jeweils einen Knoten in der Simulation darstellen. Simuliert wird hier also vor allem das Szenario, inklusive Bewegungsmuster der Geräte.

2.3.2 Weitere Simulatoren

Es existieren noch weitere Simulatoren, wobei insbesondere für größere Testbeds auch reguläre Netzwerksimulatoren verwendet werden.

ns-2 ist ein allgemeiner Netzwerksimulator. Er ist in der Lage eine Vielzahl verschiedener verkabelter und kabelloser Netzwerke zu simulieren. Insbesondere zur Anfangszeit der Forschung an DTN-Protokollen wurde ns-2 als Simulationsumgebung genutzt, da bis dahin noch keine dedizierten DTN-Simulatoren verfügbar waren.

DTNSim2 ist ein für DTNs entwickelter, inzwischen jedoch nicht mehr gepflegter Simulator. Er wurde als Verbesserung des von Sushant Jain für das *Routing in Delay Tolerant Network[JFP04]* Paper entwickelte DTNSim geschrieben. DTNSim2 ist in Java geschrieben.

[20] http://www.netlab.tkk.fi/tutkimus/dtn/theone/

3 Auswahl der Technologien und Verfahren

3.1 DTN-Software

Für die Umsetzung einer Anwendung im vorgeschlagenen Szenario kommen verschiedene Vorgehen in Frage. Einige der in Abschnitt 2.2 auf Seite 5 vorgestellten Projekte könnten als Basis für die Entwicklung der Anwendung genutzt werden. Zusätzlich gibt es die Möglichkeit, eine vollständig eigene Implementierung durchzuführen.

Als Plattform für die Umsetzung des Systems wurde Android[1] gewählt. Android ist ein Betriebssystem für Smartphones das von Google entwickelt wird und auf einer Vielzahl von Mobilgeräten lauffähig ist.

Um die Entwicklung zu vereinfachen soll auf ein bestehendes Projekt zurückgegriffen werden. Aus diesem Grund schränken sich die Auswahlmöglichkeiten in gewissen Maße ein. Die Entwicklung von Android-Anwendungen geschieht in der Java-Programmiersprache, es gibt jedoch auch Möglichkeiten native Anwendungen mit Hilfe des Android NDK[2] zu entwickeln.

Die zwei als Basis relevantesten Projekte sind Bytewalla (siehe Abschnitt 2.2.2 auf Seite 6) und Haggle (siehe Abschnitt 2.2.1 auf Seite 6). Bytewalla ist eine rein in Java entwickelte Android-Anwendung, während Haggle das NDK nutzt und nur eine Java-Schnittstelle zur Verwendung des Service bietet. Haggle ist ein älteres Projekt und ist auf mehreren Plattformen lauffähig, während Bytewalla in der Lage ist mit vielen anderen Bundle-Protokoll-Implementierungen zu kommunizieren.

Die Wahl fiel letztendlich auf Bytewalla, da es sich als native Android-Anwendung potenziell besser in dessen Ökosystem einpasst. Ausserdem implementiert es mit dem Bundle-Protokoll einen bereits standardisierten RFC und ist damit allgemeiner verwendbar.

3.2 Discovery

Für den tatsächlichen Datenaustausch der Mobilgeräte untereinander und zu stationären Netzwerkknoten kommen zwei Technologien in Frage: WiFi und Bluetooth. Beide Technologien sind in nahezu allen Android-Smartphones vorhanden und ermöglichen eine relativ schnelle Datenübertragung über kurze bis mittlere Reichweiten. An dieser Stelle sollen ein paar wichtige Eigenschaften von WiFi und Bluetooth im Zusammenhang mit

[1] http://www.android.com/
[2] http://developer.android.com/sdk/ndk/index.html

der Discovery erläutert werden. Anschließend wird konkreter auf die Optimierung von Bluetooth-Discovery eingegangen.

3.2.1 WiFi

Das vorgestellte Szenario (siehe1.1) würde es ermöglichen, die Datenübertragung zwischen einem der Sensorknoten und den Mobilgeräten per WiFi durchzuführen. Hierzu könnte der Sensorknoten als Access Point (AP) konfiguriert werden, zu dem sich das Mobilgerät verbindet sobald es in Reichweite gelangt. Die Art der Discovery kann in diesem Fall sogar asynchron stattfinden, da das Mobilgerät nur den Sensor suchen muss, nicht aber umgekehrt. Das Mobilgerät muss entsprechend das WiFi aktiv halten und regelmäßig aktiv nach einem Sensorknoten suchen. Äquivalent könnte eine Verbindung des Mobilgeräts zu einem Server, der die Sensordaten letztendlich aufnehmen und verarbeiten soll genauso durchgeführt werden.

