HUMBOLDT-UNIVERSITÄT ZU BERLIN

INSTITUT FÜR INFORMATIK

SYSTEMARCHITEKTUR

Performance of DHT over WMN

Anselm Ringleben

Gutachter: Prof. Dr. Jens-Peter Redlich

Betreuer: Mathias Kurth

01.03.2009

Inhaltsverzeichnis

1 Einleitung 3

1.1 Drahtlose Maschennetzwerke 3

1.2 Peer-to-Peer Systeme 4

1.2.1 Chord 5

1.2.2 Pastry 7

2 Methodik zum Vergleich von DHTs 8

2.1 Bestehende Ansätze 9

2.2 Methodik für drahtlose Umgebungen 9

3 Simulation und Auswertung 11

3.1 JiST/SWANS 11

3.2 Implementierung 12

3.3 Parameter 13

3.4 Durchführung der Messungen 13

3.5 Auswertung der Simulation 14

3.5.1 Ergebnisse für Chord 14

3.5.2 Ergebnisse für Pastry 15

3.5.3 Vergleich der Ergebnisse 16

4 Zusammenfassung und Ausblick 17

5 Literatur 18

5.1 Websites 19

6 Anhang 19

6.1 Quellcode 19

6.2 Liste der Zufallswerte 19

6.3 Parameter 20

Abbildungsverzeichnis

Abbildung 1: Ein drahtloses Maschennetzwerk 4
Abbildung 2: Ein 6-Bit Chord Schlüsselraum 6
Abbildung 3: Ein 4-bit Pastry Schlüsselraum 7
Abbildung 4: Der SWANS-Simulator 11
Abbildung 5: Chord Ergebnisse....14
Abbildung 6: Pastry Ergebnisse 15
Abbildung 7: Chord und Pastry Ergebnisse 16

1 Einleitung

Ziel dieser Arbeit ist es, eine Methodik zum Vergleich der Leistungsfähigkeit von verteilten Hashtabellen [engl.: distributed hash table (DHT)] in drahtlosen Maschennetzwerken [engl.: wireless mesh network (WMN)] vorzustellen und diese mittels eines WMN- Simulators beispielhaft für zwei DHTs zu überprüfen.

Beide Themengebiete – DHTs und WMNs – haben in den letzten Jahren an Bedeutung gewonnen, wobei bisher nur wenige Ansätze beide miteinander in Verbindung gebracht haben, z. B. [10] in dem die DHT mit positionsabhängigem Routing über lokal naheliegende Gruppen anstelle einzelner Knoten aufgebaut wird. Die Problematik drahtloser Netzwerke im Vergleich zu kabelgebundenen Netzen, für die bisherige DHTs entworfen wurden, besteht in der Tatsache, dass das Medium unter den Knoten einer Kollisionsdomäne geteilt werden muss. Jeder Knoten muss zum Senden mit den anderen Knoten um das Medium konkurrieren. Dabei kann das Medium nur bis zu einem bestimmten Punkt ausgelastet werden, bis das es saturiert ist. Ab diesem Punkt verhält sich das gesamte System innerhalb der Kollisionsdomäne chaotisch. Hinzu kommt das Problem der Interferenz, welche zu Störungen bei der Übertragung führen kann. Das kann die Knoten zeitweise verhindern, Daten erfolgreich zu senden. Dies wirkt sich nachteilig auf die verfügbare Bandbreite der einzelnen Knoten aus und somit auch auf die Fähigkeit, Daten miteinander verzögerungsfrei auszutauschen. Beim Weiterleiten von Anfragen einer DHT wirkt sich dies auf die parallele Verarbeitung von Anfragen innerhalb der Kollisionsdomäne aus.

Aufgrund dieser Schwierigkeiten ist eine Überprüfung der Auswirkungen der veränderten Umgebung in WMNs auf DHTs sinnvoll. Interessant ist dabei ein Vergleich bestehender Implementierungen für drahtgebundene Umgebungen, um Aussagen treffen zu können, ob die betrachteten DHTs in drahtlosen Umgebungen einsetzbar sind und welche besser geeignet ist, sofern eine solche Aussage möglich ist.

1.1 Drahtlose Maschennetzwerke

Ein drahtloses Maschennetzwerk (Abbildung 1)[9] besteht aus verschiedenen Geräten, die kabellos miteinander kommunizieren. Die Knoten dieses Netzwerks, die willkürlich angeordnet sein können, werden dabei in zwei Kategorien unterteilt: Mesh Nodes und Client Stations. Mesh Nodes bilden das Maschennetzwerk und erkennen automatisch ihre Nachbarn. Sie leiten mittels Multi-Hop-Routing Pakete an andere Mesh Nodes oder Client Stations weiter und können dem Maschennetzwerk eine Verbindung zu anderen Netzwerken oder dem Internet zur Verfügung stellen. Client Stations dagegen sind nicht direkt am Maschennetzwerk beteiligt, da sie keine Pakete weiterleiten und keine Verbindung zu anderen Netzwerken bereitstellen, sondern das Maschennetzwerk benutzen.

Durch diese dezentralisierte und flexible Struktur von Maschennetzwerken, deren Netzwerktopologie sich schnell und unvorhersehbar ändern kann, ist eine zentrale Steuerung und Konfigurati-

on unpraktisch und fehleranfällig. Daher wird in Maschennetzwerken häufig Selbstorganisation benutzt. Unter diesem Gesichtspunkt bieten sich auch DHTs an, um z. B. Inhalte in WMNs zur Verfügung zu stellen. Die Autoren von [9] benutzen eine DHT zur verteilten Realisierung des Dynamic Host Configuration Protocols, des Address Resolution Protocols und zur Gateway-Entdeckung und -Auswahl.

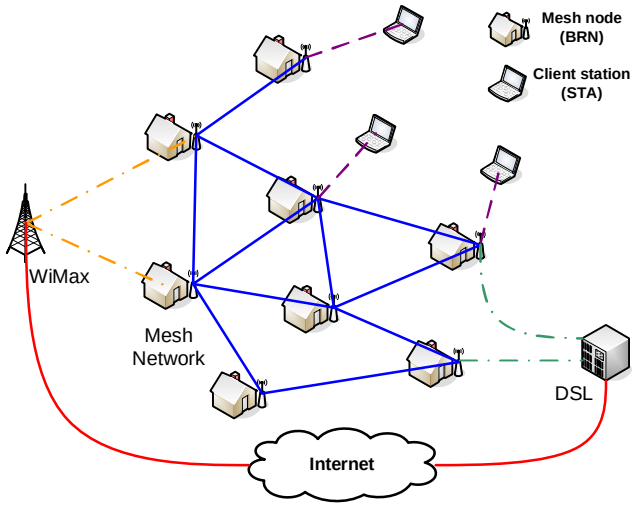

Abbildung 1: Ein drahtloses Maschennetzwerk[1]

Heutzutage spielen drahtlose Maschennetzwerke als Community-Netzwerke eine immer stärker an Bedeutung gewinnende Rolle, da sie über größere Flächen, z. B. in Städten, eine Internetverbindung zur Verfügung stellen können. Weiterhin ermöglichen sie andere Anwendungen, wie z. B. geteilte Internetzugänge, Verteilung von Ressourcen, etc.

1.2 Peer-to-Peer Systeme

Ein Peer-to-Peer-System (P2P), oder auch Overlay-Netzwerk, lässt sich als virtuelles Netzwerk beschreiben, das auf bereits existierender Infrastruktur aufsetzt und einen eigenen unabhängigen Adressraum mit eigenen Wegewahlverfahren besitzt. Dabei agieren die teilnehmenden Peers gleichberechtigt sowohl als Server als auch als Client. P2P-Algorithmen stellen die Verbindung für eine große Gruppe physikalisch verteilter Peers mit gemeinsamen Interessen her, z. B. Speicherung und Verteilung von und Suche nach Informationen und Daten. Das P2P-Protokoll regelt dabei wer mit wie vielen Nachbarn im Overlay eine Verbindung aufrechterhält, wie oft diese Nachbarn und welche Peers bei Suchanfragen kontaktiert werden. P2P-Systeme lassen sich in strukturierte und unstrukturierte Systeme unterteilen. Während in unstrukturierten Overlays die Peers Verbindungen zu zufälligen anderen Peers aufrechterhalten, definiert das Protokoll strukturierter Systeme die Beziehungen zwischen den Peers.

