Universität Konstanz

Fachbereich für Informatik und Informationswissenschaft
Lehrstuhl für verteilte Systeme

Effizienzanalyse von Routingalgorithmen in interessensbasierten mobilen sozialen Netzen

Wissenschaftliche Arbeit
zur Erlangung des Grades eines *Bachelor of Science* Information Engineering
im Fachbereich Informatik & Informationswissenschaft der Universität Konstanz

Verfasser:
Michael Zinsmaier

16. Januar 2010

Gutachter:
Prof. Marcel Waldvogel
Prof. Ulrik Brandes

Universität Konstanz
Fachbereich für Informatik und Informationswissenschaft
D–78457 Konstanz
Deutschland

Konstanzer Online-Publikations-System (KOPS)
URN: http://nbn-resolving.de/urn:nbn:de:bsz:352-opus-128866
URL: http://kops.ub.uni-konstanz.de/volltexte/2011/12886/

Michael Zinsmaier
Karlsruherstraße 2
78467 Wollmatingen
01/637211
michael.zinsmaier@uni-konstanz.de
Effizienzanalyse von Routingalgorithmen in interessensbasierten mobilen sozialen Netzen
(Performance analysis of routing algorithms in interest-based mobile social networks)
*Bachelor*arbeit Universität Konstanz, 2009.

Zusammenfassung

Diese Arbeit analysiert den in ISeek [BIW08] skizzierten *Routingalgorithmus* anhand einer Referenzimplementierung, die auf unterschiedlichen sozialen Netzwerken getestet wird. Zum Einsatz kommen hierbei die auch in Nuntifix-Modeling of Delay Tolerant Networks [Isl08] verwendeten Datensätze. Bei ISeek handelt es sich um einen *Routingalgorithmus*, der Informationsanfragen in einem *DTN* (delay tolerant network) an Antwortressourcen zustellt. Der Fokus liegt hierbei auf sozialen Netzwerken, deren transitive Verbindungen (Small World Phänomen [Mil67]) von dem Algorithmus ausgenutzt werden.

In einem einfachen Anwendungsszenario könnte man sich zum Beispiel vorstellen, dass alle Studenten des Fachbereichs Informatik der Universität Konstanz mit ihren Handys, Netbooks und Laptops ein solches Netzwerk über Bluetooth und WLAN-Verbindungen erzeugen. Jeder Studenten würde dann auf seinen Geräten ein Wissensprofil ablegen, mit dessen Hilfe der Algorithmus diejenigen Studenten finden kann, die sich zur Beantwortung einer Frage zum Beispiel über eine Band etc. eignen. Dabei folgt der Algorithmus den tatsächlichen Kontakten der Benutzer, transitiv über mehrere Ebenen, von Freunden zu Freundesfreunden usw. Auf dem technischen Netzwerk (WLAN, Bluetooth) wird so über real existierende Kontakte ein soziales Datennetz gebildet, welches sich zur Informationsvermittlung eines transitiv erschlossenen Bekanntenkreises bedient. Der Algorithmus kommt dabei ohne zentrale Steuerung und globales Wissen aus, alle Entscheidungen werden auf den beteiligten Geräten lokal getroffen. ISeek positioniert sich hierbei zwischen *Direct Contact* und *Flooding*. Das Netzwerk soll also weder stupide mit Nachrichten geflutet werden, noch soll die Nachrichtenausbreitung zu stark gehemmt werden. Um diese Ziele zu erreichen verwendet der Algorithmus nicht etwa alle Kontakte, sondern beschränkt sich auf wenige relativ sichere Routen zur Verbreitung der Nachrichten. Als besondere Herausforderungen stellen sich deshalb die Identifizierung geeigneter Parameter zur Dämpfung der durch den Algorithmus verursachten Netzwerkbelastung und die Identifizierung und Kodierung der zum *Routing* verwendeten Metainformationen, sowie deren Aggregation zur Gewährleistung der Transitivität dar.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Überblick

Mit ISeek wird ein Algorithmus beschrieben und evaluiert, der für die speziellen Anforderungen von *MANETs* entwickelt wurde. Bei einem *MANET* (mobile ad hoc network) handelt es sich um ein sich selbst-konfigurierendes mobiles Funknetzwerke. In einem solchen Netzwerk fungiert jeder Teilnehmer als Router, Vermittler und Empfänger. *MANETs* kommen heute durch die starke Verbreitung von Laptops, Handys und anderen mobilen Geräten mit den nötigen hardwaretechnischen Voraussetzungen in vielen Bereichen unseres täglichen Lebens vor. Außerdem können sie in Gegenden mit mangelhafter Infrastruktur als Ersatz für normale Netzwerke dienen. Sie eignen sich jedoch nicht für traditionelle *Routingverfahren*, da sie über besondere herausfordernde Einschränkungen und Eigenschaften verfügen. Betrachtet man eine Unterkategorie der *MANETs*, die *DTNs* (delay tolerant network) werden die Herausforderungen noch größer, da die *Routingalgorithmen* hier mit hohen Verzögerungen bei der Pfadentwicklung zurechtkommen müssen.

- Verbindungen zwischen Knoten sind oft nur von geringer Dauer und schwer oder gar nicht vorhersehbar
- In einem *DTN* existiert im Regelfall kein durchgehender Pfad vom Start zum Ziel
- Sowohl der Speicher der Geräte (Handys) als auch die zur Verfügung stehende Bandbreite sind engen Beschränkungen unterworfen, so das ein einfaches Fluten des Netzes mit den Nachrichten keine Option darstellt.

ISeek wurde so entworfen, dass er all diesen Herausforderungen begegnen kann. So setzt ISeek auf lokale Entscheidungen, der Startpunkt einer Nachricht muss das Ziel nicht notwendigerweise kennen. Jeder Teilnehmer wird seiner Vermittlerrolle dadurch gerecht, dass er sein eigenes lokales Umfeld beobachtet und Nachrichten an geeignete Vermittler und Empfänger weiterleitet. Jeder Teilnehmer merkt sich für alle Verbindungen, die ihm

als besonders stabil und verlässlich erscheinen welche anderen Teilnehmer über diese Verbindung erreichbar sind. Dabei wird nicht nur die direkte Erreichbarkeit berücksichtigt, sonder es fließen auch die transitiv in mehreren Schritten erreichbaren Teilnehmer mit ein. Der Algorithmus löst die beschriebenen Herausforderungen also wie folgt:

- ISeek geht mehr als nur einen Weg um so gegen das Ausfallen von Kanten und Teilnehmern gewappnet zu sein.
- Lokale intelligente Entscheidungen, die die transitive Nachbarschaft mit einbeziehen ermöglichen das Erreichen des Ziels auch dann wenn der Startknoten dieses nicht kennt.
- Um die Auslastung des Netzwerkes möglichst gering zu halten werden Nachrichten nur verschickt, wenn der Empfänger Teil einer Kette von Teilnehmern auf dem Weg zu einem Zielknoten ist.

Zusätzlich lässt sich der Algorithmus durch die Definition einer guten Verbindung und durch die Größe des berücksichtigten Umfeld sehr leicht an verschiedene Rahmenbedingungen anpassen. In dem Abschnit 1.2 werden die maßgeblichen verwandten Arbeiten und Lösungsideen vorgestellt und in Abschnitt 1.3 wird ausgeführt, wie sich ISeek zwischen ihnen einordnet.

1.2 Verwandte Arbeiten

Soziale Kontakte in Mobilen Netzwerken

Die Autoren zeigen in [MGC+07], dass *DTN-Routing* Protokolle, Firewalls und *P2P Filesharingsysteme* im mobilen Kontext (PDAs, Handys etc.) von der Ausnutzung Sozialer Informationen profitieren können.
Als Datengrundlage wird eine vom "Reality Mining Project" [MIT] zur Verfügung gestellte 101 Tage umfassende Aufzeichnung der Personenkontakte von 100 Studenten verwendet. Die teilnehmenden Studenten wurden mit speziellen Handys ausgestattet, die während des Testzeitraums jeden Bluetoothkontakt mit Studienteilnehmern und "fremden" Geräten registrierten. In der Arbeit werden lediglich die Treffen zwischen den Teilnehmern ausgewertet und in zwei Gruppen aufgeteilt. Einerseits Fremdkontakte, alle Personenpaare, die sich an weniger als 10 von 100 Tagen treffen, andererseits Kontakte zwischen Freunden, also Personenpaaren, die sich an 10 oder mehr Tagen treffen. Die 10 Tages Grenze wird von den Autoren als untere Grenze für eine Freundschaft angesehen und entspricht etwa einem Treffen pro Woche. Auf diese Art werden 6,9% der Personenpaare als Freunde definiert, zwischen denen aber 65,3% der Treffen stattfinden. Mit weiteren statistischen Untersuchungen wird gezeigt, dass die Anzahl der Kontakte für die Teilnehmer größtenteils aus der Vergangenheit vorhersehbar ist und von der Uhrzeit und dem Wochentag abhängt. Außerdem wird gezeigt, dass sowohl der Freund- als auch der Fremdgraph *scale-free* sind und das der Freundgraph *Cluster* bildet.

Mit diesen Ergebnissen erstellen die Autoren einen Simulator, der basierend auf dem MIT-Graphen, für einen Zeitraum von zwei Wochen die typischen Muster für Freund- und Fremdkontakte erzeugt, allerdings ohne Informationen über die Dauer dieser Kontakte mit einzubeziehen. Der so erzeugte synthetische Datensatz wird gegen die original Daten validiert und die Anzahl der Teilnehmer (20.000) sowie Anzahl der Kontakte wird so angeglichen, dass eine möglichst hohe Übereinstimmung erreicht wird. Der Simulator dient als Grundlage um den Nutzen Sozialer Information in den drei Themengebieten untersuchen zu können.

Zuerst widmen sich die Autoren dem *Routing* in *DTNs*. In [MGC+07] beschrieben werden die Auswirkungen auf die folgenden vier *Routing* Protokolle:

- *Direct Contact* die Nachricht wird nur weitergeleitet falls Kontakt zum Zielknoten besteht.
- *Forward-to-1-Person* die Nachricht wird einmal an einen Relaisknoten weitergeleitet und dann von diesem oder dem Startknoten zum Zielknoten.
- *Forward-to-2-Persons* die Nachricht wird an zwei Relaisknoten weitergeleitet und dann von diesen oder von dem Startknoten zum Zielknoten.
- *Forward-to-all* die Nachricht wird an alle Knoten weitergeleitet, die der Startknoten trifft und dann von einem dieser Knoten oder dem Startknoten zum Zielknoten.

Als Vergleich wird außerdem *Epidemic Routing* untersucht, bei dem jeder Knoten die Nachricht an alle Knoten weiterleitet, auf die er trifft. Ohne Soziale Information ist *Epidemic Routing* wesentlich schneller als alle anderen Algorithmen und übertrifft mit 100% ausgelieferten Nachrichten auch *Forward-to-all* als nächstbesten Algorithmen mit nicht einmal 40% bei weitem. Im zweiten Versuch werden dagegen die Nachrichten nicht zufallsverteilt im Netzwerk versendet sondern zwischen 100 Freundpaaren. Außerdem werden alle vier Algorithmen so verändert, dass sie nur noch Kanten zu Freunden berücksichtigen. Mit dieser neuen Konfiguration werden wesentlich bessere Ergebnisse erzielt, die sowohl in Geschwindigkeit als auch in der Anzahl der ausgelieferten Nachrichten zu *Epidemic Routing* vergleichbar sind.

Als zweites wird der Nutzen einer Firewall beschrieben, die Soziale Informationen verwendet um Datenverkehr mit Fremden abzulehnen. Die Autoren simulieren die Ausbreitung eines Wurms von einem infizierten Gerät auf die Population. Hierbei sind 5% der Geräte für den Wurm anfällig und 1,5% dieser Geräte werden im zweiten Schritt mit der Sozialen Firewall bestückt. Dadurch verlangsamt sich die Ausbreitung des Wurms so stark, dass die 50% Marke an infizierten Geräten erst nach 14 Tagen und damit 5 Tage später als ohne Firewall erreicht wird. Die Firewall ist damit dicht an den optimalen Werten, die erreicht würden, wenn jeglicher Verkehr geblockt würde, ermöglicht aber noch die Kommunikation mit allen Freundknoten.

Zuletzt untersuchen die Autoren noch welche Auswirkungen eine Beschränkung auf Freundkanten auf *P2P Systeme* hat. Es zeigt sich, dass eine solche Einschränkung die Anzahl der erfolgreichen Datentransfers auf 1/3 senkt und *P2P Systeme* daher nicht auf Freundkontakte beschränkt werden sollten.

Nicht-deterministisches Routing in P2P Netzwerken

In [RK02] beschreiben die Autoren einen *Hybrid Routing* Algorithmus für *P2P* Netzwerke, der im Nahbereich mittels wahrscheinlichkeitsbasierter Ortung sucht und falls nötig auf deterministische *P2P Algorithmen* zurückgreift. Die Autoren zeigen, das dieser Algorithmus nahe gelegene Ziele schneller als rein deterministische Algorithmen findet.
Für jeden Server wird aus den Servern mit der geringsten Latenz eine Menge von Nachbarn gebildet. Zu jedem Nachbarn speichert der Server für jedes Dokument die Wahrscheinlichkeit das Dokument über diesen Server zu finden. Diese Wahrscheinlichkeiten können mit Hilfe einer von den Autoren definierten Datenstruktur: den *Attenuated Bloomfiltern* mit konstantem Platzverbrauch gespeichert werden. Ein *Attenuated Bloomfilter* ist ein *Stack* der Höhe h von *Bloomfiltern* [BM02] und wird jeweils einer Verbindung zugewiesen. Der *Bloomfilter* B_i ($i \in (1..h)$) speichert hierbei die Information über alle Dokumente, die auf Nachbar i-ten Grades entlang der Verbindung vorhanden sind. Wird eine Anfrage ausgeführt sucht der Server in seinen *Attenuated Bloomfiltern* nach einem Empfänger. Da höhere Level kürzere Pfade bedeuten und ein Treffer mit höherer Wahrscheinlichkeit korrekt ist (kein *False Positiv*) werden die *Attenuated Bloomfilter* absteigend durchsucht. Um Updates effizient durchführen zu können speichert jeder Server auch die Sicht seiner Nachbarn auf eine Verbindung, so dass er die Änderung in den *Attenuated Bloomfiltern* seiner Nachbarn beim Hinzufügen eines Dokuments berechnen und an diese weiterleiten kann.
Die Autoren testen ihren Algorithmus mit zwei unterschiedlichen deterministischen Algorithmen: *Tapestry* und *Home-Node Location*. Getestet wird mit einem Simulator, der Graphen mit 5100 Knoten erzeugt in zwei Varianten:

- statisch: Die Dokumente verändern sich während des Tests nicht.
- dynamisch: Während des Tests kommen zusätzliche Dokumente dazu.

Die Ergebnisse zeigen deutlich, dass der auf *Attenuated Bloomfiltern* basierte Algorithmus Dokumente entweder schnell findet oder schnell fehlschlägt, so dass ein deterministischer Algorithmus übernehmen kann. Existiert kein lokales Ziel im von dem *Attenuated Bloomfilter* abgedeckten Umkreis, sind die Mehrkosten trotzdem relativ gering. Existiert ein solches Ziel werden die rein deterministischen Algorithmen dagegen deutlich unterboten.

Routing in Delay Tolerant Networks

Sushant et al. untersuchen in [JFP04] wie *DTN-Routingalgorithmen* von Wissen über das Netzwerk profitieren können und zeigt auf, dass begrenzte Kenntnisse weit unterhalb globalen Wissens zur Konstruktion effizienter Algorithmen ausreichen.
Die Autoren konzentrieren sich auf *DTNs* mit bekannter Netzwerktopologie wie sie z.B. von erdnahen Satelliten gebildet werden. Untersucht werden sechs verschiedene Algorithmen, von denen zwei als Vergleichswerte herangezogen werden. Die untere Schranke bildet ein Algorithmus der kein Wissen erfordert, da er die Nachricht einfach immer über eine willkürlich gewählte Kante weiterleitet, während die obere Schranke durch den *Linear Programming* Ansatz mit globalem Wissen über Kontakte, *Queues* und die Netzwerkauslastung gebildet wird. Die vier restlichen Algorithmen basieren auf einer *Dijkstra Adaption* (siehe Tabelle 1.1), so dass alle die Struktur des Netzwerks als bekannt voraussetzen. Sie unterscheiden sich in der Menge der zur Verfügung stehenden Informationen und damit in ihrer Kostenfunktion:

Abk.	**Algorithmus**	**verwendetes Wissen**
MED	Minimum Expected Delay	Durchschnittliche Wartezeit pro Kante
ED	Earliest Delivery	zukünftige Kontakte
EDLQ	ED with Local Queue	ED + lokale Warteschlange
EDAQ	ED with All Queue	ED + Warteschlangen aller Knoten

Tabelle 1.1: Übersicht über die in [JFP04] verwendeten Algorithmen.

Die Algorithmen werden an einem bildlichen Dorfszenario mit drei Kanten und zwei Knoten (eine der Kanten stellt einen Satelliten dar) und an einer komplexeren Welt, die aus den Busrouten von San Francisco gebildet wird getestet. Die Versuche werden mit variierender Bandbreite, variierender Reichweite der Sender (führt zu einer Erhöhung der Kontaktzahl) und unterschiedlichen Pufferkapazitäten getestet.
Die Autoren zeigen, dass vor allem in lückenhaften Netzwerken und unter hoher Netzwerklast die intelligenteren Algorithmen, die über genaue Kenntnisse zukünftiger Kontakte und Wissen über die *Queue* Belastung verfügen wesentlich besser funktionieren. Allerdings reichen lokale Kenntnisse über die *Queues* auf den Knoten aus, globale Kenntnisse führen nicht zu wesentlichen Verbesserungen.

