Ernst W. Mayr Angelika Steger Sven Kosub (Hrsg.)

Theoretische Grundlagen des Internets

Hauptseminar
Lehrstuhl für Effiziente Algorithmen
Fakultät für Informatik, Technische Universität München
Sommersemester 2002

Ausgewählte Seminararbeiten

Vorwort

Im Sommersemester 2002 fand am Lehrstuhl für Effiziente Algorithmen bereits zum zweiten Mal ein Hauptseminar zum Thema „Theoretische Grundlagen des Internets“ statt. Ziel des Seminars war, gemeinsam mit den Teilnehmern an Hand neuerer und neuester wissenschaftlicher Publikationen einen Überblick über die algorithmischen und mathematischen Hintergründe des Internets zu gewinnen und als Grundlage für zukünftige Lehr- und Forschungsaktivitäten zu konservieren. Eine besondere Betonung bei der Seminardurchführung lag deshalb auch auf der Anfertigung von Seminararbeiten.

Insgesamt wurden zwölf Themen aus den Bereichen Kommunikation, Protokolle, World Wide Web und Semistrukturierte Daten ausgegeben. Der vorliegende Band enthält die sieben Besten der Seminararbeiten.

Wir danken allen Teilnehmern für das Gelingen einer interessanten und abwechslungsreichen Seminarveranstaltung.

Oktober 2002

Ernst W. Mayr
Angelika Steger
Sven Kosub

Inhaltsverzeichnis

Kommunikation

Protokolle

World Wide Web

Semistrukturierte Daten

Dienstleistungsqualität in Virtuellen Privaten Netzwerken

Julian Würmser

Technische Universität München
wuermser@informatik.tu-muenchen.de

Zusammenfassung. Dieser Text dient als Einführung in das Thema Virtuelle Private Netzwerke (VPN). Er soll die grundlegenden theoretische Probleme bei der Konstruktion eines VPNs aufzeigen. Der Schwerpunkt dieser Ausführung liegt im Bereich der Dienstleistungsqualität. Die Aspekte der Bandbreitenreservierung und Ausfallssicherheit werden genauer untersucht und Lösungsansätze präsentiert.

1 Einleitung

Das Firmennetzwerk hat sich in den letzten Jahren zu einer überaus unternehmenskritischen Institution entwickelt. Anwendungen, wie Elektronische Mail, digitale Terminplanung, Ressourcenverwaltung und Dokumentensharing sind aus der heutigen Arbeitswelt kaum noch wegzudenken. Die Firmen, die über mehrere geographische Standorte verfügen und die Vorteile eines privaten Netzwerkes dennoch nutzen wollten, mussten bis vor kurzem kostenintensive Leitungen von Service-Providern mieten. Seit der Einführung des Virtuellen Privaten Netzwerkes (VPN) bietet sich jedoch die Möglichkeit, das Internet als Backbone für private Netzwerke einzusetzen. Die Vorteile einer solchen Lösung liegen auf der Hand. Neben einer immensen Kostensenkung, sinkt die Ausfallswahrscheinlichkeit, da im robusten Internet meist Alternativrouten existieren. Außerdem können sich Mitarbeiter von fast jedem beliebigen Standort auf das firmeneigene Netzwerk zugreifen. Für einen effektiven Einsatz haben sich drei Anforderungen an das VPN herauskristallisiert (vgl. [4]):

- *Geschlossene Packetverkapselung:* Datenpackete, die innerhalb eines VPNs transportiert werden, müssen nicht zwingend TCP/IP-konform sein. Deshalb ist es wichtig, dass sie ohne Einschränkung an Protokoll oder Adressierung in IP-Packete gekapselt werden können.
- *Datensicherheit:* Daten, die über das Firmennetzwerk versandt werden, sollen der Öffentlichkeit nicht frei zugänglich sein. Es wird eine sehr hohe Anforderung an den Datenschutz gestellt. Diese Anforderungen werden entweder durch ein striktes Routen oder durch eine Verschlüsselung erfüllt.
- *Garantie der Dienstleistungsqualität:* Für viele Anwendungen ist es wichtig, dass zeitkritische Daten rechtzeitig übertragen werden. Deshalb entstehen zunehmend Forderungen nach VPNs, die eine gegebene Bandbreite und Ausfallssicherheit garantieren.

In den letzten Jahren lag der Schwerpunkt bei der Weiterentwicklung des VPNs vor allem auf der Befriedigung der Sicherheitsaspekte. Tunnelmechanismen (vgl. [4, S. 3]) wie IP/IP, GRE Tunnels, IPSec und MPLS garantieren inzwischen hohe Sicherheitsstandards und gewährleisten eine packetunabhängige Verkapselung der Daten. Anwendungen wie Voice-over-IP und der zunehmenden Trend von Firmen, Software nicht mehr selbst zu installieren, sondern über das Internet von sogenannten Application-Service-Providern zu beziehen, rückt die Bandbreitenanforderung in den Vordergrund der Entwicklung. Der Aspekt der Dienstleistungsqualität (Quality of Service) erfährt neuen Aufschwung, da diese neuen Anwendungen auf zuverlässige und konstante Internetanbindungen angewiesen sind. Dieser Text beschäftigt sich mit den theoretischen Grundlagen, die es ermöglichen, eine im Voraus festgelegte Bandbreite auf möglichst effiziente Weise zur Verfügung zu stellen und eine Ausfallssicherheit zu garantieren.

2 Modelle und Definitionen

2.1 Modellierung eines VPN

Um die Dienstleistungsqualität eines VPNs analysieren und verbessern zu können, ist es wichtig, ein abstraktes Model eines VPN zu formulieren. Ein Netzwerk, bestehend aus Rechnern und Verbindungsleitungen, kann bekanntlich als ein Graph G bestehend aus einer Menge von Knoten V und Kanten E interpretiert werden. So lässt sich auch ein Teilsegment des Netzwerks, in unserem Fall das VPN, als Teilgraph $T = (V', E')$ auffassen. Terminals sind die Knoten $P \subseteq V'$ im VPN, die es zu einem geschlossenen Netzwerk zu verbinden gilt.

2.2 Modellierung der Bandbreitenanforderung

Es existieren zwei Modelle für die Spezifikation der Bandbreitenanforderung: Das etwas ältere Pipe Model und das Hose Model (vgl. [1]). Im Laufe der Jahre setzte sich jedoch zunehmend das Hose Model durch, da es im Gegensatz zum Pipe Model nur die Festlegung der maximale Sendekapazität B_i^{out} bzw. Empfangskapazität B_i^{in} eines jeden Terminals $i \in P$ benötigt. Außerdem lassen sich mit dem Hose Model Multiplexingvorteile nutzen, was in der Realität meist effizientere Lösungen liefert.

Die Datenflüsse, die zwischen den einzelnen Terminals auftreten können, werden in einer $|P| \times |P|$-Flussmatrix darstellen. Jeder Eintrag $d_{ij} \geq 0$ in dieser Matrix beschreibt die Stärke eines auftretenden Flusses von Terminal i nach Terminal j. Datenflüsse müssen nicht unbedingt symmetrisch sein. Wir sprechen von einer *gültigen Flussmatrix*, wenn alle Flüsse, die in einem Terminal einlaufen bzw. auslaufen, dessen Kapazität nicht übersteigen. Für alle $i \in P$ gilt

$$\sum_{j \in P} d_{ij} \leq B_i^{out} \quad \text{und} \quad \sum_{j \in P} d_{ji} \leq B_i^{in}.$$

3 Bandbreitenreservierungsproblem

3.1 Formulierung der Problemstellung

Nun führen wir die Funktion $C_T(i,j)$ ein, die die reservierte Kapazität auf der Strecke i nach j für $i,j \in V'$ beschreibt, ein. Des Weiteren definieren wir die Gesamtkosten, die für die reservierten Bandbreiten in einem Graphen $T = (V', E')$ auftreten mit C_T. Es gilt $C_T = \sum_{(i,j)\in E'} C_T(i,j)$.

Mit Hilfe der eingeführten Funktionen lässt sich nun das *Bandbreitenreservierungsproblem im VPN* formulieren: Ziel ist es, in einem gegebenem Graphen G für eine Menge P von Terminals mit entsprechenden Bandbreitenanforderungen einen optimalen Subgraphen T_{Opt} zu finden, der für jede gültige Flussmatrix eine Route zur Verfügung stellt. Optimalität bezieht sich auf die zu reservierende Bandbreite: $C_{T_{Opt}} = \min_{T\subseteq G} C_T$.

3.2 Differenzierung der Problemstellung

Das oben eingeführte Problem des Bandbreitenreservierungsproblem lässt sich durch zusätzliche Kriterien genauer spezifizieren (vgl. [3]):

- Struktur des Graphen,
- Symmetrie oder Asymmetrie von Empfangs- und Sendekapazität,
- Reservierungsbegrenzungen.

An die Struktur des zu reservierenden Graphen lassen sich verschiedene Anforderungen stellen. Der Graph kann beliebig gewählt werden, ein Baum sein oder andere topographische Einschränkungen erfüllen. In der Praxis wird oft die Baumstruktur bevorzugt, da diese Vorteile in Bezug auf die Adressierung in einem Netzwerk und dessen Skalierbarkeit bietet. Des Weiteren erhalten wir eine extreme Vereinfachung der Problemstellung, wenn wir davon ausgehen, dass die Sendekapazität eines Terminals seiner Empfangskapazität entspricht, da wir bei den Netzverbindungen die Richtung des Datenflusses völlig vernachlässigen können. Es gilt dann $B_i^{in} = B_i^{out}$. Leider können wir diese Einschränkung in der Realität nicht immer vornehmen und müssen auch den allgemeinen Fall betrachten. Auch zeigt sich in der Realität, dass es bei der Reservierung von Leitungskapazitäten Grenzen gibt. Diese Obergrenzen an den zu reservierenden Kapazitäten gestalten das Bandbreitenreservierungsproblem wesentlich schwieriger.

3.3 Symmetrischer Baum

In diesem Abschnitt wird das Bandbreitenreservierungsproblem mit einigen Einschränkungen behandelt. Der gesuchte Subgraph soll ein Baum sein, die Sende- bzw. Empfangskapazitäten sollen symmetrisch sein ($B_i = B_i^{in} = B_i^{out}$) und der globale Graph G weist keinerlei Beschränkungen hinsichtlich der reservierbaren Bandbreiten auf. Die Lösung für diesen Spezialfall ist, wie wir hier zeigen werden, in polynomieller Zeit berechenbar.

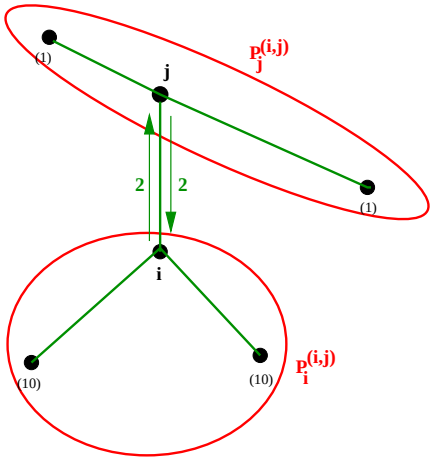


Abb. 1. Symmetrischer Baum.

Bevor wir uns mit der Konstruktion des Spannbaums, der alle Terminals verbindet, befassen, betrachten wir erst die Quantität der zu reservierenden Bandbreite. Wir gehen davon aus, dass die Topographie des zu reservierenden Baumes T bereits existiert und wir uns lediglich mit der Minimierung der Kosten $C_T = \sum_{(i,j)\in E'} C_T(i,j)$ befassen müssen. Unter der Annahme einer symmetrischen Kapazität für jeden Knoten ist klar, dass für die zu reservierende Bandbreite pro Kante (i,j) gilt: $C_T(i,j) = C_T(j,i)$. Betrachten wir nun eine Kante (i,j). Wir bezeichnen die Komponenten, in die der Baum T beim Wegfallen der Kante (i,j) zerfällt, als $P_i^{(i,j)}$ und als $P_j^{(i,j)}$. Hierbei bezeichnet $P_i^{(i,j)}$ die Komponente, die den Knoten i enthält und $P_j^{(i,j)}$ die andere. Für eine Kante (i,j) wird die zu reservierende Bandbreite immer durch die schwächeren der beiden Komponenten definiert:

$$C_T(j,i) = C_T(i,j) = \min\left\{ \sum_{l\in P_i^{(i,j)}} B_l, \sum_{l\in P_j^{(i,j)}} B_l \right\}.$$

Für die gesamte Bandbreitenreservierung in einem Baum T und einem Knoten $v \in T$ führen wir nun die Abschätzungsfunktion

$$Q(T,v) =_{\text{def}} 2\sum_{l\in P} B_l d_T(v,l)$$

ein. (Die Distanzfunktion $d_T(v,l)$ bezeichnet die Anzahl der Kanten auf dem kürzesten Pfad von Knoten v nach l.). Wir können zeigen, dass für jeden Baum T, der alle Terminals P beinhaltet, folgende Eigenschaften gelten:

Proposition 1. *1. Es gibt einen Knoten $w \in T$ ´, so dass $C_T = Q(T,w)$.*
2. Für alle Knoten $v \in T$ gilt $C_T \leq Q(T,v)$.

Beweis. Um die obigen Eigenschaften zu beweisen, konstruieren wir aus T einen gerichteten Baum T_{dir}. Wir geben jeder Kante $e = (i, j)$ in T nach folgenden Kriterien eine Richtung:

- wenn $\sum_{l \in P_i^{(i,j)}} B_l < \sum_{l \in P_j^{(i,j)}} B_l$, zeigt e in Richtung i,
- wenn $\sum_{l \in P_i^{(i,j)}} B_l > \sum_{l \in P_j^{(i,j)}} B_l$, zeigt e in Richtung j,
- wenn $\sum_{l \in P_j^{(i,j)}} B_l = \sum_{l \in P_j^{(i,j)}} B_l$, dann zeigt e in die Richtung der Komponente, die ein beliebiges Blatt $\hat{l}$ enthält.