[1] [9], Fig. 1, Seite 2

Unstrukturierte P2P-Systeme sind völlig dezentralisiert und bieten somit keine zentrale Komponente, deren Ausfall das Netzwerk lahmlegen könnte. Die Suche nach gespeicherten Informationen oder Dateien wird meist mittels einfachem Flooding oder Random-Walk-Algorithmen implementiert. Dies benötigt einen geringen Speicheraufwand für einzelne Knoten, erzielt allerdings eine schlechte Suchperformance und garantiert keinen Erfolg oder eine Antwort, auch wenn die gesuchte Information im Overlay gespeichert ist.

Strukturierte P2P-Systeme teilen sich ein gemeinsames Funktionsprinzip. Jedem teilnehmenden Peer wird ein eindeutiger *m*-Bit Schlüssel zugewiesen, z. B. durch das Anwenden einer Hashfunktion auf die IP-Adresse. Um sich in die Struktur des Systems einzugliedern, kontaktieren die Peers ihre *r* nächsten Knoten gemäß einer Metrik über die Schlüssel für das jeweilige P2P-Protokoll. Zusätzlich bauen die Knoten Verbindungen zu weiter entfernten Peers als Abkürzungen für die Suche im Overlay auf. Um die für eine Datei verantwortlichen Peers zu ermitteln, wird die Information auf den gleichen Schlüsselraum abgebildet, so dass der gemäß Metrik nächste Knoten dafür zuständig ist. Aufgrund dieses Hash-basierten Ansatzes werden strukturierte P2P-Netzwerke häufig auch als DHTs bezeichnet. Durch die Struktur solcher DHTs kann garantiert werden, dass Anfragen immer beantwortet werden, sei es durch die erfolgreiche Übertragung der Datei oder durch eine entsprechende Nachricht, dass die gesuchte Datei nicht vorhanden ist. Während DHTs die negativen Eigenschaften von unstrukturierten P2P-Systemen nicht besitzen, müssen sie auch unter Churn, dem kontinuierlichen Prozess neu hinzukommender oder ausfallender Peers, funktionsfähig bleiben.

In den folgenden zwei Abschnitten werden zwei spezielle Implementierungen von DHTs detaillierter vorgestellt, die für die Arbeit verwendet wurden: Chord und Pastry.

1.2.1 Chord

Chord [1], [3] ist ein strukturierter Peer-to-Peer-Algorithmus, der beschreibt, wie neue Peers dem Overlay beitreten, wo Dateien gespeichert werden und wie diese von anderen Peers abgerufen werden können. Chord ordnet die Peers logisch in einem sogenannten Chord-Ring auf Modulo-Basis an, so dass die Schlüssel in aufsteigender Richtung im Uhrzeigersinn angeordnet werden. Der erste Knoten, welcher auf Peer *z* folgt, wird Nachfolger *s* von *z* genannt. Der nächste Knoten gegen den Uhrzeigersinn wird Vorgänger *p* von *z* genannt. Alle Dateien, deren Schlüssel zwischen Peer *z* und dessen Nachfolger *s* fällt, werden bei Knoten *s* gespeichert. Abbildung 2 zeigt einen 6-Bit Chord-Ring mit zehn Knoten und sieben Datenobjekten.

Um einem Chord-basierten Overlay-Netzwerk beitreten zu können, muss Peer *z* einen beliebigen schon am Overlay teilnehmenden Knoten *x* kennen. Dies ist ein generelles Problem in P2P-Netzwerken und wird meist durch eine Liste bekannter Peers, oder durch einen zentralen Bootstrap-Knoten gelöst (dies ist der Ansatz in dieser Arbeit). Peer *x* kann nun den Schlüssel des direkten Nachfolgers für den beitretenden Peer *z* über den Such-Algorithmus identifizieren und Peer *z* mitteilen. Sobald Peer *z* den Schlüssel von seinem direkten Nachfolger *s* kennt, benachrichtigt er diesen, dass er der neue direkte Vorgänger ist. Sobald der alte Vorgänger *p* von *s* diesen während

der periodischen Stabilisierung kontaktiert, teilt *s* Peer *p* seinen neuen Nachfolger *z* mit. Peer p teilt nun *z* mit, dass *z* der Vorgänger von *p* ist. Knoten *p* ist nun vollständig in den Chord-Ring integriert. Prinzipiell würde ein Peer seine Vorgänger und Nachfolger vor dem Verlassen der DHT in ähnlicher Weise kontaktieren, jedoch geschieht dies in realen Umgebungen selten.

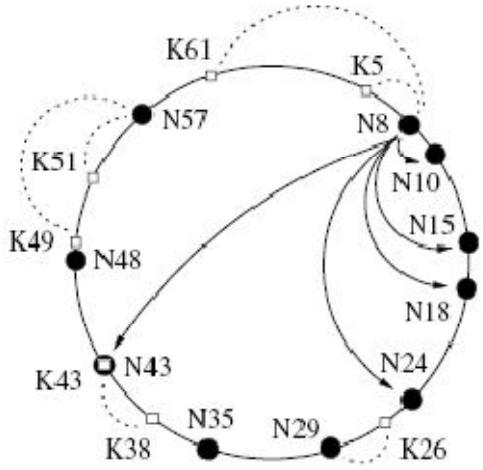

Abbildung 2: Ein 6-Bit Chord Schlüsselraum. Gepunktete Linien zeigen welche Knoten für welche Schlüssel verantwortlich sind. Schwarze Linien zeigen die Finger von Knoten N8 an.[2]

Um die Stabilität unter Churn gewährleisten zu können, führt Chord regelmäßig Maintenance-Routinen durch. Dazu speichert jeder Peer die Schlüssel der ersten *r* Nachfolger im Uhrzeigersinn in der Successorlist. Dadurch kennt jeder Peer die nächsten *r* – 1 Knoten im Ring, sobald einer der Nachfolger ausfällt. Um Änderungen in der Nachbarschaft erkennen zu können, kontaktiert jeder Peer seinen direkten Vorgänger *p* und Nachfolger *s* und stimmt seine Successorlist mit diesen ab. Sollte ein Peer nicht auf diese Anfrage reagieren, wird er aus der Liste entfernt und der nächste Peer kontaktiert. Das Chord-Protokoll spezifiziert keine Mechanismen, um die gespeicherten Dateien redundant im Ring abzulegen.

Die gesamte Funktionalität von Chord basiert auf seinem Suchalgorithmus. Um eine Datei im Overlay zu speichern, sucht der Peer nach dem ersten Knoten, dessen Schlüssel größer als der Schlüssel der abzuspeichernden Datei ist. Zum Abrufen der Datei zu einem späteren Zeitpunkt wird dieselbe Suchroutine benutzt. Bei einer Anfrage leitet jeder Peer diese an seinen Nachfolger weiter, bis der verantwortliche Peer erreicht wurde. Die Antwort wird anschließend direkt an den ursprünglichen Peer gesendet. Um die Suche abzukürzen, speichert jeder Peer sogenannte Finger in der Fingerlist. Der *i*. Eintrag in dieser Fingerlist enthält den Schlüssel des ersten Knotens, dessen Schlüssel mindestens $2^i - 1$ auf dem Chord-Ring entfernt ist. Somit hat Peer z mit Schlüssel id_z die Finger $id_z + 2^i - 1$, für $i = 1$ bis m, wobei m die Anzahl der Bits des Schlüssel ist. Bei der Suche nach einer Datei, sendet Peer *z* die Anfrage an den Finger, dessen Schlüssel direkt vor dem Schlüssel der Datei liegt. Sollte der Finger die Anfrage nicht beantworten können, leitet er die Anfrage entsprechend weiter. Auf diese Weise können Anfragen mit $O(log_2(n))$ Nachrichten beantwortet werden.