Analyse Sozialer Netzwerke für optimiertes Routing

In [DH07] stellen die Autoren mit *SimBet Routing* einen Algorithmus vor, der mithilfe von *Betweennes Centrality* und *Similarity* Maßen eine Auslieferungsgeschwindigkeit erreicht, die *Epidemic Routing* nahe kommt und den zum Vergleich herangezogenen Algorithmus *ProPHET Routing* übertrifft.
In der Veröffentlichung wird vorgeschlagen Techniken aus der Analyse sozialer Netzwerke

zum *Routing* in *DTNs* zu verwenden. Einerseits wird ein lokales (*egocentric*) *Betweenes Centrality* Maß definiert um Brücken zwischen *Clustern* erkennen und ausnutzen zu können, andererseits wird, basierend auf der Anzahl gleicher Nachbarn, ein Ähnlichkeitsmaß zwischen Knoten definiert. Knoten, die nach diesem Maß eine hohe Ähnlichkeit zum Zielknoten aufweisen sind mit hoher Wahrscheinlichkeit im gleichen *Cluster* und daher gute Relaisstationen. Der Algorithmus entscheidet beim Aufeinandertreffen zweier Knoten aufgrund der berechneten *Betweenes* und *Similarity* Maße für jede Nachricht jeweils welcher Knoten besser zur Weiterleitung geeignet ist.
Die Autoren evaluieren *SimBet Routing* mit Hilfe eines vom "Reality Mining Project" [MIT] zur Verfügung gestellten Datensatzes, der alle Bluetoothkontakte zwischen den Handys von 100 Studenten während eines 101 tägigen Experiments beinhaltet (siehe [MGC+07]). Durchgeführt werden zwei unterschiedliche Tests, beim Ersten sendet jeder Knoten an alle anderen Knoten eine Nachricht. Hier erreicht *SimBet Routing* eine Nachrichtenverzgerung, die nur 40% über dem von *Epidemic Routing* liegt und sendet dabei um den Faktor 33 weniger Nachrichten. Beim zweiten Test werden Nachrichten zwischen den am schwächsten verbundenen Knoten verschickt. Da *SimBet Routing* versucht zentralere Knoten zu erreichen liegt es hier klar vor *ProPHET*.

1.3 Einordnung der Arbeit in den Kontext

Von den vier vorgestellten Arbeiten sind die ersten beiden besonders wichtig für ISeek. Ähnlich wie in [MGC+07] werden Kontaktinformationen zur Identifizierung von Freunden / Freundverbindungen genutzt. Neben der grundsätzlichen Argumentation sind vor allem die Ergebnisse aus [MGC+07] zur Auswahl geeigneter Parameterbereiche sehr nützlich. Aus [RK02] stammt eine weitere Kernidee von ISeek. Die dort beschriebene Probabilistische Komponente des Hybridalgorithmus bildet entspricht in ihrer Idee in großen Teilen der von ISeek erschlossenen transitiven Umgebung, deshalb wird auch die Datenstruktur des *Attenuated Bloomfilters* in einer leicht abgewandelten Form übernommen. Im Gegensatz zu [RK02] arbeitet ISeek allerdings nicht auf *P2P Netzwerken*, sondern auf *MANETs*.
Die anderen beiden Veröffentlichungen zeigen alternative Ansätze im Bereich der *DTNs*, so basiert [JFP04] auf der Annahme, dass Kenntnisse über die Netzwerktopologie verfügbar sind. Solche Daten werden für soziale Netze aus Datenschutzgründen normalerweise nicht erhoben. Zwar baut ISeek für eine lokale Umgebung indirekt ebenfalls Topologiewissen auf, aber hier ergibt sich das transitive Wissen der über eine Verbindung erreichbaren Informationen aus den *Attenuated Bloomfiltern* und kann deshalb nicht mehr ohne weiteres auf einzelne Personen zurückgeführt werden. Der in [DH07] verwendete Ansatz stellt schließlich eine alternative Herangehensweise zu ISeek dar, auch hier wird versucht unter der Ausnutzung der Strukturen sozialer Netze einen Algorithmus mit vergleichbarer Effektivität und besserer Effizienz als *Flooding* zu implementieren, allerdings setzt [DH07] auf bekannte Netzwerkmaße zur Identifizierung von Brücken und zentralen Knoten, während ISeek eine Art "Directed Flooding"

implementiert.

Im Gegensatz zu den zitierten Arbeiten ist es nicht das Ziel von ISeek zu einer bestimmten Ressource zu *routen*, sondern in einem Netzwerk möglichst viele nahe gelegene Informationsträger zu finden, die über zu einer Anfrage passende Kenntnisse verfügen. Dieser Ansatz ist am ehesten mit der "Replica-Idee" von [RK02] vergleichbar und stellt eine Übermenge des klassischen "Ein Ziel Routings" im Sinne eines Zielrechners dar. ISeek kombiniert dazu die Ideen von [MGC+07] und [RK02] und überträgt sie auf einen neuen Kontext. Darüber hinaus werden die α–Parameter aus [MGC+07] weiterentwickelt und es wird ein komplett neuer Parameter, der λ–Wert eingeführt. Dieser wählt innerhalb eines vordefinierten Bereichs zusätzliche eigentlich zu schwache Kanten aus, um auch mit eher schwach verbundenen Graphen arbeiten zu können.

1.4 Glossar

- **Betweeness Centrality:**
  bezeichnet in der Graphenanalyse die relative Bedeutung eines Knotens in einem Netzwerk in Bezug auf die Anzahl der kürzesten Pfade zwischen beliebigen Paaren von Knoten, die diesen Knoten enthalten.

- **Bin:**
  wörtlich Gefäß oder Eimer bezeichnet in der Informatik häufig ein Feld, dem alle Werte innerhalb eines Intervalls zugeordnet werden. So könnte man beispielsweise eine Woche in 7 Bins aufteilen, wobei jede Bin für einen Tag steht.

- **Direct Contact:**
  bekannter Routingalgorithmus, der Datenübertragungen nur dann zulässt, wenn ein direkter Kontakt ohne Vermittler zwischen Sender und Empfänger besteht.

- **DTN:**
  (delay tolerant network) bezeichnet eine Unterkategorie der *MANETs*, bei denen zusätzlich eine besonders große Toleranz gegenüber Verzögerungen in der Nachrichtenauslieferung erwartet wird.

- **Flooding/Epidemic Routing:**
  bekannter Routingalgorithmus, bei dem Daten grundsätzlich von jedem Teilnehmer an alle seine Kontakte übertragen werden. Das Netzwerk wird so mit der Nachricht "geflutet".

- **Hashfunktion:**
  bezeichnet in der Informatik häufig eine Abbildung, die Objekte auf Zahlenwerte innerhalb eines bestimmten Intervalls abbildet. So könnte man beispielsweise die Tage einer Woche wie folgt hashen: Montag $\rightarrow$ 1, Dienstag $\rightarrow$ 2 ...

- **Linear Programming:**
  bezeichnet eine mathematische Methode, die eine Nutzensfunktion optimiert, wobei beschränkenden Bedingungen in Form linearer Gleichungen eingehalten werden müssen.

- **MANET:**
  (mobile adhoc network) bezeichnet sich selbstorganisierende mobile Funknetzwerke, wie sie zum Beispiel von Handys, Laptops etc. aber auch von nicht geostationären Satelliten gebildet werden.

- **Routing:**
  bezeichnet die Festlegung eines Übertragungsweges zwischen einem Sender und einem Empfänger in der Telekommunikation.

- **Scale-Free:**
  wird im Kontext der Graphenanalyse verwendet und bezeichnet Netzwerke, deren Knotengradverteilung einer Potenzfunktion folgt.

- **Similarity:**
  wörtlich Ähnlichkeit. Im Textzusammenhang handelt es sich um ein Ähnlichkeitsmaß für die Knoten des Routinggraphen.

Kapitel 2

Verwendete Datenstrukturen und deren Eigenschaften

2.1 Überblick

Ein *Bloomfilter* [Blo70] ist eine einfache Datenstruktur, die es ermöglicht Mengen kompakt zu repräsentieren und Mitgliedschaftsanfragen zu beantworten. Der Einsatz von *Bloomfiltern* kann zu *False Positives* führen, aber in vielen Einsatzgebieten, wie zum Beispiel dem Datenbank -und Netzwerkbereich überwiegen die Vorteile. *Bloomfilter* zeichnen sich hierbei besonders durch ihre geringe Größe und die Möglichkeit auf von *Bloomfiltern* repräsentierten Mengen, effizient mit booleschen Operatoren zu arbeiten, aus.

[BM02] : *"The Bloom filter principle: Wherever a list or set is used, and space is at a premium, consider using a Bloom filter if the effect of false positives can be mitigated."*

2.2 Implementierung

Ein *Bloomfilter* ist ein Bitarray der Länge m, bei dem zunächst alle Bits auf 0 gesetzt sind. Er verwendet k *Hashfunktionen* $h_1, h_2, .., h_k$, die idealerweise alle denkbaren Objekte gleichmäßig auf die Zahlen aus dem Intervall $(1, .., m)$ abbilden.

$$h_i : Objekt \rightarrow (1, .., m)$$

Um die Menge M=$\{a_1, a_2, ..a_n\}$ darzustellen, werden für alle $a \in M$ die Bits $h_i(a)$ $(1 <= i <= k)$ im *Bloomfilter* gesetzt (siehe Abb. 2.1). Sollte ein Bit bereits gesetzt sein, so wird es nicht weiter verändert. Ein Element a gilt als Mitglied der durch den *Bloomfilter* repräsentierten Menge falls alle Bits $h_i(a)$ $(1 <= i <= k)$ gesetzt sind. Hier kann es zu *False Positives* kommen, falls die betreffenden Bits von unterschiedlichen anderen Elementen $a \in M$ gesetzt wurden. *False Negatives* dagegen können nicht auftreten.

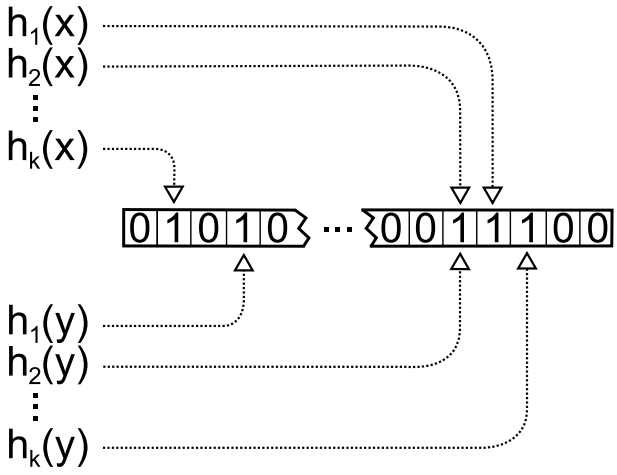


Abbildung 2.1: Die Elemente x und y werden mit einem *Bloomfilter* dargestellt. Dazu werden sie mit Hilfe von k *Hashfunktionen* $h1..hk$ auf den *Bloomfilter* abgebildet.

2.3 Bloomfilter Mathematik

Besonders wichtig für *Bloomfilter* ist die Berechnung der *False Positivrate*. Diese wird hier deshalb verkürzt hergeleitet. Die ausführliche Herleitung sowie weitere mathematische Eigenschaften von *Bloomfiltern* finden sich in [BM02].

Die Wahrscheinlichkeit eines bestimmten Bits nach dem Einfügen aller Elemente aus M noch 0 zu sein ist:

$$p' = (1 - \frac{1}{m})^{kn} \tag{2.1}$$

Definieren wir ρ als Anteil der 0 Bits nach dem Einfügen der Objekte gilt für den Erwartungswert:

$$E(\rho) = p' \tag{2.2}$$

Die Wahrscheinlichkeit eines *False Positives* f' ergibt sich als

$$f' = (1 - \rho)^k \approx (1 - p')^k \tag{2.3}$$

Und damit gilt schließlich:

$$f' \approx (1 - (1 - \frac{1}{m})^{kn})^k = (1 - p')^k \tag{2.4}$$

Ebenfalls interessant ist, dass unter der Voraussetzung, dass n und m bekannt sind

$$k = ln(2) * (\frac{m}{n}) \tag{2.5}$$

die optimale Anzahl von *Hashfunktionen* ist. Optimal insofern, als dass dieses k f' für feste n und m minimiert.

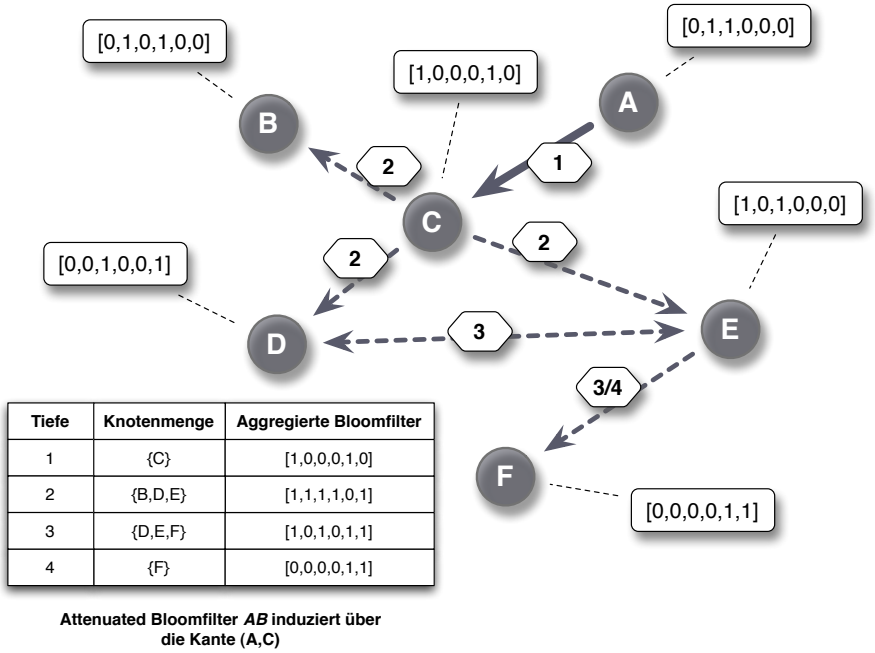


Abbildung 2.2: Die Grafik zeigt die Konstruktion des *Attenuated Bloomfilters* $AB^{\overline{AC}}$ zur Kante $\overline{AC}$ auf dem Knoten A. Die Richtung der Kanten und ihre Beschriftung zeigen an, auf welche *Bloomfilterlevel* sich eine Kante auswirkt und in welcher Richtung sie dabei genutzt wird. So wirkt sich die Kante $\overline{AC}$ auf den ersten *Bloomfilter* des Stacks aus, von C aus sind für das zweite Level die Knoten B, D, E erreichbar usw. Entscheidend für die Aufnahme eines Knotens in das i-te *Bloomfilterlevel* ist, dass er auf einem zyklenfreien Pfad der Länge i beginnend mit der Kante $\overline{AC}$ erreichbar ist. So kann zum Beispiel D über $\overline{ACD}$ und $\overline{ACED}$ erreicht werden. (siehe Algo. 3.3.1)

2.4 AttenuatedBloomfilter

Da *Bloomfilter* ein sehr einfaches aber mächtiges Konzept darstellen, findet man zahlreiche Erweiterungen, eine davon ist der *Attenuated Bloomfilter* [RK02].

Ein *Attenuated Bloomfilter* der Tiefe d ist ein Stack von d normalen *Bloomfiltern*, welcher sich dazu eignet transitive Beziehungen in Netzwerken darzustellen. Hierzu wird in dem Netzwerkgraphen jede Kante jedes Knotens mit einem *Attenuated Bloomfilter* versehen.
Abbildung 2.2 zeigt die Konstruktion des *Attenuated Bloomfilter* $AB^{\overline{AC}}$, der auf dem Knoten A der Kante $\overline{AC}$ zugeordnet ist. Der erste *Bloomfilter* des Stacks enthält Informationen über den Nachbarn ersten Grades C, der i-te ($i \in (1..d)$) *Bloomfilter* ist die Vereinigung der Informationen aller Knoten, die auf einem zyklenfreien Pfad der Länge i liegen, dessen erste Kante $\overline{AC}$ ist. Die Autoren von [RK02] berechnen die Wahrscheinlichkeit eine bestimmte Information I über eine Kante mit zugehörigem *Attenuated*

Bloomfilter AB zu erreichen wie folgt:

$$\sum_{i=1}^{d}(g(AB[i], I)) * \frac{1}{i+1} \tag{2.6}$$

wobei $AB[i]$ der i-te Bloomfilter des Stacks ist und

$$g(AB[i], I) = \begin{cases} 1 & \text{falls } I \in AB[i] \\ 0 & \text{sonst} \end{cases} \tag{2.7}$$

.

Hiervon leitet sich auch der Name *Attenuated Bloomfilter* ab, da tiefere Level im Verhältnis zu höheren Leveln abgeschwächt werden. Für uns ist im folgenden allerdings nur die transitive Eigenschaft der *Attenuated Bloomfilter* von Interesse.