Es ist klar, dass es einen Knoten in T_{dir} gibt, dessen Eingangsgrad gleich Null ist. Wir nennen diesen Knoten $a(T)$ und zeigen, dass er für jeden Baum T_{dir} eindeutig ist und die Gleichung $C_T = Q(T, a(T))$ erfüllt. Die Behauptungen ergeben sich dann aus den nachstehenden Lemmata. □

Lemma 2. *Jede Kante e in T_{dir} zeigt vom Knoten $a(T)$ weg.*

Beweis. Es sei $e = (i, j)$ eine Kante in T, so dass i näher an $a(T)$ ist als an j. Wir zeigen, dass e in T_{dir} von i nach j zeigt. Betrachten wir den Pfad von $a(T)$ nach i, so wissen wir, dass die erste Kante (u, v) auf diesem Weg von $a(T)$ wegzeigt. Es gilt $\sum_{l \in P_v^{(u,v)}} B_l \leq \sum_{l \in P_u^{(u,v)}} B_l$. Da (u, v) die erste Kante des Pfades von $a(T)$ nach i ist, gilt $P_u^{(u,v)} \subseteq P_i^{(i,j)}$ und $P_j^{(i,j)} \subseteq P_v^{(u,v)}$. Wir erhalten also

$$\sum_{l \in P_j^{(i,j)}} B_l \leq \sum_{l \in P_v^{(u,v)}} B_l \leq \sum_{l \in P_u^{(u,v)}} B_l \leq \sum_{l \in P_i^{(i,j)}} B_l.$$

Wenn nun $\sum_{l \in P_j^{(i,j)}} B_l < \sum_{l \in P_i^{(i,j)}} B_l$ gilt, so zeigt die Kante e von i nach j. Ansonsten muss $\sum_{l \in P_j^{(i,j)}} B_l = \sum_{l \in P_i^{(i,j)}} B_l$ gelten und somit wäre $P_u^{(u,v)} = P_i^{(i,j)}$ und $P_j^{(i,j)} = P_v^{(u,v)}$. Das kann gelten, wenn die Kante (i, j) mit der Kante (u, v) zusammenfällt, die aus $a(T)$ entspringt und somit von u nach v zeigt. □

Lemma 3. *Für die Kosten eines Baumes T gilt $C_T = Q(T, a(T))$.*

Beweis. Seien l ein Blatt und $e = (i, j)$ eine Kante auf dem Weg von $a(T)$ nach l (der Knoten i ist näher an $a(T)$ als j). Wir sehen, dass B_l zu den zu reservierenden Bandbreiten $C_T(i, j)$ und $C_T(j, i)$ beiträgt, da in T_{dir} die Kante e in Richtung j bzw. l orientiert ist. Allgemein ist zu erkennen, dass B_l genau in die Reservierung jedes Teilabschnittes des Pfades von $a(T)$ nach l einbezogen werden muss und auf alle anderen Reservierungen keinen Einfluss hat. □

Lemma 4. *Für jeden beliebigen Knoten v in T gilt $Q(T, a(T)) \leq Q(T, v)$.*

```
 1: procedure ComputeTreeSymmetric(G,P)
 2: T_opt := ∅
 3: für jeden Knoten v in G{
 4:     T_v := {v}
 5:     openQ:={v}
 6:     while (openQ≠ ∅){
 7:         Nimm den ersten Knoten u aus openQ
 8:         Für jede Kante (u,w) in G, so dass w nicht in T_v ist{
 9:             Füge Kante (u,w) zu T_v hinzu
10:             Hänge w an das Ende von openQ an
11:         }
12:     }
13:     Entferne alle Blätter von T_v, die keine Terminals sind
14:     if (C_{T_v} < C_{T_opt}) T_opt := T_v
15: }
16: return T_opt
```

Abb. 2. Der Algorithmus ComputeTreeSymmetric.

Beweis. Betrachten wir zunächst wieder eine Kante $e = (i, j)$. In T_{dir} ist die Kante von i nach j orientiert. Mit $\sum_{l \in P_j^{(i,j)}} B_l \leq \sum_{l \in P_i^{(i,j)}} B_l$ ergibt sich:

$$\begin{aligned} & Q(T,i) - Q(T,j) \\ & \quad = 2 \sum_{l \in P_i^{(i,j)}} B_l (d_T(i,l) - d_T(j,l)) + 2 \sum_{l \in P_j^{(i,j)}} B_l (d_T(i,l) - d_T(j,l)) \\ & \quad = -2 \sum_{l \in P_i^{(i,j)}} B_l + 2 \sum_{l \in P_j^{(i,j)}} B_l \quad \leq \quad 0. \end{aligned}$$

Die Differenz von $Q(T, i)$ und $Q(T, j)$ ist also immer kleiner gleich Null, wenn die Kante (i, j) in Richtung j orientiert ist. Da alle Kanten in T_{dir} von $a(T)$ weg orientiert sind, folgt direkt das Lemma. □

Somit gilt für den optimalen Baum T_{opt} stets $C_{T_{opt}} = Q(T_{opt}, a(T_{opt}))$. Folglich können wir die Suche nach einem optimalen Baum darauf reduzieren, dass wir für jeden Knoten v des Graphen den Baum T_v suchen, so dass $Q(T_v, a(T_v))$ minimal ist. Der optimale Baum T_{opt} entspricht dann dem Baum T_v der im Vergleich zu den anderen Bäumen T_v das kleinste $Q(T_v, a(T_v))$ hat.

Betrachten wir nun noch einmal die Abschätzungsfunktion $Q(T, v)$, so lässt sich intuitiv erkennen, dass sich diese Funktion minimieren lässt, indem wir die Distanzen jedes Knoten zum Knoten v möglichst gering halten. Es stellt sich heraus, dass der Breadth-First-Search-Algorithmus genau diese Optimierung durchführt und uns zu jedem v den lokal optimalen Baum T_v liefert. Der Algorithmus zur Lösung des Bandbreitenreservierungsproblem betrachtet also jeden Knoten im Graphen G, konstruiert zu diesem Knoten einen kostenoptimalen Spannbaum und merkt sich die zu reservierenden Kapazitäten für den entstandenen Baum. Die globale Lösung liefert der Baum, der die geringste Ka-

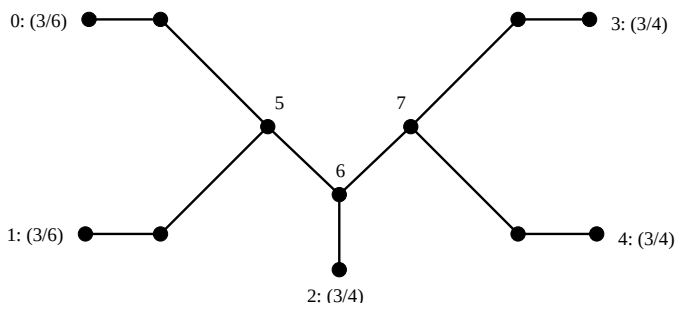


Abb. 3. Kosten für einen asymmetrischen Baum.

pazitätenreservierung benötigt. Ein Pseudoprogramm zur Berechnung des optimalen Baums T_{opt} bei gegebenem Graphen G und Terminals P findet sich unter Abbildung 2.

Für die Laufzeitabschätzung lässt sich leicht einsehen, dass der Aufwand von ComputeTreeSymetric $O(mn)$ ist,wobei $n = |V|$ die Anzahl der Knoten in G darstellt und $m = |E|$ die der Kanten. Im Programm wird die äußere For-Schleife genau n-mal abgearbeitet. In ihrem Inneren werden im ungünstigstem Fall für jeden Knoten m Kanten betrachtet. So ergibt sich $O(mn)$.

3.4 Asymmetrischer Baum (AsymT)

Bei dem Bandbreitenreservierungsproblem für einen asymmetrischen Baum wird, wie auch im symmetrischen Fall, nach einem Subgraphen, der eine Baumstruktur hat, gesucht. Jedoch muss an dieser Stelle zwischen Sendekapazität B_j^{out} und Empfangskapazität B_j^{in} unterschieden werden, da diese voneinander abweichen können. Hieraus ergibt sich auch, dass sich die Bandbreitenreservierung einer Kante je nach Richtung unterscheiden kann. Das heißt, $C_T(i,j)$ unterscheidet sich eventuell von $C_T(j,i)$ und es gilt

$$C_T(i,j) = \min\left\{ \sum_{l \in P_i^{(i,j)}} B_l^{out}, \sum_{l \in P_j^{(i,j)}} B_l^{in} \right\}$$

und

$$C_T(j,i) = \min\left\{ \sum_{l \in P_i^{(i,j)}} B_l^{in}, \sum_{l \in P_j^{(i,j)}} B_l^{out} \right\}.$$

Beispiel 5. Betrachten wir die Abbildung 3, so können wir ableiten, dass für die Kante (5,6) bzw. (6,5) eine Bandbreite von 9 bzw. 6 reserviert werden muss, da $\sum_{l \in P_5^{(5,6)}} B_l^{out} = 12$ größer als $\sum_{l \in P_6^{(5,6)}} B_l^{in} = 9$ und $\sum_{l \in P_6^{(5,6)}} B_l^{out} = 12$ größer als $\sum_{l \in P_5^{(5,6)}} B_l^{in} = 6$ ist.

Obwohl wir gesehen haben, dass sich bei einem bereits gefundenen Baum die Bandbreitenreservierung relativ einfach finden lässt, gilt folgendes Theorem [2].

Theorem 6. *Für den Fall von asymmetrischen Bandbreitenanforderungen ist das Problem der Berechnung eines optimalen VPN-Baumes, der alle Terminals verbindet,* **NP***-hart.*

3.5 Formulierung des Problems als Integer-Programm

Ein Integer-Programm-Problem ist ein Problem, das sich durch ein lineares Programm lösen lässt, das einzig und allein Integervariablen verwendet. In diesem Abschnitt wird gezeigt, dass sich das AsymT-Problem auch als Integer-Programm formulieren lässt. Um dies zu zeigen, müssen wir zuerst einige Eigenschaften des AysmT betrachten.

Eine Kante (i, j) eines AsymT bevorzugt den Knoten i, wenn die folgenden zwei Bedingungen zutreffen:

1. $\sum_{l \in P_i^{(i,j)}} B_l^{in} < \sum_{l \in P_j^{(i,j)}} B_l^{out}$ oder $(\sum_{l \in P_i^{(i,j)}} B_l^{in} = \sum_{l \in P_j^{(i,j)}} B_l^{out}$ und $P_i^{(i,j)}$ enthält einen spezifischen Knoten l) und
2. $\sum_{l \in P_i^{(i,j)}} B_l^{out} < \sum_{l \in P_j^{(i,j)}} B_l^{in}$ oder $(\sum_{l \in P_i^{(i,j)}} B_l^{out} = \sum_{l \in P_j^{(i,j)}} B_l^{in}$ und $P_i^{(i,j)}$ enthält einen spezifischen Knoten l).

Bevorzugt die Kante (i, j) weder i noch j, so heißt sie balanciert. Knoten, die auf einer balancierten Kante liegen, heißen Zentralknoten. Eine Kante bevorzugt i, wenn die schwächeren Sende- und Empfangsanforderungen in der Komponente mit dem Knoten i liegen und somit auch diese Komponente allein für die Quantität der Bandbreitenreservierung relevant ist. In unserem Beispiel ist die Kante $(5, 6)$ balanciert und die Kante $(6, 7)$ bevorzugt Knoten 7.

Wir geben nun einige Eigenschaften für balancierte Kanten an. Es sei $M = \min\{\sum_{l \in P} B_l^{in}, \sum_{l \in P} B_l^{in}\}$.

Lemma 7. *1.* [8] *Im AsymT muss auf Vorwärts- und Rückwärtskante einer balancierten Kante (i, j) zusammen genau eine Kapazität der Größe M reserviert werden. Es gilt also $M = C_T(i, j) + C_T(j, i)$.*

2. [8] *Die balancierten Kanten des Baumes T bilden eine Zusammenhangskomponente.*

Aus diesem Lemma folgt, dass alle Zentralknoten durch einen zusammenhängenden Baum verbunden sind, der nur aus balancierten Kanten besteht. Wird diese Kante gelöscht, so zerfällt der Baum in mehrere Zusammenhangskomponenten, die jeweils einen Zentralknoten enthalten. Wir nennen diese Komponenten C_v, wobei v den repräsentativen Zentralknoten bezeichnet. Sollte T keine ausgeglichene Kante enthalten, dann gibt es nur eine Komponente. Da alle Kanten in dieser Zusammenhangskomponente nicht balanciert sind, kann leicht bewiesen werden, dass es einen Knoten v gibt, von dem alle nicht balancierten Kanten wegführen. Dieser Knoten v wird dann als repräsentativer Zentralknoten aufgefasst.

Lemma 8. *1.* [8] *Für die oben eingeführten Zusammenhangskomponenten C_v zeigt sich, dass jede Kante (i, j) in C_v den Knoten j bevorzugt, sofern j weiter von v entfernt ist als i.*
2. [8] *Die Kosten der Zusammenhangskomponente C_v sind*

$$\sum_{l \in C_v \cap P} d_T(v, l)(B_l^{in} + B_l^{out}).$$

Also besteht ein AsymT T aus einer Menge von Zentralknoten core(T), die durch eine Menge balancierter Kanten bal(T) verbunden sind. Mit Hilfe der obigen Lemmata lässt sich eine Kostenfunktion

$$C_T =_{\text{def}} M|\text{bal}(T)| + \sum_{v \in \text{core}(T)} \sum_{l \in (C_v \cap P)} d_T(v, l)(B_l^{in} + B_l^{out})$$

für T aufstellen. Der hintere Teil gilt, da alle Kanten einer Komponente C_v, die Knoten bevorzugen, die am weitesten von Knoten v entfernt sind.

Wir machen Gebrauch vom Steiner-Baum-Problem. Gegeben ist ein positiv gewichteter Graph G mit einer Untermenge von Knoten (Terminals). Beim Steiner-Baum-Problem gilt es, einen (in Bezug auf die Summe der Kantengewichte) minimalen Graphen in G zu finden, der alle Terminals verbindet (vgl. [6]). Es lässt sich erkennen, dass sich jeder AsymT größtenteils durch seine Zentralknoten core(T) charakterisieren läßt. Betrachten wir eine Menge von Knoten $S(\in P)$ und definieren eine Kostenfunktion

$$C_S =_{\text{def}} Mb + \textstyle\sum_{l \in P}(\min_{v \in S} d_G(v, l))(B_l^{in} + B_l^{out}).$$

Hierbei gibt b die Anzahl der Kanten im Steiner-Baum an, die benötigt werden, um die Knoten S zu verbinden. Als nächstes zeigen wir, dass wir einen Baum $T(S)$ (d.h., einen Baum T charakterisiert durch S, der die Knoten $S \cup P$ enthält) konstruieren können, für den $C_{T(S)} \leq C_S$ gilt. Dies funktioniert in zwei Schritten:

1. Verbinde die Knoten S durch einen Steiner-Baum und füge alle benötigten Kanten zu $T(S)$ hinzu.
2. Vereinige alle Knoten von S zu einem Superknoten und konstruiere einen Breadth-First-Search-Baum mit dem Superknoten als Wurzel, der alle Knoten P enthält.