[2] [3], Fig. 8.1, Seite 96

1.2.2 Pastry

Ähnlich zu Chord speichert Pastry [3] die Schlüssel in einem Ring. Dabei werden die 128-Bit Schlüssel als Sequenz von Zahlen zur Basis 2^b so interpretiert, dass b fortlaufende Bits eine Zahl repräsentieren. Um die Struktur des Overlay zu etablieren, unterhält jeder Peer Verbindungen zu drei verschiedenen Typen von Nachbarn. Diese werden benutzt um die Routingtabelle, das Leafset und das Neighborhoodset aufzubauen.

Die Routingtabelle eines Peers enthält bis zu $\lfloor 128/b \rfloor$ verschiedene Reihen, wobei jede Reihe (2^b-1) Einträge besitzt. Die *i*. Reihe enthält nur Knoten, deren Schlüssel die ersten *i* Zahlen mit dem speichernden Peer teilen, allerdings in der *i* + 1-ten abweichen. Die tatsächliche Größe der Routingtabelle hängt von der Anzahl der Knoten im Overlay ab. Das Leafset ist eine symmetrische Liste der *k* numerisch nächsten Peers bezüglich ihrer Schlüssel. Also speichert jeder Peer die jeweils $k/2$ numerisch nächsten Knoten mit kleinerem und größerem Schlüssel. Diese Liste wird zur Stabilisierung des Overlay und für den letzten Schritt der Suche verwendet. Das Neighborhoodset enthält die *m* Peers, die physikalisch nah zum speichernden Knoten sind. Diese Liste ist absolut unabhängig von den Schlüsseln der Peers und wird nicht für Routing-Zwecke benutzt. In Pastry sind die Peers für Schlüssel verantwortlich, deren eigener Schlüssel numerisch am nächsten liegt. Abbildung 3 zeigt einen 4-bit Schlüsselraum mit fünf Knoten und sechs Datenobjekten.

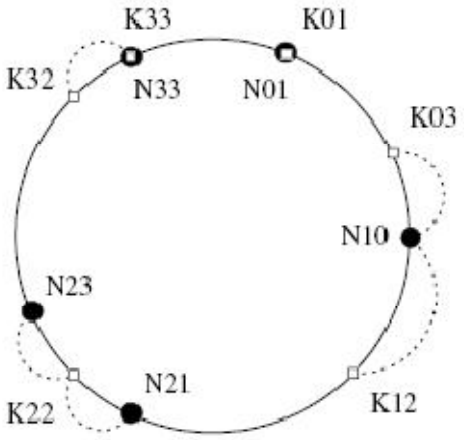

Abbildung 3: Ein 4-bit Pastry Schlüsselraum mit sechs auf fünf Knoten verteilten Schlüsseln. Numerische Nähe ist eine mehrdeutige Metrik, um Knoten Schlüssel zuzuordnen, wie für Schlüssel K22 dargestellt.[3]

Der Routing-Prozess in Pastry basiert sowohl auf der Routingtabelle und dem Leafset. Bei der Suche nach einem Schlüssel überprüft Peer *p* zunächst, ob der gesuchte Schlüssel durch das Leafset abgedeckt wird. In diesem Fall kann er die Anfrage an den numerisch nächsten Knoten im Leafset weiterleiten, der dann die Anfrage beantworten können sollte. Andernfalls leitet der Peer die Anfrage an den Knoten in der Routingtabelle, dessen Schlüssel um mindestens eine Zahl näher am gesuchten Schlüssel ist als der eigene. Um den gesuchten Schlüssel so schnell wie möglich zu erreichen, sollte den Knoten auswählen, dessen Schlüssel das größte gemeinsame Präfix zum

[3] [3], Fig. 8.2, Seite 100

gesuchten Schlüssel aufweist. Beim Aufbau der Routingtabelle, bevorzugt Pastry Peers mit guter Latenz. Für jede Zeile wählt der Peer den physikalisch nächsten Knoten aus allen numerisch für die Zeile geeigneten Knoten aus. Ein der DHT beitretender Knoten initialisiert seine Routingtabelle mit passenden Einträgen von anderen Peers. In periodischen Abständen kontaktiert er die Knoten seines Neighborhoodsets, um mit den physikalisch nahen Knoten unabhängig von ihrem Schlüssel Informationen auszutauschen. Mit diesen Knoten kann er dann geeignete Einträge mit niedrigerer Latenz austauschen. Die Wahl des Parameters *b* stellt einen Trade-Off zwischen der Größe der Routingtabelle $\lceil log_{2^b} n \rceil * (2^b-1)$ und der durchschnittlichen Anzahl der für eine Suche benötigten Hops $\lceil log_{2^b} n \rceil$ da. Global betrachtet kann die erfolgreiche Übertragung einer Nachricht garantiert werden, solange kein Knoten sein gesamtes Leafset von kleineren oder größeren Knoten verliert.

2 Methodik zum Vergleich von DHTs

Das Protokoll von DHTs beinhaltet neben Suche, Speichern und Aufnahme von neuen Knoten auch den regelmäßigen Austausch von Informationen, um die Routing-Tabellen aktuell zu halten. Hierbei wird der eigene Informationsstand den Nachbarn mitgeteilt, oder mit kurzen Anfragen überprüft, ob weitere Knoten noch im Netzwerk erreichbar sind. Grob lässt sich der Zustand eines Knotens in einer DHT in fünf Zustände unterteilen:

- Verbindungsaufbau: Der Knoten versucht, einer bereits bestehenden DHT beizutreten
- Betrieb: Der Knoten ist korrekt in die DHT eingegliedert und kann Anfragen bearbeiten oder stellen; dabei wird regelmäßig Kontakt mit anderen Knoten gehalten
- Verbindungsabbau: Der Knoten meldet sich von der DHT ab
- Fehlerfall: Es ist ein Fehler aufgetreten, der die korrekte Funktionsweise verhindert. Es kann einige Zeit dauern, bis der Knoten selbst oder andere Knoten diesen Fehler bemerken
- Zusammenführung: die Vermischung von zwei separaten DHT-Partitionen zu einer einzigen, wobei sich Zuständigkeiten der Peers für gespeicherte Werte ändern könnten

Dabei kann es durch (Verbindungs-)Fehler für Knoten notwendig werden, der DHT erneut beizutreten. Außerdem muss sich ein Knoten nicht unbedingt korrekt abmelden. Dies würde erst nach einiger Zeit den verbleibenden Knoten auffallen und dann zu einer Umstrukturierung der DHT führen.

Aufgrund der großen Unterschiede bezüglich der Implementierung zwischen verschiedenen DHTs und ihrer Charakteristika des Netzwerkverhaltens ist eine Analyse all dieser Zustände zu umfangreich und sollte separat behandelt werden. Diese Arbeit beschränkt sich lediglich auf die Betrachtung des Betriebszustands, da während der Initialisierung Unterschiede der Implementierungen und unterschiedliche Aufbaugeschwindigkeiten für unfaire Vorteile sorgen könnten. Gleiches gilt auch für den Fehlerfall oder die Zusammenführung von separaten DHT-Partitionen.

2.1 Bestehende Ansätze

Bisherige Arbeiten zum Thema der Evaluierung und der Vergleichbarkeit von DHTs haben sich hauptsächlich auf drahtgebundene Netzwerke beschränkt. Die Arbeiten [5], [6] benutzen zu diesem Zweck entweder Lookup Hopcount (die Anzahl der notwendigen Sprünge, um einen bestimmten Schlüssel anzufragen) oder Lookup Latenz (die Dauer der Anfrage nach einem bestimmten Schlüssel) und Größe der Routing Tabellen in statischen Netzwerken. Beide Werte, Lookup Hopcount und Lookup Latenz vernachlässigen dabei allerdings die benutzte Bandbreite pro Knoten, welche gerade in drahtlosen Umgebungen eine knappe Ressource wegen der Teilung des Mediums darstellt. Die Größe der Routing Tabellen spielt für Implementierungen ohne starke Speicherbeschränkungen keine große Rolle. Allerdings kann der Aufwand zur Aktualisierung der Routing Tabellen je nach Größe variieren. Dies trägt dann zu der benutzten Bandbreite pro Knoten bei.

