Institut für Parallele und Verteilte Systeme

Abteilung Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Fachstudie Nr. 155

Analyse von Netzwerksimulatoren

Daniel Reichelt Julian Trischler Tobias Stubenvoll

Studiengang: Softwaretechnik

Prüfer: Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer: Dipl.-Inf. Andreas Benzing
begonnen am: 30.04.2012
beendet am: 30.10.2012

CR-Klassifikation: I.6.3

Inhaltsverzeichnis

1 Einleitung

Im Rahmen dieser Fachstudie sollten verschiedene Netzwerksimulatoren hinsichtlich ihrer Einsatztauglichkeit in der Abteilung Verteilte Systeme (VS) beim Institut für Parallele und Verteilte Systeme (IPVS) der Universität Stuttgart überprüft werden.

Die einzelnen Schritte der Studie umfassten dabei zunächst eine kurze Angebotserfassung der verfügbaren Netzwerksimulatoren und darauf basierend die grundlegende Einarbeitung in die Arbeitsweise ereignisbasierter Netzwerksimulatoren. Zu Beginn der Studie war zu entscheiden, ob einige wenige Simulatoren in hohem Detail untersucht oder ob ein etwas allgemeiner gehaltener Überblick über eine breitere Produktpalette gegeben werden sollte. Schließlich wurde der letztgenannte Ansatz vom Betreuer vorgegeben.

Danach konnten die Anforderungen der Abteilung VS in Einzelgesprächen mit den Mitarbeitern aufgenommen werden. Diese sind in Kapitel 2 zu finden.

Auf dieser Basis konnten die Merkmale und Funktionsweisen der einzelnen Simulatoren relativ gezielt recherchiert und untersucht werden. Die Ergebnisse hierzu finden sich in Kapitel 3.

Eine Zusammenfassung und Empfehlung im Kapitel 4 bilden den Abschluss der Arbeit.

1.1 Konventionen und Terminologie

Sofern „Schichten“ oder „Netzwerkschichten“ zur Sprache kommen, ist die Rede von den Schichten des Protokollstapels nach dem OSI-Referenzmodell. Allerdings werden im Sprachgebrauch dieser Ausarbeitung die Schichten 5-7 des OSI-Modells als *Schicht 5* beziehungsweise *Anwendungsschicht* bezeichnet.

Unter „Tracefiles“ wird im Rahmen dieser Arbeit nur das Modell der Bewegungsvorschrift mobiler Knoten verstanden. Sofern Tracefiles weitere Funktionalitäten ermöglichen, ist dies explizit erwähnt. Der Begriff meint *weder* das Aufzeichnen des Programmablaufs in einem Debugger in Form von *Stacktraces*, *noch* Programmausgaben, die in einem Simulationslog protokolliert werden können, was hier als *Logging* bezeichnet wird.

2 Anforderungen

2.1 Ausgangssituation

Alle im Rahmen der Mitarbeiterbefragungen genannten Simulatoren waren Teil der Untersuchungen dieser Fachstudie. Diese waren beziehungsweise sind: ns-2, OMNeT++, PeerSim und The ONE.

In der Regel stimmen die genannten Pro-/Contra-Argumente für beziehungsweise gegen den Einsatz eines bestimmten Simulators mit den Befunden dieser Studie überein.

2.2 Arbeitsgebiete

Die in Forschung befindlichen Themen sind breit gefächert und nachfolgend kurz umrissen.

Netze in Rechenzentren Die simulierten Endsysteme (wenige 100 bis 10.000) sind virtuelle Maschinen auf physischen Hosts. Auf Transport- und Vermittlungsschicht werden Topologien in Baumstruktur auf Basis von TCP/IP und UDP/IP simuliert. Gestützt wird dies von geswitchten Ethernets. Die untersuchten Parameter sind Verzögerung, Datenrate sowie daraus abgeleitet die Netzlast und die Fairness bezüglich Bandbreitenzuteilung. Pro Experiment werden zur Mittelwertbildung 10-100 Simulationsläufe mit einer Laufzeit von mehreren Minuten (bei großen Netzen auch länger) durchgeführt.

Public Sensing Simuliert werden mobile Endgeräte, die drahtlos über WLAN, UMTS oder Bluetooth in einem Peer-to-Peer-Netz miteinander oder mit einem stationären Knoten kommunizieren. Untersucht werden hier vor allem der Energieverbrauch der Funkkommunikation und die Latenz im UMTS-Netz. Als Mengengerüste wurden im einen Fall etwa 100 Knoten, 100-1.000 Durchläufe, 12-24 Stunden Laufzeit genannt, im anderen waren es 500-10.000 Mobiltelefone, 100 Durchläufe zu je 5-30 Minuten.

globales Internet Ziel ist die Erfassung der Netzwerklast von Topologien im globalen Internet mit 1.000-10.000 Broker/Clients. Die Dauer der Experimentdurchführung wurde bei 96 parallel, aber voneinander unabhängig laufenden Simulationen mit rund fünf Tagen angegeben.

Cyber physical systems/Regelungstechnik Hier sollte ehemals auf dem CAN-Bus aufbauende Regelungstechnik auf IP-Netze mit dann 100 bis 1.000 Knoten umgestellt werden. Relevante Simulationsparameter sind hier Verzögerung und Paketverlust. Die simulierten Netzwerkschichten umfassten daher die Schichten 3 und 4.

Peer-to-Peer-Routingprotokolle Ein Knoten führt hierbei mehrere unterschiedliche Protokolle aus, wobei ausgewertet wird, wie viele Nachrichten zum Beitritt und Verlassen des jeweiligen Systems erforderlich sind, bis die gewünschte Topologie aufgebaut ist. Hierbei schwankt die Anzahl an beteiligten Peers von wenigen 10 bis auf mehrere 1.000. Ein Experiment wird bis zu 100 mal ausgeführt, dies dauert in simulierter und Echtzeit maximal einige Minuten.

VANET Untersucht wird die Funkkommunikation über WLAN von Fahrzeug zu Fahrzeug sowie Fahrzeug zu Infrastruktur. Die Simulationslaufzeit wurde mit 1-2 Stunden angegeben, ausgewertete Parameter sind die auf Schicht 3 üblichen.

Complex Event Processing Hierbei werden Ereignisse kommuniziert und analysiert. Erschwert wird das Ganze durch das Hinzukommen von Mobilität, etwa Kraftfahrzeugen. Z.B. bremsen Fahrzeuge und fahren schnell wieder an, die Folge kann ein Unfall sein, wodurch sich ein Stau bildet. Der Strom an Positionsdaten wird in Form von Bandbreite untersucht, weitere Parameter sind die Latenz und Nachrichtenanzahl im Netz. Die Autos stoßen die Kommunikation über WLAN, UMTS oder GSM durch ihre Positionsmeldungen an die Infrastruktur an, anhand der eingehenden Daten fragen die Server ggf. andere Datenquellen nach weiteren Informationen. Bei Experimenten kommen 100-1.000 Fahrzeuge und etwa 20 Serverknoten zum Einsatz. Die 1.000 Sekunden simulierter Zeit

kosten zwischen einer Stunde und zwei Tagen an Hardwareressourcen, in denen zwischen 10 und 100 Ausführungen inbegriffen sind.

3 Die Simulatoren

Im Folgenden werden die untersuchten Simulatoren in alphabetischer Reihenfolge erläutert.

3.1 CloudSim

Sofern nicht anders angegeben wurden die folgenden Informationen dem Paper [1] oder der Website [2] entnommen.

Das „Cloud Computing and Distributed Systems (CLOUDS) Laboratory“ der Universität Melbourne brachte die Version 1.0 im April 2009 heraus, derzeit aktuell ist die Version 3.0.1, die Mitte Oktober 2012 veröffentlicht wurde. CloudSim ist eine Java-Bibliothek und setzt daher die JVM als Laufzeitumgebung voraus.

Verbreitung findet CloudSim vor allem bei Anbietern und teilweise auch Nutzern von Cloud-Diensten zur Angebots- beziehungsweise Kostenoptimierung. Beispielsweise verwenden die US-amerikanischen HP Labs CloudSim in ihrer Forschung über Ressourcenzuweisung in der Cloud sowie der energieeffizienten Nutzung der Ressourcen eines Rechenzentrums.

Netzwerktopologien für CloudSim basieren auf dem BRITE-Format, welches zum gleichnamigen, mittlerweile eingestellten Projekt gehört. Allerdings dürfte Letzteres keinen Einfluss auf die Zukunftssicherheit haben. Das Projekt steht unter der LGPL. Die Weiterentwicklung des Projekts wird überwiegend durch die Universität vorangetrieben, seit dem letzten Release sind im Versionskontrollsystem [3] nur Fehlerbeseitigungen zu verzeichnen. Allerdings existiert eine zwar kleine, aber aktive Google Groups Community [4].

3.1.1 Einsatzzwecke

CloudSim wird vornehmlich zur Simulation großer Cloud-Computing-Rechenzentren mit ihren typischen Topologien und Anwendungen, welche Nachrichten austauschen, eingesetzt. Dies umfasst für virtualisierte Server unter anderem anpassbare Richtlinien für die Zuteilung von Host-Ressourcen an virtuelle Maschinen sowie den Energieverbrauch der simulierten Ressourcen. Darüber hinaus können auch föderierte Clouds simuliert werden. Dies sind Clouds, die vom Endverbraucher transparent als eine Einheit genutzt werden können, jedoch über die physikalischen Grenzen eines einzelnen Rechenzentrums hinaus über mehrere Standorte verteilt betrieben werden. CloudSim unterstützt ein dynamisches Einfügen von Simulationselementen. Simulationen können gestoppt und zu einem späteren Zeitpunkt fortgesetzt werden.

CloudSim reiht sich im Ganzen eher weniger in die Riege klassischer Netzwerksimulatoren ein, da sich mit ihm eher die für Rechenzentren typischen Netze sowie deren Vernet-

zung simulieren lassen. Dabei liegt das Hauptaugenmerk zumeist auf Optimierungsproblemen der den Vernetzungen zugrundeliegenden Strukturen und der Ressourcenverteilung auf verschiedenen Ebenen. Auf niedrigen Ebenen geht es hierbei beispielsweise um die Zuteilung von Hardwareressourcen an virtuelle Maschinen oder über die Platzierung einer virtuellen Maschine – sowohl bezüglich des beherbergenden Hardware-Servers innerhalb des Rechenzentrums als auch der Auswahl des Standortes bei föderierten Clouds. Auf der höchsten, vom Endanwender kontrollierbaren, Anwendungsebene könnten selbstorganisierende, verteilte Systeme optimiert werden.

Aufgrund dieser Zielsetzungen lassen sich von Haus aus keine Netzwerkschichten der höheren Ebenen simulieren, es werden nur die Parameter Verzögerung und Bandbreite unterstützt. Damit lässt sich immerhin die Kanalbitzahl berechnen. Ebensowenig werden Tracefiles unterstützt, da ausschließlich von ortsfesten Netzwerkelementen ausgegangen wird.

3.1.2 Bewertung der vorhandenen Dokumentation

In dem Paper [1] wird relativ ausführlich auf das Konzept und die Kernkompetenzen von CloudSim eingegangen. Als Beispiele sind acht „basic“, vier „Netzwerk-“ und sechs Energieverwaltungsprojekte im Download-Archiv enthalten. JavaDoc-Kommentare sind informativ und brauchbar, gesichtete Code-Teile waren gut verständlich dokumentiert. Eine FAQ ist unter [5] abrufbar.

3.1.3 Modellierung, Durchführung und Auswertung

Wie bereits erwähnt ist CloudSim als Java-Bibliothek implementiert, die Programmierschnittstelle ist verständlich gekapselt und gut organisiert. Die Notwendigkeit für Eingriffe in den Programmcode von CloudSim selbst war im Rahmen dieser Studie nicht ersichtlich. Eine saubere Trennung der Programmcodes des Simulators selbst und der jeweiligen Simulation ist somit gut möglich.

Netzwerktopologien werden im BRITE-Format [6] beschrieben. Die Knoten IDs aus der Topologie werden anschließend Entitäten der Cloud (Brokers, Datacenter, Hardware-Server, virtuelle Server etc.) zugeordnet. Die Simulation wird unter Verwendung der Programmierschnittstelle der Bibliothek in einer Java-Applikation implementiert. Log-Ausgaben können beliebig angepasst werden. Mechanismen zur Auswertung sind nicht vorhanden, diese muss anhand der Textlogs separat erfolgen.

Bezüglich Skalierbarkeit und Performance trifft das erwähnte Paper [1] keine Aussage, lediglich dass mehrere Experimente mit 1.000 – 1.000.000 Knoten auf einer Maschine mit Dual-Xeon-CPU und 16GB RAM durchgeführt wurden.