Aus Lemma 9 folgt, dass die Berechnung eines optimalen AsymT durch das Finden einer Menge von Knoten S, wobei C_S minimal ist, gelöst werden kann.

Lemma 9. *1. Für eine Knotenmenge S gilt $C_{T(S)} \leq C_S$.*
2. Für einen VPN-AsymT T gilt $C_{core(T)} \leq C_T$.

Theorem 10. *Sei S eine Menge von Knoten, für die C_S minimal ist, so ergibt $T(S)$ den optimalen AsymT.*

Mit diesem Theorem lässt sich nun das gesuchte Integer-Programm aufstellen, das zu einem gegebenem Knoten v die Knotenmenge S findet: Wir benötigen

die 0-1-Variablen x_{ij},y_i und z_e, wobei $y_i = 1$, wenn $i \in S$, $x_{ij} = 1$, wenn ein Terminal j einem Knoten i zugeordnet wird, und $z_e = 1$, wenn eine Kante e zum Steiner-Baum gehört, der die Knoten S verbindet. Des Weiteren bezeichnet $\delta(\hat{V})$ die Menge von Kanten, für die $\delta(\hat{V}) = \{(i,j)|(i,j) \in G \wedge i \in V \wedge j \in (V \setminus \hat{V})\}$ gilt. Außerdem gilt $B_j = B_j^{in} + B_j^{out}$. Ist nun einen Knoten v aus der Menge S bekannt, so ergibt folgende Aufgabenstellung die optimale Lösung: Finde geeignete x_{ij},y_i und z_e, so dass

$$\sum_{i \in V, j \in P} d_G(i,j) B_j x_{ij} + M \sum_{e \in E} z_e$$

minimal wird, wobei folgende Einschränkungen eingehalten werden müssen:

1. Für alle $j \in P$ gilt $\sum_{i \in V} x_{ij} \geq 1$.
2. Für alle $\in V, j \in P$ gilt $y_i - x_{ij} \geq 0$.
3. Für alle $\hat{V} \subseteq V, v \notin \hat{V}$, und $j \in P$ gilt $\sum_{e \in \delta(\hat{V})} z_e - \sum_{i \in \hat{V}} x_{ij} \geq 0$.

Die erste Einschränkung besagt, dass jedes Terminal mit mindestens einem inneren Knoten i aus V verbunden ist. Die zweite stellt sicher, dass jeder innere Knoten, dem ein Terminal zugeordnet wurde, auch zu der Menge S gehört. Die dritte Bedingung besagt, dass, wenn ein Knoten zu S gehört, dieser durch Steiner-Kanten mit v verbunden werden muss.

Die Schritte zum Finden des optimalen AsymT sehen wie folgt aus:

1. Löse für jeden Knoten $v \in V$ das Integer-Program, um S_v zu berechnen.
2. Gib den Baum $T(S_v)$ zurück, dessen Kosten minimal sind.

Leider ist die Lösung des Integer-Programmes sehr aufwendig, wir können uns aber mit verschiedenen Approximationsverfahren behelfen (vgl. [5]).

4 Ausfallssicherheit

Die Dienstgüte eines VPN definiert sich nicht nur durch die Garantie einer gewissen Bandbreite, sondern auch durch das Gewährleisten von Ausfallssicherheit. Normalerweise ist die Ausfallssicherheit um so höher, je mehr Redundanz in einem System besteht. Für unseren konkreten Fall eines VPN bedeutet dies, dass wir zusätzliche Kanten in unser Netzwerk aufnehmen, die beim Ausfall einer Primärkante, deren Datenströme übernehmen. Natürlich entstehen auch hier wieder Kosten, die es zu minimieren gilt.

Wird zum Beispiel garantiert, dass ein VPN T trotz Ausfall einer beliebigen Kante, das gleiche Durchsatzvermögen gewährleistet, so muss für jede Kante in T eine oder mehrere Ersatzpfade reserviert werden, deren Gesamtkapazität gleich der Kapazität der ausgefallenen Kante ist. Selbst wenn die Kapazitäten der einzelnen Kanten vernachlässigt werden und nur nach einem Graphen gesucht wird, der die Terminals trotz Ausfall einer Kante verbindet, zeigt sich, dass eine Lösung dieses Problems **NP**-hart ist [7].

Italiano, Rastogi und Yener [9] ist es gelungen, ausgehend von einer optimalen Lösung des SymT, das Finden von optimalen(im Sinne von Bandbreitenreservierung) Ersatzpfaden auf einen konstanten Faktor zu approximieren. Ihr Algorithmus reduziert die Problemstellung auf das Problem, einen bestehenden Graphen möglichst kostengünstige Kanten hinzuzufügen, so dass sich ein zweifach verknüpfter Graph ergibt (2-approximierbar in polynomieller Zeit).

5 Ausblick

In den obigen Abschnitten wurden Algorithmen für das Bandbreitenreservierungsproblem vorgestellt, die unter gewissen Einschränkungen akzeptable Lösungen liefern. Schon für den asymmetrischen Fall lässt sich die optimale Lösung nur noch approximieren, da das Problem **NP**-hart wird. Noch problematisch wird es, wenn wir die zur Reservierung bereitstehenden Leitungskapazitäten nach oben beschränken. Die Approximationsverfahren liefern dann sehr schlechte Lösungen. Ähnlich verhält es sich auch im Bereich der Ausfallssicherheit. Schon sehr einfache Anforderungen an die Ausfallssicherheit sprengen den Rahmen der exakten *und* effizienten Berechenbarkeit. Jedoch werden hier sehr gute Approximationsansätze gefunden.

Literatur

1. N. G. Duffield, P. Goyal, A. G. Greenberg, P. P. Mishra, K. K. Ramakrishnan, J. E. van der Merwe. A flexible model for resource management in virtual private networks. In: *Proceedings Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'99)*, S. 95–108. ACM Press, 1999.
2. S. Even. *Graph algorithms.* Computer Science Press, Rockville, MD, 1979.
3. A. Gupta, J. M. Kleinberg, A. Kumar, R. Rastogi, B. Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. In: *Proceedings 33rd ACM Symposium on Theory of Computing (STOC'2001)*, S. 389–398. ACM Press, 2001.
4. B. Gleeson, A. Lin, J. Heinanen, G. Armitage, and A. Malis. A framework for IP based virtual private networks. *IETF Request for Comments 2764*, 2000.
5. D. S. Hochbaum (Hrsg.). *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, MA,1997.
6. J. Schlichting, J. Hartmann, H. Pohl. Virtual Private Networks auf IPsec-Basis - Test und Evaluierung ausgewählter Produkte. In: *Proceedings Fachtagung Sicherheit in Informationssystemen (SIS'2000)*, vdf Verlag, Zürich, 2000.
7. S. Khuller, U. Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41(2):214-235, 1994.
8. A. Kumar, R. Rastogi, A. Silberschatz, B. Yener. Algorithms for provisioning virtual private networks in the hose model. In: *Proceedings Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'2001)*, S. 123–134. ACM Press, 2001.
9. G. F. Italiano, R. Rastogi, B. Yener. Restoration algorithms for virtual private networks in the hose model. In: *Proceedings IEEE Conference on Computer Communication (INFOCOM'2002).* IEEE Computer Society Press, 2002.

Messungen in großen Netzwerken

Franz Prilmeier

Technische Universität München
prilmeie@informatik.tu-muenchen.de

Zusammenfassung. Häufig fehlen in großen, unstrukturierten Netzwerken Aussagen über wichtige Leistungsparameter. Dies kann verschiedene Gründe haben. Oft liegt es jedoch daran, dass das Netzwerk in einer unkontrollierten Art und Weise gewachsen ist. Um ein solches Netzwerk sinnvoll weiterentwickeln zu können, ist es notwendig, verschiedene Messungen durchzuführen, um diese Leistungsparameter zu erhalten. Eine sehr wichtige Messung ist dabei die Erkennung der Netzwerktopologie. Viele weitere Messungen bauen auf Erkenntnissen dieser grundlegenden Messung auf.
Dieser Artikel wird dabei einen Überblick über einige Arbeiten geben, die sich mit Messungen, Messmethoden und/oder Metriken für große Netzwerken wie z.B. dem Internet beschäftigen. Dabei wird die Erkennung der Netzwerktopologie im Vordergrund stehen, da diese eine der grundlegendsten Fragestellungen darstellt.

1 Einleitung

Seit den letzten fünf Jahren gibt es verstärkt Bestrebungen, Messungen in größerem Netzwerken wie z.B. dem Internet durchzuführen. Dass diese Messungen nicht nur für Theoretiker interessant sind, liegt auf der Hand, da viele Messergebnisse Entscheidungen durch Netzwerkadministratoren oder Personen mit ähnlichem Berufsprofil erleichtern. Interessante Leistungsparameter eines Netzwerks werden z.B. in [13] vorgestellt. Die vorliegende Arbeit versucht dabei einen groben Überblick über die Erkennung von verschiedenen Leistungsparametern zu geben, indem einige erst kürzlich erschiene Arbeiten kurz zusammengefasst und, falls möglich, miteinander verglichen werden.

Daher wird in Abschnitt 2 zuerst eine der grundlegendsten Eigenschaften eines Netzwerkes ermittelt, die Topologie. Aufbauend auf dieser Information können dann weitere Messungen durchgeführt werden. Dabei werden dort vier Arbeiten vorgestellt, die sich intensiv mit diesem Thema beschäftigen. Alle Ergebnisse und Methoden dieser Arbeiten werden dabei, soweit sinnvoll, miteinander verglichen. Der Abschnitt schließt mit einer Bewertung der erreichten Ergebnisse ab.

Abschnitt 3 beschäftigt sich mit dem relativ jungen Begriff der Internet-Tomographie. Unter Netzwerk-Tomographie wird allgemein eine Schlussfolgerung auf direkt nicht messbare Sachverhalte aus einfacheren, messbaren Daten verstanden. Damit wird ein großer Teil von denkbaren Messungen oder Fragestellungen abgedeckt. Bei der Tomographie steht jedoch nicht die Messung selbst

im Vordergrund, sondern mehr die verwendete Methode. Dabei unterscheidet man zwei Hauptrichtungen, zum einem die aktive Messung über End-zu-End Verbindungen, zum anderen eine passive Messung durch das Protokollieren von echtem Netzwerkverkehr. Drei Arbeiten, die sich mit diesem Begriff beschäftigt haben, werden dort behandelt.

Einen ganz anderer Weg für Messungen in Netzwerken wird in 4 vorgestellt. Auf die Topologie des Netzwerkes wird vollständig verzichtet. Die Arbeit, die diesen Weg vorschlägt, untersucht diese Vorgehensweise anhand des Problemes der Erkennung von Netzwerkfehlern. Dabei wird mehr Wert auf die analytische Seite als auf die tatsächliche Durchführung dieses Ansatzes gelegt.

Den Abschluss bilden die Abschnitte 5 und 6. In 5 werden noch einige weitere interessante Arbeiten genannt, die leider nicht ausführlich behandelt werden konnten. Die Arbeit schließt mit Abschnitt 6, in dem das Thema Messen in Netzwerken, wie es sich in den vorangegangenen Abschnitten darstellt, zusammengefaßt wird.

2 Netzwerk-Topologie

Viele große Netzwerke wie z.B. das Internet erlebten im letzten Jahrzehnt ein exponentielles Wachstum von angeschlossen Rechnern. Da das Internet ein aus vielen dezentralisierten Teilnetzwerken bestehendes Netzwerk darstellt, gibt es ad hoc keinen Überblick, wie die Rechner verteilt sind und wie sie miteinander verbunden sind. Eine Topologie des Internets würde genau einem solchen Überblick entsprechen. Dabei werden Rechner als Knoten, Verbindungen zwischen den Rechnern als Kanten dargestellt. Üblicherweise wird das Internet als ungerichteter Graph modelliert. Dies ist durch die Protokolle der Transportschicht TCP/UDP bedingt, auf welche sich das Internet stützt. Beide Protokolle verlangen Quittierungen, daher muss Kommunikation in beiden Richtungen möglich sein. Nur [2] behauptet, dass das Internet aufgrund spezieller Richtlinien zum Routing einen gerichteten Graphen darstellt.

Zwei der wichtigsten Verwendungsmöglichkeiten einer derartigen Karte nennt [19]. Zum einen ist sie für den Netzwerkadministrator einer Domäne äußerst interessant. Dort kann er einerseits Sicherheitsschwachstellen oder Flaschenhälse im Netzwerk schnell identifizieren, andererseits bietet ihm diese Karte eine exzellente Grundlage um das Netzwerk evtl. umzustrukturieren bzw. um neue Elemente zum Netzwerk hinzuzufügen. Andererseits gibt es viele Algorithmen, die eine Topologie bzw. Adjazenzlisten zur Berechnung von Sachverhalten voraussetzen. Zu nennen wären hier Algorithmen zur Berechnung von Flüssen in Netzwerken oder Algorithmen, die minimale Spannbäume berechnen. Eine solche Karte würde jedoch noch ganz andere Interessenten finden, wie z.B. Militärs [4, 5]. Mit dieser Karte lassen sich allerdings die Rechner nicht geographisch zuordnen, obwohl es in letzter Zeit Bestrebungen gibt genau dies zu erreichen [18, 12]. Dabei soll der Domain-Name-Service-Dienst (DNS) um Ortsinformationen erweitert werden.

Die Topologie von Teilnetzwerken wird oft durch Pläne dokumentiert, die bei der Erstellung des Netzwerks angelegt wurden. Diese Pläne spiegeln oft auf Grund schlechter Pflege oder eigenmächtiger Änderungen des Netzwerks nicht die Verhältnisse des tatsächlichen Netzwerkes wider.