Kapitel 3

Simulator

3.1 Überblick

Im Rahmen des Bachelorprojektes haben wir uns mit der Funktionsweise und Simulation von *DTNs* auseinandergesetzt. Um ein besseres Verständnis für diesen speziellen Netzwerktyp zu entwickeln wurde als praktischer Teil des Projektes ein eigener Netzwerksimulator implementiert, der speziell auf den Umgang mit *DTNs* zugeschnitten ist. Der Simulator besteht aus einem konsolenbasierten Kern, der die Simulation durchführt und die Ergebnisse in Form eines Reports in eine *MySql* Datenbank speichert und einer graphischen Oberfläche (Abb. 3.1), die diese Ergebnisse für Tetstgraphen etc. darstellen kann um ein intuitiveres Verständnis der Algorithmen zu ermöglichen.

3.2 Wesentliche Merkmale

Die im folgenden erläuterten Merkmale (Realsimulation, Eventbasiertheit, Determinismus) spiegeln die grundlegenden Entwurfsentscheidungen wieder, die zu Beginn des Bachelorprojektes getroffen wurden. Gleichzeitig definieren sie die wesentlichen Charakteristika des Simulators und ermöglichen eine grobe Vorstellung von dessen Funktionsweise.

- Realsimulation: es wird aufgrund einer Initialisierungsdatei eine exakte Simulation durchgeführt. Kanten werden nicht durch Wahrscheinlichkeiten bestimmt sondern zu festen Zeitpunkten hinzugefügt und entfernt.
- Eventbasiert: der Simulator arbeitet eventbasiert, so dass er einerseits Zeiträume in denen nichts passiert überspringen kann und andererseits mit einem kontrollierenden Thread auskommt, der aufgrund der Events die richtigen Knoten informiert.
- Deterministisch: alle Zufallsentscheidungen, sei es in der Initialisierungsphase, bei der Auswahl des nächsten Simulationsevents oder in den Algorithmen selbst werden vom gleichen Zufallsgenerator getroffen. Da dessen Initialisierungswert mit den

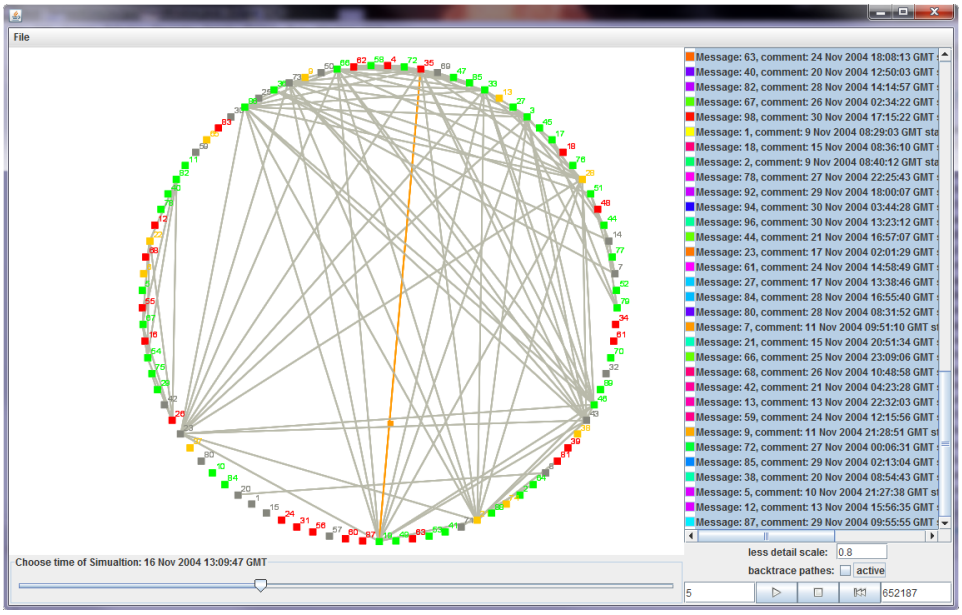


Abbildung 3.1: Graphische Oberfläche des Simulators. Dargestellt wird ein Teil des MIT Cell Tower Datensatzes. Auf der rechten Seite kann man die unterschiedlichen Nachrichten erkennen, am unteren Rand das Benutzerinterface, das beliebige Zugriff in den Zeitstrom ermöglicht. Auch ein kontinuierliches "Abspielen" der Simulation ist möglich.

erzeugten Reports gespeichert wird, besteht die Möglichkeit eine Simulation exakt zu wiederholen. Dies ist insbesondere für Debugging oder Verständnissfragen sehr sinnvoll, da man tatsächliche oder scheinbare Fehler exakt reproduzieren kann. Außerdem ist es so problemlos möglich Algorithmen mit verschiedenen Parametern nach der gleichen Initialisierungsphase auszuführen und ihre Ergebnisse zu vergleichen.

Auf den vom Simulator bereitgestellten Schnittstellen werden dann die Algorithmen definiert. Ein Algorithmus reagiert dabei auf die vom Simulator aufgerufenen Ereignisse, so würde beispielsweise ein *Floodingalgorithmus* auf Knoten A vom Simulator über eine neu gebildete Kante $\overline{AB}$ informiert werden und als Folge daraus eine Übertragung über diese Kante einleiten. Die Verwaltung der Übertragung wird dann wieder vom Simulator selbst vorgenommen.

3.3 Initialisierung der Simulation

Der Simulator muss vor der eigentlichen Simulation geeignet initialisiert werden. Im Falle von ISeek bedeutet dies, dass alle Knoten mit ihren eigenen *Bloomfiltern* und den *Attenuated Bloomfiltern* für ihre Kanten ausgestattet werden müssen. Der Algorithmus 3.3.1 wird verwendet um die *Attenuated Bloomfilter* zu erzeugen. Hierzu wird in der Methode *createAttenuatedBloomfilter* für jede Kante ein *Attenuated Bloomfilter* angelegt, der

dann mit Hilfe der Methode *travers* gefüllt wird. *Travers* führt dabei eine Art rekursiver Tiefensuche aus.
Soll der *Attenuated Bloomfilter* für eine Kante E des Knotens K erzeugt werden, so wird *travers* auf dem über E mit K verbunden Knoten K' von *createAttenuatedBloomfilter* aufgerufen. K' wird zum aktuellen Pfad hinzugefügt (Zeile 2). Falls die maximale Tiefe noch nicht erreicht ist wird anschließend für jede Kante E' von K', die nicht zu einem Knoten zurückführt, der bereits auf dem aktuellen Pfad liegt (Zeile 4 – 7) der über E' erreichbare Knoten in das passende Level des *Attenuated Bloomfilters* eingetragen (Zeile 8) und *travers* wird auf diesem Knoten aufgerufen.
Wird die maximale Tiefe erreicht geht der Algorithmus einen Schritt zurück, daher wird der letzte Eintrag vom aktuellen Pfad entfernt und die Tiefe um eins verringert (Zeile 14 – 15). Die Rekurrsion endet hier, als nächstes werden die Geschwister des vorletzten Knotens abgearbeitet ... bis schließlich alle Kinder von K' abgearbeitet wurden.

Algorithm 3.3.1: AttenuatedCreator

```
1  createAttenuatedBloomfilter (Node node, Integer max_Depth)
2  travers (Node node, AttenuatedBloomfilter ab, NodeSet path, Integer
   actual_Depth, Integer max_Depth)
```

Procedure `createAttenuatedBloomfilter`

Input: Node node, Integer max_Depth
Output: AttenuatedBloomfilter for each edge of the node

```
1  begin
2      result ← new AttenuatedBloomSet
3      foreach RoutingEdge edge of node do
4          ab ← new AttenuatedBloomfilter
5          path ← new NodeSet
6          node2 ← new Node
7          actual_Depth ← new Integer
8          path ∪ node
9          node2 ← edge.oppositeNode(node)
10         actual_Depth ← 1
11         ab ← travers (node2, ab, path, actual_Depth, max_Depth)
12         result ∪ ab
13     end
14 end
```

Procedure `travers`

Input: Node node, AttenuatedBloomfilter ab, NodeSet path,
Integer actual_Depth, Integer max_Depth
Output: AttenuatedBloomfilter

```
1  begin
2      path ∪ node
3      if actual_Depth < max_Depth then
4          foreach RoutingEdge edge of node do
5              node2 ← new Node
6              node2 ← edge.oppositeNode(node)
7              if node2 ∉ path then
8                  ab[actual_Depth] ∪ node2
9                  actual_Depth ← actual_Depth + 1
10                 ab ← travers (node2, ab, path, actual_Depth, max_Depth)
11             end
12         end
13     end
14     path ← path \path.lastElement()
15     actual_Depth ← actual_Depth - 1
16 end
```

Kapitel 4

Algorithmus – ISeek

4.1 Überblick

Der hier getestete *Routing Algorithmus ISeek* ist eine Implementierung und Konkretisierung der in [BIW08] vorgestellten Idee. Umgesetzt wurde bisher allerdings nur der Transport der Anfrage zum Ziel, die Implementierung des intelligenten Rücktransportprotokolls sowie ein dynamisches Aktualisieren der *Bloomfilter* verbleiben als zukünftige Arbeiten. Eine weitere grundlegende Designentscheidung war die Festlegung auf *Attenuated Bloomfilter*, die bereits von den Autoren von ISeek bevorzugte der drei in [BIW08] vorgestellten Möglichkeiten. Der Algorithmus wird in der Terminologie des Simulators im folgenden auch als "Device Algorithmus" bezeichnet, da lokal auf jedem Knoten/*Device* eine Instanz von ihm läuft.

4.2 Grundidee

Auf jedem Knoten, der Teil eines ISeek Netzwerkes, ist läuft eine Instanz des ISeek Algorithmus. Diese verwaltet die Kanten des Knoten und einen *Bloomfilter*, in den alle Informationen, für die sich der Knoten als Ziel eignet *gehasht* werden. Aus den Kanten wählt der Algorithmus eine Teilmenge aller Kanten, die besonders gut zum Weiterleiten von Nachrichten geeignet ist. Zu jeder dieser Kanten wird ein *Attenuated Bloomfilter* gespeichert. Dieser *Attenuated Bloomfilter* wird auf dem durch die ausgewählten Kanten definierten Teilnetzwerk erzeugt und fasst in seiner k-ten Ebene alle über diese Kante in k Schritten erreichbaren *Bloomfilter* der Knoten zusammen. Außerdem werden auf jedem Gerät bereits erhaltene und selbst erzeugte Nachrichten, die weitergeleitet werden sollen, gespeichert. Zu jeder dieser Nachrichten gehört eine Zieldefinition, die aus den *Hashwerten* der Information besteht, die ein Zielkonten in seinem *Bloomfilter* enthalten soll.

Wird nun eine der ausgewählten Kanten aktiv, das heißt kommen zwei Geräte in

Kommunikationsreichweite, so überprüft jeder der beiden Algorithmen, ob sein lokaler Nachrichtenspeicher eine Nachricht enthält, deren Zieldefinition zu einer der Ebenen des *Attenuated Bloomfilters* der Kante passt. Liegt ein solcher Treffer vor bedeutet dies, dass entweder der Nachbarknoten oder ein innerhalb der maximalen Transitivität erreichbarer dem Nachbarknoten bekannter Knoten als Ziel in Frage kommt und die Nachricht wird übertragen.

4.3 Erweiterungen

Dieser grundlegende Algorithmus wurde im Laufe des Projektes um drei wichtige Features erweitert.

- Verbesserung des Nachrichtenspeichers
- Aktionsradius für Nachrichten
- *Direct Contact* Subprotokoll

Der Nachrichtenspeicher wurde so erweitert, dass er auch die zum *Routing* einer Nachricht geeigneten Kanten speichert. Auf diese Art muss nicht bei jedem Kontakt für alle Nachrichten berechnet werden, ob eine Übertragung sinnvoll ist. Außerdem erlaubt eine solche Verwaltung der Transportwege zusätzliche Verbesserungen, wie die Verhinderung von Mehrfachübertragungen über die gleiche Kante und die Unterdrückung von Rückübertragungen an Knoten, von denen eine Kopie der Nachricht ausgegangen ist. Das zweite wichtige Feature basiert auf einem in [RK02] vorgestellten Ansatz und lässt sich am besten als Aktionsradius von Nachrichten beschreiben. Wie sich in frühen Tests gezeigt hat kann es auf Graphen mit hohen Füllraten in den tiefsten Ebenen der *Bloomfilter* zu einem wiederholten Weiterleiten aufgrund von *False Positives* kommen. Das heißt eine Nachricht wird nur aufgrund eines *False Positives* an einen Nachbarknoten weitergeleitet und stoppt auf diesem dann nicht, sondern erkennt wieder in der tiefsten Ebene eines *Bloomfilters* einen vermeintlichen Treffer usw. eine solche Nachricht verbreitet sich dann ohne Legitimation durch die *Bloomfilter* unkontrolliert im Netzwerk. Um diesem Phänomen entgegenzuwirken startet jede Nachricht mit der Erlaubnis sich über alle *Bloomfilterebenen* auszudehnen. Diese Erlaubnis wird aber bei jeder Übertragung dekremtiert, so dass eine Nachricht z.B. nach einer Weiterleitung nicht mehr aufgrund von Informationen im tiefsten Boomfilterlevel versendet wird. Diese Einschränkung ist legitim, da der Zielknoten, der auf dem 1. Knoten einen Treffer im tiefsten Boomfilterlevel ausgelöst hat nach einer Übertragung im zweit tiefsten Boomfilterlevel stehen muss. Wird dagegen ein echter Zielknoten erreicht, so wird die Übertragungserlaubnis zurückgesetzt, da von diesem validen Punkt aus eine neue Suche gestartet werden kann. Als letzte Verbesserung wurde schließlich *Direct Contact* als Subprotokoll aufgenommen, so dass zwei Knoten die sich treffen bei einer direkten Übereinstimmung des *Bloomfilters* eines Knotens mit einer der gespeicherten Nachrichten des anderen Knotens Daten

übertragen dürfen unabhängig davon ob die Kante zwischen den Knoten zu den verwalteten Kanten gehört oder nicht. Diese Optimierung wurde interessant, als sich gezeigt hat, dass *Direct Contact* auf dem sehr dichten Cell Tower Datensatz (siehe Abb. 8.1) in der Lage war ISeek nach einigen Tagen zu übertreffen.

4.4 Implementierung

Da der Algorithmus (siehe Algorithmus 4.4.1) auf NetworkSim (siehe Kapitel 3) getestet wird, muss er an das Eventmodell des Simulators angepasst werden. Er reagiert dabei auf die folgenden Events:

- edgeAdded: wenn eine Kante dazukommt
- createMessage: zum Startzeitpunkt einer Nachricht ausgeführt
- transmissionAborted: Reaktion des Simulators auf ein *edgeRemoved* Ereigniss auf einer aktiven Kante
- messageTransmitted: Reaktion des Simulators auf die erfolgreiche Übermittlung einer Nachricht.

Algorithm 4.4.1: DeviceAlgorithm

Data: Node node, TaskSet tasks: (TargetNode x Message), MessageSet messages, Integer max_Depth

```
1  edgeAdded ()
2  createMessage ()
3  transmissionAborted ()
4  messageTransmitted (Message message)
5  forwardTransmission ()
6  getTargets (Message message)
```

Während das *edgeAdded* Ereigniss auf den beiden Kantenkonten aufgerufen wird, reicht es beim *createMessage* Ereigniss aus den Knoten zu informieren, der die neue Nachricht erstellen soll. Die *transmissionAborted* und *messageTransmited* Ereignisse werden schließlich auf beiden Knoten der Kante und all ihren direkten Nachbarn aufgerufen. Dies ist notwendig, da die beiden Konten durch das Beenden einer Übertragung für ihre Nachbarn wieder erreichbar werden.

Der implementierte Algorithmus 4.4.1 lagert einen Großteil der Arbeit in die Hilfsmethode *forwardTransmission* aus, die alle zur Verfügung stehenden Übertragungsoptionen prüft und falls möglich eine Übertragung einleitet. Diese Methode und die konzeptionell wichtige *messageTransmitted* Methode werden daher im folgenden näher beschrieben.

Wie oben erläutert trifft ein *messageTransmitted* Ereigniss auf dem Sender, dem Empfänger und allen Nachbarknoten des Senders und Empfängers ein. Der Sender entfernt die erfolgreich übermittelte Nachricht aus seinem Aufgabenspeicher (Zeile 2-4), während der Empfänger, falls er die erhaltene Nachricht bereits kennt, den Sender aus dem Aufgabenspeicher streicht (Zeile 6–8). Ist die Nachricht dem Empfänger bisher unbekannt, so wird der Aufgabenspeicher um die zur Übertragung der Nachricht geeigneten Kanten erweitert und die Nachricht dem Nachrichtenspeicher hinzugefügt (Zeile 9–15). Ist der Empfänger zusätzlich ein echter Zielknoten, der die gesuchte Information enthält wird die Übertragungserlaubnis der Nachricht zurückgesetzt (Zeile 10–12). Für alle Knoten, die das Ereignis empfangen wird anschließend mit *forwardTransmission* geprüft, ob sich durch die veränderte Situation neue Übertragungsmöglichkeiten ergeben haben (Zeile 17). *forwardTransmission* testet zuerst ob der Knoten selbst zurzeit überhaupt übertragen kann (was nicht der Fall ist falls er bereits sendet oder empfängt), um dann einen dazu passenden empfangsbereiten Nachbarn zu suchen (Zeile 2–6). Ist ein solcher Nachbar gefunden wird überprüft, ob es sich bei der Kante um eine der zum *Routing* ausgewählten Kanten handelt und ob der Aufgabenspeicher eine passende Nachricht zu diesem Knoten enthält. Falls ja wird die Übertragungserlaubnis dekremtiert und der Simulator mit der Übertragung zwischen den beiden Knoten beauftragt (Zeile 8–14). Falls es sich nicht um eine der ausgewählten Kanten handelt wird das *Direct Contact* Subprotokoll genutzt und der Algorithmus überprüft, ob eine Nachricht im Nachrichtenspeicher zu dem *Bloomfilter* des Empfängers passt (Zeile 15–21). Die Übertragungserlaubnis wird in diesem Fall auf -1 gesetzt (Zeile 18) um eine Weiterverbreitung der Nachricht zu verhindern.