3.2 GTNetS

Die letzte Version des GTNetS – Georgia Tech Network Simulator – stammt vom Oktober 2008 und ist damit zum Zeitpunkt der Fachstudie vier Jahre alt. Seither sind keine Aktivitäten mehr zu verzeichnen [7]. Der Verweis auf das alte Mailinglistenarchiv ist ein toter Link, auf dessen Fehlerseite sich der Juni 2004 als letztes Änderungsdatum befindet. Auch die aktuelle Mailingliste führt ins Datennirvana [8]. Somit muss der Simulator unserer Meinung nach als tot betrachtet werden. Da aus ihm Code zu ns-3 floss und auch einige Entwickler Code zu ns-3 beisteuerten [9], ist stark davon auszugehen, dass sich die Konzentration der Entwickler von ihrem ursprünglichen Projekt auf ns-3 verlagert hat.

Federführend wurde das Projekt von George Riley und seinen Studenten am Georgia Institute of Technology entwickelt. Das amerikanische Verteidigungsministerium sowie die National Science Foundation förderten GTNetS mit zwei Projekten [10].

Sowohl unter unixoiden Betriebssystemen, wie Linux, Solaris und Mac OS X, als auch auf Windows kann der Simulator eingesetzt werden [11]. Laut der Georgia Tech-Universität eignet sich der Simulator für mittlere bis große Netzwerkexperimente. [7] Was genau darunter zu verstehen ist, eine repräsentative Knoten-, Verbindungs- oder Nachrichtenanzahl, ließ sich trotz intensiver Recherche von uns nicht ermitteln. GTNetS ist in C++ geschrieben, entsprechend ist eine Laufzeitumgebung hierfür erforderlich. Zur verwendeten Lizenz ließen sich keine Hinweise finden, er benutzt jedoch ein ebenfalls am Georgia Institute of Technology entwickeltes Modul für das Border-Gateway-Protokoll, welches unter der GNU GPL steht. Auch die von der Boston University stammende BRITE-Anbindung ist unter einer eigenen Lizenz veröffentlicht, die die freie Weitergabe und Codemodifikationen für Forschungs- und Ausbildungszwecke erlaubt, entsprechend muss GTNetS ebenfalls unter einer ähnlichen und GPL-kompatiblen Lizenz stehen, sofern er selbst keine Lizenzverstöße begeht [12].

3.2.1 Einsatzzwecke

Der Simulator ist in der Lage ab Schicht 2 alle aufwärts zu simulieren. Von Haus aus bringt er eine Unterstützung für IEEE 802.3 und IEEE 802.11 mit. Sowohl IPv4 als auch das Border-Gateway-Protokoll lassen sich direkt auf der Vermittlungsschicht verwenden. Außerdem kann der Anwender zwischen statischen oder dynamischen (NIx-Vector-Routing) Berechungsverfahren wählen oder eigene Definitionen verwenden. Auf der Transportebene stehen Klassiker wie TCP und UDP bereit. FTP und HTTP sind auf Schicht 5 nach eigenen Aussagen bereits als Server und Client fertig vorhanden [13]. In der API-Dokumentation werden auch Protokolle aus dem Bereich der Peer-to-Peer-Systeme, wie z.B. Gnutella oder Chord, aufgelistet [14]. Einen Hauptfokus legen die Entwickler auf die klare Trennung von Protokollstapelschichten. Zugleich soll es in diesem Simulator einfach möglich sein, Pakete zwischen benachbarten Schichten mit Zusatzinformationen zu versehen, wenn sie nach unten gereicht werden, bzw. diese in der Rückrichtung wieder zu entfernen [7].

So lassen sich beliebige Parameter der verschiedenen Netzwerkschichten, wie Paketverlust, Nachrichtenanzahl, Datenrate oder Kanalbitzahl simulieren.

An Tracefile-Formaten unterstützt GTNetS BRITE, dieses aber nur unter *nix, nicht jedoch Windows [15]. Für die Bewegungsmuster stehen Random-Waypoint-Algorithmen und statisch definierte Routen zur Verfügung [13]. Demnach kann GTNetS sowohl für statische als auch für dynamische Netzwerksimulationen eingesetzt werden.

3.2.2 Bewertung der vorhandenen Dokumentation und Einarbeitung

Die Website enthält einige informative Seiten und gute Zusammenfassungen, die kurz und knapp das Wichtigste erläutern. So sind vor allem die Startseite sowie die Liste der Funktionalitäten sehr gute Einstiegspunkte [7, 13]. Auf der anderen Seite sind aber auch – wie zuvor bereits erwähnt – völlig veraltete Rubriken zu finden, etwa die Mailingliste [8]. Das System für die Fehlerverfolgung ist nicht öffentlich zugänglich. Bei Fehlerberichten sollen direkt die Entwickler an der Universität per E-Mail kontaktiert werden [16]. Dies schließt eine größere Entwicklungsbeteiligung der Gemeinschaft von vorn herein aus.

Eine mit Doxygen generierte API-Dokumentation ist zwar vorhanden, hat aber das selbe Problem wie ns-2. Ausschließlich die Code-Struktur ist erfasst, zusätzliche Kommentare sind nicht vorhanden, sodass diese Form der Beschreibung nicht weiterhilft [14]. Im Code selbst sind jedoch in Kommentaren die einzelnen Klassen und Methoden sauber dokumentiert – leider spiegelt sich dies in der generierten HTML-Dokumentation nicht wider. Auch erschwert die rohe Doxygen-Auszeichnung stellenweise das flüssige Lesen der Doku im Quellcode selbst.

Die FAQ ist recht kurz und behandelt viele Problembehebungen bei Übersetzungsfehlern [15].

Der veröffentlichte, gepackte Tarball enthält neben dem eigentlichen Code auch einige Beispieleprogramme. Leider sind in ihm aber auch Verschmutzungen durch unsachgemäßes Bedienen des Versionsverwaltungssystems SVN enthalten: jeden Ordner ziert ein `.svn`-Ordner. Auch von den Entwicklern zu Debugzwecken angelegte Dateien, Speicherabbilder sowie Objektdateien sind in der offiziellen Veröffentlichung enthalten [12], was unseren Projekteindruck zusätzlich trübte.

Das Handbuch macht mit seinen knapp 150 Seiten einen guten Eindruck. Auch enthält es größere Beispielcodeauszüge. Extra Tutorials haben wir auf der Website keine verlinkt gefunden. Die Musterquellen im Handbuch selbst haben jedoch Tutorialcharakter. Auch sind zu Beginn des Referenzmanuals die grundlegenden Konzepte und Architektur GTNetS' beschrieben. Im Gegensatz zur vorhin bemängelten Doxygen-API-Dokumentation ist im Handbuch eine mit weiterführenden Erläuterungen versehende Klassendokumentation enthalten [10].

3.2.3 Modellierung, Durchführung und Auswertung

Hinweise darauf, dass beim Verwenden des Simulators Kerncode angefasst werden muss, konnten wir nicht finden. Der eigene Quelltext kann also getrennt vom Kern entwickelt werden.

Die mitgelieferte Standardbibliothek des Simulators macht einen vernünftigen und brauchbaren Eindruck. Aus den enthaltenen Beispielen ist ersichtlich, dass sofort mit der eigentlichen Arbeit begonnen werden kann. Zwischenformate sind keine vorhanden, es wird direkt in C++ das Szenario kodiert. Eigene Komponenten können bei geschicktem Vorgehen in anderen Experimenten wiederverwendet werden.

GTNetS hat eine aus UNIX bekannte API für die Anbindung der Anwendungs- an die Transportschicht mit Sockets [7].

Über eine Qt-basierte und sehr simple Benutzeroberfläche können Nachrichtenflüsse in der aufgebauten Topologie visualisiert werden [17]. Jedoch wird hierfür das mittlerweile schon in die Jahre gekommene und veraltete Qt 3.3 genutzt [18].

Der Simulator bringt eigene Möglichkeiten zum Schreiben von Textprotokollen mit, die nach dem Experimentlauf mit Werkzeugen wie Gnuplot ausgewertet werden können. Schichtenweise kann sehr einfach festgelegt werden, ob Informationen mitgeschnitten werden sollen oder nicht. Die zeitliche Auflösung der Protokolldatei kann ebenfalls mit einer einzelnen Anweisung eingestellt werden. Während des Experimentlaufs sammelt GTNetS optional verschiedene statistische Informationen, die am Ende zusammengefasst und in Histogrammen ausgegeben oder Tabellenkalkulationen weiterverarbeitet werden können [10].

Über die Wissenschaftlichkeit der mit den GTNetS-Modellen erstellten Experimente liegen uns keine Erkenntnisse vor.

[19] nennt ein Experiment bezüglich Skalierbarkeit, das am Pittsburgh Supercomputer Center durchgeführt wurde. Eine Simulation mit etwa 3.700 Knoten auf einer HP Alpha CPU dauerte 560 Sekunden, rund 480.000 Knoten auf 128 CPUs wurden in 870 Sekunden simuliert.

3.3 JiST/SWANS

JiST, Java in Simulation Time, liegt uns in der Version 1.0.6 vom März 2005 vor und ist damit zum Zeitpunkt der Fachstudie über sieben Jahre alt. Die Simulationsengine für diskrete Ereignisse stammt von der Cornell-Universität in den USA und wurde vom dortigen Verteidigungsministerium gefördert. Die Software steht unter einer eigenen Lizenz, die die freie Weitergabe und Modifikation für nichtkommerzielle, wissenschaftliche und Ausbildungszwecke erlaubt.

SWANS, Scalable Wireless Ad hoc Network Simulator, ist ein Aufsatz auf JiST und lt. Hersteller für riesige Szenarien mit mehreren 10.000 – 1.000.000 Knoten ausgelegt [20].

Über den Simulator existieren neben sehr vielen Vorträgen und Präsentationen auch eine Abschlussarbeit und wissenschaftliche Artikel, die allesamt sehr übersichtlich auf der offiziellen Projektwebsite aufgeführt sind [21]. Inwieweit das Projekt eine noch aktive Community hat, ließ sich von uns nicht ermitteln.

3.3.1 Einsatzzwecke

SWANS unterstützt die Simulation der Netzwerkschichten 1–5 [22], wobei für jede davon schon einige vorgefertigte Protokolle existieren, etwa Zone Routing Protocol und On-Demand Distance Vector Routing zum Routen als auch UDP und TCP auf der Transportebene. Entsprechend sollten sich sehr viele unterschiedliche Parameter simulieren lassen.

Hinweise auf die direkte Unterstützung von Tracefiles konnten wir keine finden. Dafür ist der Simulator, wie bereits oben erwähnt, für sehr große Szenarien ausgelegt, die mehrere 100.000 Knoten umfassen können. Dies wird durch die Vermeidung von Kontextwechsel und einer großen Speichereffizienz erreicht, so ist etwa ein Nachrichtenobjekt nicht veränderbar und kann daher von mehreren Knoten gleichzeitig referenziert werden.

Trotz Java ist JiST ziemlich performant, er sticht im speziellen Vergleich sogar ns-2 in Durchsatz und Speicherverbrauch aus [23, 24].

JiST/SWANS ist für dynamische Ad-Hoc-Netze geeignet [25].

3.3.2 Bewertung der vorhandenen Dokumentation und Einarbeitung

JiST ist einer der wenigen Simulatoren, dessen Konzepte im Handbuch recht gut beschrieben sind. Auch ist in das Handbuch ein kleines Tutorial integriert.

Für diesen Simulator existieren viele verschiedene wissenschaftliche Artikel. Allgemein ist die verfügbare Dokumentation, allen voran das Benutzerhandbuch, ziemlich brauchbar. Allerdings enthält das Handbuch Quellcode als eingebettete Grafiken, was das schnelle Kopieren nicht erlaubt [26].

Die SWANS-JavaDoc-API ist auf den ersten Blick gut gestaltet. Viele stichprobenartig überprüften Stellen in der Dokumentation waren kurz, aber auf den Punkt gebracht und gut verständlich. Bei näherer Betrachtung waren uns die dargebotenen Informationen doch allerdings immer wieder etwas zu dürftig, sodass Fragen nur durch die die Studie des Javaquelltexts beantwortet werden konnten. Im Quellcode selbst sind Kommentare i.d.R. nur an komplexeren Stellen zu finden, sie sind sehr knapp, jedoch ausreichend und verbessern die Lesbarkeit.

3.3.3 Modellierung, Durchführung und Auswertung

Wie bei den meisten anderen Simulatoren muss der Kern nicht angefasst werden, um Netzwerksimulationen zu schreiben. Der Benutzercode ist vom Simulatorcode getrennt.

Die mitgelieferte Standardbibliothek enthält alle grundlegenden Bausteine, die für Netzwerksimulationen benötigt werden. Wenn eigene Komponenten entsprechend gestaltet sind, können sie auch in anderen Projekten wiederverwendet werden.