Für die Kommunikation zwischen zwei Mobilgeräten kann dieses Verfahren so nicht ohne weiteres eingesetzt werden. Es ist zwar möglich, ein Android-Mobilgerät als kabellosen Access Point einzurichten, dies bringt jedoch folgende Nachteile mit sich:

- Es ist unter Umständen nur auf Geräten möglich, auf denen vollständig priviligierter („root“) Zugriff vorhanden ist. Viele neuere Android-Versionen bringen diese Funktion jedoch auch standardmäßig mit.
- Zumeist sind die Geräte nicht in der Lage, gleichzeitig als AP zu fungieren und sich selbst mit anderen Access Points zu verbinden. Dies war beispielsweise auf denen im Forschungsprojekt verwendeten Geräten, dem Google Nexus S und dem HTC Desire, nicht möglich.
- Die Bereitstellung eines Access Points bringt einen stark erhöhten Energieverbrauch mit sich.

WiFi kennt einen Ad-hoc-Modus, der es möglich macht ohne die Bereitstellung eines Access Points mit anderen Geräten zu kommunizieren. Dieser Kommunikationsmodus ist jedoch nur auf wenigen Geräten möglich, die ebenfalls priviligierten Zugriff erlauben müssen. Seit Version 4.0 wurde jedoch die WiFi-Direct-Spezifikation in Android integriert, welche eben solche Ad-hoc-Verbindungen ermöglicht. Bisher ist nur das Google Galaxy Nexus[3] in der Lage, diese Funktion zu nutzen.

Der Scan nach nach WiFi-Direct-Geräten und der nach WiFi-Access Points ist unabhängig voneinander und müsste bei Nutzung beider Technologien auch unabhängig voneinander durchgeführt werden.

3.2.2 Bluetooth

Bluetooth hat seinen Hauptnutzen als Peer-to-Peer-Kommunikationstechnologie. Es kann im vorliegenden Szenario besonders zur Kommunikation zwischen Mobilgeräten verwendet werden. Werden die Sensorknoten ebenfalls mit Bluetooth ausgestattet, so würde die

[3] http://www.google.de/nexus/

Datenübertragung zwischen Mobilgeräten und zwischen Mobilgerät und Sensorknoten gleich gestaltet sein. Die Discovery müsste entsprechend synchron funktionieren, da die beteiligten Geräte alle Discovery-Modi durchlaufen müssen.

Ein spezifischer Vorteil von Bluetooth ist, dass es mit seinem *Inquiry*-Mechanismus bereits eine Möglichkeit zur Discovery implementiert. Dieser Mechanismus hat jedoch einige Eigenschaften, die seine effektive Nutzung einschränken:

- Der Bluetooth Inquiry benötigt im Gegensatz zum passiven Betrieb eine hohe Leistung, kann also nicht ständig durchgeführt werden wenn auf den Batteriestatus geachtet werden soll.
- Geräte sind zwar in der Lage, einen Scan durchzuführen während sie bereits in Verbindung mit einem anderen Gerät stehen, die Verbindung ist in diesem Fall jedoch stark eingeschränkt.

3.2.3 Optimierung der Bluetooth-Discovery

Die relevanten Eigenschaften bei der Optimierung von Discovery-Mechanismen sind der Energieverbrauch und die Zeit bis zur Entdeckung eines Nachbarknotens. Hieraus ergeben sich die konkret zu optimierenden Metriken:

- die *mean discovery time*[4] ist definiert als die Zeit die vergangen ist seit zwei Geräte in Reichweite gekommen sind, bis zur Zeit in der sie sich als potentielle Nachbarn entdecken.
- die *mean power consumption*[5] beschreibt die Leistungsaufnahme in einem bestimmten Zeitraum.