Die Autoren von [4] schlagen zu diesem Zweck einen zusammengefassten Wert in sich stark ändernden Umgebungen vor: die Kosten-Performance-Abschätzung. Kosten werden dabei durch den Verbrauch der wichtigsten Ressource in Netzwerkumgebungen (der verfügbaren Bandbreite) in übertragenen Bytes pro Knoten pro Sekunde gemessen. Performance wird mittels Lookup Latenz bestimmt. Da der Hopcount zur Latenz beiträgt, wird der Hopcount nicht separat betrachtet. Dieses Maß wird auch für diese Arbeit übernommen. Dabei wird in dieser Arbeit Churn außen vor gelassen, um diese Störquelle auszuschließen. Allerdings reicht dieser Wert alleine nicht aus, um das Verhalten von DHTs in drahtlosen Umgebungen zu erklären. So gehen gerade in drahtlosen Umgebungen viele Pakete durch Übertragungsfehler verloren, was erhebliche Auswirkungen auf die Funktionsweise der DHT haben kann.

Die Autoren von [9] benutzen das Verhältnis der erfolgreich gesendeten zu insgesamt gesendeten Paketen (Zuverlässigkeit), die Anzahl der Pakete pro Anfrage, die Anzahl der Paket-Kollisionen auf dem drahtlosen Medium und die Reaktionszeit. Von diesen Werten wird die Zuverlässigkeit übernommen, um eine Einschätzung über die Auswirkungen der drahtlosen Übertragung auf Anfragen zu bekommen.

2.2 Methodik für drahtlose Umgebungen

Ziel der Arbeit ist es, einen geeigneten Ansatz vorzustellen, mit dem ein Vergleich der Leistungsfähigkeit verschiedener DHTs in drahtlosen Maschennetzwerken möglich ist. Für ein Maß zum Vergleich der Leistungsfähigkeit von DHTs werden in dieser Arbeit drei Werte benutzt:

- **Kosten-Performance-Abschätzung:** Welche durchschnittliche Lookup Latenz in Millisekunden wird bei der durchschnittlich verbrauchten Bandbreite pro Knoten im Beobachtungszeitraum erreicht?
- **Zuverlässigkeit:** Wie viele der Anfragen wurden im Beobachtungszeitraum im Durchschnitt erfolgreich beantwortet?

- **Netzlast:** Wie hoch ist die durchschnittliche Auslastung des Mediums im Beobachtungszeitraum innerhalb der Kollisionsdomäne?

Für die Kosten-Performance-Abschätzung werden bei der Lookup Latenz alle erfolgreich beantworteten Anfragen gewertet. Sobald eine Anfrage wegen Übertragungsfehlern verworfen wird, geht dies nicht in die Lookup Latenz mit ein. Bei der durchschnittlich verbrauchten Bandbreite pro Knoten werden alle Pakete betrachtet, die vom Application Layer (DHT und Anwendung) an den Adapter gesendet werden, also auch Maintenance-Pakete. Dazu wird die Größe der Pakete in Byte für jeden Knoten protokolliert. Der präsentierte Wert ergibt sich dann aus dem Durchschnitt der gesendeten Bytes pro Knoten dividiert durch die Anzahl der betrachteten Sekunden. Für die Zuverlässigkeit werden allein Anfragen betrachtet. Dazu werden alle versendeten Anfragen und alle Antworten protokolliert. Maintenance-Traffic spielt bei diesem Wert keine Rolle. Für die Netzlast protokolliert jeder Knoten die betrachtete Auslastung des Mediums auf dem Physical Layer. Dabei liegen alle Knoten innerhalb einer Kollisionsdomäne. Die präsentierten Werte ergeben sich aus dem Mittelwert über alle Simulationsläufe. Somit zählen zu diesem Wert nicht nur die Daten aus dem Application Layer, sondern auch alle Daten der niedrigeren Schichten. Für eine Unterteilung der Netzlast in Maintenance-Traffic und Datenverkehr der DHT wird die durchschnittliche Auslastung des Mediums auch ohne Anfragen an das Overlay gemessen.

Das Anfrageprofil wurde so gewählt, dass alle Knoten in regelmäßigem Abstand Anfragen für zufällige Schlüssel an die DHT stellen. Dies gewährleistet eine gleichmäßige Auslastung des Mediums und zeigt zugleich das Maximum der gleichzeitig aktiven Knoten auf, sobald es zu einer Überlastung des Mediums kommt. Dies mag zwar nicht Anfragemuster in der Realität wiederspiegeln, jedoch ist eine Vergleichbarkeit komplett zufallsbasierter Muster nicht zwangsläufig möglich. So könnte es z. B. durch eine übermäßige Häufung von Anfragen innerhalb eines kurzen Zeitraums zu einer Überlastung des Mediums kommen, was nur die Aussage zuließe, dass dies Muster nicht für die drahtlose Umgebung geeignet ist.

Da die verfügbare Bandbreite unter allen Knoten innerhalb einer Kollisionsdomäne aufgeteilt werden muss, und alle Knoten das gleiche Anfrageschema besitzen, spielt hier die Anzahl (Dichte) der Knoten innerhalb einer Kollisionsdomäne eine entscheidende Rolle. Je mehr Knoten der DHT beitreten, desto größer ist die Netzlast. Daher lässt die Netzlast eine direkte Aussage über die jeweils vertretbare Dichte für die DHTs zu. Ergänzend dazu liefert die Betrachtung der Zuverlässigkeit eine Aussage, welche Auswirkungen die Netzlast auf das korrekte Funktionieren der DHT hat. Kosten und Performance lassen direkte Aussagen zu, welche Eigenschaften eine DHT besitzt und ermöglichen dahingehende Vergleiche. Dies erleichtert Entscheidungen über die Wahl einer bestimmten DHT für den Einsatz unter vorher bekannten Rahmenbedingungen.

Aufgrund des geteilten Mediums und den damit entstehenden Effekten werden bei der Auswertung nur Messungen im nicht-saturierten Bereich miteinander verglichen. Im nicht-saturierten Bereich können prinzipiell alle Anfragen abgearbeitet werden, sodass sich in der Kosten-Performance-Abschätzung eine charakteristische Kurve ergibt. Im Falle eines saturierten Mediums erhöhen sich schlagartig die Queuing-Delays und die Knoten können ihre Pakete nicht mehr senden. Dies Verhalten lässt keine Rückschlüsse über das Verhalten der DHTs zu, sondern beschreibt lediglich die Eigenschaften des drahtlosen Mediums im saturierten Betrieb.

Die Zusammenstellung dieser Messwerte deckt somit Kosten und Leistungsfähigkeit ab, mit denen eine Abschätzung in drahtgebundenen Umgebungen möglich ist. Für eine Einschätzung der Auswirkungen der drahtlosen Umgebung auf die Funktionsweise von DHTs werden die Netzlast und die Zuverlässigkeit herangezogen. Weitere Messwerte würden beide Bereiche ergänzen, jedoch keinen zusätzlichen Informationsgewinn bedeuten.

3 Simulation und Auswertung

Als Simulationsumgebung wurde der „Berlin RoofNet Simulator“ [13] gewählt, der auf dem „Java in Simulation Time / Scalable Wireless Adhoc Network Simulator“ (JiST/SWANS) [11] basiert und diesen um gewisse Funktionalität erweitert.