Ein eigener Übersetzer schreibt die kompilierten Klassen-Dateien um, sodass diese über einen mitgelieferten Simulationskern gestartet und weiterhin in einer normalen JVM aus-

geführt werden können. Dadurch kann lt. Handbuch zum einen eine erweiterte Simulationsfunktionalität bereitgestellt und andererseits gewohnt in Java programmiert werden [27].

Mittels Reflection und über die Java-Scriptingengine verfügbare Schnittstellen lassen sich Experimente recht einfach konfigurieren [28]. So können schnell einfache Experimente im Scriptcode zusammengestöpselt werden.

Event-Methoden in den Javaklassen werden nach dem zusätzlichen Übersetzungsvorgang in Event-Objekte umgewandelt, die bei einem Aufruf im Code zur Laufzeit in Warteschlangen gestellt und abgearbeitet werden. Somit können sich diese Ereignisfunktionen selbst aufrufen, was in einer normalen Situation sonst zu einem Programmabbruch aufgrund eines Stapelüberlaufs durch die Endlosrekursion führen würde. So aber kann ein Ereignis endlos ein neues Ereignis gleichen Typs erzeugen [26]. Die Simulationszeit schreitet nur nach Aufruf einer speziellen `sleep(n)`-Methode fort [29]. Um blockierende Ereignisse in Verbindung mit Simulationszeitfortschritten zu implementieren, muss die Ereignismethode mit einer `throws JistAPI.Continuation`-Exception deklariert werden, die jedoch weder geworfen noch gefangen wird; sie dient einzig dem JiST-Übersetzer als Hinweis beim Umschreiben der mit `javac` erzeugten Class-Dateien [30]. Das zu Beginn des Handbuchs aufgeführte Ziel, möglichst ganz gewohnt in Java zu programmieren [31], steht mit diesem Vorgehen unserer Meinung in deutlichem Widerspruch. Allgemein wird die bekannte Semantik aus Java an einigen Stellen hierdurch tiefgreifend verändert.

Um Speicherplatz zu sparen, können sogenannte *timeless*-Objekte verwendet werden, die im Gegensatz zu anderen nicht als Kopie sondern per Referenz übergeben werden [32].

Über `GuiLog` können Ereignisse im Simulator zu Debugzwecken in einem grafischen Fenster angezeigt werden [33]. An Auswertungsmöglichkeiten der durchgeführten Experimente ist sonst nichts direkt im Framework enthalten, sodass die übliche Auswertungstechnik über selbst angefertigte Logdateien und Gnuplots angewandt werden kann.

Zur wissenschaftlichen Verifizierung liegen uns trotz wissenschaftlicher Artikel zu JiST/SWANS keine gesicherten Erkenntnisse vor.

3.4 ns-2

Der Network Simulator Version 2 – ns-2 – steht unter der GNU GPLv2 und wurde Ende 2011 in der Version 2.35 ein letztes Mal veröffentlicht. Er ging im Jahr 1989 aus einer Abspaltung des REAL-Netzwerksimulators hervor. Das amerikanische Verteidigungsministerium unterstützte seit 1995 die ns-2-Entwicklung mit mehreren Projekten. Auch Sun Microsystems und andere große namhafte Softwarehersteller sowie Forschungseinrichtungen trugen zu ns-2 bei [34].

ns-2 scheint teilweise immer noch ein De-facto-Standard zu sein, jedenfalls wird er von fast allen anderen Simulatoren referenziert und schneidet im direkten Vergleich bezüglich einzelner Funktionsmerkmale teilweise um Größenordnungen schlechter ab [23, 24, 19, 35, 36, 37]. Allerdings schrumpft die Benutzerzahl von ns-2 stetig, da er zwar sehr mächtig, aber dennoch ziemlich kompliziert zu bedienen ist und andere Simulatoren ihn im Funk-

tionsumfang auf- oder gar überholen und um Längen benutzerfreundlicher sind. Diese Erfahrungen haben auch viele unserer Befragten aus der Abteilung Verteilte Systeme gemacht: Sie hatten am Rande oder gar direkt mit ns-2 zu tun, fanden ihn aber aber dennoch, gelinde gesagt, sehr stark verbesserungswürdig. Auch sind noch im Bugtracker auf SourceForge ungelöschte Spameinträge vom März zu finden [38]. Zudem wird ns-2 zu Gunsten seines Nachfolgers ns-3 nicht mehr aktiv weiterentwickelt, sondern nur noch gewartet [39, 40], sodass wir für neue Forschungsprojekte dringend von ns-2 abraten.

Als Programmiersprachen kommen C++ für den eigentlichen Kern und OTcl für die Skripte zum Einsatz. Entsprechend sind eine plattformspezifische C++- sowie eine OTcl-Laufzeitumgebung erforderlich. Der Simulator ist auf allen gängigen Systemen lauffähig: FreeBSD, Linux, SunOS, Solaris, POSIX-ähnliche Betriebssysteme und Windows, sogar Cygwin werden lt. Entwicklerangaben unterstützt. Auf der Website sind alle Anforderungen im Detail und für spezielle Module aufgelistet [41].

3.4.1 Einsatzzwecke

Fast alle Szenarien lassen sich lt. Dokumentation mit ns-2 umsetzen, sofern die Topologie nicht allzu groß ist. Der Simulator unterliegt nur einigen wenigen funktionalen Einschränkungen, die allesamt auf einer extra Seite vorbildlich aufgeführt sind [42].

Es können alle Schichten simuliert werden [43]. Auf der Sicherungsschicht stehen z.B. CSMA/CD sowie CSMA/CA fertig zur Verfügung [44]. SPF- und DV-Routing sind Implementierungen der Routingschicht, ebenso wie eine zentralisierte und eine Shared Tree Multicast-Umsetzung [45]. Neben den Klassikern TCP und UDP stehen auch exotischere Protokolle wie SCTP, SRM, PLM und DCCP auf der Transportschicht zur Verfügung [46]. Auf der Anwendungsschicht sind u.a. FTP, Telnet und HTTP vertreten, Letzteres in verschiedenen Ausprägungen, z.B. Client, Server und Cache [47].

Der Simulator verwendet sein eigenes Tracefile-Format, das auch *new trace format* oder *Nam Trace Format* genannt wird – Nam, Network Animator, ist ein Tcl/Tk-Animations-Werkzeug zum Visualisieren der Simulationen. Dieses Format unterstützt nicht nur Positionsinformationen, sondern neben vielen weiteren Daten auch Paketinformationen und -regeln, mit denen selbige weitergeleitet, verworfen oder gesendet und empfangen werden können [48, 49].

ns-2 eignet sich nach eigenen Angaben für vergleichsweise recht einfache Netzwerkexperimente. Bei großen Experimenten ist auf der Website von 1.000 Knoten und 2.000 Verbindungen die Rede [50]. Dafür gibt es aber bzgl. der simulierbaren Parameter kaum Einschränkungen. Es lassen sich mit ihm eigentlich alle Beliebigen analysieren, etwa Paketverlust, Nachrichtenrate, Verzögerung, Datenrate und Netzwerklast.

Durch die vielen mitgelieferten Modelle ist es möglich, sowohl kabelgebundene als auch kabellose Knoten gut zu simulieren.

3.4.2 Bewertung der vorhandenen Dokumentation und Einarbeitung

Eine FAQ ist zwar vorhanden, bringt in den meisten Fällen aber eher nichts. Hier werden Fragen wie „Wo finde ich $Foo?“ mit „Auf der Website unter $Bar“ beantwortet [51].

Die Abschnitte *Commands at a glance* im Handbuch sind da schon deutlich geeigneter. Sie beschreiben am Ende jeden Kapitels wichtige Befehle, die in Simulationsskripten häufig Anwendung finden [52]. Das Handbuch dient gleichzeitig als API-Beschreibung, eine gesonderte ist zwar verfügbar, jedoch ergaben stichprobenartige Überprüfungen, dass nur die Codestruktur erfasst wurde, eine weitere Dokumentierung, die einen tieferen Sinn ergeben würde, erfolgte nicht. Zudem stammt die letzte Generierung der API-Dokumentation aus dem Jahr 2004 [53].

Dokumentation zu allgemeinen Konzepten ist vorhanden, allerdings wird man von der Vielzahl erschlagen. Tutorials sind ebenfalls en masse abrufbar.

Leider scheint es so, als ob die jeweiligen Dokumentationsautoren niemals ihre Werke selbst gesichtet hätten. Das Handbuch ist stellenweise absolut unlesbar, da das zum Übersetzen verwendete `latex2html` die Quellcodes nicht korrekt transformierte und so LaTeX mit C++ und OTcl gemeinsam vermischt auftauchen. Jede der drei Sprachen ist mit ihren vielen Sonderzeichen für sich alleine schon schwierig zu lesen, aber in Vermengung mit den anderen entsteht eine nur sehr schwer verständliche Mixtur, die das Verständnis des Gelesenen nicht erleichtert [54]. Die PostScript- oder PDF-Versionen sind der HTML-Version deshalb vorzuziehen, jedoch sind sie lesezeichenverwaltungstechnisch etwas schwieriger handzuhaben.

Der Vorteil, dass vor allem das Handbuch sehr ausführlich ist (knapp 650 HTML-Seiten oder 430 DIN A4-Seiten), ist gleichzeitig auch nachteilhaft. So muss erst eine große Menge an Dokumentation gelesen werden, bevor mit der eigentlichen Arbeit begonnen werden kann. Auch ist fraglich, inwieweit die beschriebenen Details noch aktuell sind, stammen einige Informationen noch aus dem Jahr 2000 [55].

Zur Vorgehensweise: Zuerst haben wir die Website und insbesondere die FAQ gelesen, was sich, wie oben erwähnt, als unnötig herausstellte. Anschließend war das Tutorial von Marc Greis an der Reihe, das einen recht guten Eindruck für den Einstieg machte, gefolgt vom Handbuch.

3.4.3 Modellierung, Durchführung und Auswertung

Wie die Erfahrungen der Abteilung Verteilte Systeme zeigen, muss im Gegensatz zu allen anderen betrachteten Simulatoren bei ns-2 i.d.R. immer der Simulatorcode mit angefasst werden. Eine Trennung von Benutzer- und Kerncode scheint nicht vorhanden zu sein. Die Wiederverwendung von Komponenten ist dadurch schwierig bis gar nicht möglich.

Um ein sehr einfaches Szenario umzusetzen, kann dies komplett im OTcl-Skript erledigt werden. Möchte man allerdings über das Verdrahten von bestehenden Komponenten hinaus, kommt man nicht um C++ herum. Zudem erfordert anscheinend in manchen Situationen das Bearbeiten von *.tcl-Dateien eine Neukompilierung von ns-2 [55].

Mitgeliefert wird eine riesige Anzahl an vorgefertigten Komponenten. Die größte Schwierigkeit besteht eher darin, die für das jeweilige Szenario am besten Passende herauszufinden. Für die allermeisten Module sind eine Vielzahl an Implementierungen mit subtilen Implementationsunterschieden verfügbar.

Dass mit Tcl als Skriptsprache nicht alles einfacher wird, zeigt folgendes Beispiel: Da es Probleme bei der Kommunikation zwischen C++ und Tcl bei Arrays gibt, ist die Anzahl an Verkehrsklassen auf vier beschränkt. Um dies zu ändern, muss der C++-Code angepasst werden [56]. Zudem illustriert das gleiche Beispiel, dass Komponenten teilweise hochgradig experimentell sein können und unter Umständen auch verschiedene Codestücke aus ursprünglich völlig unterschiedlichen Quellen zusammenführen.

Für die Auswertung der Experimentergebnisse bringt ns-2 selbst nichts mit. Hier muss auf die Analyse selbst angefertigter Textlogs zurückgegriffen werden. Das Tutorial schlägt hierfür eine Xgraph-Auswertung vor [57].

Mittels einer riesigen Testsuite wird ein Großteil von ns-2 auf Fehler überprüft. Inwieweit sich hieraus eine fundierte wissenschaftliche Verifizierung ableiten lässt, ist fraglich, zumal die Entwickler selbst schreiben, dass nicht aller Code davon abgedeckt ist und jeder Forscher selbst sich vor Beginn über die verwendeten Codeteile informieren soll [58]. Dennoch wird ns-2 oft eine gute Wissenschaftlichkeit bescheinigt oder zumindest nachgesagt.