Procedure `edgeAdded`

```
1 begin
2 |  forwardTransmission ()
3 end
```

Procedure `createMessage`

```
1  begin
2  |  message ← new Message
3  |  message.level ← max_Depth
4  |  tasks ∪ getTargets(message)
5  |  messages ∪ message
6  |  forwardTransmission ()
7  end
```

Procedure `transmissionAborted`

```
1  begin
2  |  forwardTransmission ()
3  end
```

Procedure `messageTransmitted`

Input: Message message

```
1  begin
2  |  if node = message.Sender then
3  |  |  tasks ← tasks \ (message.Receiver, message)
4  |  end
5  |  else if node = message.Receiver then
6  |  |  if node already has message then
7  |  |  |  tasks ← tasks \ (message.Sender, message)
8  |  |  end
9  |  |  else
10 |  |  |  if node is real target then
11 |  |  |  |  message.level = max_Depth
12 |  |  |  end
13 |  |  |  tasks ∪ getTargets(message)
14 |  |  |  messages ∪ message
15 |  |  end
16 |  end
17 |  forwardTransmission ()
18 end
```

Procedure `getTargets`

Input: Message message

```
1  begin
2      result ← new TaskSet
3      foreach AttenuatedBloomfilter ab of node do
4          level ← new Integer
5          level ← ab.firstMatch(message)
6          if level != -1 && level < message.level then
7              result ∪ (ab.targetNode, message)
8          end
9      end
10 end
```

Procedure `forwardTransmission`

```
1  begin
2      if node is idle then
3          foreach Edge edge of node do
4              node2 ← new Node
5              node2 ← edge.oppositeNode(node)
6              if node2 is idle then
7                  message ← new Message
8                  if edge is routingEdge then
9                      if ∃(n, m) ∈ tasks : n = node2 then
10                         message ← m
11                         message.level ← message.level-1
12                         sendMessage(message,node2)
13                     end
14                 end
15                 else
16                     if ∃m ∈ messages : match(m, node2.bloom) then
17                         message ← m
18                         message.level ← -1
19                         sendMessage(message,node2)
20                     end
21                 end
22             end
23         end
24     end
25 end
```

Kapitel 5

Datensätze

5.1 Überblick

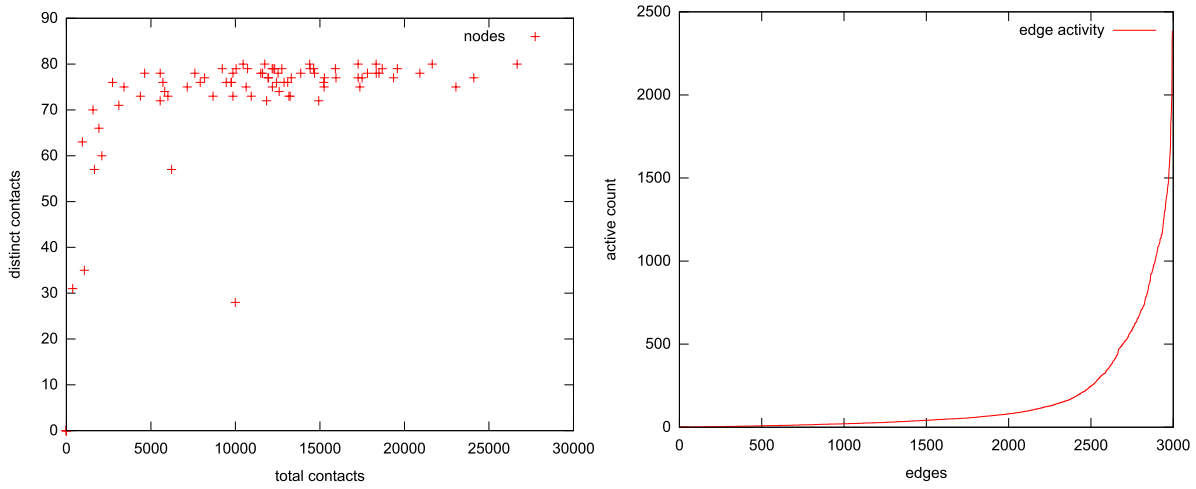
Um eine realistische Simulation zu ermöglichen benötigen wir Daten über die "Personenkontakte" einer sozial verbundenen Gruppe, über einen mehrwöchigen Zeitraum. Solche Datensätze sind aufgrund der damit verbundenen Datenschutzbedenken schwer zu erzeugen und auch fertige Datensätze stehen nur in geringer Anzahl zur Verfügung. Verwendet werden, der auch von einigen der Eingangs vorgestellten Wissenschaftlichen Arbeiten genutzte "MIT Reality Mining", sowie der "IBM Watson" Datensatz. Beide Datensätze sind auf [Cra] öffentlich zugänglich. Obwohl alle drei Quellen versuchen das gleiche Verhalten abzubilden, sind sie recht unterschiedlich. Der MIT Cell Tower Datensatz ist der mit Abstand dichteste der drei. In der Mitte liegen die IBM -Aufzeichnungen und schließlich folgen mit großem Abstand die MIT Bluetooth Daten.

5.2 MIT Reality Mining

Hier wurden 100 Nokia 6600 Handys mit einer speziellen Software ausgestattet, die es ermöglicht Daten über Bluetoothkontakte, Handynutzung und den genutzten Sendemasten (genauer gesagt dessen Antenne) zu sammeln. Die Daten wurden während des Studienjahres 2004/2005 von 75 Angehörigen des "MIT Media Laboratory" und 25 Mitgliedern der benachbarten "MIT Sloan Business School" erhoben. Insgesamt sind so 350.000 Stunden (40 Jahre) zusammenhängender Daten über menschliches Verhalten und Interaktion verfügbar. Für die Versuche verwenden wir die beiden daraus extrahierten Subdatensätze, die auch in [Isl08] eingesetzt werden.

5.2.1 MIT Cell Tower Datensatz

Der Cell Tower Datensatz basiert auf den gesammelten Daten über Sendemastantennen und besteht aus insgesamt 89 Knoten, von denen zwischen dem 01.11.04 um 01:52:44



(a) Knotenkontakte (b) Kantenkontakte

Abbildung 5.1: Dargestellt wird für jeden Knoten die Anzahl der unterschiedlichen Kontakte vs. Gesamtzahl der Kontakte. Und für jede Kante die Anzahl der Kontakte insgesamt. Die Daten beziehen sich auf den MIT Cell Tower Datensatz Abb. 8.1 und vermitteln einen Eindruck von der Konnektivität des Graphen.

und dem 01.12.04 um 09:00:26 jeder Kontakt auf wenige Sekunden genau festgehalten ist. Der Monat November wurde für die Experimente von [Isl08] gewählt, da es sich um den aktivsten Monat des Datensatzes handelt, der aber trotzdem repräsentativ für alle anderen Monate ist. Daher ist er auch hier bestens geeignet.

Analysiert man den Datensatz näher, so fallen einem vor allem die auch in [MGC+07] beschriebenen wochentags und uhrzeitabhängigen Muster und Wiederholungen auf. Die Wochenenden sind klar zu erkennen und durch deutlich geringere Aktivitäten gekennzeichnet. Um einen Eindruck von der wechselnden Aktivität der Kanten zu bekommen, hilft ein Blick auf Abb. 8.1. Hier steht jedes Rechteck für einen Tag, wobei die vom Rechteck eingenommene Fläche der Gesamtzahl der Kanten entspricht. Für jeden Tag sind drei Schichten dargestellt, von denen die oberste für Kanten steht die gegenüber dem Vortag neu hinzugekommen sind. Die mittlere Schicht repräsentiert Kanten, die sowohl am Vortag als auch an diesem Tag aktiv waren und die letzte Schicht schließlich steht für alle Kanten, die am Vortag aktiv waren, es aber jetzt nicht mehr sind. Dargestellt wird der Zeitraum von Montag dem 01.11.04 (links oben) bis Mittwoch dem 01.12.04 (rechts unten).
Neben den erwähnten Mustern ist vor allem der starke Wechsel der aktiven Kanten auffallend. Nur knapp mehr als die Hälfte der Kanten bleibt unter der Woche über zwei aufeinander folgende Tage aktiv, während die restlichen Kanten ihren Status verändern.

Außerdem wird deutlich, dass die 89 Knoten zwar im Schnitt Kontakt zu 60-80 an-

deren Knoten haben Abb. 5.1 a.), aber der Großteil dieser Kontakte nur sehr schwach ausgeprägt ist Abb. 5.1 b.). So sind allein 2,6 % der Kanten Knotenpaaren zuzuordnen, die sich während der gesamten Aufzeichnung nur einmal sehen und über 600 der 2995 Kanten werden durch maximal 10 Kontakte gebildet. Andererseits gibt es sehr aktive Kanten mit bis zu 2477 Kontakten.

5.2.2 MIT Bluetooth Datensatz

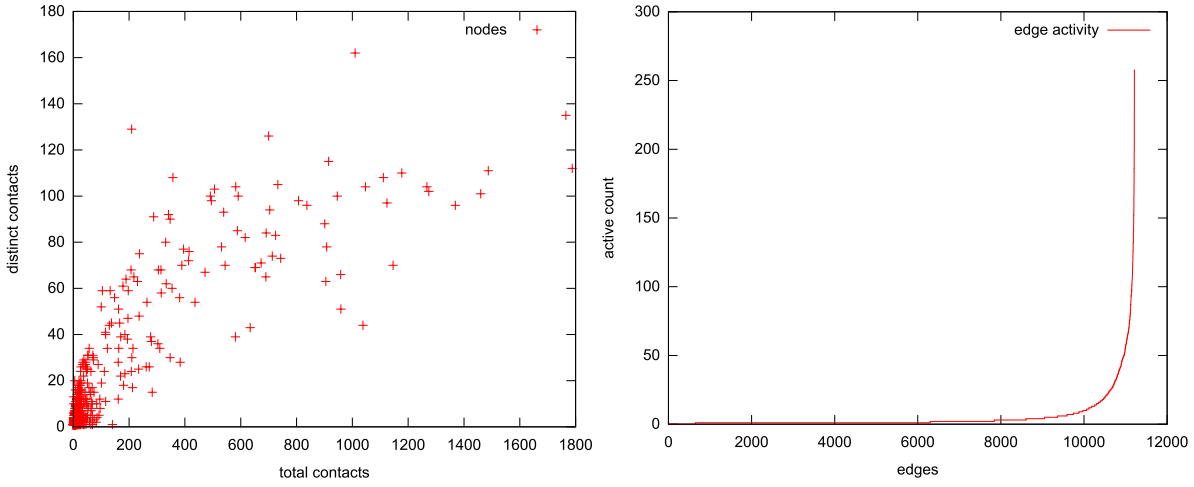
Der MIT Bluetooth Datensatz basiert auf den durch die Handys protokollierten Bluetoothkontakten. Wie Eingangs erwähnt handelt es sich hier um den am schlechtesten verbundenen Graphen, was sich auch direkt im Verhältnis der 1785 Knoten zu nur 11213 Kanten wiederspiegelt. Wir verwenden hier das gleiche Setup wie für den Cell Tower Datensatz, der Simulationszeitraum ist also wiederum analog zu [Isl08] der November 2004.

Bei genauerer Betrachtung können auch hier die wochentags und zeitabhängigen Muster aus [MGC+07] beobachtet werden. Betrachten wir hier Abb. 8.2, die vom Aufbau her Abb. 8.1 entspricht fällt sofort auf, dass relativ gesehen weniger Kanten aktiv sind. Dies ist vor allem insofern gravierend, da das Kanten/Knoten Verhältnis sowieso schon viel schlechter ist. Ansonsten ist auch hier eine deutliche Abnahme an den Wochenenden, sowie eine starke Variation der aktiven Kanten zu beobachten. Da der zugrunde liegende Zeitraum der gleiche ist wie beim Cell Tower Datensatz und zumindest eine Teilmenge der Knoten identisch ist, entspricht dies auch den Erwartungen.

Schaut man sich Abb. 5.2 sieht man sofort einige wesentliche Unterschiede zum MIT Cell Tower Datensatz. Während in Abb. 5.1 die Anzahl der unterschiedlichen Kontakte pro Knoten bei nahezu allen Knoten konstant bei etwa 80 liegt, ergibt sich hier eher ein "Kegel" mit einer massiven Häufung im niedrigen Bereich. Die meisten Knoten haben unter 40 unterschiedliche Kontakte und unter 200 Kontakte insgesamt. Im Vergleich dazu sind es im MIT Cell Tower Datensatz etwa 80 unterschiedliche Kontakte (mehrere 10.000 insgesamt). Außerdem fällt auf, dass die Aktivität der Kanten noch ungleicher verteilt ist als im MIT Cell Tower Datensatz. So sind über 10.000 der 11213 Kanten 10 mal oder seltener aktiv. Der höchste Aktivitätswert liegt bei 258 (zum Vergleich 2477 beim MIT Cell Tower Datensatz).

5.3 IBM Watson Datensatz

Der letzte Datensatz stammt schließlich von IBM. Hier wurde in drei Gebäuden des "Watson Research Centers" mit Hilfe des *SNMP Netzwerkprotokolls* aufgezeichnet welches *Wireless Device* (meist Laptops) zu welchem Zeitpunkt an welchem *Acess Point* angemeldet war. Aufgezeichnet wurde vom 20.07.2002 bis zum 17.08.2002 mit einer Auflösung von 5 Minuten. Auch hier werden wieder die in Nuntifix [Isl08]



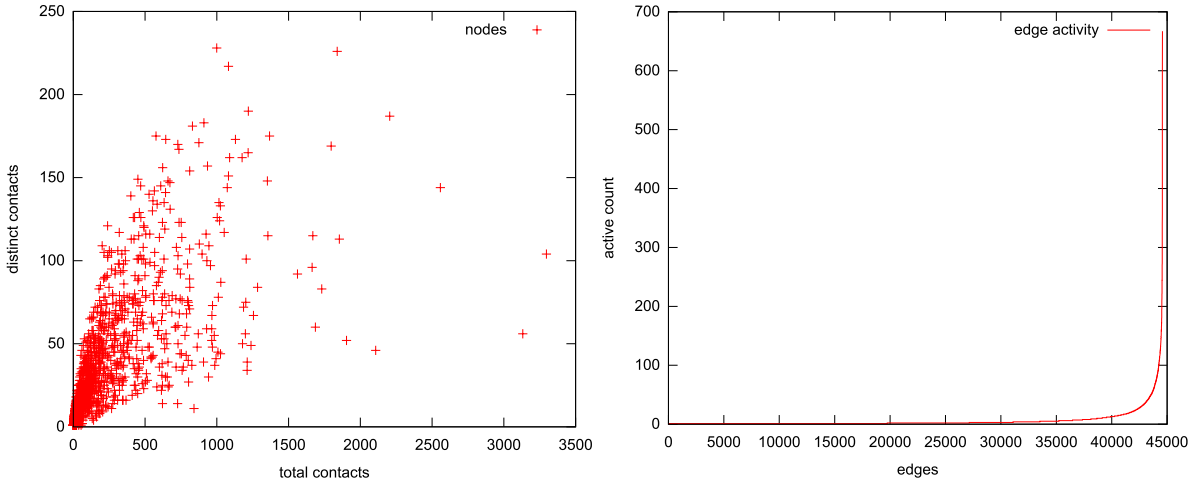
(a) Knotenkontakte (b) Kantenkontakte

Abbildung 5.2: Dargestellt wird für jeden Knoten Anzahl der unterschiedlichen Kontakte vs. Gesamtzahl der Kontakte. Und für jede Kante die Anzahl der Kontakte insgesamt. Die Daten beziehen sich auf den MIT Bluetooth Datensatz Abb. 8.2 und vermitteln einen Eindruck von der Konnektivität des Graphen.

verwendeten Datensätzen genutzt. Um die IBM Daten in ein gemeinsames Format mit den MIT Daten zu überführen wurde nur das größte der 3 Gebäude (der größte *Cluster*) verwendet. Um die Auflösung zu erhöhen wird angenommen, dass sich die Geräte während der 5 Minuten Intervalle konstant verhalten. In den wenigen Fällen, in denen sich das Verhalten zwischen zwei Intervallgrenzen verändert wird angenommen, dass der Wechsel nach 2,5 Minuten stattgefunden hat.

Der IBM Graph stellt in allen Bereichen einen Mittelweg zwischen den beiden MIT Datensätzen dar. So hat er mit 1351 Knoten wesentlich mehr Teilnehmer als der MIT Cell Tower Graph, gleichzeitig aber noch rund 400 weniger als der Bluetooth Graph. Die Anzahl der Kanten liegt mit 44574 in einem deutlich besseren Verhältnis zur Anzahl der Konten als bei Bluetooth und entspricht mit ungefähr 1:32 ziemlich exakt der des MIT Cell Tower Datensatzes.