3.1 JiST/SWANS

Java in Simulation Time (JiST) ist ein diskreter Ereignissimulator [engl.: discrete-event simulator], der auf einer herkömmlichen Java Umgebung aufsetzt. Er basiert auf einem Ansatz, der traditionelle Systeme und sprachbasierte Simulatoren zusammenführt. Dabei ist Transparenz ein Hauptaugenmerk. Simulationsprogramme müssen für JiST nicht in einer Simulator-spezifischen Sprache geschrieben werden und benötigen keine speziellen Systemaufrufe. Stattdessen konvertiert JiST eine existierende Java Virtual Machine in eine Simulationsplattform durch Umschreiben des Bytecodes zur Laufzeit. Um die Objektorientierung von Java auf Simulationen anzupassen, erweitert JiST Java um Entities, die logische Anwendungsobjekte kapseln und zeitlich unabhängige Simulationskomponenten anzeigen. Aufrufe an Entities repräsentieren dabei Simulationsereignisse. Diese Aufrufe sind nicht-blockierend und werden zum zugehörigen Simulationszeitpunkt ausgeführt.

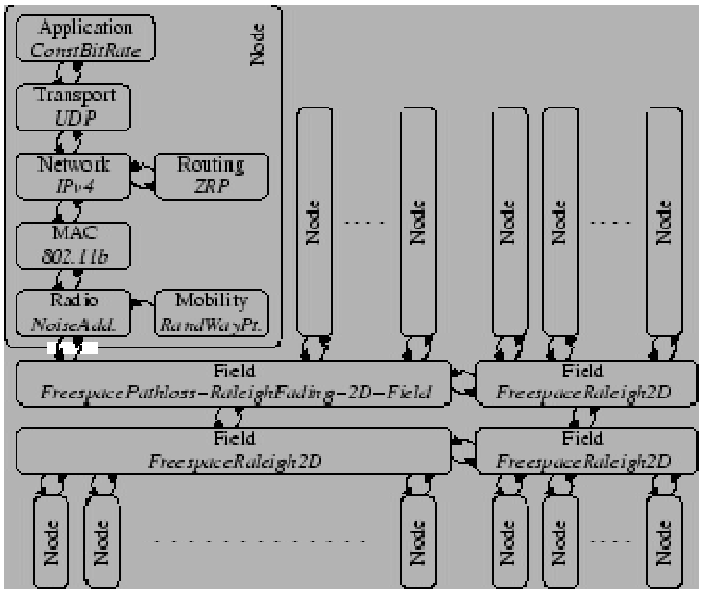

Abbildung 4: Der SWANS-Simulator besteht aus ereignisgesteuerten Komponenten, die so konfiguriert werden können, dass das gewünschte Netzwerk simuliert wird.

Der „Scalable Wireless Adhoc Network Simulator“ (SWANS) basiert auf JiST und ermöglicht die Simulation kompletter drahtloser Netzwerkkonfigurationen. Seine Eigenschaften ähneln denen von ns-2[4] und GloMoSim[5]. SWANS (Abbildung 4[6]) besteht aus unabhängigen Softwarekomponenten für verschiedene Typen von Applikationen, Netzwerkstacks, Routing, Signalübertragung und Fehlermodelle, sowie Bewegungsmodelle. Jede SWANS-Komponente ist als JiST-Entity implementiert und interagiert mit anderen Komponenten nur über Event-basierte Interfaces.

3.2 Implementierung

Die Wahl für die zwei DHTs fiel auf Chord und Pastry. Chord stellt mit seiner großen Verbreitung einen de facto-Standard dar. Pastry wurde wegen des anderen Ansatzes zur Verteilung von Informationen gewählt.

Für die DHT-Implementierungen wurde der Quellcode aus dem OverSim: P2P Simulation Framework [12] von C++ nach Java portiert und angepasst. Jener Simulator ermöglicht zwar Simulationen von DHTs in drahtgebundenen Umgebungen, jedoch stellt er keinerlei Funktionalität für WMNs bereit. Alle dortigen Implementierungen wurden laut der Entwickler gemäß der beschriebenen Funktionalität programmiert.

Die Kommunikation innerhalb der DHT und zwischen DHT und Anwendung erfolgte über Objekte. In Java wurde dies beibehalten. Beim Senden dieser Pakete werden diese mittels Java-Serialisierung übertragen. Dies ist eine Vereinfachung, die allerdings auch zu einer Vergrößerung der über das WMN übertragenen Pakete beiträgt. Dies muss bei direkten Vergleichen zu Testläufen in realen WMNs beachtet werden.

Bei der Portierung wurde der modularisierte Ansatz soweit übernommen, dass die DHT ein Overlay-Netzwerk bereitstellt, welches dann von beliebigen Anwendungen benutzt werden kann. Dazu wurden Overlay und Anwendung in einer JIST/SWANS-Entität zusammengeführt. Dadurch wird der logische Zusammenhang zwischen Knoten im WMN, Overlay-Knoten und Anwendung wiedergespiegelt. Jeder Knoten im WMN ist zugleich auch ein Knoten der DHT und betreibt die Anwendung. Da OverSim ereignisgetrieben funktioniert, ruft die Simulationsanwendung zum passenden Zeitpunkt die entsprechende Funktion in Anwendung und Overlay auf.

Als Anwendung wurde die DHTTestApp aus OverSim übernommen, welche in regelmäßigen Abständen Put-Anfragen zu zufällig generierten Schlüsseln an das Overlay sendet und Erfolge, Misserfolge und ausbleibende Antworten protokolliert. Dadurch ist eine Unterscheidung der aus dem Knoten aus- und eingehenden Pakete in Anwendungsdaten und administrative DHT-Daten möglich.

[4] ns-2: http://www.isi.edu/nsnam/ns/
[5] GloMoSim: http://pcl.cs.ucla.edu/projects/glomosim/
[6] [11]: SWANS User Guide: Design Highlights

Die DHT wird durch den ersten aktiv werdenden Knoten initialisiert, an dem sich alle folgenden anmelden. Die Anwendung wird erst nach erfolgter Stabilisierung der Link-Tables, der simulierten Knoten und der DHT aktiv.

3.3 Parameter

Die simulierte Fläche ist eine 100 m² große, freie Fläche, auf der 5, 8, 10 und 15 Knoten in separaten Läufen zufällig positioniert werden. Alle Knoten verfügen über einen 802.11g-Adapter und bekommen zu Beginn der Simulation eine feste IP- und MAC-Adresse zugeteilt. Die Kommunikation erfolgt über UDP. Die Daten, die von der Test-Applikation angefragt werden, sind immer 20 Zeichen lang.

Eine genaue Auflistung der Parameter ist im Anhang in Kapitel 6.3 zu finden.

3.4 Durchführung der Messungen

Bei der Sammlung von Messergebnissen ist die Wiederholbarkeit von Messungen besonders wichtig, um Überprüfbarkeit gewährleisten zu können. Dazu werden die Messungen in einem WMN-Simulator durchgeführt, da dieser eine kontrollier- und zugleich wiederholbare Plattform zur Überprüfung und Auswertung der Messungen bietet.

Um jeden Simulationslauf wiederholbar zu machen, wird die Simulation mit einem vorher festgelegten Seed gestartet (siehe Kapitel 6.2). Alle benötigten Zufallswerte werden durch einen Pseudo-Zufallszahlengenerator ermittelt. Der übergebene Seed bestimmt die Folge der erzeugten Zahlen.

Jeder Knoten ist zugleich Mesh-Node im WMN, Peer in der DHT und stellt Anfragen an die DHT. Jedem Knoten wird eine Vorlaufzeit von 20 Sekunden zur Stabilisierung der Link-Tabelle eingeräumt. Danach wird der DHT 10 Sekunden Zeit gegeben, sich aufzubauen und zu stabilisieren. Im Anschluss daran werden DHT und Anwendung in jedem Knoten für eine Zeit von 10 Sekunden aktiv, während der Anfragen gesendet werden. Jeder Knoten generiert alle 100 Millisekunden eine Anfrage an die DHT, somit versucht jeder Knoten in den 10 Sekunden 100 Pakete zu senden.

Eine Anfrage gilt als erfolgreich, sofern eine positive Antwort erhalten wurde. Dies ermöglicht auch die Messung der Latenz der Anfrage. Geht eine Anfrage verloren, trägt sie dennoch zur Gesamtzahl der gesendeten Anfragen bei.