Wie in Abschnitt 2.1.2 auf Seite 5 erwähnt wurde, ist es zumindest grundlegend nötig, den passiven und aktiven Discovery-Teil mit zufälligen Parametern zu versehen. In Bluetooth wird der aktive Modus als *inquiry*, der passive als *inquiry scan* bezeichnet. Entsprechend können die Zeiten, in denen sich ein Gerät in einem der Modi befindet wie folgt dargestellt werden [DARD07, BFM04]:

$$\begin{aligned} T_{inquiry} &= C_{inquiry} + random(0,\, 2\,V_{inquiry}) \\ T_{scan} &= C_{scan} + random(0,\, 2\,V_{scan}) \end{aligned}$$

Wobei $C_{inquiry}$, $V_{inquiry}$, C_{scan} und $V_{inquiry}$ jeweils die konstanten und variablen Teile der Aufenthaltszeit in einem Modus darstellen. Drula, *et al* [DARD07] schlagen zusätzlich vor, die Suchdauer und das Suchintervall variabel zu gestalten und untersuchten eine Reihe an Parametrisierungen für diese Werte. Basierend auf Simulationen und Tests auf Bluetoothgeräten haben sie die Leistungsaufnahme und die Entdeckungszeit ausgewertet. Basierend auf diesen Werten haben sie 5 Konfigurationen ausgewählt. Diese bilden ein

[4] dt.: Mittlere Zeit zur Entdeckung
[5] dt.: Mittlere Leistungsaufnahme

Spektrum von einem aggressiven Modus, der eine geringe Entdeckungszeit aber hohe Leistungsaufnahme aufweist, zu einem sehr „faulen“ Modus, welcher eine sehr geringe Leistungsaufnahme aber hohe Entdeckungszeiten hat.

Da keiner dieser Modi ideal ist sollte die Auswahl des aktuellen Modus entsprechend dynamisch gestaltet werden. Hierzu schlägt Drula zwei Vorgehen vor:

Recent activity level scheme[6]: In diesem Verfahren wird in aggressivere Modi geschalten, wenn ein Gerät ein anderes Gerät gefunden hat. Dieses Verfahren basiert auf der Tatsache, dass Menschen sich nicht „zufällig“ bewegen, sondern dazu tendieren sich an bestimmten Orten oft aufzuhalten, und sich zwischen diesen hin- und herbewegen. An den Orten, an denen sie sich aufhalten ist es zusätzlich wahrscheinlich, dass sich auch andere Netzwerkteilnehmer dort aufhalten[GP05]. Werden lange keine weiteren Geräte gefunden wird wieder in konservativere Modi gewechselt.

Location of past activity scheme[7]: Dieses Verfahren nutzt Ortsdaten aus, die beispielsweise über GPS kommen können. Das Gerät merkt sich Orte, an denen es zuvor andere Teilnehmer entdeckt hat und wechselt zu aggressiveren Modi wenn es diese Orte erneut aufsucht. Dieser Modus hat offensichtlich eine Lernphase, sodass er zu Beginn tendenziell schlechter agiert, im Laufe der Zeit jedoch effektiver wird.

Im Ergebnis der Tests konnte festgestellt werden, dass diese beiden Verfahren im Schnitt 50% weniger Leistung pro erfolgreich zustande gekommenem Kontakt benötigen und 4,6% respektive 8,6% bessere Performanz gegenüber einer naiven, leistungssparenden Methodik liefern. Diese Ergebnisse sind jedoch durch Simulationen basierend auf simplen Bewegungsmustern erreicht worden.

3.3 Simulation

Zur Auswertung der betrachteten Verfahren können verschiedene Werkzeuge benutzt werden. Besonders interessant sind dabei Simulatoren, die in Abschnitt 2.3 auf Seite 10 vorgestellt wurden. Im Rahmen dieser Forschungsarbeit wurde dabei besonders der ONE-Simulator betrachtet. Ein Vorteil dieses Simulators ist die Möglichkeit, Kartenmaterial zu verwenden um das gewünschte Szenario abzubilden. Im Allgemeinen wird für eine Simulation in ONE eine solche Karte mitsamt Bewegungsmustern für die Netzwerkteilnehmer erstellt. Hierbei sind bereits viele Muster vorhanden die direkt verwendet werden können. Unter anderem sind Muster für zufällige Bewegungen, feste Routen (beispielsweise für ÖPNV) und stationäre Knoten möglich.