Wie in Abb .8.3 erkennbar ist, ist allerdings das Verhältnis der aktiven Kanten zu den inaktiven Kanten noch schlechter als im Bluetooth Datensatz, so dass die Konnektivität insgesamt zwischen der, der beiden MIT Datensätzen liegt (wesentlich besseres Kanten/Knoten Verhältnis als Bluetooth aber etwas schlechteres aktive Kanten/ inaktive Kanten Verhältnis). Ansonsten können auch hier die wochentagsabhängigen Muster leicht erkannt werden. Die Wochenenden fallen besonders schwach aus, was vermutlich daran liegt, dass durch die fest installierten *Acess Points* in den Gebäuden des "Watson Research Centers" nur Kontakte in den Gebäuden und damit bei der Arbeit aufgezeichnet werden können. Betrachtet man weiterhin Abb. 5.3 sieht man, dass die Verteilung



(a) Knotenkontakte (b) Kantenkontakte

Abbildung 5.3: Dargestellt wird für jeden Knoten die Anzahl der unterschiedlichen Kontakte vs. Gesamtzahl der Kontakte. Und für jede Kante die Anzahl der Kontakte insgesamt. Die Daten beziehen sich auf den IBM Watson Datensatz Abb. 8.3 und vermitteln einen Eindruck von der Konnektivität des Graphen.

zwischen schwachen und starken Knoten wieder eher kegelförmig ausfällt und zumindest in der Anzahl der unterschiedlichen Kontakte dicht an das Niveau des MIT Cell Tower Datensatzes herankommt, allerdings bei den Kontakten insgesamt, wenn gleich besser als bei Bluetooth, immer noch weit hinter dem Cell Tower Datensatz liegt. Ebenfalls interessant ist, dass in diesem Datensatz die Knoten mit den höchsten "Distinct Contact" Werten (> 200) auftreten. Die Verteilung der Kantenaktivität ist schließlich wieder ungefähr auf dem Niveau des Bluetooth Datensatzes und damit auch extrem unsymmetrisch. So sind über 39.000 Kanten höchstens 10 mal aktiv während die aktivste Kante auf 667 Kontakte kommt.

Kapitel 6

Versuchsaufbau

6.1 Vorbereitung der Datensätze

Die in den MIT Datensätzen enthaltenen Kontaktdaten reichen vom 01.11.2004 um 01:52:44 bis zum 01.12.2004 um 09:00:26. Um störende Einflüsse durch "halbe" Tage zu eliminieren beschränken wir uns auf den Zeitraum vom 02.11.2004 um 00:00:00 bis zum 30.11.2004 um 24:00:00. Aufgrund der gleichen Argumentation wird der vom 19.07.2002 um 22:00:10 bis zum 18.08.2002 um 22:04:58 reichende IBM Datensatz auf den Zeitraum zwischen dem 20.07.2002 00:00:00 und dem 17.08.2002 24:00:00 beschnitten. Insgesamt stehen also jeweils 29 Tage für die Simulation zur Verfügung. Das Zeitfenster beginnt und endet bei den MIT Daten an einem Dienstag und umfasst vier Wochenenden. Die IBM Daten beginnen an einem Samstag und überdecken ebenfalls vier Wochenenden, allerdings sind fünf Samstage enthalten.

Für die Simulation müssen den Kanten der Graphen α–Werte zugeordnet werden. Da, wie in Abb. 8.1-8.3 ersichtlich ist, die Aktivität von Kanten zwischen zwei aufeinander folgenden Tagen stark schwankt und an Wochenenden auf ein sehr niedriges Niveau zurückgeht, werden die α–Werte aufgrund von 10 zufällig aus dem Simulationszeitraum ausgewählten Tagen berechnet. Dadurch ergibt sich für die statische Initialisierung ein realistischer Wert, der die wochentags abhängigen Muster des Datensatzes wiedergibt. Mit diesen Werten wird dann der *Attenuated Creator Algorithmus* (Algo. 3.3.1) ausgeführt, der für jeden Knoten *Attenuated Bloomfilter* der Tiefe $d \in N$ für alle zum *Routing* ausgewählten Kanten ($\alpha - Kanten \cup \lambda - Kanten$) anlegt.

6.2 Spezifikation der Simulationsparameter

6.2.1 Übersicht

Insgesamt gibt es neben den fest gewählten *Bloomfiltern* und Nachrichten noch drei weitere variable Parameter (siehe Tab. 6.1) Diese sind so gewählt, dass sie sich gegenseitig

Tiefe (d)	**Alpha (α)**	**Lambda (λ)**
2, 3, 4	70, 80, 90	30, 40

Tabelle 6.1: Für die unterschiedlichen Simulationsreihen verwendete Parameterwerte

ergänzen. Die Wahl der Tiefe hat dabei den größten Einfluss auf die Anzahl der Treffer, da mit ihr die Größe der transitiven Nachbarschaft und damit letztlich die Anzahl der bekannten Knoten festgelegt wird. Danach folgen der λ–Wert und zuletzt der α–Wert, deren Einfluss auf die Selektion der verwendeten Kanten zu den gewählten Knoten beschränkt bleibt. Bei der Auswahl und Austarierung der Parameter wurde darauf geachtet, dass sich ein möglichst weicher Übergang zwischen den Parameterkombinationen ergibt. Um die Anzahl der Simulationen in einem menschlich erfassbaren Rahmen zu halten ist es hier besonders wichtig, durch Vortests und Probeläufe die Parameterbereiche der Alpha (α), Tiefen (d) und Lambda(λ)–Werte auf ein sinnvolles Maß zu reduzieren.

6.2.2 Bloomfilter

Jedem Knoten muss ein *Bloomfilter* zugeordnet werden. Hierfür wird ein Elementpool von etwa 100.000 Elementen aus einer Musikdatenbank (siehe [Fre]) verwendet. Jedes Element wird durch 8 Einträge in einem *Bloomfilter* der Größe 2^{18} (entspricht 32kB) repräsentiert. Eine Herausforderung ist es hier eine geeignete initiale Füllrate für die *Bloomfilter* zu wählen. Nach Experimenten mit verschiedenen Werten zwischen 1% und 25% erscheint es am sinnvollsten die *Bloomfilter* mit 1% zu befüllen (330 Elemente). Wie in Abb. 6.1 ersichtlich ist, bleibt die *False Positivrate* für bis zu 100 Aggregationen in einem akzeptablen Bereich, so dass sich die *Bloomfilter* auch für Simulationen mit großer transitiver Nachbarschaft eignen.

Während der Simulationen durchgeführte Messungen bestätigen diesen Wert. Zwar ergeben sich im schlimmsten Fall (siehe Tab. 6.2) Füllraten von bis zu 90% und damit eine theoretische *False Positivrate* von 46%. Aber die Höchstwerte können auch nur in den tiefsten Leveln der *Bloomfilter* auftreten und werden durch den Algorithmus nach einem Schritt (Weiterleitung auf den nächsten Knoten + Auswahl des nächst höheren *Bloomfilters*) automatisch korrigiert.

Alternativ lässt sich die Wahl der Füllrate auch mit Hilfe der oben vorgestellten Formel (2.5) für optimale *Bloomfiltergrößen* begründen. Betrachtet man die Größe des *Bloomfilters* und die Anzahl der *Hashfunktionen* als gegeben und stellt die Formel um

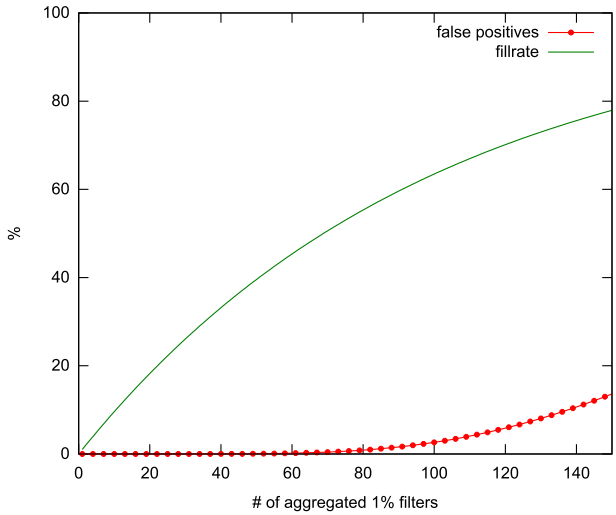


Abbildung 6.1: Füll- und *False Positivrate* der *Bloomfilter* unter der Voraussetzung, dass alle Elemente der *Bloomfilter* mit einer optimalen *Hashfunktion* auf die 2^{18} Bits verteilt werden.

erhält man:

$$n = \ln(2) * \frac{m}{k} = \ln(2) * \frac{2^{18}}{8} = 22713,05 \tag{6.1}$$

Der *Bloomfilter* ist also für 22.713 Elemente optimal. Setzt man 330 Elemente pro aggregiertem *Bloomfilter* an, ergibt sich für das Experiment eine optimale Belegung bei der Aggregation von 68,8 *Bloomfiltern*. Dieser Wert ist offensichtlich sowohl für den MIT Cell Tower Graphen mit seinen 89 sehr stark verbundenen Knoten als auch für den Bluetooth Datensatz mit seiner sehr geringen Dichte bestens geeignet. Lediglich der IBM Graph verfügt sowohl über viele Knoten als auch ein relativ dichtes Kantennetz. In der Tat treten hier auch die gemessenen Höchstwerte auf, aber wie oben begründet wird führt dies nur zu einer geringen Mehrbelastung des Netzes.

IBM Alpha (α): 70 Lambda (λ): 40 Tiefe (d): 4
α–Kanten: 8698 λ–Kanten: 18847 Vereinigung: 20792

BloomLevel	**durchschnittliche Füllrate**	**theoretische FalsePositivrate**
0	0.0100	1.0092E-16
1	0.1881	1.5664E-6
2	0.6773	0.0443
3	0.9039	0.4458

Tabelle 6.2: Die Tabelle stellt die höchsten während der Simulation aufgetretenen Füllraten dar. Aber während im tiefsten Level im Durchschnitt 44,6% der Treffer *False Positives* sind, fällt dieser Wert bereits einen Schritt weiter auf nur noch 4,4%, so dass die reale Mehrbelastung gering ausfällt.

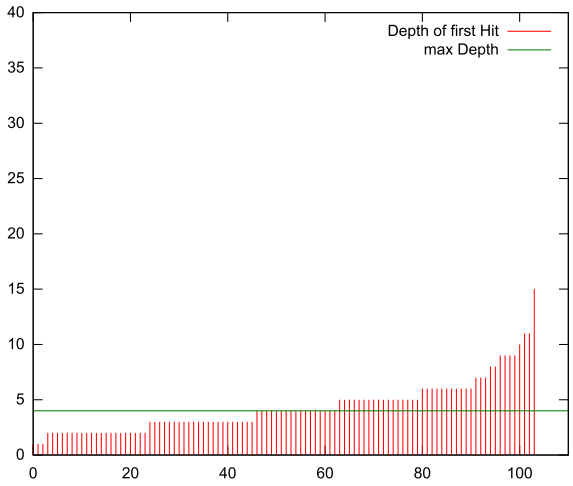
Bei den in Tab. 6.2 gezeigten Werten muss beachtet werden, dass es sich hier um die absoluten Extremwerte der Simulation handelt. So liegt auch im tiefsten *Bloomfilterlevel* bei 80% der durchgeführten Simulationen die durchschnittliche theoretische Fehlerrate unter 1%. Die gewählte *Bloomfiltergröße* passt also zu einem breiten Spektrum von Testwerten und Graphen und ist damit für die von uns durchgeführten Simulationen bestens geeignet. Insgesamt können mit der Wahl der Initialen Füllrate von 1% störende Einflüsse der *False Positives* für alle getesteten Parameterkombinationen ausgeschlossen werden. Deshalb wurden für die Simulationen für jeden Knoten ein individueller *Bloomfilter* mit dieser Anfangsbelegung generiert.

6.2.3 Nachrichten

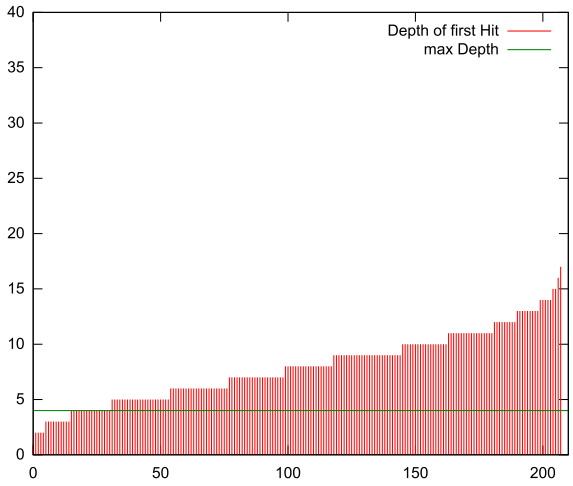
Der zweite wichtige Faktor sind die Nachrichten, die im Kontaktgraphen verschickt werden. Für unsere Simulationen werden 100 Nachrichten generiert, von denen jede einen Startknoten und ein zufällig aus dem Elementpool gewähltes Anfrageitem hat. Die Evaluation mit den bereits generierten *Bloomfiltern* ergibt für den MIT Cell Tower Graphen 117 , für BlueTooth 592 und für die IBM Daten 486 Zielknoten. Als Zielknoten gilt hierbei jeder Knoten, der sich für mindestens eine der 100 Nachrichten als Ziel eignet. Die Größe aller Nachrichten wird mit 64kByte angesetzt, genug um einen unkomprimierten *Bloomfilter* und eine Textnachricht aufnehmen zu können. Somit lässt sich eine Nachricht innerhalb von einer Sekunde über eine Bluetoothverbindung (723,2 kBit/s) übertragen.

6.2.4 Tiefen-Parameter

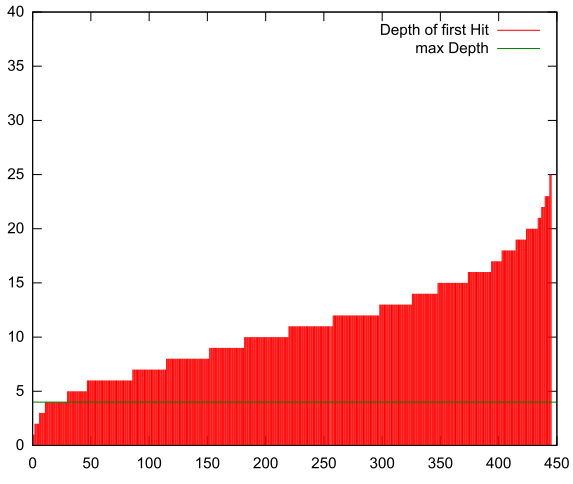
Prinzipiell erscheinen Tiefenwerte < 10 aufgrund des *Small World Phänomens* [Mil67] als aussichtreiche Kandidaten. Allerdings sind zu große Werte bedenklich, da mit zunehmender transitiver Tiefe die Anzahl der Daten in einem *Bloomfilterlevel* jeweils stark ansteigt (vgl. Tab. 6.2). Außerdem ist ein Algorithmus, der weniger Daten halten muss und weniger Kontaktzeit zum Austauschen und Aktualisieren der Profile verbraucht prinzipiell zu bevorzugen. Interessant ist in diesem Zusammenhang auch die von *Flooding* beim ersten Erreichen eines Knotens verwendeten "Optimalen Tiefe", optimal in dem Sinn, dass es sich jeweils um die Tiefe handelt, die die Nachricht am schnellsten zum Ziel zustellen kann. Abbildung 6.2 stellt diese "Optimale Tiefe" für alle drei Datensätze dar. Gezeigt werden die *Floodingsimulationen* aus der ersten Testreihe. Die Ergebnisse der anderen Testreihen sind vergleichbar. Man sieht, dass im MIT Cell Tower Graphen eine Tiefe von 4 aufgrund der hohen Konnektivität und geringen Anzahl an Knoten bereits ausreicht, um über 50% der Nachrichten optimal auszuliefern. Der Bluetooth Datensatz dagegen ist so schlecht verbunden, dass er von Tiefen über 15 nicht mehr profitiert während der IBM Datensatz, der wiederum relativ groß ist und über eine mittlere Dichte verfügt auch von hohen Tiefenwerten bis etwa 20 noch profitieren kann. Es zeigt sich also, das es den perfekten Tiefenwert nicht gibt, da er in Abhängigkeit zu dem zugrundeliegenden Graphen steht. Um nun einen sinnvollen Tiefenbereich zu finden wurden mit unterschiedlichen



(a) MIT Cell Tower



(b) MIT Bluetooth



(c) IBM Watson

Abbildung 6.2: Das Diagramm zeigt zu jedem Zielknoten für alle Nachrichten die bei der schnellsten möglichen Auslieferung verwendete Tiefe (d) an. Die wagerechte Linie entspricht dabei der höchsten von uns verwendeten Tiefe. (Versuche aus der ersten Testreihe)

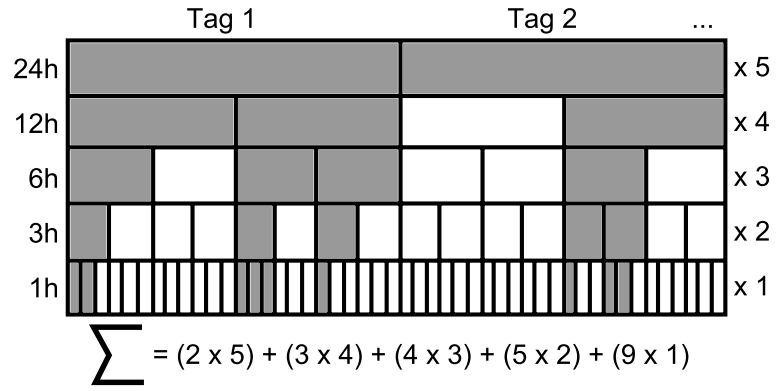


Abbildung 6.3: Berechnung des α–Gewichtes einer Kante über den Zeitraum von zwei Initialisierungstagen. Aktive *Bins* sind grau gefärbt. Die Summenformel illustriert am dargestellten Beispiel, wie das α–Gewicht berechnet wird.