3.5 ns-3

Der Simulator ns-3 entstand 2005 im Rahmen von Forschungsprojekten an der Georgia Tech und der University of Washington. Ziel der Projekts ist es, einen Nachfolger für ns-2 zu etablieren, der eine unkomplizierte Modellierung ermöglicht, eine schönere Architektur bietet und die Realitätsnähe der Simulationen verbessert. Derzeit liegt die Software in der Version 3.15 von August 2012 vor [9]. ns-3 ist wie sein Vorgänger in C++ geschrieben, es existiert jedoch auch eine optionale Anbindungen an Python 2, das dann auch als Laufzeitumgebung installiert sein muss. Dadurch lassen sich Experimente etwas einfacher gestalten, allerdings ist die API nicht vollständig von Python ansprechbar und hält sich im Gegensatz zu sehr vielen anderen Pythonwrappern auch nicht an die De-facto-Standards bzgl. Bequemlichkeiten für Programmierer oder Namenskonventionen [59, 60]. Sie macht den Eindruck eines sehr dünnen Wrappers. Der Simulator ist nicht völlig neu geschrieben, sondern verwendet auch Komponenten von *GTNetS* und *YANS*. Die Abwärtskompatibilität zu ns-2 und damit auch Tcl wurde aufgegeben, da den Entwicklern dafür nach eigenen Angaben die Ressourcen fehlten [9]. ns-3 ist ähnlich weit verbreitet wie Omnet++ und besitzt ebenfalls eine sehr aktive Entwicklergemeinschaft. Veröffentlicht ist ns-3 unter der GNU GPLv2. Aufgrund der freien Lizenz, der aktiven Community und den regelmäßigen Veröffentlichungen kann man ns-3 als verhältnismäßig zukunftssicher einstufen.

3.5.1 Einsatzzwecke

Der Simulator ist in der Lage alle fünf Schichten zu simulieren. Von Haus aus sind Protokolle wie IEEE 802.11 (WLAN bzw. WiFi) und 802.16 (WiMAX) sowie LTE, OLSR,

AODV und viele weitere fertig implementiert [61].

Anscheinend ist der Simulator auch für Durchführung von Emulationen prädestiniert. So verwendet er stellenweise echten Code aus verschiedenen Frameworks und u.a. auch dem Linux-Protokollstapel, sodass es in naher Zukunft möglich sein soll, Anwendungen nativ über ns-3 laufen zu lassen. Zumindest ist dies lt. eigenen Angaben Gegenstand aktueller Forschung [61].

Für mobile Netze stehen Standard-Bewegungsmodelle wie das *Random Walk Model* oder das *Random Waypoint Model* zur Verfügung. Interessant dürfte für viele Nutzer auch der Import von ns-2 Bewegungsdateien [62] sein, da viele Aufzeichnungen in diesem Format vorliegen.

Über die Anzahl der simulierbaren Knoten machen die Autoren von ns-3 keine Angaben, da diese maßgeblich von der Komplexität des Modells abhängt. Simuliert werden können alle für Computernetze gängige Parameter, wie Paketverluste, Datenrate, Verzögerung, etc. Darüber hinaus sind sehr fortgeschrittene Energieverbrauchsmodelle verfügbar, die nicht nur den Energieverbrauch an sich, sondern auch verschiedene Batterietypen (wie Li-Ionen-Akkus [63]) simulieren können.

Der Fokus liegt auf der Simulation dynamischer Netze [61].

3.5.2 Bewertung der vorhandenen Dokumentation und Einarbeitung

Allgemein ist zu Beginn zu sagen, dass verschiedenerlei Informationen en masse vorliegen. Die zugrundeliegenden Konzepte und die Architektur von ns-3 sind im Handbuch gut beschrieben. Auch erläutert es auf recht tiefer Ebene die Abläufe im Simulator und empfohlenen Vorgehensweisen sowie Richtlinien bei Schreiben von neuem Simulationscode [64]. Es beinhaltet zudem einige Anleitungen mit größeren Codebeispielen. Auch ist ein äußerst ausführliches Tutorial vorhanden, allerdings deckt dies nur den C++-Teil ab [65]. Aufgrund der starken Anlehnung an die C++-API sollte es jedoch ohne großen Aufwand möglich sein, äquivalenten Pythoncode zu erzeugen, im gepackten Tarball sind zur Unterstützung nicht nur für Python einige kleine Beispiele enthalten [66]. Der Wikiseite zur Pythonanbindung ist zu entnehmen, dass es evtl. beim Einbinden von Python zu Problemen kommen kann und man etwas Handarbeit leisten muss, um diese zum Laufen zu bekommen [59]. Von der Anwenderschaft und den Entwicklern wird ein Wiki gepflegt, das weitere Informationen vorhält [67]. Darin werden die beiden FAQs für Anwender [68] und Entwickler [69] zur Verfügung gestellt, wobei v.a. Letztgenannte sehr informativ ist. Zudem ist eine große Anzahl weiterer Tutorials zu finden [70].

In der API-Dokumentation verfügt nur ein kleiner Bruchteil über erläuternde Kommentare, vom Rest ist ausschließlich die Codestruktur erfasst. Bei allen stichprobenartig überprüften Stellen waren die Zusatzkommentare sehr dürftig und nur von geringem Mehrwert [71].

Eine punktuelle Sichtung des Codes ergab, dass Codekommentare entweder sehr ausführlich oder gar nicht vorhanden sind [66].

Speziell für aktive Entwickler existiert viel weiterführendes Material [72].

3.5.3 Modellierung, Durchführung und Auswertung

Eine große Anzahl an Komponenten wird in der Standardbibliothek des Simulators bereits mitgeliefert, sodass sich einfache Szenarien schnell und unkompliziert implementieren lassen. Hierfür können sowohl C++ oder Python als Sprachen gewählt werden. Da, wie bereits erwähnt, nicht alles von Python aus ansprechbar ist, muss evtl. in speziellen Fällen das Szenario in C++ implementiert werden. Es ist allerdings unklar, in wie weit die Anbindung tatsächlich fortgeschritten ist. Eine grafische Oberfläche oder Zwischenformate sind zum Erstellen des Experiments nicht vorhanden. Eigene Komponenten sollten sich problemlos in anderen Simulationen wiederverwenden lassen.

Bei der Verwendung von C++ soll dem Handbuch folgend der `new`-Operator nicht zum Erstellen neuer Objekte verwendet werden. Stattdessen soll dies mittels den Template-Methoden `Create` bzw `CreateObject` erfolgen, je nach zugrunde liegendem Datentyp. Zur Speicherverwaltung wird ein spezieller referenzzählender Zeiger zurückgeliefert [73].

Für die Auswertung liefert ns-3 ein riesiges Tracing-System mit. Das zugrunde liegende Objekt- und Attributsystem bildet hierfür die Grundlage, über Rückruffunktionen können beliebige Daten mitgeschnitten werden. Somit ist es möglich, nahezu überall Routinen einzuhängen, um Daten abzugreifen und bspw. vor dem Protokollieren zu filtern oder das Ausgabeformat umzuschreiben [74]. Auch in diesem Fall wird Gnuplot empfohlen, um die Daten zu visualisieren. Dazu wird eine direkte Anbindung mit ausgerollt, sodass unmittelbar aus dem C++-Code Graphen und Diagramme erzeugt werden können [75]. Mittels der Qt 4-Oberfläche NetAnim können während eines Laufs aufgezeichnete Experimente im Nachhinein abgespielt und visualisiert werden [76]. Da ns-3 Paketmitschnitte im pcap-Format erzeugt, kann der Netzwerkverkehr auch in anderen Werkzeugen, wie beispielsweise Wireshark, ausgewertet werden [65].

Lt. Hersteller ist ns-3 echtzeitfähig und kann auch in Testbeds integriert oder auf virtuellen Maschinen ausgebracht werden. Mit dem mitgelieferten Echtzeitsteuerprogramm ist es möglich die Simulationszeit mit z.B. der Systemzeit synchron zu halten [77].

Nach eigenen Aussagen ist die wissenschaftliche Verifizierung schwierig, allerdings wird versucht, durch Dritte die existierenden Modelle überprüfen zu lassen. Auch stammen die meisten Modelle von Wissenschaftlern [61]. Somit bleibt es auch in diesem Fall doch wieder jedem Forscher selbst überlassen, die verwendeten Komponenten auf Tauglichkeit zu untersuchen.

3.6 OMNeT++

Omnet++ ist ein Simulations-Framework, das von András Varga an der Technical University of Budapest entwickelt wurde und, nachdem Varga die Universität verließ, in seiner eigenen Firma OpenSim Ltd. weitergeführt wird [78]. Neben Omnet++ existiert auch noch die kommerzielle Version Omnest. Omnet++ existiert bereits seit Mitte der 90er Jahre, die aktuelle Version 4.2.2 ist vom März 2012. Der Simulationskernel und die Utility-Klassen sind in C++ geschrieben [79]. Die enthaltene IDE basiert allerdings auf Eclipse, das heißt, wenn sie genutzt werden soll, wird die Java-Laufzeitumgebung

ebenfalls benötigt. Neben ns-2 gehört Omnet++ zu den verbreitetsten Simulatoren überhaupt. Allerdings sieht sich Omnet++ nicht als Netzwerksimulator an sich, sondern als Infrakstruktur für jede Art von Simulation, für die der diskrete Ereignis-basierte Ansatz geeignet ist [80]. Das Framework steht unter seiner eigenen *Academic Public License*, die für den privaten und wissenschaftlichen Gebrauch ähnliche Rechte wie die GNU GPL einräumt [81]. Omnest hingegen muss lizenziert werden und bietet dafür neben der Erlaubnis für den kommerziellen Gebrauch vor allem auch noch einen besseren Support, einen Installer und kompilierte Bibliotheken für Windows, sowie einige Zusatzfunktionalitäten (z.B. SVG-Export, SystemC-Anbindung) [82]. Zudem besitzt Omnet++ auch noch eine sehr aktive Community, und eine rege genutzte Mailingliste (in einem unserer Interviews wurde allerdings die Meinung geäußert, dass dort fast nur absolute Anfängerfragen und sehr spezielle C++-Probleme diskutiert werden). Durch den kommerziellen Support, die relativ freie Forschungslizenz und die große Community ist für Omnet++ daher die Zukunftssicherheit kein ernsthaftes Problem.

3.6.1 Konzepte und Tools

3.6.1.1 NED

Da Omnet++ keine eigenen domainspezifischen Funktionen (wie beispielsweise die Simulation von Kommunikationsnetzen) bereitstellt, werden diese im Abschnitt zu INET-MANET beschrieben.

Grundsätzlich möglich ist laut dem offiziellen Handbuch mit Omnet++ [80]:

- die Modellierung von kabelgebundenen und kabellosen Kommunikationsnetzen
- die Modellierung von Protokollen
- die Modellierung von Bediensystemnetzwerken
- die Modellierung verteilter Hardwaresysteme (Multiprozessoren etc.)
- die Validierung von Hardwarearchitekturen
- das Bewerten von Performanceaspekten komplexer Softwaresysteme
- jede Anwendung, die sich mit dem Ansatz der diskreten Ereignisse und Nachrichtenaustausch modellieren lässt

Einen eher ungewöhnlichen Weg geht Omnet++ bei Beschreibung der Netzwerke und Module. Während das Verhalten der Knoten in C++ implementiert werden, existiert zusätzlich noch die domänenspezifische Sprache NED, was für „Network Description“ steht. Der Grund für die eigene Sprache ist laut András Varga, dass XML zu wortreich und unübersichtlich ist, während Skriptsprachen (wie Tcl oder Python) nicht die Bearbeitung mit Texteditor und grafischem Modellierungstool im Roundtrip-Verfahren ermöglicht hätten [9]. Es existiert dennoch eine automatisierte Möglichkeit, Netzwerke zwischen NED und XML ohne Informationsverlust zu konvertierten, was vor allem dann nützlich ist, wenn Netzwerktopologien automatisiert bearbeitet werden sollen. Spätesten seit Omnet++ 4.0

ist NED zu einer sehr mächtigen Sprache herangereift. Es lassen sich Hierarchien bilden, um komplexere Modelle zu beschreiben, mit Interfaces lassen sich mehrere Implementierungen für ein Modul schreiben, die dann je nach Parametrisierung verwendet werden. Zudem wurde in NED auch Vererbung umgesetzt, so kann beispielsweise von einem generischen TCP-Client eine FTP-Client abgeleitet werden. Dazu kommen noch innere Typen, Annotationen mit Metadaten (z.B. das grafische Layout) und eine an Java angelehnte Paketstruktur [80].

Durch diese Spracheigenschaften werden gut geschriebene Komponenten wiederverwendbar und Komponentenbibliotheken wie INET ermöglicht.