Deshalb wurden in den letzten Jahren diverse Arbeiten darüber veröffentlicht, wie die Topologie eines Netzwerkes automatisiert erkannt werden kann.

2.1 Messmethoden

Eine gemeinsame Bestrebung aller Arbeiten ist es, die eingesetzten Methoden so einfach wie möglich zu halten, d.h. so wenig Voraussetzungen an das Netzwerk wie möglich zu machen, da viele historisch gewachsene Netzwerke über verschiedene technische Stände verfügen. Die eingesetzten Methoden selbst lassen sich in zwei Grundrichtungen klassifizieren:

- Messung der Topologie nur mit Bordmitteln, z.B. `traceroute` oder `ping`,
- Messungen mit eigens zu diesem Zweck entwickelten Programmen.

Den ersten der beiden Wege beschreibt [19]. Dabei werden die Bordmittel `ping` bzw. `broadcast ping`, `traceroute` und `DNS zone transfer` in geeigneter Weise kombiniert. Dies wird ergänzt durch eine Reihe von Heuristiken, die aus dem Netzwerk der Cornell University extrahiert wurden. Alle Heuristiken dienen dazu, die Subnetmaske eines Netzwerks oder gültige IP-Adressen eines Netzwerks zu erraten. Zusätzlich wird noch SNMP (Simple Network Managament Protocol [3]) erwähnt, das für diesen Zweck besonders gut geeignet sei. Viele kommerzielle Werkzeuge, die Topologien erkennen können, wie OpenView von Hewlett-Packard oder Tivoli von IBM, basieren auf diesem Protokoll. Leider ist SNMP für die Messung der Topologie größerer Netzwerke nicht anwendbar, da sicherheitsbewusste Netzwerkadministratoren dieses in der Regel deaktivieren. So antworteten bei Messungen im Netzwerk der Cornell University nur 8% der Rechner auf SNMP-Anfragen. Deshalb ist eines der Hauptziele dieser Arbeit durch andere Methoden die Ergebnisse zu erreichen, die SNMP liefern würde. Ein Einsatz der vorgestellten Methoden im Internet ist nicht sinnvoll, da die angewendeten Heuristiken aus lokalen Verhältnissen im Netzwerk der Cornell University extrahiert wurden. Daher wird für die Topologie-Erkennung im Internet eine andere Vorgehensweise vorgeschlagen. Dort wird BGP (Border Gateway Protocol [17]) eingesetzt, um gültige Netzwerknummern zu erlangen. Dann werden in diesen Netzwerkdomänen verschiedene nach einer Heuristik ausgewählte Rechner mit dem `traceroute` Programm adressiert.

Auf diese Weise werden neue Rechneradressen erlangt, die wiederum adressiert werden, um evtl. Umleitungen zu erkennen. Ein grundlegendes Problem bei dieser Messmethode stellt die Tatsache dar, dass bei einer Messung an nur einem Ort eine Menge von meist disjunkten Pfaden als Ergebnis erhalten wird. Verbindungen zwischen den Pfaden werden mit dieser Messmethode kaum erkannt. Um dieses Problem zu umgehen wird vorgeschlagen, die Messung an mehreren voneinander unabhängigen Plätzen im Internet vorzunehmen.

Der zweite Weg wird z.B. in [8] beschritten. Dort wird auf ein extra für diesen Zweck entwickeltes Programm (Mercator) zurückgegriffen. Dieses baut wie auch `traceroute` auf ICMP (Internet Control Message Protocol [16, 7]) auf. Dadurch werden alle Rechner entlang eines Pfades identifiziert. Mit Hilfe des TTL (Time to live) Attributes wird dabei die Entfernung jedes Rechners, der sich entlang des Pfades zurückgemeldet hat, ermittelt. Dabei wird der Wert dieses Attributes solange inkrementiert, bis der jeweilige Rechner erreicht wurde. Bereits ermittelte Entfernungen zu Rechnern werden, um Netzwerklast zu vermeiden, kein zweites Mal verifiziert, sondern es werden nur die Entfernungen zu Rechnern ermittelt, die neu erfasst wurden. Diese Methode wird als *Path Probing* bezeichnet. Eine zweite Methode, *Source-Routed Path Probing* baut auf das nach [15] zulässige Routing vom Ursprungsrechner auf. Da dieses für Denial-of-Service-Attacken missbraucht werden kann, ist es bei nur sehr wenigen Rechnern aktiviert. Nur 8% aller Router im Internet ließen *source-routing* zu. Mit dieser zweiten Methode können alternative Wege zu Rechnern ermittelt werden. [19] schlug zu diesem Zweck Messungen von verschiedenen Orten aus vor, hier wird dies durch die geschickte Nutzung von *source-routing* umgangen. Um festzustellen, ob ein Rechner diese Eigenschaft aufweist, wird versucht, über diesen Rechner einen weiteren Rechner auf einem nicht-zugewiesenen Port anzusprechen. Wird dabei mit einem `ICMP port unreachable` geantwortet, so ist dieser Rechner fähig, *source-routing* einzusetzen.

Da die Möglichkeit besteht einem Rechner verschiedene IP-Adressen zuzuordnen, muss diese Abbildung wieder umgekehrt werden. Dafür wird eine Heuristik namens *alias probe* verwendet. Es werden dazu UDP-Pakte an einen nichtexistenten Port eines Rechners gesendet. Ist dabei die Quelladresse der `ICMP port unreachable`-Antwort eine von der Zieladresse verschiedene, so wurde hier ein Alias für ein und den selben Rechner erkannt. In [19] wurde diese Tatsache nicht behandelt.

Während den Messungen wurden diverse Probleme durch Fehlkonfigurationen oder Softwarefehler von Routern erkannt. Obwohl die meisten derartigen Fehler nur störend sind, gibt es einen Fehler, der je nach Auftrittshäufigkeit das Potenzial hat, die Güte der ermittelten Messergebnisse zu beeinflussen. Dabei wurde nichtdeterministisch das TTL-Attribut nicht erniedrigt.

Die Ergebnisse sind nach eigenen Aussagen vielversprechend, Vergleiche mit den tatsächlichen Topologien zweier mittlerer Internet Service Provider (ISP) erbrachten beinahe vollständige Übereinstimmung. Eine einzige Verbindung zwischen zwei Rechnern wurde nicht erkannt. Ein Vergleich der Ergebnisse mit [4] ergab eine weitaus bessere Ausbeute an erkannten Netzwerkelementen. Die Autoren mutmaßen, dass eine verteilte Messung noch bessere Ergebnisse erzielen könnte.

Ein weiterer Vertreter der zweiten Methode ist [10]. Diese Arbeit ist während der Arbeit von CAIDA[1] entstanden. Das dazu entwickelte Programm (Skitter) ähnelt in den Methoden stark dem in [8] diskutierten Mercator. Auch hier wird ICMP verwendet, und auch hier wird das TTL-Attribut zur Erkennung

[1] Cooperative Association for Internet Data Analysis, `http://www.caida.org`.

von Pfadlängen verwendet. Auf *source-routing* wird hingegen verzichtet. Das Auflösen von Aliase einzelner Router wurde durch den gleichen Ansatz wie in [8] vorgenommen. Allerdings wird das Programm in einem verteiltem Umfeld eingesetzt, die Messungen wurden gleichzeitig an 18 gezielt ausgewählten Orten im Internet durchgeführt. Damit wurde das Problem der Vollständigkeit der Topologie angegangen. Zusätzlich wurde diese Messung durch eine große Liste an bereits bekannten Internetadressen vorbereitet. Bei Mercator hingegen wurde auf diese Liste bewusst verzichtet.

Leider sind keine vergleichbaren Messergebnisse vorhanden, so dass die Leistungsfähigkeit von Skitter im Gegensatz zu Mercator nicht verglichen werden kann. Es wurden zwar drei Messungen aufgeführt, diese lassen sich jedoch aufgrund der speziellen Messziele nicht mit den Ergebnissen von Mercator vergleichen. Dies liegt auch daran, dass [8] kaum Aussagen über Messergebnisse macht.

2.2 Sicherheitsaspekte

Verschiedene Arbeiten berichten darüber, dass ihre tatsächlich durchgeführten Messungen auch Sicherheitsfragen betreffen. So berichten [8, 4], dass ihre Messungen mit Angriffen durch Hacker verwechselt wurden und sie dadurch teilweise in Erklärungsnöte kamen. Die Autoren von [4] haben daher in ihren Datagrammen den Zweck der Aktivität als Nutzdaten mitgeschickt.

Auch stellen Messdaten eines Netzwerkes einen nicht unerheblichen Wettbewerbsfaktor bei ISP dar. Daher wird oftmals wie z.B. in [4] vermutet, dass Rechner von ISP keine ICMP-Meldungen verschicken. Dies kann jedoch durch [8] nicht bestätigt werden, diese Situation konnte dort nur bei einem ISP festgestellt werden. Eine allgemein zugängliche Topologie eines dem Internet angeschlossenen Netzwerkes kann für Angriffe auf das Netzwerk [4] missbraucht werden, kann jedoch auch der Abwehr bzw. der Rückverfolgung solcher Angriffe dienen.

2.3 Visualisierung

Ein ganz anderer Aspekt der Internet-Topologie ist die Visualisierung der Daten. In [19, 8] wird über die Darstellung der Daten keine Aussage gemacht. In [4] werden Möglichkeiten diskutiert, wie Topologie-Daten visualisert werden können. Die Diskussion wird vor allem durch Performance- und Übersichtlichkeitsfragen geprägt. Letztendlich wird ein minimal aufspannender Baum berechnet, der dann gedruckt werden kann. In [10] werden dagegen vier Methoden gezeigt, wie die Topologie eines Netzwerkes selbst visualisiert werden kann.

AS-Core-Graph. Es wird versucht, Mengen von gültigen IP-Adressen auf sog. Autonome Systeme (AS) abzubilden, welche im wesentlichen mit ganzen ISP vergleichbar sind. Die Abbildung wird mit Hilfe von Informationen, die aus BGP-Tabellen extrahiert wurden erreicht. Dies erleichtert die Visualisierung deutlich, da Teilnetzwerke komplett ausgeblendet werden, und nur noch die Verbindungen

zwischen diesen Teilnetzwerken für die Karte relevant sind. Um zusätzliche Übersicht zu ermöglichen, wird der Graph auf einen Kreis abgebildet, in dem alle AS mit ihrem Verbindungsgrad von innen nach außen abnehmend abgebildet werden. Eine weitere Anordnung wurde durch Ortsinformationen angewendet, die durch das Programm NetGeo [12] ermittelt wurden. Der Winkel, in dem ein AS auf dem Kreis aufgetragen wurde, entspricht dabei dem ermittelten Längengrad des Firmensitzes des jeweiligen ISP.

Ein interessantes Ergebnis dieser Visualisierung ist, dass die meisten Verbindungen von Europa nach Asien einen Umweg über Amerika nehmen. Direkte Verbindungen zwischen AS in Europa und Asien existieren hingegen kaum.

Dispersionsgraph. Auf Grund der zahlreichen Vereinfachungen durch Gruppierung von IP-Adressen zu AS, wird die Nachbarschaft einzelner Rechner verdeckt. Durch den Dispersionsgraphen wird die Nachbarschaft einzelner Rechner zu anderen AS visualisiert. Dazu wird ein Diagramm verwendet, bei dem in x-Richtung die Entfernung zweier Rechner angetragen wird, in y-Richtung wird dann der Anteil eines AS anteilsmäßig eingetragen. Verschiedene AS werden durch Farben und Nummern voneinander im Graphen getrennt.

Hyperbolischer Raum. Wie bei AS-Core-Graphen wird auch hier die Topologie visualisiert. Diesmal jedoch nicht als flacher Graph, sondern in einem hyperbolischen Raum. Dazu wird das Programm `walrus` verwendet. Dies verschafft einen detaillierten Überblick über die Topologie, dennoch geht der globale Zusammenhang bei dieser Methode nicht verloren.

Bidirektionale Pfade. Bei dieser Methode wird die Betrachtung auf nur einige Teile eines Pfades fokussiert. Dadurch lassen sich alternative Routen, gemeinsame Routen und Routen, die bei der Lastbalancierung ausgewählt wurden, schnell erkennen.

2.4 Diskussion

Alle vier Arbeiten verwenden für ihre Messungen das ICMP-Protokoll. Während [19] dies indirekt über die Implementierung von `traceroute` vornimmt, so wird das ICMP-Protokoll und dessen Eigenschaften direkt von [8, 10, 4] ausgenutzt. Der Einsatz des eigentlich viel besser geeignetem SNMP-Protokoll, das auch von kommerziellen Programmen verwendet wird, ist aufgrund der schlechten Verfügbarkeit nicht sinnvoll, um eine Topologie des Internets zu erstellen.

Insgesamt lässt sich feststellen, dass das Problem, die Messung der Internet-Topologie, noch nicht zufriedenstellend gelöst werden konnte. Die Hindernisse, die diesem Ziel entgegenstehen sind vielfältig. Auf der einen Seite stehen die niedrigen technischen Vorraussetzungen, die für eine solche Messungen im Internet angenommen werden können. Dies hat unter anderem den Grund, dass die Dienste, die viele Rechner anbieten, in Furcht vor möglichen Angriffsflächen

auf das Nötigste reduziert wurden. Zum anderen kann das Internet aufgrund der Vielzahl von angeschlossenen Rechnern nur in einer längeren Messung einigermaßen vollständig kartographiert werden. Die Anzahl und die Position der angeschlossenen Rechner kann in einem solchen Zeitraum keinesfalls als stabil betrachtet werden. Dann verhindern spezielle Routingschemata das Erkennen aller Verbindungen zwischen einzelnen Rechnern. Zusätzlich ist das Erkennen einer Topologie von außen für einige ISP oder andere Netzwerkbetreiber im Internet schlicht unerwünscht und wird daher durch Abschaltung von den für diesen Zweck wichtigen Protokollen ICMP oder SNMP verhindert. An die Möglichkeit, dass UDP-Pakete von diversen Firewalls blockiert werden könnten, wird in keiner der hier behandelten Arbeiten gedacht. Schließlich sind die erzielten Ergebnisse nicht zwangsläufig reproduzierbar und stark vom Ort der Messung abhängig.