Für alle ausgehenden Pakete eines Knotens wird deren Größe in Bytes gespeichert. Aus dem Durchschnitt über alle Knoten lässt sich zusammen mit dem betrachteten Simulationszeitraum (10 Sekunden) die durchschnittlich benutze Bandbreite errechnen (Bytes/Knoten/10 Sekunden).

3.5 Auswertung der Simulation

In diesem Abschnitt wird anhand der Resultate der Simulationen aufgezeigt, welche Aussagen sich über das Verhalten von herkömmlichen DHTs in WMNs treffen lassen. Dabei werden beide DHTs zuerst separat betrachtet und anschließend miteinander verglichen.

3.5.1 Ergebnisse für Chord

In Abbildung 5 sind die Ergebnisse der Simulationen für Chord dargestellt.

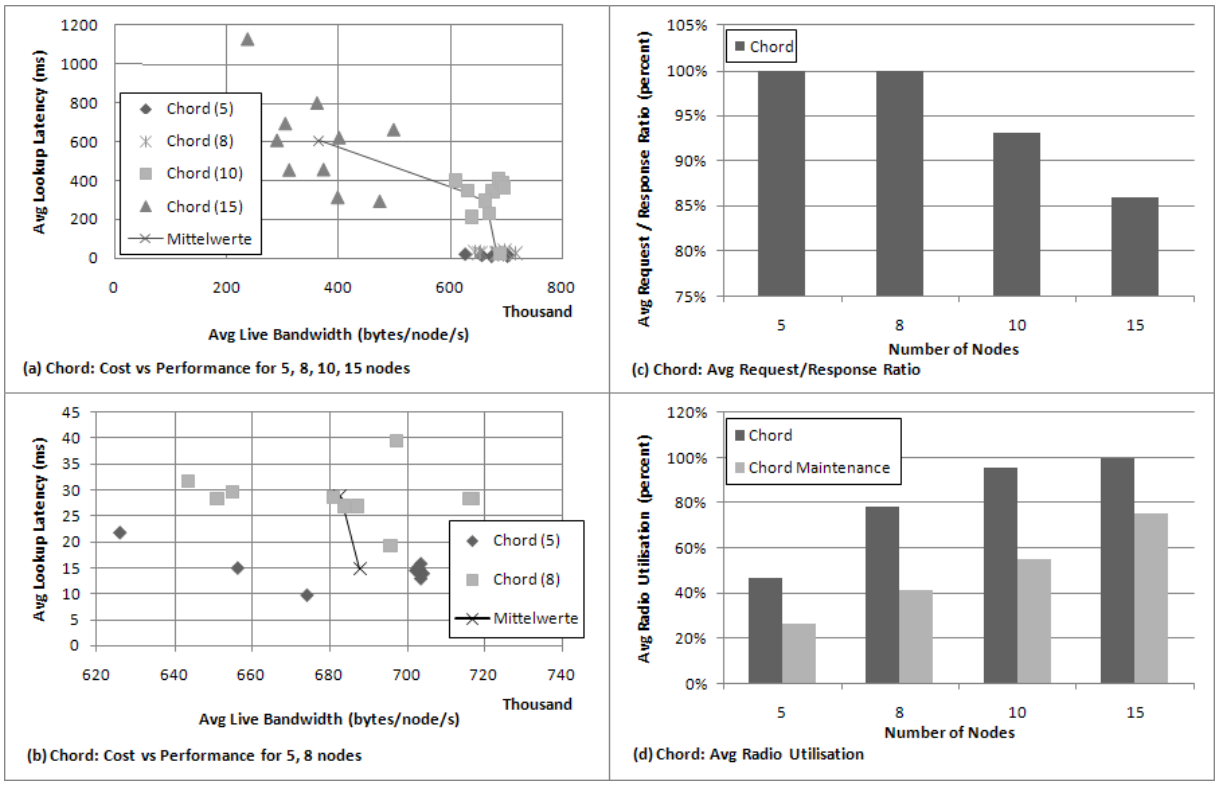

Abbildung 5: Chord Ergebnisse

Abbildung 5(a) zeigt die Kosten-Performance-Abschätzung für die Messungen mit fünf, acht, zehn und fünfzehn Knoten. In diesem Diagramm sind hauptsächlich die Unterschiede zwischen nicht-saturiertem und saturiertem Medium zu sehen. Wie in Abbildung 5(d) zu sehen ist, beträgt die durchschnittliche Netzlast bei zehn Knoten 95,6%. Dabei steigt die durchschnittliche Lookup Latenz im Mittelwert von acht Knoten bei *28,8 ms* um den Faktor 10 auf *300,2 ms* bei zehn Knoten. Hier kommen die bereits erwähnten Queuing-Delays zum Tragen und das erklärt die hohe Varianz der Latenzen ab zehn Knoten.

Abbildung 5(b) zeigt die Kosten-Performance-Abschätzung für den nicht-saturierten Bereich (also Messungen mit fünf und acht Knoten). Der Mittelwert der Lookup Latenz liegt für fünf Knoten bei *15 ms* und steigt für acht Knoten auf *28,8 ms* an. Der Mittelwert der durchschnittlich benutzten Bandbreite sinkt um *4950 bytes/node/s* beim Schritt von fünf zu acht Knoten.

Abbildung 5(c) zeigt die Zuverlässigkeit. Hier ist zu sehen, dass die Zuverlässigkeit im nicht-saturierten Bereich bei 100% liegt, im saturierten Bereich auf 93% für zehn Knoten und auf 86% für fünfzehn Knoten einbricht. Die Zuverlässigkeit wird also direkt durch die Anzahl der Knoten innerhalb der Kollisionsdomäne und damit durch die Belegung des Mediums beeinflusst.

Abbildung 5(d) zeigt die durchschnittliche Netzlast nach Maintenance-Traffic und Regelbetrieb getrennt. Für fünf Knoten macht der Maintenance-Traffic 55% des gesamten Datenverkehrs aus, für acht Knoten 52%. Dieser Faktor wird einerseits durch die Vergrößerung der Pakete durch die Java-Serialisierung bewirkt, andererseits durch die Parameter und die generelle Funktionsweise von Chord beeinflusst.

3.5.2 Ergebnisse für Pastry

In Abbildung 6 sind die Ergebnisse der Simulationen für Pastry dargestellt.