In folgendem Beispiel wird der Aufbau von NED verdeutlicht. Es wird dabei ein Netzwerk mit mehreren Knoten aufgebaut, von denen einige mit einem Channel mit 100 Mb/s verbunden sind[1]:

```
network Network
{
  types:
    channel C extends ned.DatarateChannel {
      datarate = 100Mbps;
    }
  submodules:
    node1: Node;
    node2: Node;
    node3: Node;
    ...
  connections:
    node1.port++ <--> C <--> node2.port++;
    node2.port++ <--> C <--> node4.port++;
    node4.port++ <--> C <--> node6.port++;
    ...
}
```

3.6.1.2 Der Messagecompiler

Nachrichten werden in Omnet++ als C++-Objekte ausgetauscht. Um Arbeit zu sparen wurde mit dem Tool opp_msgc ein Messagecompiler eingeführt, mit dem die C++-Klassen aus einer einfachen Beschreibung, die sich an Strukturen aus C anlehnt, generiert wird. Eine Nachricht kann also folgendermaßen definiert werden.

```
packet AwesomePacket
{
  string message;
  int payload;
};
```

[1] Beispiel entnommen aus [80]

3.6.2 Bewertung der vorhandenen Dokumentation und Einarbeitung

Bei der verfügbaren Dokumentation setzt sich Omnet++ besonders positiv von den meisten anderen Simulatoren ab. Alleine das Handbuch besitzt ohne Anhang fast 300 Seiten. Dazu kommen noch mehrere andere Anleitungen und Screencasts. Im Web lassen sich verschiedene Tutorials finden, erwähnenswert ist hier das „TicToc Tutorial“, dass sich entlang des in Omnet++ enthaltenen TicToc-Beispiels hangelt. Außerdem existiert ein Wiki und eine sehr aktive Google Group. Es existiert auch eine Dokumentation des Simulator-Quellcodes, allerdings ist diese für den normalen Benutzer weniger interessant, da man Omnet++ ohne dessen Kenntnis problemlos nutzen kann. Hier ist also der kommerzielle Support deutlich zu spüren: Während es in den meisten untersuchten Simulatoren nicht viel mehr als einen Quickstart-Guide und den Quellcode gibt, hat man bei Omnet++ mehr Dokumentation, als man in der Regel lesen möchte.

3.6.3 Modellierung, Durchführung und Auswertung

Bei der Installation von Omnet++ benötigt das Kompilieren selbst die meiste Zeit, zuvor müssen lediglich einige in der Installationsanleitung beschriebenen Pakete installiert werden, der Quellcode heruntergeladen und entpackt und dann ein Konfigurationsskript und `make` ausgeführt werden.

Die Modellierung in Omnet++ beginnt mit dem Erstellen des Netzwerks. Dies geschieht entweder, indem man eine NED-Datei (s.o.) von Hand schreibt und/oder sie im grafischen Editor bearbeitet. Da die grafische Repräsentation und der Code selbst sofort nach einer Bearbeitung des jeweils anderen angepasst werden, kann man beim Erstellen der NED-Datei beliebig oft hin- und herwechseln. Anschließend muss die Funktionalität der beschriebenen (atomaren) Module implementiert werden. Dies geschieht in C++, man leitet von `cSimpleModule` ab und implementiert mindestens die Funktion `handleMessage()`. Zusammengeführt wird schließlich alles in einer Konfigurationsdatei (`omnetpp.ini`), in welcher das verwendete Netzwerk ausgewählt, verschiedene Durchläufe mit verschiedenen Parametern versehen und Seeds für den Zufallsgenerator angegeben werden können.

Die Simulationsdurchläufe an sich können aus der IDE gestartet werden, dann hat der Nutzer auch die Möglichkeit die Visualisierungen der Simulation zu nutzen. Dies geschieht mit der grafischen Schnittstelle `Tkenv`. Standardmäßig werden dort die Top-Level-Module angezeigt und der Nachrichtenaustausch zwischen ihnen animiert dargestellt. Hierbei handelt es sich um eine Live-Animation, das heißt, man sieht die Visualisierung während simuliert wird. Das kann für das Debugging des C++-Codes praktisch sein, bedeutet aber auch, dass man nicht zurückspulen oder einzelne Situationen wiederholt abspielen kann [9].

Für den Start im Bash-Modus gibt es diverse Möglichkeiten: mit einem selbstgeschriebenen Bash-Skript oder mit dem Hilfsprogramm `opp_runall`, das Makefiles erstellt und ausführt. Für die Verwendung in Clustern wird Xgrid (für Mac OS X), Akaroa und oProbe genannt [9].

Für die Ausgabe der Ergebnisse gibt es grundsätzlich drei verschiedene Arten von Ergeb-

nissen und Ausgabeformaten: *scalars* (einzelnen Werte), *vectors* (eine Abfolge von Werten samt Zeitstempeln) und *statistics* [80] (z.B. ein Histogramm). Diese werden zeilenorientiert als Text in Dateien geschrieben. Für die grafische Auswertung ist bereits ein Werkzeug in die IDE integriert. Mit dem Programm `scavetool` können die Ergebnisse in einige andere Formate umgewandelt werden, um sie dann zum Beispiel als CSV-Datei von GNUPlot auswerten zu lassen. Als weitere Auswertungsprogramme und -bibliotheken werden im Handbuch genannt: NumPy, SciPy, MatPlotLib, MATLAB, Octave, ROOT, Grace, verschiedene Tabellenkalkulationen und die Programmiersprache GNU R, für die bereits Pakete für die Auswertung von Omnet++-Ergebnissen existieren [80].

Darüber hinaus steht es dem Nutzer auch frei eigene Output-Manager als Plugins zu schreiben und so beispielsweise eine direkte Datenbankanbindung zu realisieren [9].

3.6.4 OMNeT++/INETMANET

INET und INETMANET sind Erweiterungen für OMNeT++, wobei INETMANET seinerseits auf INET aufbaut. Im Folgenden werden diese beiden Erweiterungen als Einheit betrachtet und als solche beschrieben.

3.6.4.1 Einsatzzwecke

INETMANET bringt viele Protokollimplementierungen auf den OSI-Schichten zwei bis fünf mit. Eine vollständige Auflistung und Dokumentation ist unter [83] zu finden. Es folgt eine kurze Übersicht der Wichtigsten:

Schicht	**Protokolle**
Sicherungsschicht	Ethernet, PPP, 802.11
Vermittlungsschicht	IPv4, IPv6
Transportschicht	TCP, UDP, SCTP

Für die Modellierung der Anwendungsschicht werden einige Klassen zur Verfügung gestellt, die verschiedene domänenspezifische Grundfunktionen bereitstellen, wie zum Beispiel `TCPBasicClientAp` `TCPGenericSrvApp` oder `PingApp`.

Mobilitätsmodelle werden reichlich mitgeliefert. Hierbei ist zwischen dynamischen Modellen und einer Anbindung an Simulatoren für Bewegungsmodelle über Tracefiles zu unterscheiden [84]:

Art	Beschreibung	Komponente
Tracefile	ANSim (Simulator)	ANSimMobility
Tracefile	BonnMotion (Simulator)	BonnMotionMobility
Tracefile	SUMO (Simulator)	TraCIMobility
Modell	Chiang's Random Walk	ChiangMobility
Modell	Kreisbewegung	CircleMobility
Modell	konstante Geschwindigkeit	LinearMobility
Modell	Gauss-Markov	GaussMarkovMobility
Modell	lineare Bewegung	LinearMobility
Modell	lineare Verteilung	LinearNodeDistributionMobility
Modell	Bewegung unter Berücksichtigung von Trägheit	MassMovement
Modell	„Mobility Model for wireless“ Body Area Networks“	MoBAN
Modell	Zufallsroute	RandomWPMobility
Modell	Rechtecksbewegung	RectangleMobility
Modell	statische Platzierung in rechteckigem Gitter	StaticGridMobility
Modell	Traktorbewegung auf einem Acker	TractorMobility

Die für die Modellierung des Energieverbrauchs nötige Infrastruktur ist ebenfalls bereits vorhanden, das heißt es existiert zumindest die Implementierung einer simplen Batterie, welche eine Kapazität hat, die über sogenannte `DrainMsg` verringert oder erhöht werden kann. Direkt verwendbare Energieverbrauchsmodelle werden jedoch nicht mitgeliefert.

3.6.4.2 Bewertung der vorhandenen Dokumentation und Einarbeitung

Die Dokumentation [83] von INETMANET ist sehr übersichtlich und gut verständlich aufbereitet. Allerdings ist eine gute Kenntnis der Funktionsweise von OMNeT++ erforderlich, da die Implementierungen sowohl anhand ihrer Integration in das gesamte Framework als auch ihrer API beschrieben werden. Die Bewegungsmodelle sind in den NED-Dateien im Quellcode gut dokumentiert. Vereinzelt finden sich Doxygen-Tags mit kurzen Erklärungen oder Verweisen auf Papers zum jeweiligen Modell in den Header- und Quelldateien und somit in der generierten Doxygen-Dokumentation, die im Download-Paket enthalten ist.

3.6.4.3 Modellierung, Durchführung und Auswertung

Die in INETMANET implementierten Netzwerkprotokolle können mit den jeweils benachbarten Schichten kommunizieren. Beispielsweise muss beim Versand eines IP-Datagramms die MAC-Adresse des Zielgeräts von Schicht drei an Schicht zwei übergeben werden. Dies ist durch das Anhängen eines Objekts mittels `setControlInfo()` an das eigentliche Datum möglich.

Viele höherschichtige Simulationskomponenten sind als zusammengesetzte Komponenten

implementiert, was die durchdachte Architektur des Gesamtsystems und die Wiederverwendbarkeit der Komponenten deutlich macht.

In Gesprächen mit Mitarbeitern der Abteilung Verteilte Systeme war hinsichtlich der Skalierbarkeit von Szenarien mit seinerzeit 100.000 Knoten die Rede. Allerdings ist dies ohne Berücksichtigung der Komplexität der simulierten Modelle oder der Hardwareausstattung kaum aussagekräftig.

3.7 PeerSim

PeerSim liegt in der Version 1.0.5 vom 29. September 2009 vor und ist zum Zeitpunkt der Untersuchung damit drei Jahre alt. Er wird von den beiden EU-Projekten BISON und DELIS gefördert und hauptsächlich an der Uni Trento in Italien betreut. Als Java-Projekt benötigt dieser Simulator die Java Runtime Engine in der Version 1.5 oder höher. Der Simulatorcode ist vom Benutzerquellcode getrennt, d.h. es muss beim Entwickeln von Simulationsszenarien nicht der Kerncode angefasst werden.

Was die Zukunftssicherheit angeht sind uns keine Lizenzänderungen in naher Zukunft bekannt. Der Simulator steht momentan unter der GPL [85]. Die Communitygröße scheint äußerst begrenzt zu sein, die Mailingliste ist als tot zu betrachten[2]. Auch ist die Motivation bei PeerSim mitzuwirken für neue Anwender eher gering, denn es soll nichts mehr in den Kern aufgenommen werden, wenn dann sind neue Bausteine als extra Zusatzpakete zu verschnüren [87]. Zum Zeitpunkt der Fachstudie sind 19 verschiedene Zusatzpakete verfügbar. Außerdem verwendet PeerSim als Versionsverwaltungssystem noch immer das etwas in die Jahre gekommene CVS.

3.7.1 Einsatzzwecke

Der Simulator ist für große Peer-to-Peer-Systeme konzipiert. Dazu gibt es einige vorgefertigte Protokolle, z.B. für *Pastry*, *Chord*, *BitTorrent* und viele weitere, die von anderen Anwendern als Zusatzpakete zur Verfügung gestellt werden [88].

PeerSim ist nur in der Lage die Transportschicht zu simulieren. Eine Information darüber, dass Tracefiles unterstützt werden, konnte von uns nicht gefunden werden. Lt. [89] (Seite 1) ist PeerSim für die Simulation von mehreren Millionen Knoten geeignet.

Welche Parameter sich simulieren lassen, ist nicht explizit vom Hersteller genannt, wir gehen allerdings davon aus, dass alle gängigen auf Schicht 4 (außer Verzögerung) simulierbar sind.

3.7.2 Bewertung der vorhandenen Dokumentation und Einarbeitung

Es gibt zwei Tutorials, die uns allerdings für den Einstieg eher ungeeignet erscheinen. Die sich uns als Neuanwender stellenden Fragen wurden kaum beantwortet. Ein Dokument,

[2] i.d.R. eine einzige E-Mail pro Jahr, ausschließlich Ankündigungen neuer Versionen [86]

aus dem die groben Konzepte hervorgehen, ist nicht vorhanden oder wurde von uns nicht gefunden. Sollte es dennoch eines geben, spricht dies dennoch nicht für die Dokumentation, da es ziemlich versteckt sein muss. Auch ist keine FAQ vorhanden. Selbst die offizielle Website zum Simulator enthält nur wenig Informationen.

Die API-Dokumentation ist als JavaDoc vorhanden und halbwegs brauchbar. Da der Simulator und damit auch die dokumentierte Codebasis recht klein ist, konnten wir uns nahezu die gesamte API-Dokumentation anschauen. Dies ist nötig und auch empfehlenswert, sollte sich der Leser mit dem Simulator näher beschäftigen wollen (vgl. nächsten Abschnitt).