Ein ganz anderes Problem ist, dass das verwendete ICMP-Protokoll nur Rechner erfasst, die Pakete über IP weiterleiten. Wie [4] feststellt, wird dadurch z.B. der ISP Cable and Wireless beinahe zu einem einzigen Knoten zusammengefasst, da hier Netzwerkpakete über ATM weitergeleitet werden. Diese Methode wird sicherlich nicht nur bei Cable and Wireless eingesetzt.

3 Internet-Tomographie

Unter der Tomographie eines Netzwerkes wird die Ermittlung komplexer Netzwerk-Leistungsparameter mit Hilfe von statistischen Methoden aus Messergebnissen verstanden, die aus einfacheren Fragestellungen ermittelt wurden. Dabei wird die Messung häufig von verschiedenen Orten aus gleichzeitig durchgeführt. Der Begriff selbst ist noch recht jung und wurde zuerst 1996 von Vardi geprägt [23]. Auch die vorher behandelte Topologie kann als Tomographie gesehen werden. Kürzlich dazu erschienene Arbeiten sind [20, 6, 1]. Die Arbeit [1] wird hier jedoch in dem Sinn nicht weiter betrachtet, als dass bereits [20] diese Arbeit diskutiert.

Dabei unterscheidet [20] zwischen zwei typischen Problemstellungen. Zum einen der Schluss auf Datentransfercharakteristika auf Grund von Datenflussmessungen, zum anderen der Schluss auf Netzwerkcharakteristika mit Hilfe von End-zu-End-Messungen. Typische Vertreter für die erste Problemstellung ist nach [20] die Berechnung der durchschnittlichen Paketverluste. Ein konkretes Beispiel der zweiten Problemstellung ist die Ermittlung der Auftrittsrate von anormalen Paketen, das sind z.B. ICMP-Nachrichten. Weiterhin wird die normale Vorgehensweise für solche Aufgabenstellungen diskutiert. Da es sich hierbei um statistische Probleme handelt, liegt es auf der Hand, einen Maximum-Likelihood-Schätzer zu suchen. Um diesem Problem zu begegnen, wird versucht eine Methode zu finden, um aus den Messdaten die ursprüngliche Funktion zurückzugewinnen. Als letzte Vorgehensweise wird die Erwartungsmaximierung genannt. Am Ende der Arbeit wird noch ein Überblick über andere Arbeiten in diesem Bereich gegeben, unter anderem auch [1]. Insgesamt ist [20] ein guter Einstieg in das Thema Netzwerk-Tomographie.

Etwas umfangreicher als [20] ist [6]. Auch dort wird versucht einen Überblick über das Thema Netzwerk-Tomographie zu geben. Dabei werden die dort disku-

tierten Fragestellungen im Gegensatz zu [20] tatsächlich gemessen und die Messergebnisse bewertet. Konkret sind das Erkennung der Topologie, die Erkennung von Verlustraten und die Verteilung der Verzögerungsraten entlang der einzelnen Verbindungen. Insgesamt wird festgestellt, dass die Forschung im Bereich Tomographie noch sehr jung ist. Am Ende der Arbeit werden die nächsten logischen Schritte zur Erforschung der Netzwerk-Tomographie aufgezeichnet. Ein Fernziel ist, sinnvolle Modelle entwickeln zu können, an denen Eigenschaften von Netzwerken simuliert werden können. Die Arbeit schließt mit der Frage, ob Methoden aus der Signalverarbeitung entwickelt werden können, um mit deren Ergebnisse Netzwerke sinnvoll zu erweitern.

Eine weitere Untersuchung zu Tomographien auf mathematischer Ebene ist auch [5]. Dort wird das Thema Entfernungsrealisierungen von Rechnern behandelt. Ausgehend von einer Menge von Rechner (Knoten) $\mathcal{S}$ und einer Entfernungsmatrix D wird ein Graph $\mathcal{G}$ gesucht, so dass die Entfernungen aller Knoten in $\mathcal{G}$ minimal sind. Speziell wird hier untersucht, wie schwierig sich diese Fragestellung aus algorithmischer Sicht darstellt. Dabei wurden folgende Ergebnisse erzielt.

1. Das allgemeine Problem, von einer gegebenen Menge an Rechnern aus die Entfernungen zu realisieren, ist nicht nur schwierig, sondern ist zusätzlich nicht stabil lösbar. Kleinste Variationen der Entfernungsmatrix $\mathcal{D}$ können die Lösung in der Größenordnung von $|\mathcal{S}|$ beeinflussen. Es gibt zudem Fälle, in denen eine Matrix $\mathcal{D}'$ existiert, für die $d'_{i,j} \geq d_{i,j}$ gilt, die Lösung selbst jedoch besser ist. Approximationen der Lösung sind ebenfalls schwierig.
2. Eine Aufweichung der Fragestellung ist **NP**-vollständig. Diese Aufweichung wird das schwache Problem zur Entfernungsrealisierung genannt. Dabei wird nicht mehr gefordert, dass die gegebenen Entfernungen exakt eingehalten werden müssen, sondern die Lösung höchstens gleich gute Entfernung enthalten darf.
3. Das schwache Problem zur Entfernungsrealisierung kann in polynomieller Zeit bis auf den Faktor zwei approximiert werden.

Abwandlungen der vorgestellten Messtechnik sind auch für andere Tomographien anwendbar.

4 Eine etwas andere Messmethode

Ein ganz anderer Ansatz hinsichtlich der Messmethode wird bei [11] verfolgt. Dort wird versucht, Netzwerkfehler zu erkennen. Ein Netzwerkfehler im Sinne dieser Arbeit liegt vor, falls das Netzwerk durch den Ausfall von Verbindungen oder Knoten in Teilkomponenten zerfällt. Dabei werden die der Mindestgröße der erkennbaren Teilkomponenten und die maximale Anzahl auftretender Fehler durch frei wählbare Parameter festgelegt.

Die gewählte Messmethode stützt sich auf ein System verteilter Softwareagenten. Diese stehen ständig miteinander in Kontakt, bricht der Kontakt zwischen zwei Agenten ab, so liegt ein erkennbarer Netzwerkfehler vor. Da sich ein

großes Netzwerk nicht stabil verhält, sondern sich während der Beobachtung stetig ändert wird auf die Verwendung von Topologie-Informationen verzichtet. Dafür wird jedoch das Konzept der VC-Dimension [22] verwendet, das sich auch häufig in dem Bereich der Künstlichen Intelligenz findet. Zusätzlich wird darauf verzichtet, Leistungen jederzeit zu garantieren, sondern es werden diese mit einer von den gewählten Parametern abhängigen Wahrscheinlichkeit angegeben. Dies wird durch folgende Formalisierung der Problemstellung erreicht.

Sei k die maximale Anzahl der auftretenden Fehler, n die Anzahl der im Netzwerk G vorhandenen Rechner und sei εn die Mindestgröße einer erkennbaren Teilkomponente. Die Parameter k und ε sind dabei frei wählbar. Um Unterscheidung zwischen Rechner (Knoten) oder Verbindung (Kanten) im Netzwerk (Graph) zu vermeiden, werden diese *Elemente* genannt. Sei Z eine Menge mit höchstens k Elementen. Dies sind die Elemente des Netzwerkes, die ausfallen werden, das resultierende Netzwerk ist $G \setminus Z$. Eine Teilmenge $D \subseteq V$ der im Netzwerk vorhandenen Rechner heißt (ε, k)-*erkennend*, falls für jede Menge Z sich immer ein Paar $u, v \in D$ findet, das sich in verschiedenen Zusammenhangskomponenten von $G \setminus Z$ befindet. Die Problemstellung ist nun, eine solche Menge D zu finden und dessen Mindestgröße zu bestimmen. In [11] wurde diese Untersuchung bereits durchgeführt. Die zentrale Aussage ist:

Theorem 1. *Es gibt in jedem Graph G eine (ε, k)-erkennende Menge D mit $O\left(k^3 \varepsilon^{-1} \log\left(k\varepsilon^{-1}\right)\right)$ Elementen. Eine Menge mit*

$$O\left(k^3 \varepsilon^{-1} \log\left(k\varepsilon^{-1}\right) + \varepsilon^{-1} \log \delta^{-1}\right)$$

zufällig ausgewählten Knoten besitzt diese Eigenschaft mit einer Wahrscheinlichkeit von $1 - \delta$.

Der Beweis dieser Aussage ist äußerst trickreich. Der Autor verbindet dazu die Idee der VC-Dimension [22] mit ε-Netzen [9]. Für die VC-Dimension wird ein Mengensystem benötigt. Ein Mengensystem besteht aus einer Grundmenge Ω und einer Menge $\mathcal{X}$ von Teilmengen von Ω. Die *VC-Dimension von Ω* ist die Kardinalität der größten Teilmenge A von Ω, so dass für alle $B \subseteq A$ eine Menge $X \in \mathcal{X}$ existiert mit $B = A \cap X$. Es stellt sich heraus, dass die Begriffe (ε, k)-erkennend und ε-Netz in diesem Kontext äquivalent sind. Die VC-Dimension des Mengensystems ist dabei mit dem Parameter k gleichzusetzen. Die eigentliche Hauptaufgabe besteht darin ein Mengensystem mit möglichst kleiner VC-Dimension zu finden. Um die Beweisidee zu verdeutlichen, wird zuerst die Analyse auf den Fall angewandt, dass nur Kanten ausfallen, jedoch keine Knoten. Es stellt sich heraus, dass sich in diesem Fall ein Mengensystem mit VC-Dimension von maximal $2k + 1$ finden lässt. Im allgemeinen Fall, d.h. es können entweder Knoten oder Kanten ausfallen, findet sich ein Mengensystem mit VC-Dimension $O\left(k^3\right)$.

Mit Hilfe dieses Konzepts, so der Autor, würden sich noch Messungen ganz anderen Typs realisieren lassen. Da dieses Ergebnis noch relativ jung ist und Grundlagenforschung darstellt, gibt es leider derzeit weder weiterführende Untersuchungen noch Aussagen über praktische Erfahrungen mit diesem Ansatz.

Dieser Ansatz ist in dem Sinne sehr ungewöhnlich, als dass er gänzlich neue Wege beschreitet, indem auf die Netzwerktopologie verzichtet wird. Andererseits ist es auf diese Weise unmöglich zu bestimmen, wie viele Fehler aufgetreten sind, wo sie aufgetreten sind, oder wie viele Teilnetzwerke durch die Störung entstanden sind. Diese Fragestellungen bedürfen weitere Untersuchungen.

5 Weitere Arbeiten

Es gibt unzählige weitere Arbeiten zum Thema Messungen im Internet. Dieser Abschnitt soll einen kurzen Überblick über andere Dokumente geben, die aus Platzgründen oder vom Thema her bisher in dieser Arbeit nicht berücksichtigt wurden.

Die Simulation von Netzwerken ist eine brauchbare Vorgehensweise, um Metriken oder zugehörige Messergebnisse in einem Modell zu überprüfen. Dazu bietet [21] einen sehr guten Einstieg. Dort wird die allgemeine Meinung, dass das Internet hierarchisch geordnet ist, widerlegt. Dies geht aus dem Vergleich von Modellen hervor, die mit einem hierarchisch ordnendem Algorithmus erzeugt wurden. Modelle, die mit dem power-law erzeugt wurden, approximieren tatsächliche Netzwerke viel besser.

Auf einen etwas anderen Bereich bei Messungen im Internet zielt [14] ab. Dort wird eine Architektur für Messungen propagiert. Die Ziele dieser Architektur sind vielfältig. So wird ein Augenmerk auf die wichtigsten Entwurfskriterien großer Rahmenwerke für Software geworfen, insbesondere Skalierbarkeit, Anpassbarkeit, Bedienbarkeit und Sicherheit. Auf Grund letzterer wird auch davon abgeraten, passive Messungen durchzuführen, da hier datenschutzrechtliche Probleme entstehen können. Abschließend werden diese Merkmale an Hand des Programmes vorgeführt, das im Rahmen des NIMI-Projektes [2] im Entstehen ist.

Eine weitere Arbeit zum Thema Topologie ist [2]. Die Unterschiede zu den meisten anderen Arbeiten zu diesem Thema beginnen bereits bei der Modellierung des zu untersuchenden Graphen. Während große Netzwerke sonst als ungerichtet betrachtet werden, wird hier explizit darauf hingewiesen, dass ein Netzwerk besser als gerichtet betrachtet wird. Neben neuen Untersuchungen zum Thema Topologie, in dem vor allem die Erkennung spezieller Netzwerkstrukturen im Vordergrund steht, werden in einer Zusammenfassung viele Arbeiten abgehandelt, die sich mit dem Thema Topologie beschäftigt haben. Darunter befinden sich auch diejenigen, die in der vorliegenden Arbeit diskutiert werden.

Mit einer ganz anderen Problematik im Bereich Messungen setzt sich [13] auseinander. Dabei stehen nicht so sehr die Messungen selbst im Vordergrund, sondern die zugehörigen Metriken. Der Autor arbeitet dabei die Probleme heraus, die bei einer unvorsichtigen Formulierung einer Metrik entstehen können. Der Autor unterscheidet dabei zwischen Metriken, die aus analytischer Sicht entstanden sind und Metriken, die von empirischer Seite kommen. So kann eine Metrik aus analytischer Sicht völlig korrekt sein, sich jedoch in der Praxis

[2] National Internet Measurement Infrastructure, `http://ncne.nlanr.net/nimi`.

auf Grund verschiedenster technischer Gründe nicht sinnvoll umsetzen lassen. Schlimmer noch ist die Auswertung von Messergebnissen mit falschen Annahmen über die tatsächlichen Verhältnisse in einem Netzwerk. Das Ziel der Arbeit ist es, ein sinnvolles Rahmenwerk für Metriken in Netzwerken zu schaffen.

6 Zusammenfassung

Wie die vorangegangenen Abschnitte gezeigt haben, sind Messungen in großen, dynamischen Netzwerken trotz verschiedenster Bemühungen immer noch schwierige Aufgaben. Aufgrund der Größe verschiedener Netzwerke sind die Ergebnisse von Messungen zu schnell veraltet oder schlecht auswertbar. Wegen des technischen Standes eines Netzwerkes sind oft nur grundlegendste Techniken einsetzbar, um tatsächlich Messungen durchzuführen. Andere Messungen hingegen sind nicht nachvollziehbar oder abhängig vom Messstandort. Dennoch sind Messungen in Netzwerken sinnvoll und wünschenswert. Vor allem das Thema Quality of Service (QoS) verlangt nach Messungen, wie sonst sollten sich Kunden von der tatsächlichen Garantie dieser Leistung überzeugen können?