Werten Einzeltests durchgeführt. Letztlich erscheint es sinnvoll Tiefen Werte zwischen 2 und 4 zu wählen, da diese sicherstellen, dass der Algorithmus nicht zum *Flooding* degeneriert und die Anzahl der versendeten Nachrichten in einem akzeptablen Bereich bleibt. Außerdem handelt es sich dabei fast genau um den Wertebereich, der in den Ergebnissen von [RK02] präsentiert wird. Die Autoren konzentrieren sich dort auf Tiefenwerte zwischen 2 und 3.

6.2.5 α–Parameter

Der α–Wert soll schließlich wie in [MGC+07] zwischen Kontakten von Freunden und Fremden unterscheiden. Freunde ist in diesem Zusammenhang allerdings in einem technischen Sinn zu verstehen und bezeichnet Menschen zwischen denen eine stabile Kommunikation möglich ist. Da der Datensatz mit 29 Tagen nur über einen sehr kurzen Zeitraum geht, kann nicht das gleiche Verfahren wie in [MGC+07] verwendet werden. Stattdessen wird an den 10 zufällig gewählten Initialisierungstagen der Wert der Kanten bestimmt. Nach anfänglichen Versuchen mit der Aktivitätsdauer verwendet ISeek mittlerweile eine gewichtete Summe über die Aktivitätshäufigkeit. Hierbei wird ein Tag auf verschiedenen Leveln (1h, 3h, 6h, 12h und 24h) in *Bins* aufgeteilt und für jede aktive *Bin* das Gewicht der Kante erhöht (siehe Abb. 6.3). Auf Grundlage dieses α–Gewichtes können die Knoten dann mit Hilfe des α–Wertes entscheiden welche Kanten sie als Freundkanten betrachten. Jeder Knoten n bildet dazu das α–Gewicht α_g^e aller seiner Kanten e. Entsprechend bezeichnet α_{min}^n (α_{max}^n) das kleinste (größte) α_g für den Knoten n. $\alpha_w^{(n,e)}$ bezeichnet den Wert der Kante e und wird für jede Kante des Knotens n berechnet. Die Werte werden dabei mit Hilfe der Wurzelfunktion auf das Intervall (0,1) abgebildet. Ausgewählt werden alle Kanten deren berechneter $\alpha_w^{(n,e)}$ den gewählten Grenzwert überschreitet.

$$\alpha_w^{(n,e)} = \sqrt{\frac{(\alpha_g^e - \alpha_{min}^n) * 100^2}{\alpha_{max}^n - \alpha_{min}^n}} \tag{6.2}$$

Für die Auswahl geeigneter α–Parameter mussten schließlich einige Schwierigkeiten überwunden werden. Der α–Parameter hat sich von einem globalen, auf der Verbindungsdauer basierenden Parameter hin zum jetzigen Modell entwickelt, dass zusätzlich durch die λ–Kanten unterstützt wird. Auf Grundlage der gesammelten Erfahrungen und umfangreicher Einzeltests wurden die drei α–Parameter 70, 80, 90 als die vielversprechendsten ausgewählt. Auch hier ist mit einigem Aufwand eine Validierung der gewählten Werte durch [MGC+07] möglich, da man den Prozentsatz der ausgewählten Kanten als Vergleich heranziehen kann.
So werden von [MGC+07] 6,9% der Kanten als Freundkanten definiert. Es hat sich in früheren Testreihen gezeigt, dass dieser Wert für ISeek als untere Grenze dienen kann. Betrachtet man Tab. 6.3 sieht man, dass mit den gewählten Werten 70, 80, 90 auf allen drei Graphen wie gewünscht eine Streuung um die 6,9% herum erreicht wird, mit der Tendenz zu höheren Werten.

α–Wert	Bluetooth	Cell Tower	IBM Watson
70	12,19%	22,25%	9,76%
80	8,62%	11,99%	6,27%
90	6,28%	5,01%	3,89%

Tabelle 6.3: Übersicht über den Prozentsatz der Kanten, der vom α–Parameter als Freundkante definiert wird.

Durch die Vereinigung mit den λ–Kanten kommt man letztlich je nach Datensatz auf etwa 16-32% der Kanten, die zum *Routing* verwendet werden können.

6.2.6 λ–Parameter

Der letzte variable Parameter ist schließlich Lambda (λ). Der λ–Wert wurde nötig, da sich gezeigt hat, dass vor allem die dünneren Graphen bei einem reinen α–Kanten Netzwerk in zu viele *Cluster* zerfallen und zu viele Knoten gar nicht mehr erreicht werden können. Aus diesem Grund wählt jeder Knoten eine gewisse Menge von λ–Kanten. Deren Anzahl wird zufällig im Intervall von (2, $deg(n)$ * λ–Wert / 100) – wobei $deg(n)$ den Knotengrad eines Knotens n bezeichnet – bestimmt. Ein λ–Wert von 30 bedeutet also, dass jeder Knoten zwischen zwei und 30% seiner Kanten auf jeden Fall auswählt um sie zum *Routing* zu verwenden, unabhängig davon, ob diese Kanten Freundkanten sind oder nicht. Die λ–Kanten werden dann aus einer nach ihrem α–Gewicht sortierten Liste der Kanten eines Knotens in absteigender Reihenfolge ausgewählt. Die untere Grenze von mindestens zwei Kanten stellt sicher, dass jeder Knoten zum Netzwerk verbunden ist und das bei einem über eine Kante verbundenen Knotenpaar jeder der beiden Knoten zusätzlich einen Mehrwert als Relaisstation bieten kann. Nach einigen Testläufen zwischen 10% und 50% für die obere Grenze haben sich die Werte 30 und 40 herauskristallisiert, da sie die Netzwerke gut verstärken ohne jedoch die Nachrichtenanzahl zu sehr zu erhöhen. Eine Auswertung der Simulationen zeigt, dass die Kombination aus α–Kanten mit λ–Kanten

als Verstärkung sich gut an die unterschiedlichen Graphen anpassen kann. So werden zum Beispiel bei einem α–Wert von 70 auf dem MIT Cell Tower Datensatz lediglich 2,77 - 5,81% der Kanten zusätzlich zu den α–Kanten ausgewählt. Beim IBM Graphen dagegen zwischen 9,07 und 13,57% und auf dem extrem dünnen Bluetoothgraphen sogar 16,16 - 19,24%.

6.2.7 Durchgeführte Simulationen

Insgesamt werden auf allen drei Datensätzen jeweils vier unterschiedliche Simulationsreihen durchgeführt, wobei jede Simulationsreihe aus allen oben definierten Parameterkombinationen (siehe Tab. 6.1) und zwei weiteren Simulationen für *Flooding* und *Direct Contact* besteht. Jede Simulationsreihe hat dabei einen festen Starttag, an dem jeweils um 00:00:00 alle 100 Nachrichten erzeugt werden.

- Die zufällig gewählten Starttage sind der 2., 3., 13. und 17. Tag des Simulationszeitraums.
- Der Zufallsgenerator des Simulators ist für alle Simulationen auf den gleichen Startwert eingestellt, so dass jede Simulation auf einem identisch aufbereiteten Graphen beginnt.
- Als Bandbreite wird für alle Kanten einheitlich 723,2 kBit/s angesetzt. Dieser Wert entspricht der Übertragungsrate von Bluetooth I im asynchronen Modus und kann als realistischer Wert für aktuelle Datenübertragungsstandards betrachtet werden.

In der Auswertung werden wir die interessantesten und aussagekräftigsten Ergebnisse dieser 240 Einzelsimulationen näher betrachten.

Kapitel 7

Resultate

7.1 Überblick

In diesem Kapitel sollen die wichtigsten Ergebnisse der Versuche präsentiert werden. Wie zu erwarten war unterscheiden sich die Ergebnisse je nach zugrunde liegendem Datensatz sehr stark. Die unterschiedlichen Starttage fallen dagegen bei den meisten Versuchen nicht so sehr ins Gewicht, so dass wir uns im Folgenden darauf beschränken werden die beobachteten Effekte exemplarisch an der ersten Simulationsreihe zu zeigen. Andere Starttage werden nur bei größeren Unterschieden mit einbezogen. Eines der wichtigsten Ziele von ISeek ist die Reduzierung des Aufwandes gegenüber *Flooding*. Um diesen Aufwand vergleichbar zu machen unterscheiden wir zwei Nachrichtenarten:

- Transportnachrichten: Nachrichten, bei denen tatsächlich alle Daten übertragen werden.

- *Look-Up* Nachrichten: Nachrichten, bei denen der Zielknoten die Daten bereits kennt und z.B. ein einfacher *Hash-ID*-Abgleich ausreicht um das unnötige Versenden aller Daten zu unterbinden.

Als sehr aussagekräftig kann das Verhältnis zwischen erreichten Zielknoten und verschickten Transportnachrichten angesehen werden, da es die tatsächliche Netzwerkbelastung sehr gut wiederspiegelt. Aber auch das Verhältnis aller Nachrichten (Transport $\cup$ *Look-Up* Nachrichten) zu den erreichten Zielknoten ist durchaus interessant und gewährt weitere Einblicke. Tabelle 7.1 zeigt den Vergleich zwischen dem jeweils teuersten getesteten ISeek Algorithmus und *Flooding* auf allen Graphen. Um die verschiedenen Algorithmeninstanzen unterscheiden zu können wird folgende Definition eingeführt: Dx Ly Az bedeutet die ISeek-Variante mit den Parametern Tiefe(d) = x, Lambda(λ) = y und Alpha(α) = z.

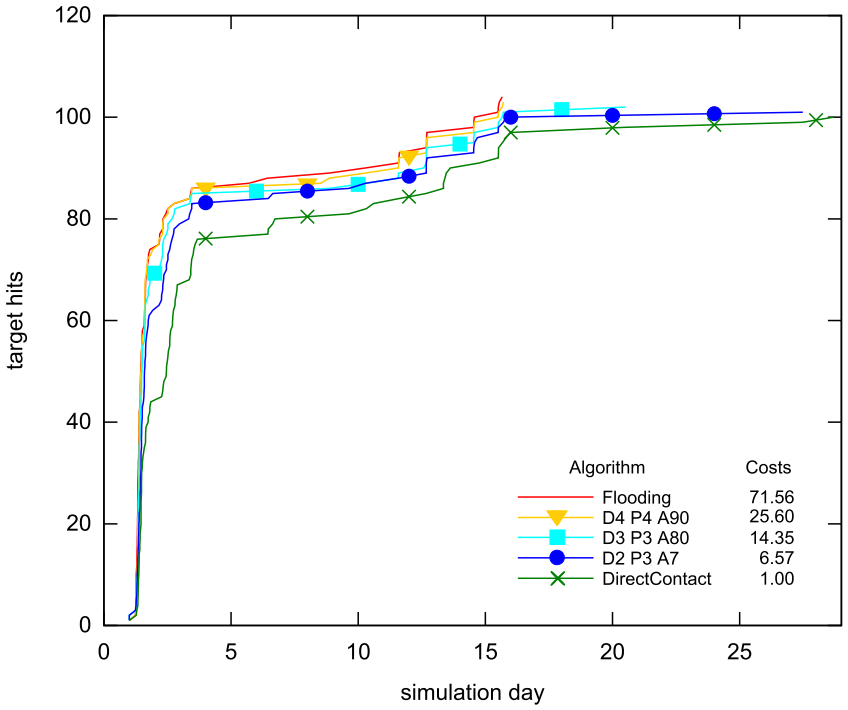
7.2 Trefferquote und Kosten ISeek vs. Flooding/Direct Contact

Wie aus Tabelle 7.1 ersichtlich ist, ermöglicht ISeek auch in den teuersten Varianten noch beachtliche Kostenersparnisse gegenüber *Flooding*. Gleichzeitig erreicht ISeek auf allen Graphen eine höhere Trefferquote als *Direct Contact*. Tabelle 8.1 schlüsselt für die erste Simulationsreihe mit Starttag 2 für alle Parameterkombinationen die Anzahl der Treffer mit ihren relativen Kosten auf. Betrachtet man hier zum Beispiel die billigsten Algorithmen D2 L3 A90 so sieht man, dass bei einer Kostensteigerung von 1,93 und 2,25 auf dem IBM und Bluetooth Graphen bereits eine Verdoppelung bzw. Verdreifachung der Treffer gegenüber *Direct Contact* erreicht wird. Lediglich für den MIT Cell Tower Graphen sind kaum Verbesserungen möglich, da *Direct Contact* bereits 100 von 117 möglichen Zielen erreicht. Auf diesem Datensatz wurden dafür allerdings beachtliche Performancegewinne im Bereich der Ausbreitungsgeschwindigkeit erreicht. Eine Analyse der besten Parameterkombinationen für den Graphen (Abb. 7.1) zeigt deutlich, dass bereits für relativ geringe Mehrkosten deutliche Steigerungen gegenüber *Flooding* möglich sind. Die teureren Varianten scheinen sogar beinahe *Floodingperformance* zu erzielen. Eine detaillierte Auswertung des Geschwindigkeitsaspektes im nächsten Unterkapitel wird diesen Eindruck bestätigen.

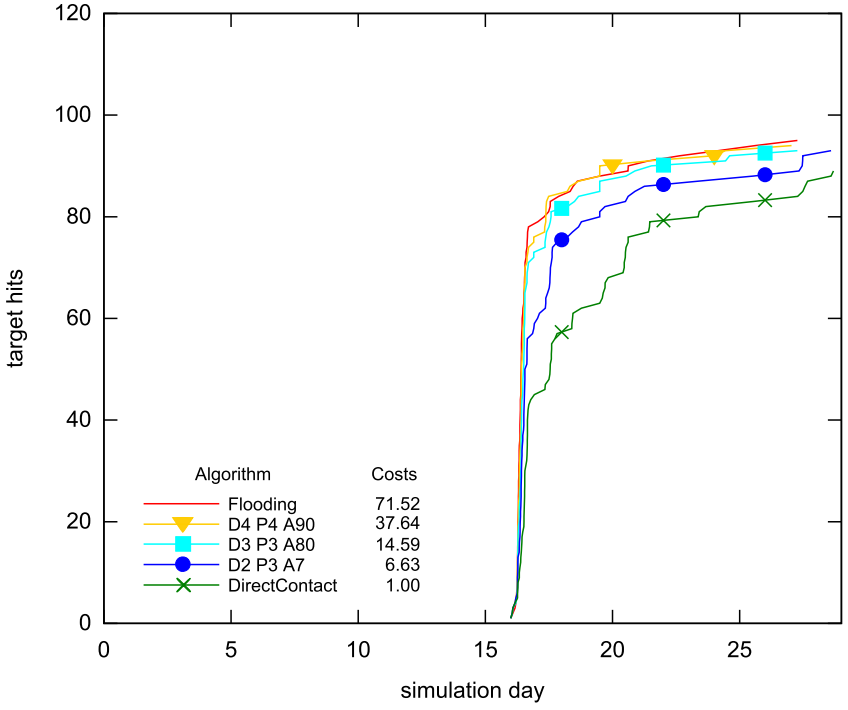
Ein ganz anderes Bild zeigt sich auf den beiden verbleibenden Graphen, hier ist sehr deutlich zu erkennen, wie durch die Steigerung der relativen Kosten mit weniger restriktiven Parametern immer mehr Treffer erzielt werden. Besonders auf dem IBM Datensatz (Abb. 7.2) wird ein breites Spektrum abgedeckt, so dass der Algorithmus durch die Wahl der Parameter an verschiedenste Anwendungsszenarien angepasst werden kann. Betrachtet man Abb. 7.2, so fallen einem drei deutliche Stufen in den Kurven auf. Diese lassen sich den Wochenenden zuordnen, an denen nur sehr wenige Personen im "Watson Research Center" waren. Besonders interessant ist auch hier die teuerste Variante D4 L4 A70, die immerhin 80,1% der Treffer von *Flooding* erreicht, dafür aber lediglich 11,4% der Transportnachrichten und sogar nur 1,5% aller Nachrichten von *Flooding* verschickt. Die relativen Kosten pro Nachricht sind damit um den Faktor 7 niedriger als bei *Flooding*. Insbesondere eignet sich ISeek offensichtlich sehr gut, um in zum IBM Datensatz ähnlichen Umgebungen zu *routen*.
Der Bluetoothdatensatz dagegen ist letztlich zu dünn, dies lässt sich neben dem eher

Graph	**Transportnachrichten**	**Transport ∪ *Look-Up* Nachrichten**
Bluetooth	98,1%	99,1%
Cell Tower	38,4%	87,3%
IBM Watson	88,6%	98,5%

Tabelle 7.1: Vergleich der teuersten getesteten ISeek Variante vs *Flooding*. Prozentuale Nachrichtenersparniss für die erste Simulationsreihe mit Starttag 2