Da der Quellcode nicht tiefer betrachtet wurde, lässt sich über die Güte der Codekommentare und des Code selbst keine Aussage treffen.

3.7.3 Modellierung, Durchführung und Auswertung

Die Standardbibliothek sollte für die meisten Szenarien ausreichend bestückt sein, uns fielen jedenfalls auf den ersten Blick keine fehlenden Komponenten auf.

PeerSim verwendet mit Ausdrücken aufgebohrte Konfigurationsdateien zum Einstellen verschiedener Simulationsparamter. So ist es dadurch möglich, eigene Variablen zu verwenden oder mit numerischen Werten in der Konfiguration direkt zu rechnen. Mittels einfachen Schlüssel-Werte-Paaren können Pseudozufallsgeneratoren, Protokolle, Anzahl an Experimentläufe usw. verändert werden [89]. Über eigenen Javacode können komplexere Szenarien relativ einfach erzeugt werden, z.B. wird zum Verbinden der Knoten die Klasse `WireGraph` erweitert und die Methode `wire(g)` implementiert [90]. Zur Modellierung gibt es keine grafische Oberfläche.

Alle entwickelten Komponenten können in beliebigen anderen Simulationsszenarien wiederverwendet werden [89].

Einige Dinge sind beim Entwickeln zu beachten, im Folgenden sollen einige äußerst dubiose Klassenimplementierungen und andere Fallstricke vorgestellt werden:

- `OracleIdleProtocol.clone()` liefert `this` zurück [91].
- Zum Initialisieren von `LinearIterator` ist zusätzlich der Aufruf von `reset(k)` erforderlich [92].
- Teilweise sind `max`-Parameter inklusive [93], teilweise auch exklusive [94].
- Zeiten liegen im Intervall [0..`Long.max`], allerdings ist die Einheit nirgends dokumentiert.
- Teilweise sind JavaDoc-API-Dokumentationen absolut unbrauchbar, als Beispiel soll hier die Methode `IncrementalFreq.toArithmeticExpression()` [95] dienen:

  > An alternative method to convert the object to String

  Wie die zurückgelieferte Zeichenkette aussieht, etwa Formatangaben, ist nicht beschrieben. Auch lassen sich anhand dieser Beschreibung absolut keine Rückschlüsse auf den

Rückgabewert ziehen.

- Alle Knoten werden von einem einzigen Prototypen geklont [96].
- Das Routernetzwerk ist statisch [97].

Mitgeliefert werden einige Komponenten, die verschiedene einfache Statistiken erzeugen und verwalten können, etwa `DegreeStats` um die Knotengrade im Netzwerkgraphen zu analysieren.

Zwar gibt es einige wissenschaftliche Artikel, die in der einen oder anderen Form Peersim behandeln [98], dennoch lässt sich über die wissenschaftliche Verifizierung von Peersim keine allgemeingültige Aussage treffen.

3.8 The ONE

Der Oportunistic Networking Environment Simulator (The ONE) entstand in den Forschungsprojekten SINDTN und CATDTN der Helsinki University of Technology und legt seinen Fokus auf delay-tolerant Networking (DTN). Hierbei konzentriert sich The ONE weniger auf die niedrigen Netzwerkschichten, sondern legt den Schwerpunkt vor allem auf anspruchsvolle Bewegungsmodelle in Städten. Die aktuelle Version 1.4.1 ist vom Januar 2011. The ONE ist in Java (1.6) geschrieben und benötigt somit die JVM als Laufzeitumgebung. Die Software ist unter der GPLv3 lizenziert und besitzt auch eine kleine Community. Dass diese allerdings sehr überschaubar ist, lässt sich an den lediglich neun aufgeführten Beiträgen erkennen [99]. Dafür wird die offizielle Mailingliste verhältnismäßig rege genutzt. Der Großteil der Entwicklung fand in den Jahren 2008/2009 statt, seither wurden noch zwei weitere Versionen in größeren Abständen veröffentlicht, ob zur Zeit noch an The ONE entwickelt wird, ist unklar.

3.8.1 Einsatzzwecke

Mit seinem Schwerpunkt auf delay-tolerant Networking abstrahiert the ONE die unteren Netzwerkschichten unter vereinfachenden Annahmen über Datenraten, Funkreichweiten und das daraus resultierende Transfervolumen. Modelliert werden Knoten, die einen store-carry-forward-Router darstellen. Für diese kann dann die Funkschnittstelle, der persistente Speicher, die Bewegung, der Energieverbrauch und natürlich das Nachrichtenrouting modelliert werden. Die Szenarien, die in the ONE simuliert werden, sind ausschließlich DTNs bestehend aus Fußgänger, Autos, etc., die sich auf Land- oder Stadtkarten bewegen. Das andere große Feld der DTNs, nämlich die Simulation von Satelliten-Netzwerken, wird von the ONE nicht unterstützt [100].

Großes Gewicht wird auf die Bewegungsmodellierung gelegt. Der Simulator besitzt ein Framework für die Generierung von Mobilitätsmodellen, das drei Arten von Bewegungmustern unterstützt: Random movement, map-constrained random movement und human behaviour based movement. Unter den kartenbasierten Bewegungsmodellen gibt es wiederum die völlig zufällige Bewegung entlang von Wegen oder der shortest-path-Ansatz, bei dem sich die Knoten auf dem kürzesten Wegen zwischen zufällig gewählten

oder festgelegten Punkten (Points of Interest) bewegen. Das komplexeste Modell ist das sog. Working Day Movement Model, das für the ONE entwickelt wurde. Es basiert auf der Annahme, dass Menschen an einem Werktag zu Hause schlafen, dann ins Büro fahren und abends mit Freunden ausgehen. Zusätzlich wird modelliert, dass sich Menschen oftmals in Gruppen bewegen. Dies kann eine Gruppe von Fußgängern sein, aber auch mehrere Personen in einem Auto oder einem öffentlichen Verkehrsmittel. Die einzelnen Knoten bewegen sich also im Gegensatz zu den üblichen Modellen nicht immer unabhängig von einander. Schließlich werden auch noch Gemeinschaften und soziale Beziehungen im Modell berücksichtigt, wie beispielsweise Knoten, die zusammen arbeiten, Freizeitaktitvitäten unternehmen oder zusammenleben.

Im Vergleich zur Echtzeit ist die Simulationsgeschwindigkeit auf einem aktuellen PC laut den Entwicklern 40 bis über 1.000 mal schneller. Die Geschwindigkeit hängt nicht nur von der Anzahl der Knoten ab, ebenfalls maßgeblich ist die Komplexität des verwendeten Bewegungsmodells und des Routingprotokolls. Dazu kommt noch, dass eine größere Funkreichweite zu mehr Begegnungen führt und dadurch auch mehr Ereignisse generiert werden. In Tests wurde bei der Vergrößerung der Reichweite von zehn auf hundert Meter, eine Verlangsamung um den Faktor fünf bis zehn festgestellt. Auch das Intervall, um das die Simulationszeit fortschreitet, ist einstellbar, wozu sich die benötige Dauer fast linear verhält. Die Anzahl der simulierbaren Knoten wird deshalb nur mit „mehreren tausend“ angegeben [37].

3.8.2 Bewertung der vorhandenen Dokumentation und Einarbeitung

Die externe Dokumentation von the ONE ist überschaubar: Es gibt einige Papers, die die theoretischen Hintergründe des Simulators beschreiben. In [37] wird ein grober Überblick über den Simulator und seine Konzepte gegeben, im etwas älteren Paper [101] wird dasselbe sogar noch etwas ausführlicher erklärt. In [102] wird das „Working Day Movement Model“ detailliert beschrieben und bewertet. [103] erläutert und evaluiert einige in the ONE implementierten DTN-Protokolle, ohne jedoch auf Details des Simulators einzugehen.

Das Benutzen des Simulators wird lediglich in der README-Datei beschrieben, die man zusammen mit the ONE herunterladen kann. Da die Informationen dort allerdings nicht besonders ausführlich sind, bleibt dem Nutzer für die meisten Anwendungen doch nichts anderes übrig, als sich den Quelltext bzw. die Javadoc-Dokumentation anzuschauen und dort die Funktionsweise nachzuvollziehen. Der Code an sich ist zumeist recht ausführlich kommentiert, in der Regel gibt es sowohl für öffentliche als auch für private Methoden Javadoc-Kommentare und auch weitere Zeilenkommentare. Tutorials oder Ähnliches stehen dem Nutzer allerdings nicht zur Verfügung.

3.8.3 Modellierung, Durchführung und Auswertung

Das Aufsetzen des Simulators ist unkompliziert, er ließ sich auf Anhieb kompilieren und mit einer Art Default-Modell starten. Für die Modellierung muss dann eine eigene Settings-Datei geschrieben werden. Praktisch ist dabei, dass alles, was nicht angegeben

wird, aus den Default-Settings „geerbt“ wird. Das bedeutet, dass die selbstgeschriebenen Settings-Dateien meist sehr schlank bleiben können. Sieben Routing-Module sind in the ONE bereits enthalten (First Contact, Epidemic, Spray and Wait, Direct Delivery, PRoPHET, MaxProp und ein „passiver“ Router für die Kommunikation mit externen Simulatoren) [104]. Die Modellierung lässt sich neben dem Batch-Modus auch mit einer grafischen Oberfläche durchführen, was für DTNs auch sinnvoll ist, da das Verständnis für verschiedene Größenverhältnisse deutlich gefördert wird, etwa: „Wie viel sind hundert Knoten auf zehn mal zehn Kilometern mit einem Radius von fünfzig Metern?“

Für die Auswertung existiert bereits eine ganze Reihe von Klassen, die die Simulation für die nachträgliche Datenaufbereitung vorbereitet. So gibt es beispielsweise eine Ausgabe im Graphviz-Format oder eine Reportklasse, die die Bewegungen im ns-2-Format ausgibt. Auch hier können noch eigene Ausgaben geschrieben werden, für die der existierende Code nicht geändert werden muss.

Für den Realismus der Ergebnisse gibts es wie in jedem Simulator auch Grenzen. Da die Simulation in endlichen Schritten erfolgt, kann es passieren, dass zum Beispiel ein kurzzeitiger Verbindungsabbruch nicht erfasst wird, weil er zwischen den Schritten stattfand. [101] Hier können die Schrittweiten zwar beliebig verkleinert werden, aber natürlich immer auf Kosten der Performance. Überhaupt nicht berücksichtigt werden die unteren Netzwerkschichten, was zur Folge hat, dass zwei Knoten, die sich in Reichweite befinden, immer mit der festgelegten Geschwindigkeit kommunizieren können. Weitere Faktoren, wie Hindernisse, Interferenz oder die Entfernung, die in der realen Welt großen Einfluss auf die Kommunikationsgeschwindigkeit haben, können nicht simuliert werden. Die Entwickler weisen ebenfalls darauf hin, dass sich mobile Geräte oft in Stromsparmodi befinden und daher die Kontaktzeiten im Simulator zu optimistisch sein könnten [101].

4 Zusammenfassung und Empfehlung

Unserer Ansicht nach sind die beiden Simulatoren OMNeT++ und ns-3 am besten für den breiten Einsatz in der Abteilung VS geeignet.

OMNeT++ Der teilweise kommerzielle Hintergrund der Entwicklung hinter OMNeT++ ist wohl der Hauptgrund dafür, dass sich dieser Simulator nicht nur in der Projektpräsentation sondern auch durch eine von Anfang an recht gut strukturierbare Einarbeitungsmöglichkeit auszeichnet. OMNeT++ ist von Haus aus eher ein wenig spezialisierter Allround-Simulator, der aber im Vergleich zu den anderen die ausgereifteste Architektur bietet. Somit ist eine solide Grundlage für Erweiterungen gegeben, die von der großen Community auch rege genutzt wird.

ns-3 Auch dieser Simulator erfreut sich einer großen Community. Wie auch sein Vorgänger kann ns-3 alle fünf Schichten simulieren. Die große Flexibilität beim Erzeugen der Simulationslogs kann bei großen Simulationen ein enormer Vorteil sein. Als eine weitere Besonderheit bietet ns-3 die Möglichkeit, unveränderten Programmecode innerhalb einer Simulation auszführen, womit die ohnehin fließende Grenze zur Welt der Emulation weiter aufgeweicht wird.

CloudSim Ist im Anwendungsfall speziell die Simulation von Clouds inbegriffen, kann der Einsatz von CloudSim erwogen werden. Als speziell dafür entwickelter Simulator bietet er kaum bis gar nicht verbreitete Funktionalitäten wie beispielsweise Energieverbrauchsmodelle für ein Rechenzentrum typischer Infrastruktur, die parametrisierbare Verteilung von Host-Ressourcen an virtuelle Maschinen und virtuellen Maschinen innerhalb eines Rechenzentrums oder bei föderierten Clouds auch über Rechenzentren hinweg.