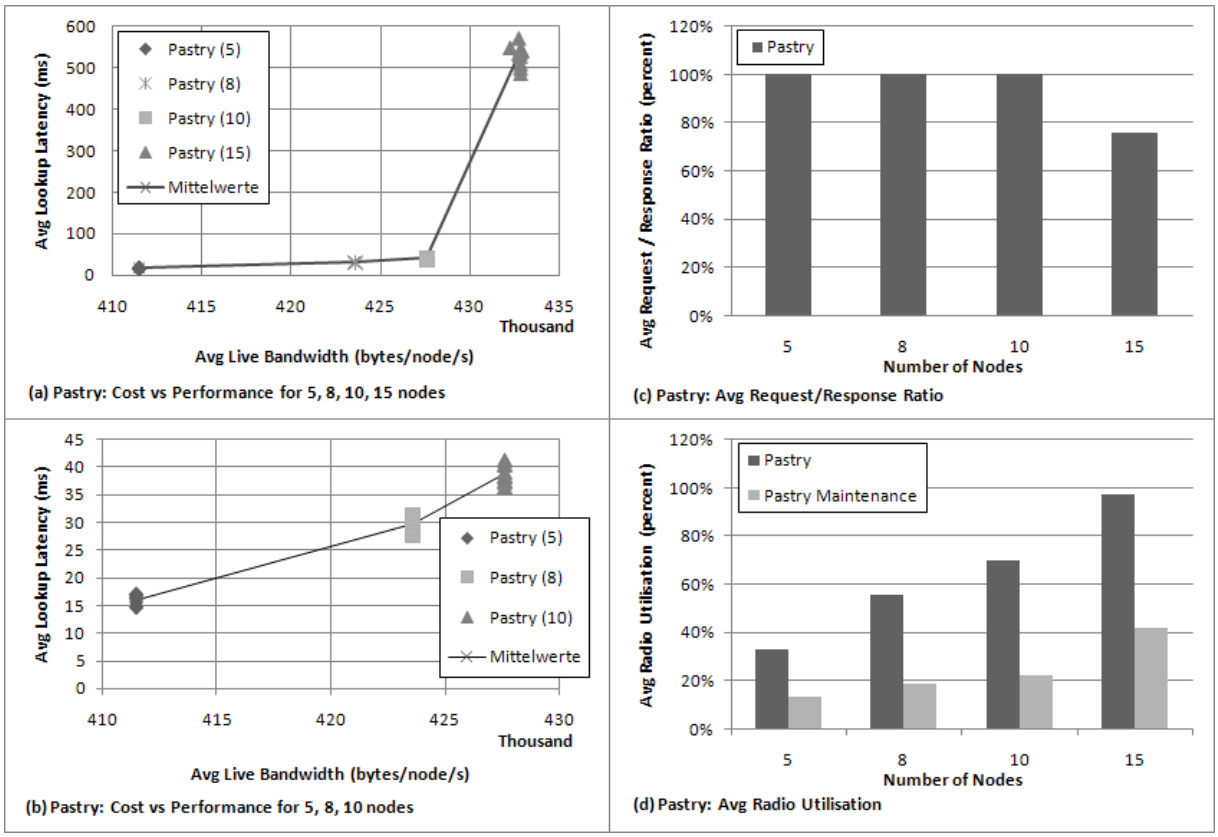

Abbildung 6: Pastry Ergebnisse

Abbildung 6(a) zeigt die Kosten-Performance-Abschätzung für die Messungen mit fünf, acht, zehn und fünfzehn Knoten. Auch in diesem Diagramm sind die Unterschiede zwischen nicht-saturiertem und saturiertem Medium zu erkennen. Abbildung 6(d) zeigt, dass das Medium mit 15 Knoten bei einer Netzlast von 97,3% überlastet ist. Die durchschnittliche Lookup Latenz steigt von *38,9 ms* bei zehn Knoten auf *528,6 ms* bei fünfzehn Knoten um den Faktor 13. Auch in diesem Fall sind die Queuing-Delays Hauptverursacher für den Anstieg.

Abbildung 6(b) zeigt die Kosten-Performance-Abschätzung im nicht-saturierten Bereich (also für fünf, acht und zehn Knoten). Der Schritt von fünf zu acht Knoten verdoppelt nahezu den Mittelwert der durchschnittlichen Lookup Latenz von *16,1 ms* auf *29,7 ms*. Gleichzeitig steigt die durchschnittlich benutze Bandbreite pro Knoten um *12119 bytes/node/s* an. Beim Schritt von acht zu zehn Knoten steigt die durchschnittliche Lookup Latenz von *29,7 ms* auf *38,9 ms*, die durchschnittlich benutzte Bandbreite steigt um *4040 bytes/node/s*.

Abbildung 6(c) zeigt die Zuverlässigkeit. Im nicht-saturierten Bereich liegt sie bei 100% und fällt im saturierten Bereich um über 20% ab. Auch hier ist der Zusammenhang zwischen Zuverlässigkeit, Anzahl der Knoten in der Kollisionsdomäne und der Belegung des Mediums zu sehen.

Abbildung 6(d) zeigt die durchschnittliche Netzlast nach Maintenance-Traffic und Regelbetrieb getrennt. Im nicht-saturierten Bereich liegt der Maintenance-Traffic zwischen 13% für fünf Knoten und 22% für zehn Knoten. Dieser Faktor wird einerseits durch die Vergrößerung der Pakete durch die Java-Serialisierung bewirkt, andererseits durch die Parameter und die generelle Funktionsweise von Pastry beeinflusst.

3.5.3 Vergleich der Ergebnisse

In diesem Abschnitt werden die Simulationsergebnisse beider DHTs miteinander verglichen und interpretiert. Abbildung 7 zeigt die Werte nebeneinander. Dabei wird der Vergleich der Werte auf den nicht-saturierten Bereich beschränkt, da hier die Charakteristika auswertbar sind.

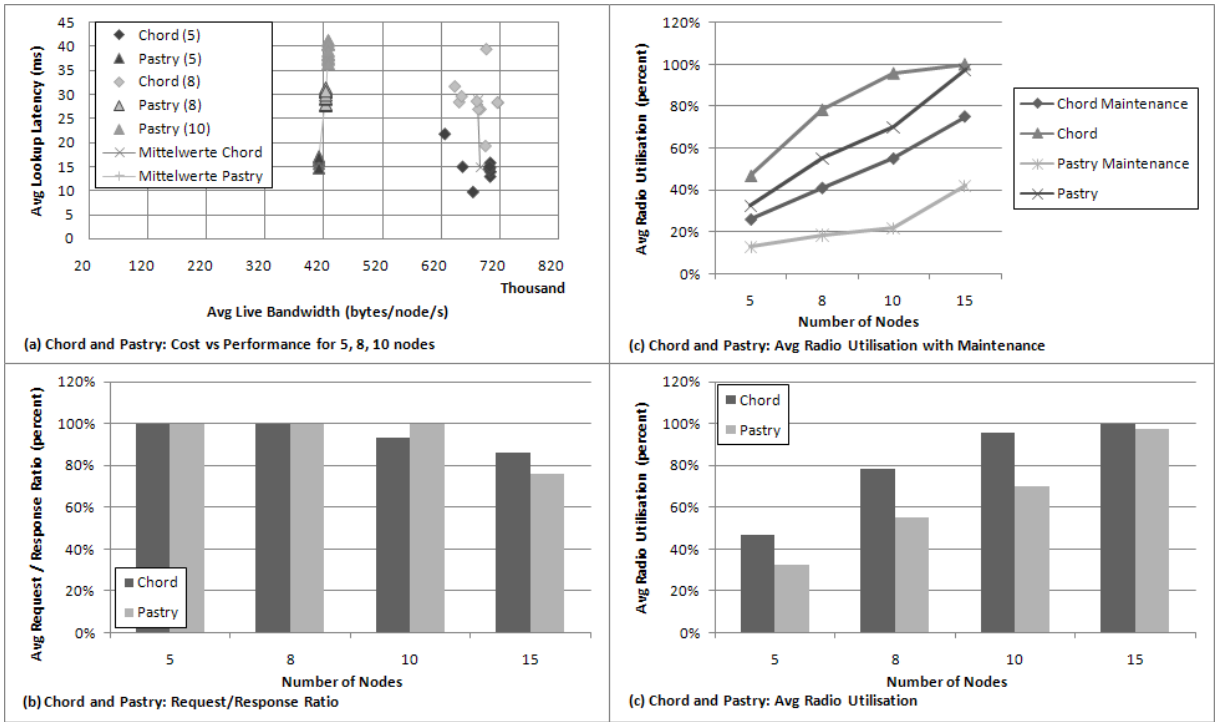

Abbildung 7: Chord und Pastry Ergebnisse

Abbildung 7(a) zeigt die Kosten-Performance-Abschätzung für beide DHTs im nicht-saturierten Bereich. Dieser liegt für Chord bei fünf und acht Knoten und für Pastry bei fünf, acht und zehn Knoten. Chord verbraucht bei gleichem Anfrageprofil im Durchschnitt ungefähr *270 kbytes/node/s* mehr Bandbreite pro Knoten. Bei der Lookup Latenz ist Chord den Mittelwerten *1 ms* schneller. Die Werte der einzelnen Simulationen zeigen für Pastry keine starke Varianz, während Chord eine hohe Varianz aufweist.

Abbildung 7(c) zeigt die durchschnittliche Netzlast zusammen mit der zugehörigen durch die Maintenance verbrauchten Netzlast für jeweils Chord und Pastry. Im nicht-saturierten Bereich (fünf und acht Knoten) weist Pastry für Maintenance einen um den Faktor 1/2 niedrigeren Wert zu Chord auf. Allerdings steigt der Maintenance-Anteil bei Pastry wesentlich weniger stark an, als Chord. Dies ist durch die andere Organisation von Pastry zu erklären.