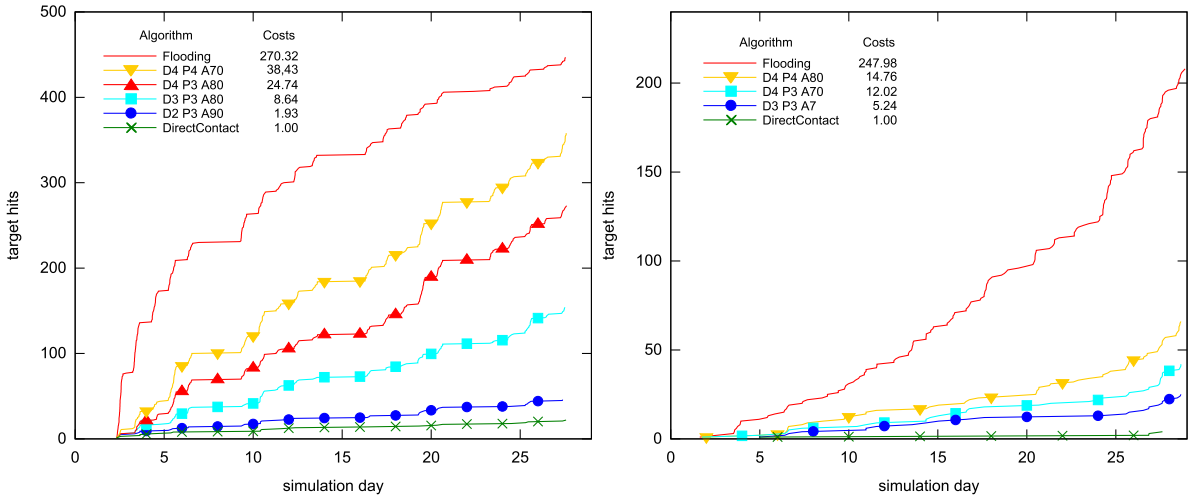


(a) Starttag 2 Cell Tower



(b) Starttag 17 Cell Tower

Abbildung 7.1: Erzielte Treffer im Zeitverlauf. Man kann deutlich erkennen das ISeek mit *Flooding* konkurrieren kann und *Direct Contact* in der Geschwindigkeit weit hinter sich lässt. (MIT Cell Tower Datensatz)



(a) Starttag 2 IBM

(b) Starttag 2 BT

Abbildung 7.2: erzielte Treffer im Zeitverlauf IBM Watson und Bluetooth Datensatz. Auf beiden Graphen kann *Flooding* nicht erreicht werden, aber *Direct Contact* wird in der Anzahl der erreichten Ziele deutlich übertroffen.

schlechten Ergebnis von nur etwa $\frac{1}{4}$ der *Flooding* Trefferquote auch direkt an den Kosten ablesen. Die D4 L4 Parameterkombination, die auf den anderen Graphen relative Kosten zwischen 25 und 38 verursacht hat, führt hier nur zu Kosten von 14,76. Dies deutet daraufhin, dass ISeek seine Tiefenabdeckung aufgrund der Struktur des Graphen nicht ausspielen kann. Allerdings handelt es sich hier auch um einen sehr schwierigen Anwendungsfall. So konnte keiner der in [Isl08] getesteten Algorithmen für diesen Datensatz gute Ergebnisse generieren. Trotzdem sind sehr deutliche Steigerungen gegenüber *Direct Contact* möglich und auch die Abstufungen, sowohl im Kostenfaktor, als auch in der erreichten Trefferquote variieren erwartungsgemäß mit den Parametern (Tab. 8.1). Sehr interessant ist auch die *Floodingkurve*, die in etwa einen quadratischen Verlauf nimmt, ein Indiz dafür, dass auch *Flooding* sehr lange braucht um sich im Netzwerk weit genug ausbreiten zu können. Insgesamt lässt sich festhalten, dass ISeek gegenüber *Direct Contact* bereits für geringe Mehrkosten eine echte Verbesserung bringt und das die Trefferquote sich erwartungsgemäß mit den Parametern entwickelt. Außerdem skaliert auch die Ausbreitungsgeschwindigkeit mit der Parametervariation.

7.3 Geschwindigkeit ISeek vs. *Flooding / Direct Contact*

Neben der Trefferzahl und der Kostenersparnis ist die Geschwindigkeit der dritte bedeutende Faktor in der Analyse von ISeek. Wie wir bereits gesehen haben erreicht *Direct Contact* lediglich auf dem sehr dichten MIT Cell Tower Datensatz eine relevante

Anzahl von Treffern. Daher wird der Geschwindigkeitsvergleich zu *Direct Contact* auch nur für diesen Datensatz evaluiert.

ISeek erreicht auf dem MIT Cell Tower Datensatzes aufgrund der hohen Dichte und geringen Größe die höchsten Ausbreitungsgeschwindigkeiten der Simulationsreihen. Wie in Abb. 7.3 zu sehen ist werden fast alle Nachrichten in unter 3 Tagen ausgeliefert, spätestens jedoch nach maximal 15 Tagen. Bedeutender noch ist aber die Tatsache, dass ISeek hier nahezu gleich schnell ist wie unbegrenztes *Flooding* und das bei Kostenersparnissen von 38,4% für Transportnachrichten und sogar 87,3% bezogen auf alle verschickten Nachrichten (Tab 7.1). Allerdings erreicht aufgrund der Struktur des Netzwerkes hier auch *Direct Contact* eine sehr hohe Anzahl von Treffern mit geringeren Kosten. Der direkte Vergleich in Abb. 7.3 zeigt jedoch, dass ISeek bei allen Nachrichten schneller oder gleich schnell wie *Direct Contact* ist, zum Teil mit beträchtlichem Abstand. Zieht man zudem noch den Zeitpunkt der Treffer hinzu (Abb. 7.1), so wird klar, dass vor allem in den ersten drei Tagen nach dem Start der Nachrichten höhere Auslieferungsgeschwindigkeiten erreicht werden und dies auch für deutlich billigere ISeek Varianten. So verursacht zum Beispiel D2 L3 A7 lediglich die sechs fachen Kosten (siehe Tab. 8.1) und erreicht trotzdem eine deutlich höhere Ausbreitungsgeschwindigkeit als *Direct Contact*. Während also auf dem MIT Cell Tower Datensatz wie erwartet keine großen Schwankungen in Bezug auf die Trefferzahl auftreten, zeigt die Auswertung der Simulationsergebnisse ganz klar, dass ISeek hier in der Lage ist zu wesentlich geringeren Kosten *Floodingperformance* zu erreichen.
Im Kontrast hierzu stehen der IBM und Bluetoothdatensatz, auf denen *Flooding* in Bezug auf Trefferzahl und Geschwindigkeit klar dominiert. *Direct Contact* hat auf diesen Graphen keine Chance und wird jeweils von allen getesteten Varianten weit zurückgelassen. Interessanter ist dagegen der Geschwindigkeitsvergleich zum *Flooding* (Abb. 7.4). Im Gegensatz zum MIT Cell Tower Graphen können hier beide Datensätze auch noch von Auslieferungszeiten deutlich oberhalb von 15 Tagen profitieren. Manche Nachrichten werden sogar erst nach 25 Tagen ausgeliefert. Davon abgesehen differenzieren sich die beiden Datensätze aber sehr stark, so dass eine getrennte Betrachtung nötig wird. Bei IBM Watson kann wie oben gesehen fast die Trefferquote von *Flooding* erreicht werden, betrachtet man aber Abb. 7.4 so wird deutlich, dass die Auslieferungsgeschwindigkeit nicht ganz so gut abschneidet. Diese liegt im Durchschnitt um etwa dem doppelten über der von *Flooding*. Besonders interessant ist, dass die zu den geordneten *Floodingnachrichten* korrespondierenden ISeek Nachrichten nur eine schwache Ordnung aufweisen, so sind zwar alle Nachrichten mit Auslieferungszeiten unter 5 Tagen auf der linken Seite zu finden, und die allermeisten Nachrichten mit zu *Flooding* äquivalenter Lieferzeit auf der rechten Seite, aber es gibt auch sehr viele Ausreißer, die sich über die komplette X-Achse verteilen. Dieser Effekt ergibt sich, weil *Flooding* auf dem IBM Graphen sehr viele Nachrichten mit Tiefen größer 4 ausliefert, und so Gebiete erschließen kann, die ISeek erst wesentlich später erreicht (vgl. Abb. 6.2).

Ganz anders ist das Ergebnis auf dem Bluetoothgraphen, wie bereits gesehen liegt

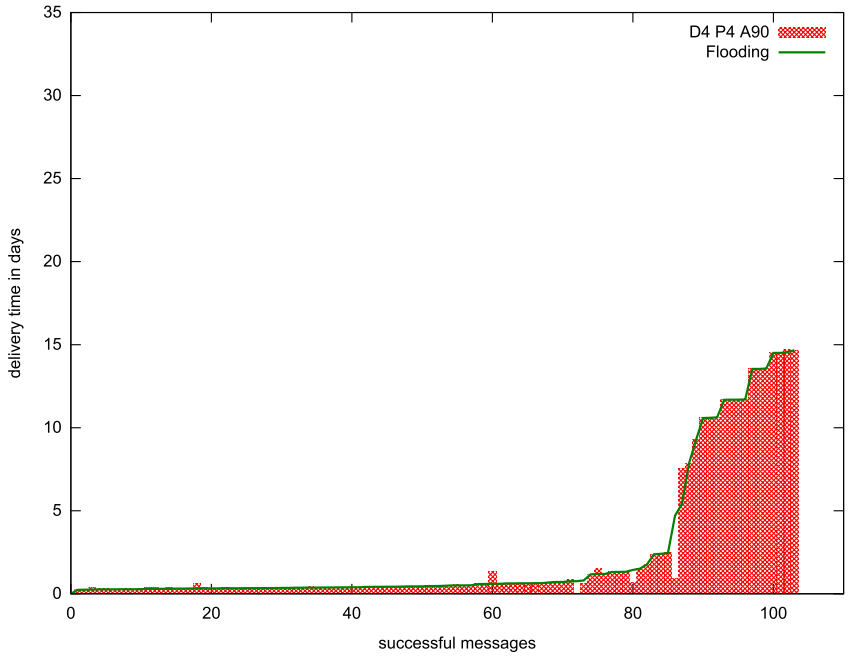
Flooding hier in der Trefferzahl mit deutlichem Abstand vorne. Bei den Nachrichten, die von beiden Algorithmen ausgeliefert werden, ist ISeek aber oftmals gleich schnell. Dieses Ergebnis steht im direkten Gegensatz zu den Ergebnissen des IBM Watson Graphen. Zieht man jedoch die im Vergleich deutlich geringere Transitivität (vgl. Abb. 6.2) in die Überlegungen mit ein, so wird klar, dass sich auch *Flooding* auf diesem Graphen anders verhält. Während auf dem IBM Graphen *Flooding* über die große Tiefe und ein Überfluten des Netzes mit Nachrichten hauptsächlich Geschwindigkeitssteigerungen erreicht, ist der Bluetoothgraph so schlecht verbunden, dass es offenbar wenige alternative Wege zu einem Ziel gibt (Geschwindigkeitssteigerungen also oft nicht möglich sind). Stattdessen profitiert *Flooding* hier vor allem von hohen Transitivitäten, weil durch diese neue Knoten überhaupt erst erschlossen werden. Diese Knoten sind dann aber für ISeek nicht mehr erreichbar. Unter diesen Gesichtspunkten erklärt sich die eher schlechte Performance von ISeek auf dem Bluetoothgraphen. Wenn Ziele aber innerhalb des von der maximalen Tiefe definierten Radius liegen, so werden sie offenbar häufig mit *Floodinggeschwindigkeit* erreicht.

7.4 Fazit

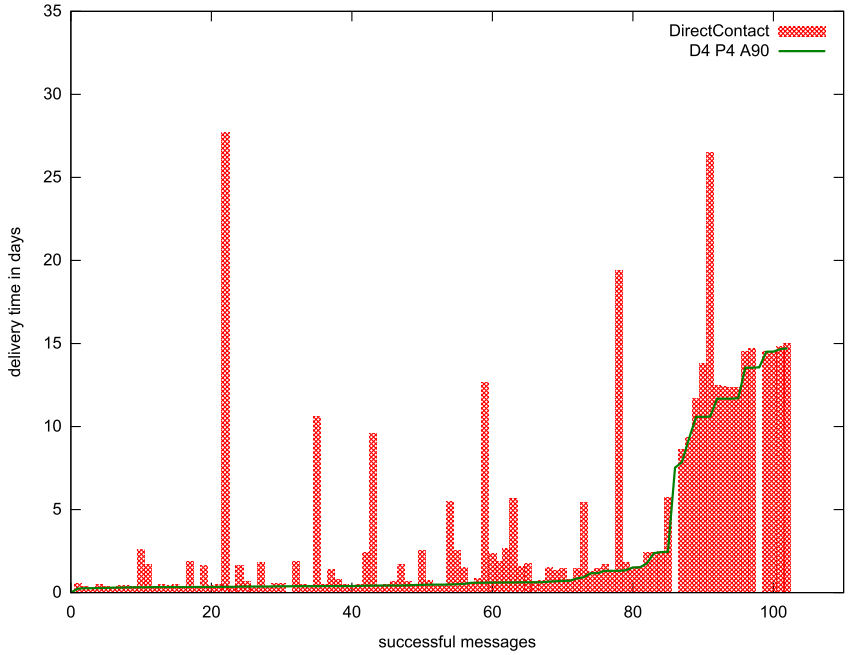
Um ISeek evaluieren zu können, bieten sich uns drei Kennzahlen an, die Ausbreitungsgeschwindigkeit, die Trefferquote und die damit verbundenen Kosten. Die Auswertung der Ergebnisse zeigt klar, dass es möglich ist ISeek über die unterschiedlichen Parameter an verschiedene Anwendungszenarien anzupassen, so sind *Low-Cost* Varianten denkbar, die lediglich um den Faktor 2-8 gesteigerte relative Kosten verursachen und trotzdem zu deutlichen Verbesserungen führen. Genauso ist es aber auch möglich *High-Cost* Varianten zu erstellen, die auf den dichteren Graphen bei hohen Kostenersparnissen trotzdem in direkte Konkurrenz zu *Flooding* treten können. Generell zeigt sich, dass je nach Struktur des Graphen die Verbesserungen vor allem in der Ausbreitungsgeschwindigkeit (Cell Tower) oder in der Trefferquote (IBM) stattfinden. Wobei für reale Szenarien natürlich häufig eine Erhöhung der Geschwindigkeit ebenfalls zu einer höheren Trefferquote führen wird, da Nachrichten normalerweise keine unbegrenzte Gültigkeit besitzen.

Der letzte Datensatz (Bluetooth) zeigt schließlich die Grenzen des Algorithmus auf. So ist es auf einem derart schlecht verbundenen Datensatz kaum noch möglich mit generalistischen Algorithmen wesentliche Erfolge zu erzielen. Weiterführende Einzeltests deuten allerdings daraufhin, dass man auf solche Datensätze abgestimmte ISeek Varianten entwickeln kann, bei denen sowohl der Tiefenwert, als auch der λ-Wert erhöht und der α-Wert stark abgesenkt werden. Einem solcher Art parametrisierten Algorithmus würden wesentlich mehr Kanten zur Verfügung stehen, allerdings würde er auf dichteren Graphen seinen Kostenvorteil gegenüber *Flooding* einbüßen. Zusammengefasst könnte man ISeek als eine Art "Directed Flooding" interpretieren, das dadurch charakterisiert wird, dass der Algorithmus lokal und dynamisch auf den Knoten agiert und damit in

der Lage ist zu *Flooding* vergleichbare Ergebnisse zu wesentlich geringeren Kosten zu liefern.

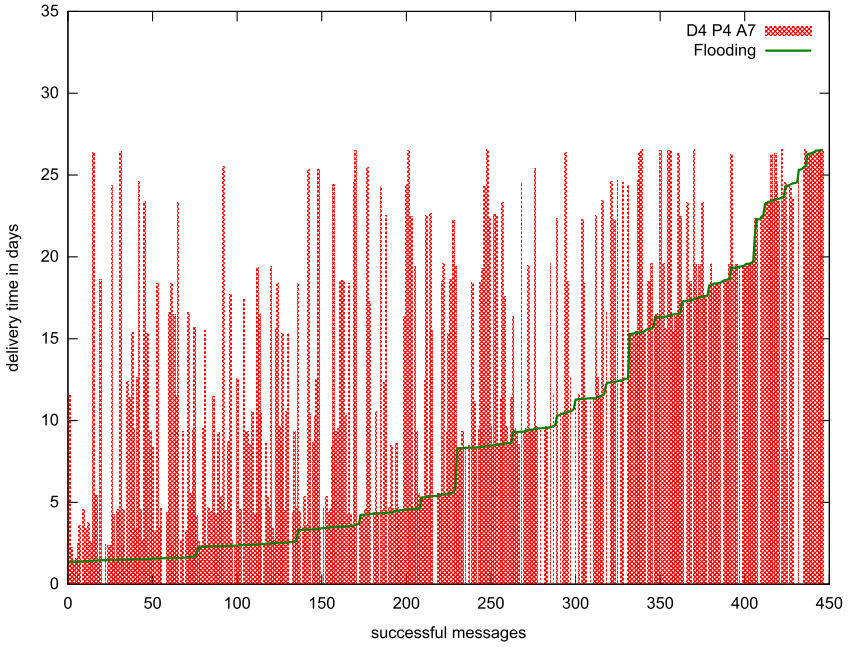


(a) ISeek (Fläche) vs. *Flooding*

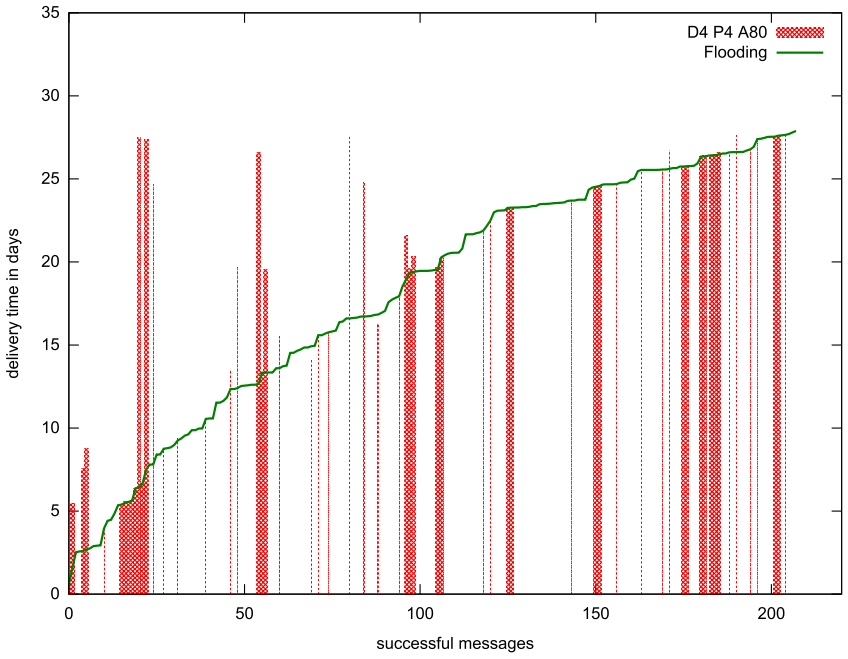


(b) Direct Contact(Fläche) vs. ISeek

Abbildung 7.3: Vergleich von ISeek mit *Flooding* und *Direct Contact* auf dem MIT Cell Tower Datensatz. ISeek erreicht die Geschwindigkeit von *Flooding* und ist bei einigen Nachrichten wesentlich schneller als *Direct Contact*.