Insgesamt sind sogar einfachste Fragestellungen, wie die Topologie eines Netzwerkes derzeit nicht befriedigend zu beantworten. Es gibt nicht einmal übereinstimmende Aussagen darüber, wie viele Rechner dem Internet angeschlossen sind. Während [14] von 30 Millionen spricht, wird etwa zwei Jahre zuvor, [13] von nur 9,5 Millionen Rechnern gesprochen. Deshalb bleibt das Thema Messungen in Netzwerken, insbesondere Netzwerk-Tomographie, ein weiterhin spannendes Thema.

Literatur

1. A. Adams, T. Bu, R. Caceres, N. G. Duffield, T. Friedman, J. Horowitz, F. Lo Presti, S. B. Moon, V. Paxson, D. Towsley. The use of end-to-end multicast measurement for characterizing internal network behavior. *IEEE Communications*, 38(5):152-158, 2000.
2. A. Broido, K. C. Claffy. Internet topology: Connectivity of IP graphs. In: *Proceedings SPIE International Symposium on the Convergence of Information Technology and Communication (ITCom+OptiCom'2001)*, Band 4526 der *Proceedings of the SPIE*, S. 172-187. SPIE Press, 2001.
3. J. D. Case, M. Fedor, M. L. Schoffstall, J. Davin. A simple network management protocol. *IETF Request for Comments 1157*, 1990.
4. B. Cheswick, H. Burch, S. Branigan. Mapping and visualizing the Internet. In: *Proceedings USENIX Annual Technical Conference*, S. 1-13. The USENIX Association, 2000.
5. F. R. K. Chung, M. W. Garrett, R. L. Graham, D. Shallcross. Distance realization problems with applications to Internet tomography. *Journal of Computer and System Sciences*, 63(3):432-448, 2001.
6. M. Coates, A. Hero, R. Nowak, B. Yu. Internet Tomography. *IEEE Signal Processing Magazine*, 19(3):47-65, 2002.
7. A. Conta, S. Deering. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6). *IETF Request for Comments 2463*, 1998.

8. R. Govindan, H. Tangmunarunkit. Heuristics for Internet map discovery. In: *Proceedings IEEE Conference on Computer Communication (INFOCOM'2000)*, S. 1371-1380. IEEE Computer Society Press, 2000.
9. D. Haussler, E. Welzl. Epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127-151, 1987.
10. B. Huffaker, D. Plummer, D. Moore, K. C. Claffy. Topology discovery by active probing. In: *Proceedings IEEE Symposium on Applications and the Internet Workshops (SAINT'2002 Workshops)*, S. 90-96. IEEE Computer Society Press, 2002.
11. J. M. Kleinberg. Detecting a network failure. In: *Proceedings 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS'2000)*, S. 231-239. IEEE Computer Society Press, 2000.
12. D. Moore, R. Periakaruppan, J. Donohoe, K. C. Claffy. Where in the world is netgeo.caida.org? In: *Proceedings 10th Annual Internet Society Conference (INET'2000)*. The Internety Society, 2000.
13. V. Paxson. Towards a framework for defining Internet performance metrics. Technischer Bericht LBNL-38952, Lawrence Berkeley National Laboratory, 1996.
14. V. Paxson, J. Mahdavi, A. Adams, M. Mathis. An architecture for large-scale Internet measurement. *IEEE Communications Magazine*, 36(8):48-54, 1998.
15. J. Postel. Internet Protocol. *IETF Request for Comments 791*, 1981.
16. J. Postel. Internet Control Message Protocol. *IETF Request for Comments 792*, 1981.
17. Y. Rekhter, T. Li. A Border Gateway Protocol 4 (BGP-4). *IETF Request for Comments 1771*, 1995.
18. E. Röscheisen. Spurensuche - Zuordnung von IP-Adressen zu Ortsinformationen. *iX Magazin für professionelle Informationstechnik*, 8:90-93, 2002.
19. R. Siamwalla, R. Sharma, and S. Keshav. Discovering Internet topology. Technischer Bericht, Department of Computer Science, Cornell University, Ithaca, NY, 1998.
20. M. Tsuru, Y. Oie. Introduction to the network tomography. Technischer Bericht IN2001-106, The Institute of Electronics, Information, and Communication Engineers, 2001.
21. H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, W. Willinger. Network topologies, power laws, and hierarchy. Technischer Bericht USC-CS-01-746, Computer Science Department, University of Southern California, 2001.
22. V.N. Vapnik, A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probabilities and its Applications*, 16:264-280, 1971.
23. Y. Vardi. Network tomography: Estimating source-destination traffic intensities from link data. *Journal of the American Statistical Association*, 91(433):365-377, 1996.

Selbstähnlichkeit des Datenverkehrs in paketorientierten Netzwerken

Patrick Bossel

Technische Universität München
bossel@in.tum.de

Zusammenfassung. Ankunftsraten in öffentlichen Telefonnetzen können gut mit Exponentialverteilungen modelliert werden, weil diese ein homogenes und vorhersagbares Verhalten über große Zeitskalen besitzen. Bei Netzwerken, die Datenpakete übertragen, funktioniert diese Modellierung wegen ihrer hohen zeitlichen und örtlichen Dynamik nicht mehr. Es müssen andere Modelle an Stelle des Poisson-Modells gefunden werden, die bessere Ergebnisse bei paketorientierten Netzen liefern. Dazu wird das Kriterium der Selbstähnlichkeit solcher Netze auf unterschiedlichen Skalierungsebenen betrachtet. Nachfolgend werden verschiedene statistische Methoden beschrieben, um Art und Grad der Selbstähnlichkeit von Messreihen (im konkreten Fall *Internet traffic traces*) festzustellen. Abschließend behandelt diese Seminararbeit auch mathematische Modelle, um selbstähnliche Prozesse zu erzeugen und die Frage, inwiefern sie geeignet sind, Netzwerkverkehr zu simulieren, dessen statistische Eigenschaften - insbesondere Skalierungscharakteristiken - möglichst gut mit denen des gemessenen Verkehrs übereinstimmen.

1 Einleitung

Beim Messen der Ankunftsraten in öffentlichen Telefonnetzen ist festzustellen, dass diese, über Stunden betrachtet, zum größten Teil konstant sind. Die Anzahl der Anrufe in diesen Stunden-Intervallen ist also um einen konstanten Faktor λ verteilt, der auch als Intensitätsrate bezeichnet wird. Diese Beobachtung der Ankunftszeiten kann mit einer Exponentialverteilung modelliert werden. Die Exponentialverteilung steht in engem Zusammenhang mit der Poisson-verteilung denn sobald die Ankunftszeiten exponentialvereilt sind, ist die Anzahl der Ankünfte bis zu einem Zeitpunkt poissonverteilt. Ein Poisson-Prozess ist ein stochastischer Prozess, bei dem die Anzahl der Ankünfte bis zum Zeitpunkt t Poisson-verteilt sind mit Parameter λ_t.

Betrachten wir einzelne Stichproben einer Exponentialverteilung, so haben diese die Eigenschaft, sich um einen bestimmten Wert zu häufen. Sie werden andererseits an den Enden der Verteilung sehr selten. Diese Häufungseigenschaft spiegelt das sich wiederholende Verhalten von Anrufern während eines normalen Arbeitstags wider [4]. Die stündliche Ankunftsrate steigt während den Hauptgeschäftszeiten und fällt auf den Abend hin ab. Was zusätzlich auf Konstanz hindeutet, ist, dass die Dauer der Telefonate gut vorhersehbar ist.

Werden statt des öffentlichen Telefonnetzes paketgesteuerte Netzwerke wie das Internet betrachtet, scheint eine Exponentialverteilung zur Modellierung nicht mehr geeignet zu sein. Um die Unterschiede zu veranschaulichen wird in Abbildung 1 aus [8] eine Poisson-Modellierung des Internetverkehrs (linke Seite) mit einem real gemessenen Netzwerkdatenstrom verglichen, dessen Zeitskalen sich auf der rechten Seite befinden. Diese Spur von Netzwerkdaten wurde an einem zentralen Link gesammelt, der eine große amerikanische Firma mit dem Internet verbindet.

Das Modell hat den selben Erwartungswert und die selbe Varianz wie der gemessene Datenstrom. In den einzelnen Reihen sind unterschiedliche Zeitskalen dargestellt. Falls sich das Netzwerkverhalten im Internet durch ein Poisson-Modell darstellen ließe, müssten gleiche Zeitskalen ähnliches Aussehen haben, was auf die konstante Ankunftsrate λ zurückzuführen wäre.

Die oberste Reihe enthält einen zufällig ausgewählten Zeitplot, jeder der beiden Spuren auf einer Skala von 100ms. Das heißt, in dem Plot der Länge 6 Sekunden gibt jeder Punkt die Anzahl der in einem 100ms-Intervall beobachteten Pakete an. Die zweite Reihe gibt eine um den Faktor 10 vergröberte Zeitskala an: Jeder Punkt gibt jetzt die Anzahl der Paketankünfte in einer Sekunde an über einen Zeitraum von einer Minute. Die schwarzen Regionen zeigen den Ausschnitt der in der Reihe darüber dargestellt ist. Zu beachten ist, dass nicht nur die Skalierung der X-Achse, sondern auch die der Y-Achse sich vergrößert.

Die dritte Reihe wächst noch einmal um den Faktor 10 und die vierte um den Faktor 6, die nun das Intervall einer ganze Stunde beschreibt. Es gibt einen offensichtlichen Unterschied zwischen dem Poisson-Modell und dem eigentlichen Verkehr: Mit zunehmend vergröberter Zeitskala glättet sich der Poisson-Verkehr immer mehr, während dies beim gemessenen Verkehr nicht der Fall ist.

Dieser Unterschied ist von größter Bedeutung bei der Simulation und Konstruktion von Netzwerken: Bei der Verwendung eines Poisson-Prozesses zur Simulation der Paket-Ankünfte, können wir nie sicher sein, ob der Netzwerkknoten der Paketflut standhält oder durch zu lange Ankunftsraten über einem bestimmten Level übersättigt wird. Falls der Paketverkehr im Internet sich durch eine Exponentialverteilung beschreiben ließe, wäre, unter Verwendung der mittleren Ankunftsraten über lange Zeitskalen z.B. die Größen für Speicher oder Warteschlangen (*buffers*) leicht festzulegen. Doch weil der reale Internetverkehr sehr viele Bursts beinhaltet, ist eine weit höhere Dynamik des Netzwerkverkehrs-Systems anzunehmen. Deswegen müssen neue Modelle, die dieser Variabilität besser Rechnung tragen, gefunden werden.

2 Der Varianz-Zeit-Plot als Indikator von Selbstähnlichkeiten im Datenverkehr

Betrachten wir den aktuellen Netzwerkverkehr rechts in Abbildung 1, ist festzustellen, dass der Plot, über alle Zeitskalen hinweg, die gleiche ausbruchslastige Struktur aufweist. Diese Eigenschaft wird als *Selbstähnlichkeit* bezeichnet. Falls Zählungen über die Zeit (wie die Anzahl der Pakete) diese Eigenschaft besitzen,

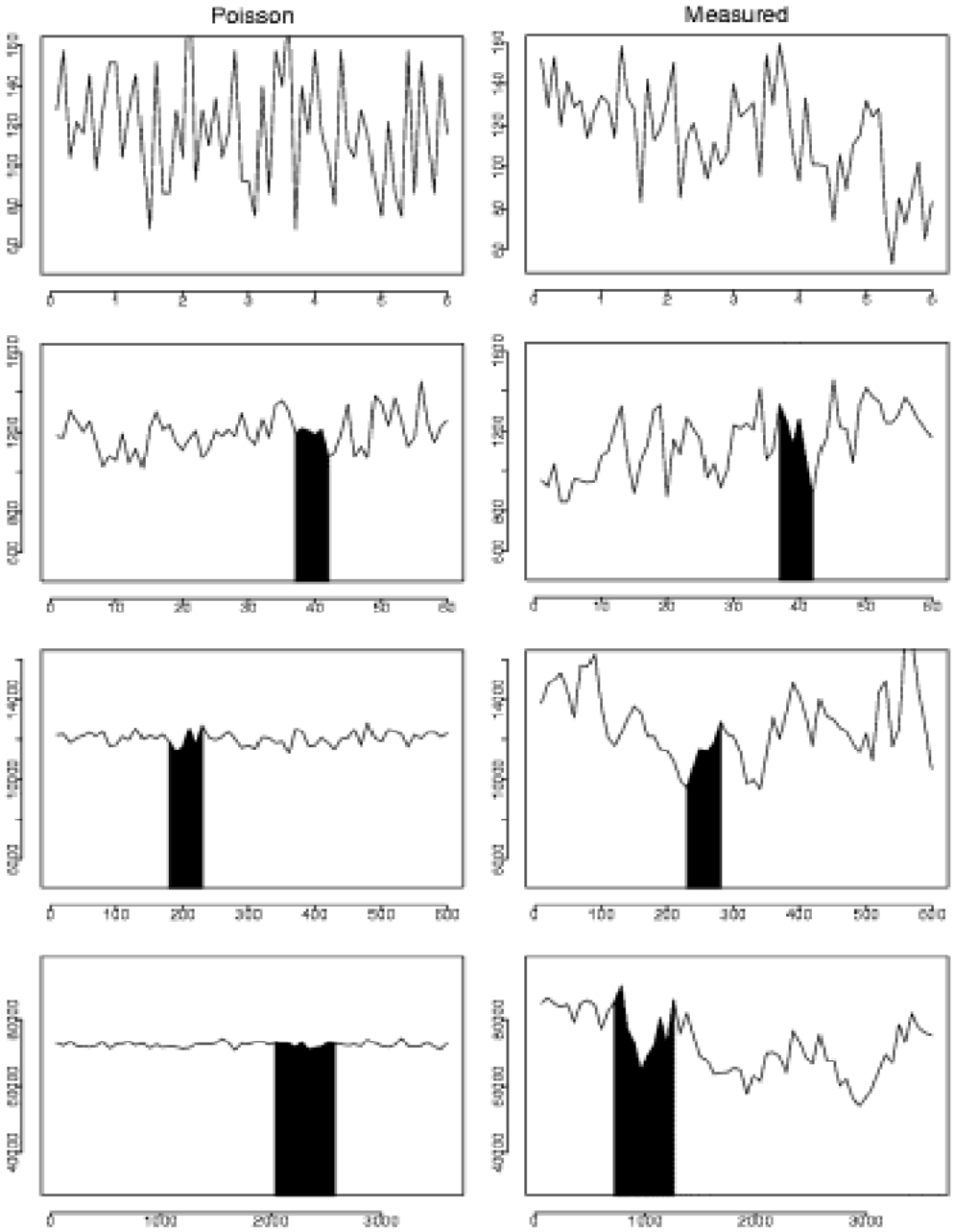


Abb. 1. Eine Poisson-Modellierung im Vergleich mit einem aktuellen Internetverkehr.

müssen wir auf plötzliche große Ausbrüche (*packet bursts*) gefasst sein. In [7] wird darauf hingewiesen, dass der Internetverkehr, der durch die Übertragung von FTP-Datenpaketen zustande kommt, sogar wesentlich durch die stärksten Bursts dominiert wird.