4 Abschluss

4.1 Stand der Implementierung

Eine grundlegende Anwendung zur Anzeige der Wetterdaten der Sensoren wurde bereits zuvor erstellt. Im Rahmen dieser Forschungsarbeit wurde dazu begonnen, die zuvor von Grund auf eigene Implementierung durch eine Bytewalla-basierte Variante zu ersetzen. Hierzu musste Bytewalla zunächst durch zwei Funktionalitäten ergänzt werden, einen *Bluetooth Convergence Layer* und eine Implementierung der *Bluetooth Discovery*, da diese in Bytewalla bisher nicht zur Verfügung standen. Die grundlegend modulare Natur der DTN2-Implementierung sollte dies vereinfachen. Im Laufe der Entwicklung kamen jedoch einige Unzulänglichkeiten in der Portierung zu Tage, die eine Anpassung verschiedener, nicht zwangsläufig relevanter Anwendungskomponenten mit sich ziehen.

Um die weitere Entwicklung zu vereinfachen wurde aus diesem Grunde begonnen, weite Teile der Bytewalla-Implementierung zu ersetzen. In diesem Zusammenhang sollen zusätzlich Teile der Anwendung umstrukturiert werden, um diese an die Konventionen und Möglichkeiten der Java-Programmiersprache und der Android-Umgebung anzupassen.

Beispiele für bisher durchgeführte Änderungen sind:

- Erstellung einer grafischen Benutzeroberfläche zur Anpassung der DTN-Konfiguration, basierend auf Android SharedPreferences[1] und PreferenceActivities[2].
- Ersetzung eigens erstellter Klassen die äquivalente oder bessere Alternativen in den Standardbibliotheken besitzen, hierzu gehören eine ganze Reihe von eigenen *Collection*-Klassen und Thread-basierte Hilfsklassen.
- Zahlreiche kleinere Anpassungen zur Verbesserung der Lesbarkeit und Struktur des Quelltextes.

Die Arbeit an der Implementierung ist noch im Gange. Die aktuelle Implementierung der Bluetoothverbindungen ist bisher nicht zuverlässig lauffähig, was der Hauptgrund für die grundlegende Neustrukturierung des Bytewalla-Codes ist.

Die Nutzung des ONE-Simulators zur Evaluation von den im Laufe dieser Forschungsarbeit recherchierten Verfahren wurde begonnen. Die Erstellung von im Simulator nutzbarem Kartenmaterial wurde durchgeführt, sodass entsprechende Evaluationen im weiteren Verlauf der Arbeit durchgeführt werden können. Für eine effektive Evaluation müssen aussagekräftige Bewegungsmodelle erstellte werden, die dem Szenario entsprechen.

[1] http://developer.android.com/reference/android/content/SharedPreferences.html
[2] http://developer.android.com/reference/android/preference/PreferenceActivity.html

4.2 Abschluss

Im Laufe der Forschungsarbeit wurden eine Vielzahl an Anwendungen, Projekten und Verfahren recherchiert, die die Grundlage für sehr ausführliche weitere Bearbeitung des Themas bilden. Insbesondere wurden Möglichkeiten zur Verbesserung der Discovery untersucht. Es ist offensichtlich, dass die Discovery die Grundlage für die Funktionalität im vorgestellten Szenario bildet und Untersuchungen zu Routingprotokollen zweitrangig sind. Die Nutzung von modernen Smartphones und leistungsstarken Sensorknoten sorgt dafür, dass übliche Routingverfahren vorraussichtlich für das Szenario ausreichen, und die Forschungs- und Evaluationsaktivitäten sich deswegen auf das weniger bearbeitete aber nicht weniger wichtige Thema der Discovery beschränken können. Insbesondere scheint die in Abschnitt 3.2.3 auf Seite 13 beschriebene Discovery-Methodik eine ausgezeichnete Grundlage für weitere Verbesserungen in diesem Bereich. Insbesondere besteht großes Potenzial, diese Methodiken in Testbeds oder auf echten Geräten zu testen.

Literaturverzeichnis

[BE10] Dan Brown and Daniel Ellard, *DTN IP Neighbor Discovery (IPND)*, 2010.

[BFM04] Diego Bohman, Matthias Frank, and Peter Martini, *Performance of symmetric neighbor discovery in bluetooth ad hoc networks*, on Mobile Ad-hoc (2004), 138–142.

[CYDC06] M.C. Chuah, Peng Yang, B.D. Davison, and Liang Cheng, *Store-and-Forward Performance in a DTN*, Vehicular Technology Conference, 2006. VTC 2006-Spring. IEEE 63rd, vol. 1, IEEE, 2006, pp. 187–191.