(a) ISeek (Fläche) vs. Flooding IBM



(b) ISeek(Fläche) vs. Flooding BT

Abbildung 7.4: Vergleich von ISeek mit *Flooding* auf dem IBM Watson und Bluetooth Datensatz

Kapitel 8

Anhang

8.1 Datensatzgrafiken Abb. 8.1 - 8.3

Die drei Grafiken zu den Datensätzen vermitteln einen Eindruck von dem Aktivitätslevel des Graphen während des Simulationszeitraums. Es wird für jeden Tag des Simulationszeitraums ein Rechteck dargestellt. Jedes Rechteck enthält drei markierte Schichten. Alle Schichten zusammen stellen die Veränderung der Kanten gegenüber dem Vortag dar. Die oberste Schicht steht hierbei für neu hinzugekommene Kanten, die mittlere Schicht für Kanten die an beiden Tagen aktiv sind und die unterste Schicht für Kanten die am Vortag aktiv waren und es jetzt nicht mehr sind. Die Fläche der Schichten und die Restfläche des Rechtecks sind hierbei proportional zur Anzahl der repräsentierten Kanten.

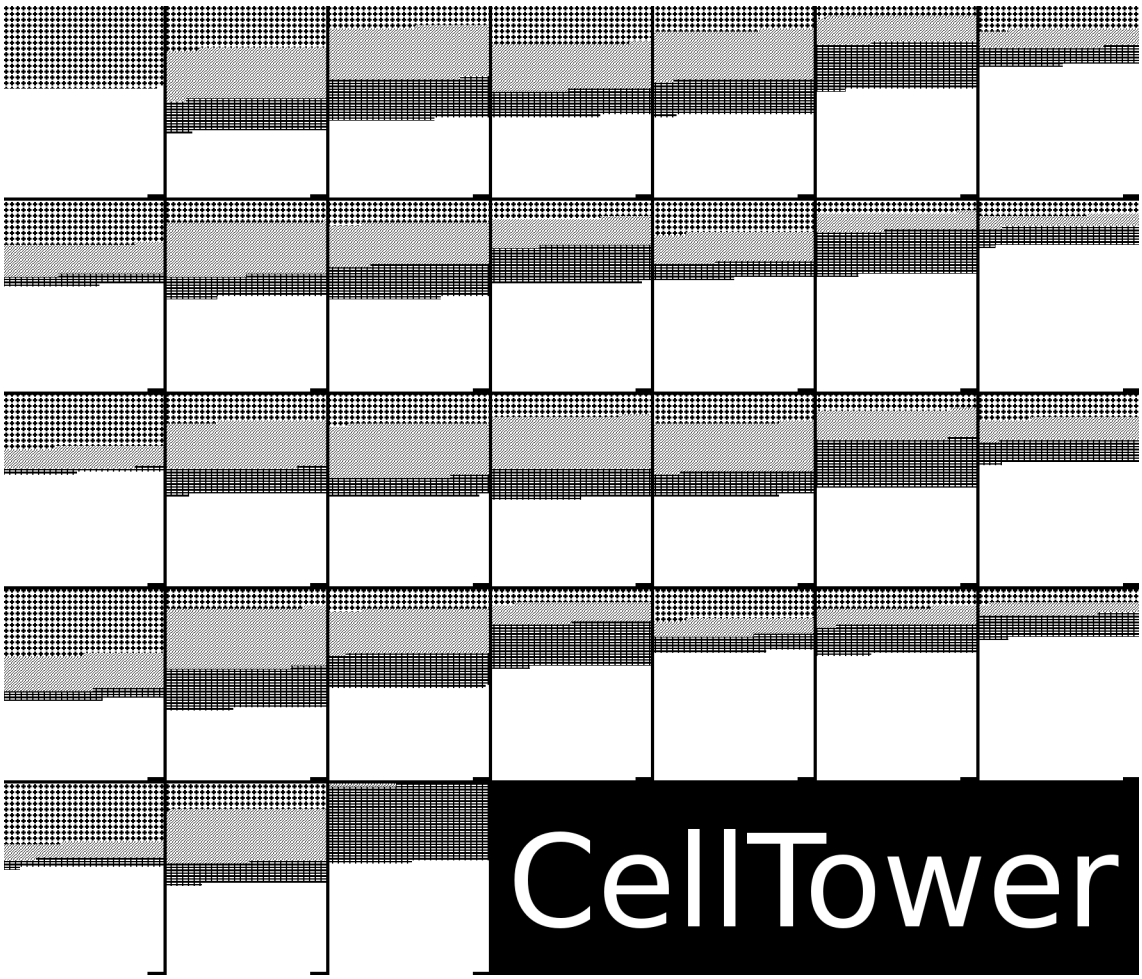


Abbildung 8.1: von Montag 01.11.2004 bis Mittwoch 01.12.2004: Auffallend sind besonders die deutlich erkennbaren Wochenenden (rechte Spalten) und der stetige Wechsel zwischen aktiven und inaktiven Kanten (alle drei Schichten sind grob gleichgroß)

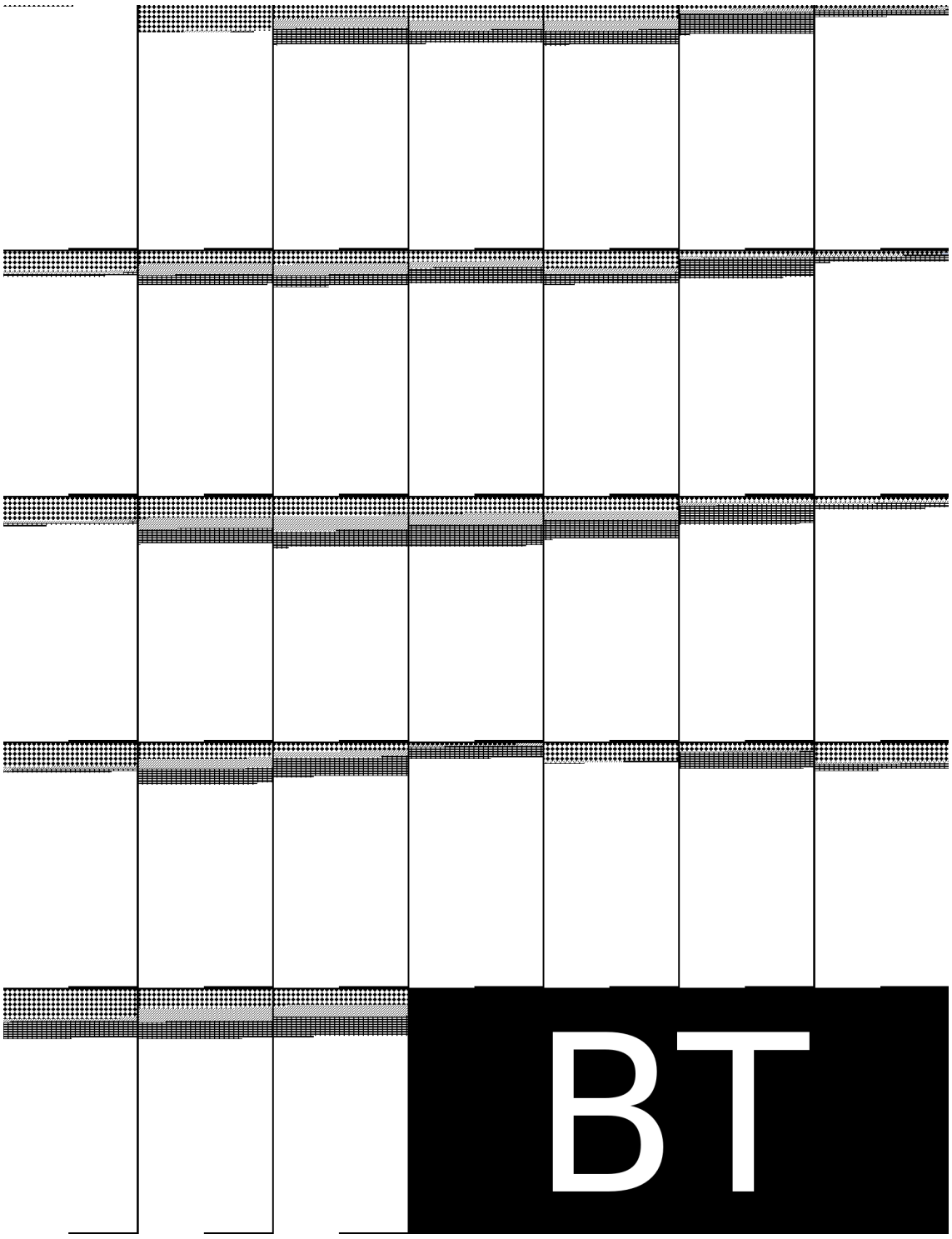


Abbildung 8.2: von Montag 01.11.2004 bis Mittwoch 01.12.2004: Auffallend sind auch hier die Wochenenden (rechte Spalten) und der stetige Wechsel zwischen aktiven und inaktiven Kanten (die mittlere "konstante" Schicht ist im Verhältniss schwach ausgeprägt.)

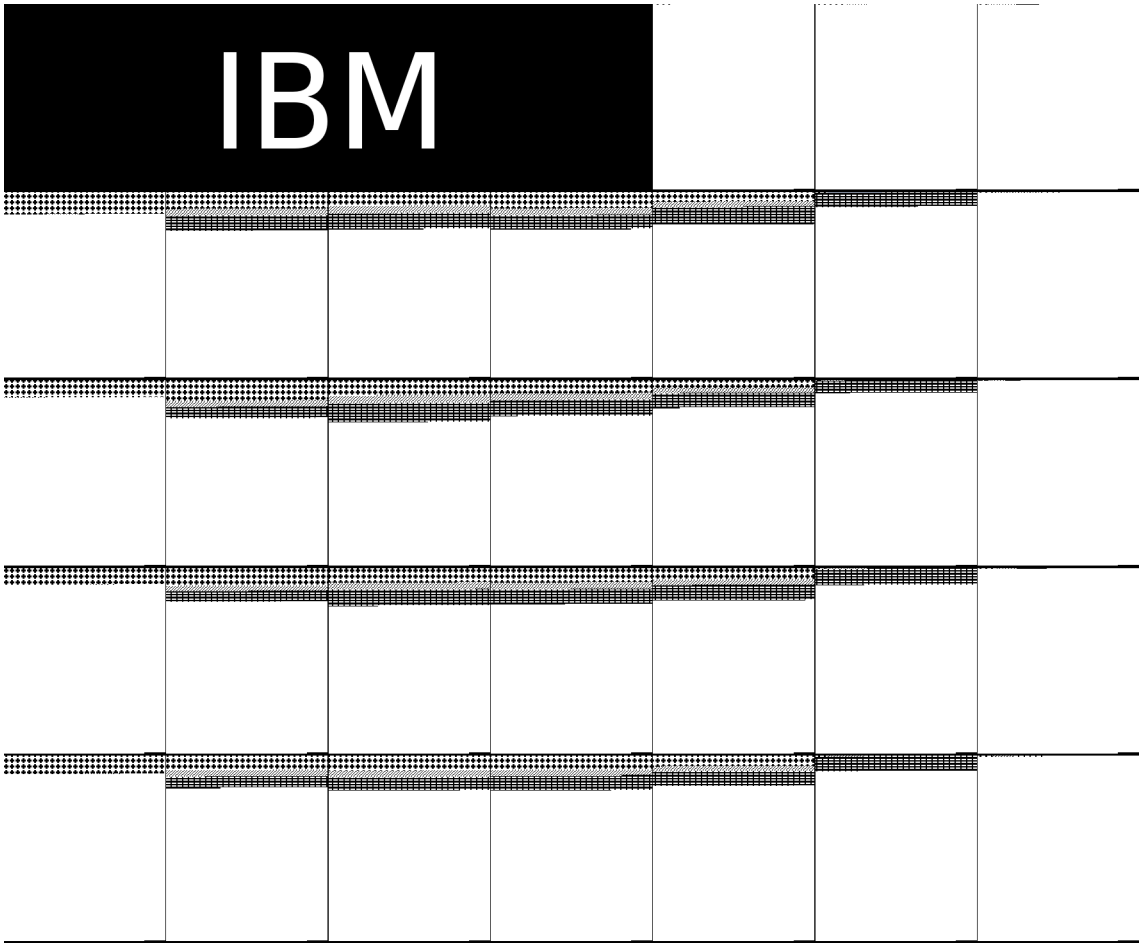


Abbildung 8.3: von Freitag 19.07.2002 bis Sonntag 18.08.2002: Neben den sehr schwachen Wochenenden und der üblichen Variation fällt hier vor allem das schlechte aktive Kanten / inaktive Kanten Verhältnis einiger Tage auf.

8.2 Ergebnisse Simulationsreihe 1

Parameter			Bluetooth		Cell Tower		IBM	
Tiefe (d)	**(λ)**	**(α)**	**Treffer**	**Kosten**	**Treffer**	**Kosten**	**Treffer**	**Kosten**
2	30	90	12	2.25	101	3.23	46	1.93
		80	12	2.25	100	4.26	46	1.96
		70	12	2.25	101	6.57	48	2.10
	40	90	14	2.43	102	5.39	69	2.72
		80	14	2.43	102	5.95	69	2.80
		70	15	2.4	102	8.02	70	2.91
3	30	90	23	5.22	101	11.15	143	8.10
		80	23	5.35	102	14.35	154	8.64
		70	25	5.24	102	22.94	163	9.31
	40	90	27	6.89	103	18.29	221	12.63
		80	29	6.45	103	20.24	228	13.25
		70	29	6.97	103	26.28	231	13.85
4	30	90	35	11.74	102	22.94	269	23.35
		80	35	11.23	102	28.25	273	24.74
		70	42	12.02	102	40.93	280	26.45
	40	90	60	14.55	103	25.60	339	36.51
		80	66	14.76	103	37.47	346	35.94
		70	64	15.61	103	44.48	358	38.43
Flooding			208	247.89	104	71.56	447	270.32
Direct Contact			4	1.00	100	1.00	22	1.00

Tabelle 8.1: Ergebnisse der ersten Simulationsreihe (Starttag zwei). Kosten beziffern die Anzahl der nötigen Transportnachrichten um ein Ziel zu erreichen, Treffer die Anzahl der Treffer. Maximal möglich sind für MIT Bluetooth 592, für MIT Cell Tower 117 und für IBM Watson 486 Treffer.

Literaturverzeichnis

[BIW08] Sebastian Kay Belle, Arshad Islam, and Marcel Waldvogel. I seek for knowledge: Exploiting social properties in mobile ad hoc networks. In *Proceedings of IFIP Wireless Days 2008*, 2008.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[BM02] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey, 2002.

[Cra] Crawdad. `http://crawdad.cs.dartmouth.edu/data.php`.

[DH07] Elizabeth M. Daly and Mads Haahr. Social network analysis for routing in disconnected delay-tolerant manets. In *MobiHoc '07: Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, pages 32–40, New York, NY, USA, 2007. ACM.

[Fre] Freedb. `http://www.freedb.org/en/`.

[Isl08] Muhammad Arshad Islam. Nuntifix-modeling of delay tolerant networks : A technical report. Technical report, Bibliothek der Universitt Konstanz, Universittsstr. 10, 78457 Konstanz, 2008.

[JFP04] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 145–158, New York, NY, USA, 2004. ACM.

[MGC+07] Andrew Miklas, Kiran Gollu, Kelvin Chan, Stefan Saroiu, Krishna Gummadi, and Eyal de Lara. Exploiting social interactions in mobile systems. In *Proceedings of UbiComp 2007*, pages 409–428, September 2007.

[Mil67] Stanley Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.

[MIT] Mit reality mining. `http://reality.media.mit.edu/`.

[RK02] S. C. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *In Proceedings – INFOCOM 2002*, volume 3, pages 1248 – 1257, 2002.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Konstanz, 16. Januar 2010

Michael Zinsmaier