Um Selbstähnlichkeit im WAN-Verkehr festzustellen, gibt es veschiedene Methoden, von denen eine hier beschrieben wird: Da alle möglichen Werte für die Paketanzahl in Frage kommen, hat die Verteilung der in der Verkehrsspur gemessenen Pakete, eine sehr hohe Varianz. Es lässt sich beobachten, dass die Varianz, bei steigender Zeitskala sehr langsam abnimmt.

Diese langsame Abnahme der Varianz tragen wir gegenüber der Aufsummierungsstufe M der Zeitintervalle (Schritte von einer Zeitskala zur nächsten) auf und erhalten den Varianz-Zeit-Plot der Verkehrsspur. Je größer M wird, umso größer ist auch die Zeitskala und die Anzahl der Werte, die gemittelt werden. Die X-Achse gibt die Größe der Varianz der gemittlelten Stichprobe an und die Y-Achse die dazugehörige Varianz. Es lässt sich also durch Betrachtung der Varianz über der Zeit eine Analyse der Bursts vornehmen und diese leicht mit denen vergleichen, die sich bei einer Exponentialverteilung entwickeln würden. Für Zählprozesse, die eine schnell fallende Autokorrelationsfunktion besitzen, so wie z.B. Poisson-Prozesse, verhält sich die Varianz eines auf M aufsummierten Prozesses wie $1/M$-mal die Varianz eines nicht aufsummierten Prozesses. Für Prozesse mit langlebigeren Autokorrelationsfunktionen ist die Abnahme der Varianz schwächer. Solche Prozesse besitzen Abhängigkeiten über lange Zeitspannen hinweg (*long range dependence*), die mit der Stärke der Autokorrelationsfunktion gemessen werden (Messung der Höhe der zeitlichen Variabilität).

Eine interessante Anmerkung: Poisson-Prozesse eignen sich dennoch zur Simulation von *remote-login* (TELNET-Session Ankünfte können z.B. durch eine Exponentialverteilung modelliert werden). Dies hat nichts mit den oben beobachteten Netzverkehrsmessungen zu tun, die die übertragenen Pakete während einer solchen Sitzung darstellen. Ähnliche Resultate erhalten wir bei der Beobachtung von Benutzern, die eine WWW-Sitzung starten (Öffnen eines Browsers) oder einen File-Transfer-Prozess initiieren. Aber alles, was sich während dessen ereignet, hat höhere Ausschläge und kann deswegen nicht durch die beschränkte Variabilität von Poisson-Prozessen ausgedrückt werden.

3 Statistische Grundlagen

Um zukünftige Missverständnisse zu vermeiden, wird hier erklärt, was es mit den statistischen Fachbegriffen (stationäre) Kovarianzfunktion, Korrelationskoeffizient und Autokorrelationsfunktion auf sich hat und wie die Autokorrelationsfunktion mit kurz- bzw. langzeitlicher Abhängigkeit zusammenhängt.

Die *Kovarianzfunktion* ist durch $\gamma = \mathbf{E}\left[(X - \mathbf{E}X)(Y - \mathbf{E}Y)\right]$ bestimmt. Sie gibt den Zusammenhang zwischen den Zufallsvariablen X und Y an. Die Zufallsvariablen, aus denen das Produkt besteht, sind um null zentriert, d.h. es gilt $\mathbf{E}(X - \mathbf{E}X) = 0$ und $\mathbf{E}(Y - \mathbf{E}Y) = 0$. Es ist leicht festzustellen, dass das Produkt positiv ist, wenn X und Y tendenziell einen gleichsinnigen linearen Zusammen-

hang aufweisen, hingegen negativ, wenn X und Y einen gegensinnigen linearen Zusammenhang besitzen [3]. Hier bedeutet gleichsinnig, dass falls X größer (bzw kleiner) wird, auch Y steigt (bzw. fällt). Gegensinnig bedeutet, dass, wenn X größer (bzw kleiner) wird, Y fällt (bzw. steigt). Sind die beiden Variablen nicht zusammenhängend (unkorreliert), also von einander unabhängig, besitzt folglich der Korrelationskoeffizient den Wert null. Da wir hier die Autokorreliertheit von einer Variablen (also die Stärke ihrer Abhängigkeit über die Zeit) betrachten, ist die *Autokovarianzfunktion* eines stochastischen Prozesses X hier definiert für alle ganzen Zahlen k durch

$$\gamma(t,k) =_{\text{def}} \mathbf{E}\left[(X_t - \mathbf{E}X_t)(X_{t-k} - \mathbf{E}X_{t-k})\right].$$

Die Autokovarianzfunktion ist symmetrisch, d.h. $\gamma(t,k) = \gamma(t-k,-k)$. Für den Spezialfall $k=0$ ergibt sich die Varianzfunktion $\gamma(t,0) = \mathrm{Var}X_t$. Im Allgemeinen hängt $\gamma(t,k)$ sowohl von t als auch von k ab.

Im Folgenden wird das wichtige Prinzip der Stationarität definiert, das es ermöglicht, die Momentfunktionen in vielen Fällen zu vereinfachen.

Definition 1. *1. Ein stochastischer Prozess X ist* kovarianzstationär*, falls es μ und γ_k für alle k gibt, so dass $\mathbf{E}X_t = \mu$ und $\gamma(t,k) = \gamma_k$ für alle t gilt.*

2. Ein stochastischer Prozess X_t ist streng stationär*, wenn für beliebige $t_1, \ldots, t_n$ und für alle s gilt*

$$F_{t_1,\ldots,t_n}(X,...,X) = F_{t_{1+s},\ldots,t_{n+s}}(X,...,X).$$

Häufig wird für Kovarianzstationarität auch der Begriff der schwachen Stationarität verwendet. Zu beachten ist jedoch, dass ein stochastischer Prozess streng stationär sein kann, ohne kovarianzstationär zu sein, nämlich dann, wenn die Varianz (oder die Kovarianz) nicht existiert. Existieren die ersten beiden Momentfunktionen, so folgt aus der strengen Stationarität die Kovarianzstationarität.

Definition 2. *Die* Autokorrelationsfunktion (ACF) ϱ *eines kovarianzstationären stochastischen Prozesses ist definiert als*

$$\varrho_k =_{\text{def}} \frac{\gamma_k}{\gamma_0} = \frac{\gamma_k}{\sigma^2}.$$

Die ACF ist normiert auf das Intervall $[-1,1]$ und erleichtert somit die Interpretation von Autokovarianzstrukturen verschiedener stochastischer Prozesse. Da gefordert wurde, dass der Prozess kovarianzstationär ist, hängt die ACF nur von einem Parameter, der Verzögerung k ab. Häufig wird die ACF als Funktion von k gezeichnet, und ergibt das sogenannte *Korrelogramm*. Dies ist ein wichtiges graphisches Instrument, um lineare Abhängigkeitsstrukturen des Prozesses aufzudecken.

Wir unterscheiden folgende Arten von Abhängigkeiten bei kovarianzstationären Prozessen:

1. *Short-Range Dependence (SRD)*, falls für die ACF gilt $\varrho_k \sim c^{-k}$ mit geeigneter Konstante c und $\sum_{k=0}^{\infty} \varrho_k < \infty$.
2. *Long-Range Dependence (LRD)*, falls für die ACF gilt $\varrho_k \sim k^{-\beta}$ für $0 < \beta < 1$ und $\sum_{k=0}^{\infty} \rho_k = \infty$.

4 Mathematische Eigenschaften von selbstähnlichen Prozessen

Für den Netzwerkdesigner bedeutet die Selbstähnlichkeits-Eigenschaft des Internetverkehrs Folgendes: Egal, wie groß die Bandbreite eines Links auch ist, es wird Bursts geben, für die sie nicht ausreicht. Anders als beim Verkehr in öffentlichen Telefonnetzen (der durch einen Poisson-Prozess simulierbar ist und bei dem daher die Wahrscheinlichkeiten für Bursts exponentiell mit der Größe seiner Skalierung abnehmen), der eine konstante Senderate über große Zeitintervalle hat, variiert die Senderate bei selbstähnlichen Prozessen auch dann noch, wenn große Zeitintervalle betrachtet werden.

Mathematisch lassen sich vier verschiedene statistische Methoden angeben, um Art und Grad dieser Selbstähnlichkeit von Messreihen (in unserem Fall *Internet traffic traces*) festzustellen [5].

Definition 3. *Sei* $\{X_t\}_{t=0}^{\infty}$ *ein zeitlich diskreter, kovarianzstationärer stochastischer Prozess mit Mittelwert* μ*, Varianz* σ^2 *und Autokorrelationsfunktion* ϱ_k*. Für jede* Aggregierungsstufe m *sei der Prozess* $X^m = \{X_k^m\}_{k=1}^{\infty}$ *definiert als*

$$X_k^m =_{\text{def}} \frac{X_{(k-1)m+1} + \cdots + X_{km}}{m}.$$

Die auf m aufaggregierten Messreihen X^m besitzen dann

1. *langsam fallende Varianzen*, d.h. ihre Varianzen nehmen langsamer ab als $1/m$,
2. *Langzeitabhängigkeiten*, d.h. die X^m besitzen eine über Skalen hinweg konstante Korrelationsstruktur, wenn $m \to \infty$,
3. eine *spektrale Dichte*, die der des Originals unterliegt, d.h. für einen selbstähnlichen Prozess gilt $f(\lambda) \sim \lambda^{-\gamma}$ für $0 < \gamma < 1$, wobei λ die Frequenz und $f(\lambda)$ die Dichte darstellen.

Mathematisch ausgedrückt bedeutet damit, dass X^m *exakt selbstähnlich* ist, falls gilt

- $\mathrm{Var} X^m = \sigma^2 m^{-\beta}$,
- $\varrho_k^m = \varrho_k$ und
- $X = m^{1-H} X^m$ für alle $m > 0$,

bzw. dass der Prozess X^m *asymptotisch selbstähnlich* ist, falls gilt

- $\mathrm{Var} X^m = \sigma^2 m^{-\beta}$ für $m \to \infty$,
- $\varrho_k^m = \varrho_k$ für $m \to \infty$,
- $X = m^{1-H} X^m$, für $m \to \infty$.

Das allgemein gebräuchlichste Maß zur Analyse der Selbstähnlichkeit ist der *Hurst-Parameter* H, der mit der *rescaled adjusted range statistic* berechnet wird:

4. *R/S-Statistik:* Es soll $\mathbf{E}R(n)/S(n) \sim cn^H$ für $0 < H < 1$ gelten, wobei $R(n) =_{\text{def}} \max\{0, W_1, W_2, \ldots, W_n\} - \min\{0, W_1, W_2, \ldots, W_n\}$ mit $W_j =_{\text{def}} (X_1 + X_2 + ... + X_k) - k\overline{X}(n)$ für den Mittelwert $\overline{X}(n)$ und die Varianz $S^2(n)$ einer n-elementigen Stichprobe.

Mit den mathematischen Methoden 1, 2 und 4 werden in [5] Ethernet-Verkehrsspuren zwischen 1989 und 1992 auf Selbstähnlichkeit hin analysiert und so der Grad von Selbstähnlichkeit geschätzt. Dieser gibt den Grad der Ausbruchslastigkeit oder Selbstähnlichkeit an (Selbstähnlichkeit bei $0,5 < H < 1$) Die Parameter β und γ lassen sich einfach in H ausdrücken: $\beta = 2 - 2H$ und $\gamma = 2H - 1$. Die Methoden, die auf Formulierung 1 und 4 basieren, erlauben nur eine grobe Schätzung; mit derjenigen, die auf Formulierung 3 basiert, können sogar Konfidenzintervalle für H angegeben werden. Diese wird in [2] dazu benutzt, um Selbstähnlichkeiten auf der TCP-Ebene zu analysieren. Dieses Analyseverfahren (MRA) wird später noch etwas genauer beschrieben.

Das Ergebnis: Ethernet-Verkehr ist selbstähnlich und der Grad der Selbstähnlichkeit steigt mit zunehmender Verkehrslast (d.h. Verkehrslast $\to \infty$ dann folgt $H \to 1$). Dies trifft nicht nur auf den Verkehr zwischen Hosts innerhalb des Ethernets zu sondern auch auf den Verkehr zwischen Ethernet und Internet, der die gleiche selbstähnliche Eigenschaft zeigt.