GTNetS Wie bereits erwähnt scheint die Entwicklung von GTNetS eingestellt zu sein. Allgemein hinterließ GTNetS keinen allzu modernen Eindruck. Da Codeteile in ns-3 weiter Verwendung finden und ehemalige GTNetS-Entwickler auch direkt zu ns-3 beigetragen haben, ist hinsichtlich der Zukunftssicherheit Letzterer zu bevorzugen.

JiST/SWANS Die herausragensten Pro-/Contra-Argumente sind zum einen die hohe Skalierbarkeit (bis zu 1 Mio. Knoten in Ad-Hoc-Netzen), zum anderen der Widerspruch dass zwar aus Java gewohnte Programmierparadigmen wie gewohnt weiterverwendet werden können sollen, dann aber ein Bytecode-Recompiler zum Einsatz kommt, der die Java-Semantik stellenweise stark verändert. Zudem ist die letzte Veröffentlichung bereits über sieben Jahre her.

ns-2 Bereits im Rahmen der Mitarbeitergespräche zeichnete sich eine unabgesprochene Kohärenz der Meinungen über ns-2 ab. Stärkster Kritikpunkt ist mit Sicherheit die schlechte Kapselung des Simulatorcodes, beispielsweise ist die notwendige Neukompilierung des C++-Codes nach einer Änderung an Tcl-Skripten der Wartbarkeit kaum zuträglich. Die Technik neuerer Simulatoren sticht ns-2 in Bezug auf Performance inzwischen sehr deutlich aus. Zudem ist die Lernkurve in der Einarbeitungsphase vergleichsweise steil.

PeerSim PeerSim sticht nicht durch besondere Funktionsvielfalt oder ausgereifte Dokumentation hervor. Skalierbarkeit ist stark ausgeprägt, allerdings lassen sich nur rudimentäre Modelle simulieren. Trotz des Namens sollten OMNeT++ oder ns-3 für Peer-to-Peer-Simulationen besser geeignet sein.

The ONE The ONE Simulator ist ebenfalls auf ein sehr enges Anwendungsfeld beschränkt. Wenn jedoch DTNs in Städten (vor allem mit Mobiltelefonen und änlichem) simuliert werden sollen, ist dieser Simulator durchaus geeignet. Eine Besonderheit ist das Workingday-Model, dass die Bewegungmuster von Menschen, die einen Werktag in der Stadt verbringen, sehr detailliert beschreibt. The ONE ist im Gegensatz zu den „großen“ Simulatoren eher schlecht dokumentiert, dafür kann man durch die geringere Komplexität auch wieder einiges an Einarbeitungsaufwand einsparen.

A Matrix der Funktionsmerkmale

	CloudSim	**GTNetS**	**JiST/SWANS**	**ns-2**
Allgemein				
Version/Alter	3.0.1/Oktober 2012	2.0/Oktober 2008	1.0.6	2.3.5/November 2011
Programmiersprachen	Java	C++	Java	C++, OTcl
Verbreitung	Anbieter/Nutzer von Cloud-Diensten	unbekannt (keine Aktivitäten sichtbar)	unbekannt	sehr große Verbreitung (allerding abnehmende Tendenz)
Vorteile	reines Java	relative einfache Handhabung	reines Java	de-facto Standard, viele vorhandene Komponenten
Nachteile	kein typischer Netzwerk-Simulator	Projekt nicht mehr aktiv	verändert stellenweise Java-Semantiken	Modellierung ist kompliziert und sehr zeitaufwändig
Einsatzzwecke/Simulierte Elemente				
Netzwerkschichten	5	2–5	1–5	1–5
Vorgefertigte Protokolle (Auswahl)	nein	IEEE 802.3, IEEE 802.11, IPv4, Border-Gateway-Protokoll, FTP, HTTP	802.11b, IPv4, ZRP, DSR, AODV, UDP, TCP	CSMA/CD, CSMA/CA, TCP, UDP, HTTP, FTP u.v.m
Tracefiles	nein	ja	nein	ja
Skalierbarkeit	bis 1 Mio. Knoten	mehrere tausend	bis 1 Mio. Knoten	wenige tausend
Paketverlust	nein	ja	ja	ja
Nachrichtenanzahl	indirekt	ja	vermutlich	ja
Verzögerung	ja	ja	vermutlich	ja
Datenrate	ja	ja	vermutlich	ja
Kanalbitzahl	indirekt	ja	vermutlich	ja
Energieverbrauchsmodell	ja	ja	nein	ja
Netzwerktypen	ortsgebunden, aber dynamisch	statische und mobile Netzwerke	Ad-Hoc-Netze	statische und mobile Netzwerke
Dokumentation				
Tutorials	nein	nein (allerdings anschauliche Beispiele im Handbuch)	im Handbuch	ja
Konzeptbeschreibung	gut	gut	sehr gut	ja
API-Dokumentation	gut	vorhanden, aber wenig hilfreich	gut	im Handbuch enthalten und extern (veraltet und nur Codestruktur)
Codekommentierung	gut	gut	gut	gut

	CloudSim	**GTNetS**	**JiST/SWANS**	**ns-2**
FAQ	ja	ja	nein	ja (allerdings wenig hilfreich)
Verwendete Technik				
Zwischenformate	BRITE-Format (Netzwerktopologien)	–	Bytecode	–
Kapselung von System- und simulationsspezifischem Code	sehr gut	gut	sehr gut	schlecht
Standardbibliotheksgüte	gut	gut	gut	gut (da extrem umfangreich)
Art der Modellbeschreibung	Javacode, BRITE-Topologie	C++-Code (Prgrammatisch)	Javacode/Skripte & Konfigurationsdateien	haupsächlich in OTcl
Art der Auswertung	Logdateiauswertung	Logdateiauswertung (gnuplot), graphische Oberfläche)	Logdateiauswertung	Logdateiauswertung (z.B. XGraph)

	ns-3	OMNeT++/ INETMANET	Peersim	The ONE
Allgemein				
Version/Alter	3.15/August 2012	4.2.2/März 2012 (INET:2.0.0/August 2012 und INETMANET: Juli 2010)	1.0.5/September 2009	1.4.1/Januar 2011
Programmiersprachen	C++ (Anbindung an Python)	C++/NED (eigene domain specific language)	Java	Java
Verbreitung	große Community	große Community	sehr kleine, wenig aktive Community	kleine, aktive Community
Vorteile	durchdachtes Konzept, benutzerfreundlich	durchdachte Architektur, eigene IDE	reines Java, gute Skalierbarkeit	realistische Bewegungsmodelle
Nachteile	nicht kompatibel zu ns-2, Phython-Anbindung bietet nicht alle Funktionalitäten	sehr komplex, hoher Einarbeitungsaufwand	wenig Dokumentation, sehr rudimentäre Funktionalität	keine unteren Netzwerkschichten
Einsatzzwecke/Simulierte Elemente				
Netzwerkschichten	2–5	2–5	Nur Overlaynetze	Anwendungsschicht/Overlaynetze
Vorgefertigte Protokolle (Auswahl)	IEEE 802.11, IEEE 802.16, LTE, OLSR, AODV, IPv4, IPv6, UDP, TCP u.v.a.	Ethernet, PPP, 802.11, IPv4/6, TCP, UDP, SCTP u.v.a.	einige einfache Beispielnetze	Verschiedene Routing-Protokolle für DTN (First Contact, Epidemic, Spray and Wait, Direct delivery, PRoPHET, MaxProp)
Tracefiles	ja	ja	nein	ja
Skalierbarkeit	je nach Simulation und Rechner	min 100.000 Knoten	10^7 (10^5 für komplexe Protokolle – je nach verfügbarem Speicher)	„mehrere tausend“
Paketverlust	ja	ja	nein	„von Hand“ modellierbar
Nachrichtenanzahl	ja	ja	indirekt	ja
Verzögerung	ja	ja	ja	ja
Datenrate	ja	ja	indirekt	ja
Kanalbitzahl	ja	ja	indirekt	indirekt
Energieverbrauchsmodell	ja	nur Infrastruktur	nein	ja
Netzwerktypen	Internet, VANET, Sensornetze, u.v.m.	ja	Peer-to-Peer-Systeme	delay-tolerant networking (DTN)
Dokumentation				
Tutorials	ja	ja	ja	nein
Konzeptbeschreibung	ja	sehr gut	nein	vorhanden

	ns-3	OMNeT++/ INETMANET	Peersim	The ONE
API-Dokumentation	nur für wichtige Klassen ausführlich	sehr gut	mittel	gut
Codekommentierung	wenig	gut	gut	meist gut
FAQ	ja (Developer- und User-FAQ)	ja	nein	ja
Verwendete Technik				
Zwischenformate	–	BRITE, SUMO, NED	Bytecode	Bytecode
Kapselung von System- und simulationsspezifischem Code	sehr gut	sehr gut	prinzipiell vorhanden	prinzipiell vorhanden
Standardbibliotheksgüte	sehr gut	sehr gut	gut	0
Art der Modellbeschreibung	C++-Code (Programmatisch)	NED	Javacode, Konfigurationsdateien	Javacode, Szenarien im Well-Known-Text-Format (wtk)
Art der Auswertung	Logdateiauswertung, Anbindung an gnuplot	Logdateiauswertung, GUI	Logdateiauswertung mit gnuplot, einige mitgelieferte Statistikkomponenten	Graphisch, Logdatei

B Aufteilung der Bearbeitung

	Reichelt	**Stubenvoll**	**Trischler**
CloudSim	•	•	
GTNetS	•		•
JiST/SWANS			•
ns-2			•
ns-3		•	•
OMNet++/INETMANET	•	•	
PeerSim			•
The ONE		•	

C Erfasste Aufwände

	Reichelt	**Stubenvoll**	**Trischler**
Aufwand	156	151	170

Literatur

[1] CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, 2010.

[2] CloudSim Website. `http://www.cloudbus.org/cloudsim/`. abgerufen am 28.10.2012.

[3] CloudSim Versionskontrollsystem. `http://code.google.com/p/cloudsim/source/list`. abgerufen am 28.10.2012.

[4] CloudSim Diskussionsgruppe. `https://groups.google.com/forum/?fromgroups#!forum/cloudsim`. abgerufen am 28.10.2012.

[5] CloudSim FAQ. `http://code.google.com/p/cloudsim/wiki/FAQ`. abgerufen am 28.10.2012.

[6] Rimon Barr. SWANS User Guide: Design Highlights. `http://www.cs.bu.edu/brite/user_manual/node29.html`. abgerufen am 12.07.2012.

[7] GTnetS-Website: About GTNetS. `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/`. abgerufen am 02.11.2012.

[8] GTnetS-Website: GTNetS Mailing List. `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/mail_archive.html`. abgerufen am 02.11.2012.

[9] Gross Wehrle, Günes, editor. *Modeling and Tools für Network Simulation*. Springer, 2010.

[10] Using the Georgia Tech Network Simulator. `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/docs/GTNetS_manual.pdf`. abgerufen am 02.11.2012.

[11] GTnetS-Website: Downloads. `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/downloads.html`. abgerufen am 02.11.2012.

[12] gepackter GTnetS-Tarball vom 10.10.2008. `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/software/SVNCurrent/10Oct2008/GTNetS-Oct-10-08.tar.gz`. abgerufen am 02.11.2012.

[13] GTNetS-Website: Feature List. `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/feature_set.html`. abgerufen am 02.11.2012.

[14] Georgia Tech Network Simulator (GTNetS) Class Hierarchy. `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/doxygen/hierarchy.html`. abgerufen am 02.11.2012.

[15] GTnetS-Website: Frequently Asked Questions. `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/downloads.html`. abgerufen am 02.11.2012.

[16] GTnetS-Website: Report Bugs. `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/report_bugs.html`. abgerufen am 02.11.2012.

[17] EXAMPLES/demoudpworm-dbg (large file; high quality; WMP and QuickTime, 162 MB). `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/docs/worm-anim.avi`. abgerufen am 02.11.2012.

[18] GTNetS-Website: Installation and Usage. `http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/installation_usage.html`. abgerufen am 02.11.2012.

[19] Goerge F. Riley, editor. *The Georgia Tech Network Simulator*, 2003.

[20] Zygmunt J. Haas Rimon Barr. JiST/SWANS Home page. `http://jist.ece.cornell.edu/`. abgerufen am 12.07.2012.

[21] Zygmunt J. Haas Rimon Barr. JiST/SWANS Documents. `http://jist.ece.cornell.edu/docs.html`. abgerufen am 12.07.2012.

[22] Rimon Barr. SWANS User Guide: Design Highlights. `http://jist.ece.cornell.edu/swans-user/node3.html`. abgerufen am 12.07.2012.

[23] Rimon Barr. JiST User Guide: Simulation Time. `http://jist.ece.cornell.edu/jist-user/node16.html`. abgerufen am 12.07.2012.