Abbildung 7(d) zeigt die durchschnittliche Netzlast für die beiden DHTs im Vergleich. Im nicht-saturierten Bereich für beide DHTs (fünf und acht Knoten) liegt die durchschnittliche Netzlast in der gleichen Kollisionsdomäne für Pastry um 1/3 unter dem Wert von Chord. Das zeigt, dass Chord einen wesentlich größeren Verwaltungsoverhead bei gleichem Anfragemuster aufweist. Dies ist auch in der deutlich höheren durchschnittlich verbrauchten Bandbreite pro Knoten zu erkennen. Allerdings schlägt sich dies im nicht-saturierten Bereich auf die durchschnittliche Lookup Latenz nieder.

4 Zusammenfassung und Ausblick

Zusammenfassend lässt sich ausgehend von der Dichte der Knoten innerhalb einer Kollisionsdomäne eine klare Unterscheidung beider DHTs bezüglich ihrer Leistungsfähigkeit in WMNs treffen. Im nicht-saturierten Bereich verbraucht Pastry im Mittelwert weniger Bandbreite pro Knoten als Chord. Somit geht Pastry bei ähnlicher Lookup Latenz schonender mit der verfügbaren Ressource (Bandbreite) um. Dies zeigt sich auch in der Tatsache, dass Pastry mehr Knoten in der gleichen Kollisionsdomäne verwalten kann. Abschließend lässt sich ausgehend von den Messungen sagen, dass Pastry deutlich besser in der drahtlosen Umgebung operiert. Insgesamt ist jedoch zu beachten, dass DHTs für drahtgebundene Netzwerke ohne Beachtung der Saturation in drahtlosen Umgebungen sehr schnell die zur Verfügung stehende Bandbreite in einer Kollisionsdomäne ausreizen und das Medium überlasten, sobald regelmäßig Anfragen an das Overlay gestellt werden. Dies würde noch schneller geschehen, sobald nicht nur Daten für die DHT über das WMN übertragen werden, sondern auch z. B. normaler TCP-Traffic. Da WMNs nicht alleine für einen Zweck, sondern parallel vielseitig benutzt werden, ist der Einsatz von DHTs für drahtgebundene Netze nur unter Vorsicht zu empfehlen.

Um die Relevanz der Methodik genauer zu überprüfen sind weitere Messungen mit anderen DHT-Implementierungen notwendig. Dabei könnte man auch auf Variation von Parametern eingehen, wie dies in [4] gemacht wurde. Interessant wäre auch die Simulation von DHTs für drahtlose Umgebungen, wie z.B. [10]. Weitere Variationsmöglichkeiten bieten erhöhte Übertra-

gungsraten, die eine durchaus größere Knotendichte zulassen könnten, wobei zugleich die Übertragungsreichweite verringert würde. Ein zusätzlicher Aspekt wäre die Einführung von Churn, die direkte Vergleiche zu den Ergebnissen aus [4] zuließe. Den letzten Punkt des vorhergehenden Absatzes aufgreifend wäre auch die Betrachtung des Verhaltens der DHTs mit einem teilweise durch z. B. TCP-Ströme belegten Medium interessant, um eine Aussage über eine solche Beeinflussung treffen zu können. Um den Einfluss der Java-Serialisierung auszuschließen, könnte auch ein konkretes Paketformat eingeführt werden, wodurch die Bandbreite geschont würde. Auch wären Simulationsläufe mit drahtgebundener Übertragung sinnvoll, um anhand derer die Ergebnisse mit den Resultaten aus anderen Veröffentlichungen zu diesem Thema vergleichen und den möglichen Einfluss des Simulators abschätzen zu können.

5 Literatur

[1] MIT Laboratory for Computer Science: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. Internet: http://pdos.lcs.mit.edu/chord/ [30.01.2009].

[2] R. Steinmetz, K. Werle (Eds.): Peer-to-Peer Systems and Applications, Kapitel 7: Distributed Hash Tables, Springer, 2005.

[3] H. Yu, X. Shen (Eds.): Peer-to-Peer Networking and Applications, Kapitel 8: Selected DHT Algorithms, Springer, 2008.

[4] J.Li, J. Stribling, T. M. Gil, R. Morris, Comparing the performance of distributed hash tables under churn.

[5] A. Binzenhöfer: Performance Analysis of Structured Overlay Networks, Würzburger Beiträge zur Leistungsbewertung Verteilter Systeme, Bericht 01/08.

[6] S. Rhea, T. Roscoe, J. Kubiatowicz: Structured Peer-to-Peer Overlays Need Application-Driven Benchmarks.

[7] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica: Wide-area cooperative storage with CFS.

[8] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, R. Morris: Designing a DHT for low latency and high throughput.

[9] R. Sombrutzki, A. Zubow, M. Kurth, J.-P. Redlich: Self-Organization in Community Mesh Networks: The Berlin RoofNet.

[10] F. Araujo, L. Rodrigues, J. Kaiser, C. Liu, C. Mitidieri: CHR: Distributed Hash Table for Wireless Ad Hoc Networks. http://www.di.fc.ul.pt/~ler/reports/debs05.pdf.

5.1 Websites

[11] JiST/SWANS: Java in Simulation Time / Scalable Adhoc Network Simulator http:// jist.ece.cornell.edu [30.01.2009].

[12] OverSim: The P2P Simulation Framework http:// www.oversim.org [30.01.2009].

[13] Berlin RoofNet http://sarwiki.informatik.hu-berlin.de [30.01.2009].

6 Anhang

6.1 Quellcode

Der Quellcode für die DHTs und ihre Einbindung in den BRN-Simulator ist auf Nachfrage an *ringlebe@informatik.hu-berlin.de* zu bekommen und steht unter GNU General Public License.

6.2 Liste der Zufallswerte

Die folgende Tabelle gibt die Werte (Seeds) wieder, mit denen der Zufallszahlengenerator für die einzelnen Simulationsläufe initialisiert wurde. Diese wurden für alle Kombinationen (5, 8, 10 und 15 Knoten für beide Implementierungen) benutzt und ermöglichen ein Nachvollziehen der Ergebnisse.

Lauf	1	2	3	4	5	6	7	8	9	10
Seed	15	3	22	39	42	8	16	27	3	48

6.3 Parameter

In dieser Tabelle sind die Parameter für den Simulator aufgelistet.

FieldBuilder	
FieldParams.fieldX	100
FieldParams.fieldY	100
FieldParams.pathloss	Constants.FreeSpaceParams
FieldParams.spatial_mode	Constants.Spatial_Linear
RadioBuilder	
NoiseParams.fieldX	100
NoiseParams.fieldY	100
NoiseParams.placement	Constants.Placement_Random
NoiseParams.useAnnos	true
NoiseParams.radioType	Constants.MAC_802_11g_Pure
MacBuilder	
RateBuilder.controlBitrate	Constants.Bandwidth_6Mbps
RateBuilder.dataBitrate	Constants.Bandwidth_6Mbps
M802_11Params.useAnnos	True
M802_11Params.useBitRateAnnos	False
NetBuilder	
IpParams.protocolMapper	{ Constants.NET_PROTOCOL_UDP, Constants.NET_PROTOCOL_LINK_PROBE, Constants.NET_PROTOCOL_MCEXOR, Constants.NET_PROTOCOL_FLOODING }
RouteBuilder	
BrnDsrParams.protocol	Constants.NET_PROTOCOL_MCEXOR
BrnDsrParams.forwarding	brn.swans.Constants.FORWARDING_UNICAST
BrnDsrParams.discovery	brn.swans.Constants.DISCOVERY_PROACTIVE
BrnDsrParams.floodingOffset	50000
BrnDsrParams.floodingPeriod	10000
MetricBuilder	
EtxParams.probes	{ Constants.BANDWIDTH_6Mbps, 1000 }
EtxParams.period	1000
EtxParams.tau	30000
EtxParams.globalLinkTable	True
TransBuilder.UdpParams	