[DARD07] Catalin Drula, Cristiana Amza, Franck Rousseau, and Andrzej Duda, *Adaptive energy conserving algorithms for neighbor discovery in opportunistic bluetooth networks*, Selected Areas in Communications, IEEE Journal on **25** (2007), no. 1, 96–107.

[DKNP06] Marie Duflot, Marta Kwiatkowska, Gethin Norman, and David Parker, *A formal analysis of bluetooth device discovery*, International Journal on Software Tools for Technology Transfer **8** (2006), no. 6, 621–632.

[Dom10] S Domancich, *Security in delay tolerant networks for the android platform*, Master's thesis, Royal Institute of Technology (KTH) (2010).

[FP05] E. Ferro and F. Potorti, *Bluetooth and Wi-Fi wireless protocols: a survey and a comparison*, Wireless Communications, IEEE **12** (2005), no. 1, 12–26.

[FSB+07] Kevin Fall, Keith L. Scott, Scott C. Burleigh, Leigh Torgerson, Adrian J. Hooke, Howard S. Weiss, Robert C. Durst, and Vint Cerf, *Delay-Tolerant Networking Architecture*, http://tools.ietf.org/html/rfc4838 (2007).

[FWSL11] Stephen Farrell, Howard Weiss, Susan Symington, and Peter Lovell, *Bundle Security Protocol Specification*, http://tools.ietf.org/html/draft-irtf-dtnrg-bundle-security-19 (2011).

[GP05] Joy Ghosh and SJ Philip, *Sociological orbit aware location approximation and routing in manet*, Broadband Networks, 2005. (2005), 1–10.

[HBMP06] Simon Heimlicher, Rainer Baumann, Martin May, and Bernhard Plattner, *Saft: Reliable transport in mobile networks*, Mobile Adhoc and Sensor Systems (MASS), 2006 IEEE International Conference on, no. i, IEEE, 2006, pp. 477–480.

[HCY10] Pan Hui, Jon Crowcroft, and Eiko Yoneki, *Bubble rap: social-based forwarding in delay tolerant networks*, IEEE Transactions on Mobile Computing (2010).

[HJ10] A. Haris and C.D. Jensen, *A DTN Study: Analysis of Implementations and Tools*, Ph.D. thesis, 2010.

[Hog11] Michel Hognerud, *Routing and Application Layer Optimizations for Delay-Tolerant Networks*, Master's thesis, KTH, 2011.

[JFP04] Sushant Jain, Kevin Fall, and Rabin Patra, *Routing in a delay tolerant network*, Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '04 (2004), 145.

[KOK09] A. Keränen, J. Ott, and T. Kärkkäinen, *The ONE simulator for DTN protocol evaluation*, Proceedings of the 2nd International Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, p. 55.

[LDGD11] Anders Lindgren, Elwyn Davies, Samo Grasic, and Avri Doria, *Probabilistic Routing Protocol for Intermittently Connected Networks*, http://tools.ietf.org/html/draft-irtf-dtnrg-prophet-08 (2011).

[MM04] C. Siva Ram Murthy and B.S. Manoj, *Ad Hoc Wireless Networks: Architectures and Protocols*, 2004.

[NGR09] Erik Nordström, Per Gunningberg, and Christian Rohner, *A search-based network architecture for mobile devices*, Department of Information (2009).

[RK03] M.A. Rónai and E. Kail, *A simple neighbour discovery procedure for Bluetooth ad hoc networks*, Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE, vol. 2, IEEE, 2003, pp. 1028–1032.

[SBT00] Theodoros Salonidis, Pravin Bhagwat, and Leandros Tassiulas, *Proximity awareness and ad hoc network establishment in Bluetooth*, Proceedings of the First Annual ACM Workshop on Mobile and Ad Hoc Networking and Computing (2000).

[SPR08] T. Spyropoulos, K. Psounis, and C.S. Raghavendra, *Efficient Routing in Intermittently Connected Mobile Networks: The Multiple-Copy Case*, IEEE/ACM Transactions on Networking **16** (2008), no. 1, 77–90.

[VB00] Amin Vahdat and David Becker, *Epidemic routing for partially connected ad hoc networks*, Science (2000).

[WH11] Lloyd Wood and Peter Holliday, *Using HTTP for delivery in Delay/Disruption-Tolerant Networks*, http://tools.ietf.org/html/draft-wood-dtnrg-http-dtn-delivery-07 (2011).