[24] Rimon Barr. JiST User Guide: Simulation Time. `http://jist.ece.cornell.edu/jist-user/node18.html`. abgerufen am 12.07.2012.

[25] Rimon Barr. SWANS User Guide: Introduction. `http://jist.ece.cornell.edu/swans-user/node1.html`. abgerufen am 12.07.2012.

[26] Rimon Barr. JiST User Guide: Hello World. `http://jist.ece.cornell.edu/jist-user/node4.html`. abgerufen am 12.07.2012.

[27] Rimon Barr. JiST User Guide: Architecture. `http://jist.ece.cornell.edu/jist-user/node3.html`. abgerufen am 12.07.2012.

[28] Rimon Barr. JiST User Guide: Simulation Time. `http://jist.ece.cornell.edu/jist-user/node10.html`. abgerufen am 12.07.2012.

[29] Rimon Barr. JiST User Guide: Simulation Time. `http://jist.ece.cornell.edu/jist-user/node5.html`. abgerufen am 12.07.2012.

[30] Rimon Barr. JiST User Guide: Simulation Time. `http://jist.ece.cornell.edu/jist-user/node12.html`. abgerufen am 12.07.2012.

[31] Rimon Barr. JiST User Guide: Simulation Time. `http://jist.ece.cornell.edu/jist-user/node2.html`. abgerufen am 12.07.2012.

[32] Rimon Barr. JiST User Guide: Simulation Time. `http://jist.ece.cornell.edu/jist-user/node8.html`. abgerufen am 12.07.2012.

[33] Mark Fong. JiST Graphical User Interface Event Viewer. `http://jist.ece.cornell.edu/docs/040806-jist-guilog.pdf`. abgerufen am 27.10.2012.

[34] ns-2-Website. `http://www.isi.edu/nsnam/ns/`. abgerufen am 18.06.2012.

[35] *Simulating Wireless Sensor Networks with OMNeT++*, 2005.

[36] Xiaodong Xian, Weiren Shi, and He Huang. Comparison of omnet++ and other simulator for wsn simulation. In *Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on*, pages 1439 –1443, june 2008.

[37] Teemu Kärkkäinen Ari Keränen, Jörg Ott, editor. *The ONE Simulator for DTN Protocol Evaluation*. Helsinki University of Technology, 2009.

[38] SourceForge ns-2 Bugtracker. `http://sourceforge.net/tracker/?group_id=149743&atid=775392`. abgerufen am 01.11.2012.

[39] ns-2-Wiki: Roadmap. `http://nsnam.isi.edu/nsnam/index.php/Roadmap`. abgerufen am 01.11.2012.

[40] ns-2-Wiki: Projects. `http://nsnam.isi.edu/nsnam/index.php/Projects`. abgerufen am 01.11.2012.

[41] The Network Simulator: Building Ns. `http://www.isi.edu/nsnam/ns/ns-build.html`. abgerufen am 18.06.2012.

[42] Ns Limitations. `http://www.isi.edu/nsnam/ns/ns-limitations.html`. abgerufen am 18.06.2012.

[43] ns-Handbuch: Channel Class. `http://www.isi.edu/nsnam/ns/doc/node145.html`. abgerufen am 18.06.2012.

[44] ns-Handbuch: CSMA-based MAC. `http://www.isi.edu/nsnam/ns/doc/node154.html`. abgerufen am 18.06.2012.

[45] ns-Handbuch: Routing. `http://www.isi.edu/nsnam/ns/doc/node308.html`. abgerufen am 18.06.2012.

[46] ns-Handbuch: Transport. `http://www.isi.edu/nsnam/ns/doc/node383.html`. abgerufen am 18.06.2012.

[47] ns-Handbuch: Application. `http://www.isi.edu/nsnam/ns/doc/node498.html`. abgerufen am 18.06.2012.

[48] ns-Handbuch: Revised format for wireless traces. `http://www.isi.edu/nsnam/ns/doc/node186.html`. abgerufen am 18.06.2012.

[49] ns-Handbuch: Nam Trace. `http://www.isi.edu/nsnam/ns/doc/node616.html`. abgerufen am 18.06.2012.

[50] The Network Simulator ns-2: Tips and Statistical Data for Running Large Simulations in NS. `http://www.isi.edu/nsnam/ns/ns-largesim.html`. abgerufen am 01.11.2012.

[51] The Network Simulator ns-2: Frequently Asked Questions. `http://www.isi.edu/nsnam/ns/ns-faq.html`. abgerufen am 18.06.2012.

[52] ns-Handbuch: Abschnitte Commands at a glance. `http://www.isi.edu/nsnam/ns/doc/node56.html`. abgerufen am 18.06.2012.

[53] ns-2 Class Hierarchy. `http://www.isi.edu/nsnam/nsdoc-classes/hierarchy.html`. abgerufen am 01.11.2012.

[54] Tom Henderson. ns-Handbuch: Invoking OTcl Procedures. http://www.isi.edu/nsnam/ns/doc/node10.html. abgerufen am 18.06.2012.

[55] ns-Handbuch: Class EmbeddedTcl. http://www.isi.edu/nsnam/ns/doc-stable/node26.html. abgerufen am 18.06.2012.

[56] ns-Handbuch: The JoBS algorithm. http://www.isi.edu/nsnam/ns/doc/node72.html. abgerufen am 18.06.2012.

[57] Marc Greis. Marc Greis' Tutorial: Creating Output Files for Xgraph. http://www.isi.edu/nsnam/ns/tutorial/nsscript4.html. abgerufen am 18.06.2012.

[58] The Network Simulator ns-2: Validation Tests. http://www.isi.edu/nsnam/ns/ns-tests.html. abgerufen am 18.06.2012.

[59] NS-3 Python Bindings. http://www.nsnam.org/wiki/index.php/NS-3_Python_Bindings. abgerufen am 03.11.2012.

[60] Barry Warsaw Guido van Rossum. Style Guide for Python Code. http://www.python.org/dev/peps/pep-0008/. abgerufen am 03.11.2012.

[61] ns-3-Website: What is ns-3. http://www.nsnam.org/overview/what-is-ns-3/. abgerufen am 03.11.2012.

[62] NS-3-API: Ns2MobilityHelper. http://www.nsnam.org/doxygen-release/classns3_1_1_ns2_mobility_helper.html. abgerufen am 03.11.2012.

[63] NS-3-API: LiIonEnergySource. http://www.nsnam.org/doxygen/classns3_1_1_li_ion_energy_source.html. abgerufen am 03.11.2012.

[64] NS-3-Handbuch: Organization. http://www.nsnam.org/docs/manual/html/organization.html. abgerufen am 03.11.2012.

[65] ns-3-Tutorial: Introduction. http://www.nsnam.org/docs/release/3.15/tutorial/html/introduction.html. abgerufen am 03.11.2012.

[66] gepackter ns-3-Tarball 29.08.2012. https://www.nsnam.org/release/ns-allinone-3.15.tar.bz2. abgerufen am 03.11.2012.

[67] ns-3-Wiki. http://www.nsnam.org/wiki/index.php/Main_Page. abgerufen am 03.11.2012.

[68] ns-3-Wiki: User FAQ. http://www.nsnam.org/wiki/index.php/User_FAQ. abgerufen am 03.11.2012.

[69] ns-3-Wiki: Developer FAQ. http://www.nsnam.org/wiki/index.php/Developer_FAQ. abgerufen am 03.11.2012.

[70] ns-3-Wiki: HOWTOs. http://www.nsnam.org/wiki/index.php/HOWTOs. abgerufen am 03.11.2012.

[71] ns-3 Documentation. http://www.nsnam.org/docs/release/3.15/doxygen/index.html. abgerufen am 03.11.2012.

[72] ns-3-Website: Developers Overview. http://www.nsnam.org/developers/overview/. abgerufen am 03.11.2012.

[73] NS-3-Handbuch: Object Model. http://www.nsnam.org/docs/manual/html/object-model.html. abgerufen am 03.11.2012.

[74] NS-3-Handbuch: Tracing. http://www.nsnam.org/docs/manual/html/tracing.html. abgerufen am 03.11.2012.

[75] NS-3-Handbuch: Making Plots using the Gnuplot Class. http://www.nsnam.org/docs/manual/html/gnuplot.html. abgerufen am 03.11.2012.

[76] ns-3-Wiki: NetAnim. http://www.nsnam.org/wiki/index.php/NetAnim. abgerufen am 03.11.2012.

[77] NS-3-Handbuch: RealTime. http://www.nsnam.org/docs/manual/html/realtime.html. abgerufen am 03.11.2012.

[78] Omnet++ credits. http://www.omnetpp.org/index.php/documentation/3636. abgerufen am 28.10.2012.

[79] Omnet++ wiki - omnet++ in a nutshell. http://www.omnetpp.org/pmwiki/index.php?n=Main.OmnetppInNutshell. abgerufen am 28.10.2012.

[80] András Varga. Omnet++ user manual version 4.2.2. http://www.omnetpp.org/doc/omnetpp/Manual.pdf.

[81] András Varga. Academic public license. http://www.omnetpp.org/home/license. abgerufen am 28.10.2012.

[82] Inc. Simulcraft. Omnest - omnet++ comparison. http://www.omnest.com/comparison.php. abgerufen am 28.10.2012.

[83] INET Framework for OMNeT++/OMNEST. http://inet.omnetpp.org/doc/INET/neddoc/index.html. abgerufen am 01.11.2012.

[84] GitHub: INETMANET/Mobility. https://github.com/inetmanet/inetmanet/tree/master/src/mobility. abgerufen am 01.11.2012.

[85] Gian Paolo Jesi. PeerSim Introduction. http://peersim.sourceforge.net/#intro. abgerufen am 26.06.2012.

[86] SourceForge.net peersim-announce. http://sourceforge.net/mailarchive/forum.php?forum_name=peersim-announce. abgerufen am 26.06.2012.

[87] Gian Paolo Jesi. PeerSim Extras. http://peersim.sourceforge.net/#extras. abgerufen am 26.06.2012.

[88] Gian Paolo Jesi. PeerSim User code. http://peersim.sourceforge.net/#code. abgerufen am 26.06.2012.

[89] Gian Paolo Jesi, editor. *HOWTO: Build a new protocol for the PeerSim 1.0 simulator*. Universität Trento, Italien, 2005.

[90] Gian Paolo Jesi, editor. *PeerSim HOWTO: Build a topology generator for PeerSim 1.0*. Universität Trento, Italien, 2006.

[91] PeerSim API: OracleIdleProtocol. `http://peersim.sourceforge.net/doc/peersim/core/OracleIdleProtocol.html#clone()`. abgerufen am 26.06.2012.

[92] PeerSim API: LinearIterator. `http://peersim.sourceforge.net/doc/peersim/util/LinearIterator.html#LinearIterator()`. abgerufen am 26.06.2012.

[93] PeerSim API: LinearDistribution. `http://peersim.sourceforge.net/doc/peersim/vector/LinearDistribution.html`. abgerufen am 26.06.2012.

[94] PeerSim API: UniformDistribution. `http://peersim.sourceforge.net/doc/peersim/vector/UniformDistribution.html`. abgerufen am 26.06.2012.

[95] PeerSim API: IncrementalFreq. `http://peersim.sourceforge.net/doc/peersim/util/IncrementalFreq.html#toArithmeticExpression()`. abgerufen am 26.06.2012.

[96] PeerSim API: Protocol. `http://peersim.sourceforge.net/doc/peersim/core/Protocol.html#clone()`. abgerufen am 26.06.2012.

[97] PeerSim API: RouterInfo. `http://peersim.sourceforge.net/doc/peersim/transport/RouterInfo.html`. abgerufen am 26.06.2012.

[98] Peersim Publications. `http://peersim.sourceforge.net/pubs/desc.html`. abgerufen am 06.10.2012.

[99] The ONE Simulator website. `http://www.netlab.tkk.fi/tutkimus/dtn/theone/`. abgerufen am 06.10.2012.

[100] The ONE Simulator FAQs. `http://www.netlab.tkk.fi/tutkimus/dtn/theone/qa.html`. abgerufen am 20.10.2012.

[101] Ari Keränen, editor. *Opportunistic Network Environment simulator*. Helsinki University of Technology, 2008.

[102] Jouni Karvo Frans Ekman, Ari Keränen and Jörg Ott, editors. *Working Day Movement Model*. Helsinki University of Technology, 2008.

[103] Jouni Karvo and Jörg Ott, editors. *Time Scales and Delay-Tolerant Routing Protocols*. Helsinki University of Technology, 2008.

[104] The ONE readme file. Im Simulator enthaltene readme.txt. abgerufen am 27.10.2012.

D Erklärung

Ich versichere, dass ich diese Arbeit selbständig verfasst und nur die angegebenen Hilfsmittel verwendet habe.