5 Variabilität des Netzwerkverhaltens

Es lässt sich zeigen, dass Poisson-Prozesse eine beschränkte Variabilität sowohl in der Zeit (Verkehrsprozesse sind entweder unabhängig oder verringern sich exponentiell schnell), als auch räumlich besitzen (die Verteilungen zur Simulation von verkehrsabhängigen Zählungen haben exponentiell fallende Enden). Aber für die Betrachtung von Netzwerken eignen sich nur Prozesse, deren Verteilungen hohe Variabilität haben. Diese wird mit statistischen Mitteln (durch die Verringerung der Autokorrelation) in unterschiedlichen Zeitskalen gemessen. Räumliche Variabilität können wir durch Verteilungen mit *heavy tails* unter unendlicher Varianz beschreiben, da diese die Ausbruchslastigkeit solcher Beobachtungen gut zum Ausdruck bringen. Solche Verteilungsfunktionen haben für große x-Werte ($x \to \infty$) folgendes Aussehen:

$$1 - F(x) \approx \kappa_1 x^{-\beta}$$

wobei die positive endliche Konstante κ_1 dabei nicht von x abhängt und der Tail-Index β in dem Intervall $(0, 2)$ liegt. Falls $\beta \leq 1$, hat die Verteilungsfunktion nicht nur unendliche Varianz, sondern auch unendlichen Erwartungswert. D.h. bei kleiner werdendem β befindet sich ein willkürlich großer Teil der Wahrscheinlichkeitsmasse in dem Ende der Verteilung. Betrachten wir also die Ausprägungen einer beliebigen Variable aus einer solchen Verteilung, kann diese sehr große und sehr kleine Werte mit hoher Wahrscheinlichkeit annehmen. Verteilungen mit *heavy tails* eignen sich somit zur Erzeugung ähnlicher Datenverteilungen wie in einem richtigen Netzwerk. Diese Eigenschaften besitzt z.B. die bekannte

Familie der Pareto-Verteilungen. Diese wurde ursprünglich eingeführt, um die Einkommensverteilung einer Personengruppe zu modellieren [5, 2].

Es stellt sich heraus, dass das zeitliche oder räumliche Abhängigkeitsverhalten ihrer Parameter, den betreffenden Datenverteilungsprozess fraktale Charakteristiken hervorbringen lässt. Ein Verkehrsprozess hat fraktale Charakteristiken, wenn es eine Beziehung zwischen bestimmten Quantitäten Q des zugrundeliegenden Prozesses und der Auflösung τ, der Form

$$Q(\tau) \approx \kappa_2 \tau^{f(D)}$$

gibt, wobei die positive endliche Konstante κ_2 dabei nicht von τ abhängt. Hier gibt τ die zeitliche oder räumliche Auflösung, unter der Q betrachtet wird, wieder. Die Approximation gibt an, wie sich Q in Abhänigkeit von der Auflösung τ verändert. F ist eine einfache (oft lineare) Funktion von der fraktalen Dimension D. Um fraktales Netzwerkverhalten aufzuzeigen muss die obere Relation über eine Spanne von unterschiedlichen Werten für τ Beständigkeit aufzeigen (mit Werten für D kleiner als die eingebettete fraktale Dimension).

In Abbildung 3 wird der gemessene Datenstrom (der auch in Abbildung 1 als Datenquelle dient) nun mit dem Modell eines kovarianzstationären Gaußschen Prozesses $X = \{X_k\}_{k=1}^{\infty}$ simuliert. Dieser Prozess muss fraktales Gaußsches Rauschen mit dem Hurst-Parameter $H \in [0.5, 1)$ besitzen, d.h. die Autokorrelationsfunktion zwischen X_n und $X_{n+k}, k \geq 0$ ist gegeben durch

$$\mathrm{Cor}(X_n, X_{n+k}) = \frac{1}{2}\left(|k+1|^{2H} - 2|k|^{2H} + |k-1|^{2H}\right).$$

Neben dem Anpassen des Modells durch den im Verkehr gemessenen Erwartungswert und die Varianz braucht es zusätzlich noch als weiteren Parameter den Hurst-Parameter H, welcher die Robustheit/Stärke der fraktalen Skalierung angibt. Dadurch, dass nur ein zusätzlicher Parameter gebraucht wird, ist dieses Model ausreichend einfach, um es auf zahlreiche Modellierungsszenarien zu übertragen, ohne viele Annahmen über die Parameter machen zu müssen.

6 MRA

In den 80 Jahren begann die Entwicklung der Theorie der Wavelets. Die Wavelet-Transformation besitzt nicht den Nachteil der Fourier-Transformation (das Fehlen einer Lokalisierungseigenschaft: Wenn sich das Signal an einer Stelle ändert, dann ändern sich alle Fourier-Koeffizienten), da zur Analyse des Signals zeitlich begrenzte „kleine Wellen“(Wavelets) verwendet werden. Der Hauptanwendungsbereich der Wavelet-Transformation ist die Signal- und Bildverarbeitung. Eine Basis-Funktion φ wird bei der Diskreten Wavelet-Transformation über den Bildbereich geschoben und skaliert. In Abhängigkeit davon ergibt sich jeweils eine neue Basis $\varphi_{x,t} = 2^{-t}\varphi(2^{-i}x - t)$, welche mit dem Originalsignal verglichen wird. Die passende Skalierungsfunktion ist

$$\psi(x) = \sum_{k=-1}^{N-2} (-1)^k c_{k+1} \phi(2x + k),$$

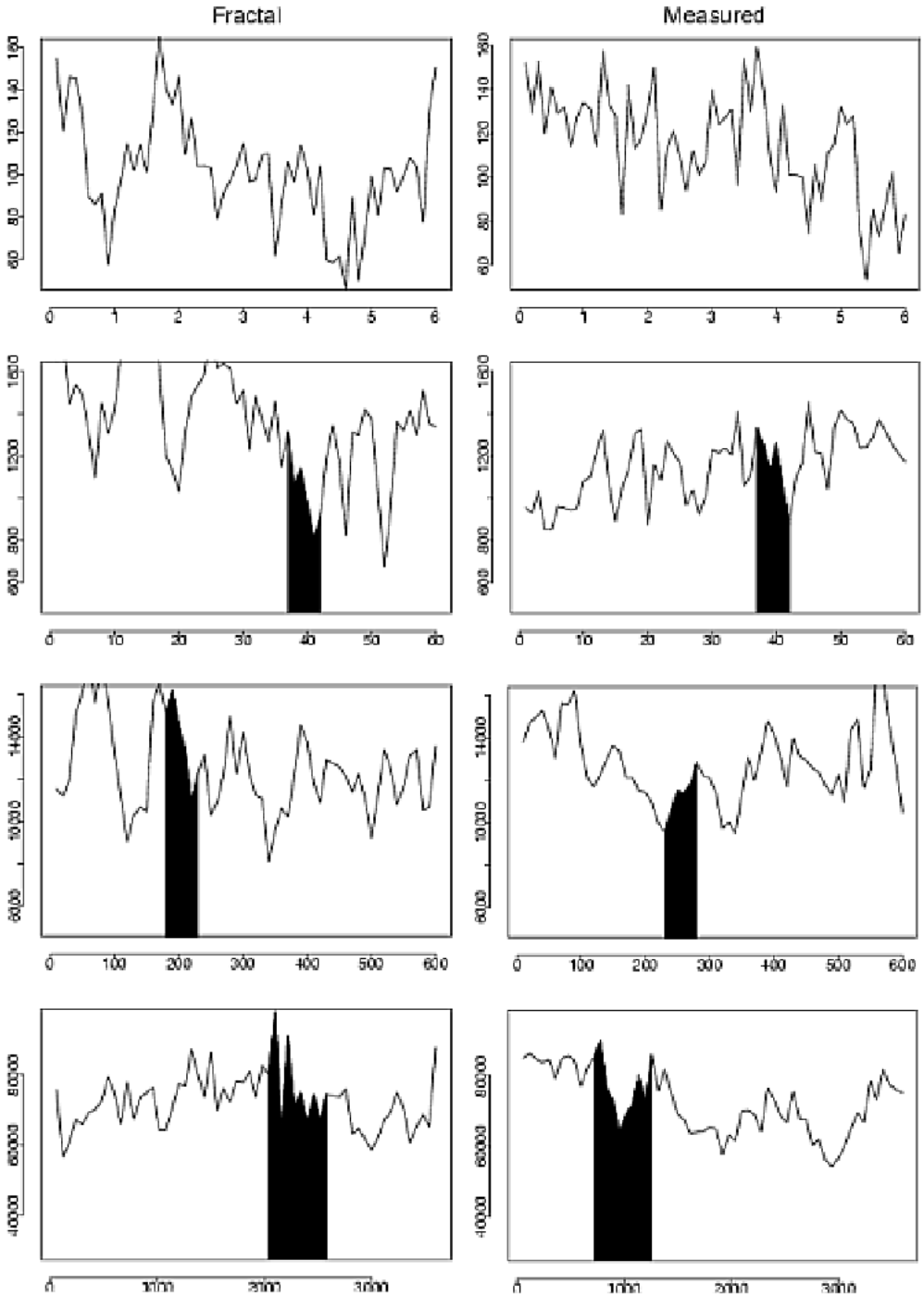


Abb. 2. Ein fraktales Modell im Vergleich mit einem aktuellen Internetverkehr.

wobei N die Anzahl der diskreten Stufen in der Basis-Funktion beschreibt und c_k einen Wavelet-Koeffizienten darstellt. Die Menge der Koeffizienten $\{c_0, \ldots, c_n\}$ wird oft auch als Filter bezeichnet.

Wichtig ist die Orthogonalität der Basis. Mit ihrer Hilfe können die Koeffizienten, die dann der Bedingung $\sum_{k=0}^{N-1} c_k = 2\delta_0$ mit der Distribution δ_0 genügen, ermittelt werden.

Die Fast-Wavelet-Transformation (FWT) war es, die die Filtertheorie aus der Signalverarbeitung mit der Wavelettheorie verband. Dazu war es sehr nützlich, daß die Bildauflösungen in mehreren Schritten um eine Konstante skaliert wurden (*downsampling*). Diesen Schritt führt die, bereits oben genannte, Skalierungsfunktion (realisiert als Tiefpassfilter) durch.

Die Details, die bei diesem Skalierungsschritt verloren gehen, werden mit Hilfe eines Hochpassfilters als Wavelet-Koeffizienten aufgefangen. Die Koeffizienten werden bei jedem Schritt gespeichert, die Tiefpassresultate werden jedoch für eine weitere rekursive Skalierung verwendet. So kann nach und nach das Originalbild in Details und einen Mittelwert (Tiefpassanteil) zerlegt werden.

MRA (*multi resolution analysis*) ist eine auf Wavelets basierende Skalierungsanalyse aus [2]. Diese liefert für kleinere Zeitskalen bessere Ergebnisse als die oben genannten Methoden. Das Signal X wird nacheinander approximiert durch $P_j X$. Jede Approximation $P_j X$ ist eine Beschreibung von X mit Auflösung 2^j . Die Information über X, die bei einem Schritt von $P_j X$ nach $P_{j+1} X$ verloren geht, kann mit den Wavelets der Form $\eta_{j,k}(t) = 2^{-j/2}\eta(2^{-j}t - k)$ ausgedrückt werden:

$$P_j X = P_{j+1} X + \sum_k \langle X, \eta_{j,k} \rangle \, \eta_{j,k}.$$

Ein Beispiel dafür ist das Haar-Wavelet η:

$$\eta(t) =_{\text{def}} \begin{cases} \frac{1}{\sqrt{2}}, & \text{falls } t \in [0, 1/2), \\ -\frac{1}{\sqrt{2}}, & \text{falls } t \in (1/2, 1], \\ 0 & \text{sonst.} \end{cases}$$

Die Terme $\langle X, \psi_{j,k} \rangle$ nennen sich Wavelet-Koeffizienten $d_{j,k}$ von X. Um sie konkret zu berechnen, werden allerdings keine Wavelets benötigt. Es gilt $d_{j,k} = (X_{2k} - X_{2k+1})/2$. Der Koeffizient $|d_{j,k}|^2$ misst die Energie eines Signals zum Zeitpunkt $t_0 = 2^j k$ und um die Frequenz $2^{-j}\lambda_0$, wobei λ_0 eine (vom Wavelet ψ abhängende) Referenzfrequenz ist. Die Summe $1/N_j \sum_k |d_{j,k}|^2$ ist ein Maß für die gesamte Energie E_j innerhalb eines Frequenzbandes der Breite 2^{-j} um die Frequenz $2^{-j}\lambda_0$. Es lässt sich zeigen, dass $\log_2 E_j = (2H-1)j + C$ ist. Tragen wir $\log_2 E_j$ nach j ab, können wir H auf Grund der Steigung schätzen. In [2] wurde auf diese Art WAN-Verkehr zwischen 1993 und 1997 analysiert, was ähnliche Resultate zeigte wie in [5]. Zusätzlich konnte bei kleinem j (kleine Zeitintervalle, hohe zeitliche Auflösung) ein anderes Verhalten als bei großem j beobachtet werden.

7 Modelle

Ein aktuelles Forschungsthema im Zusammenhang mit diesem Thema ist die Suche nach mathematischen Modellen, die *traffic traces* erzeugen, deren Skalierungseigenschaften mit tatsächlich gemessenem Internetverkehr übereinstimmen. Ein Hauptkriterium für ein „gutes“Modell ist die Frage, ob seine Struktur eine sinnvolle Entsprechung zu der physikalischen Struktur des Internets und seinen Protokollen hat. Gesucht sind also *structural models* nicht nur *operational models*, die zwar auch die gewünschte Paketsdichteverteilung liefern könnten, aber einer Black-Box gleichkommen würden. Es ließe sich daher schlechter bei ihnen abschätzen, ob bzw. wie gut sie auch für zukünftigen Verkehr geeignet sein werden, wenn sich sowohl die Netzwerkstruktur, als auch die Applikationen, die den Verkehr produzieren verändern.

Eine Modellierung mit den Bezeichnungen *immigration death process* oder $M/G/\infty$ *queuing model* [1] hat sich bisher vor allem für LAN-Verkehr bewährt. Ein selbstähnlicher Prozess wird darin als Aggregation von einzelnen Prozessen (*sessions*) modelliert. Bei näherer Betrachtung zeigen sich jedoch Mängel des Modells, die daher rühren, dass in WAN mehrere Protokollschichten existieren, das $M/G/\infty$-Queuing-Model aber nur zwei modelliert. In [2] wird daher der Transport-Layer genauer untersucht, d.h. die Anzahl der TCP-Verbindungen pro Zeitintervall.

Ein anderer Ansatz sind Energy-Scale-Plots wie sie in Kapitel 6 eingeführt wurden. Mit ihnen konnte in [2] gezeigt werden, dass nur für reinen TELNET-Verkehr die Ankunftszeiten von TCP-Verbindungen Poisson-verteilt sind (jede TELNET-Session entspricht einer TCP-Verbindung [6]). Mit zunehmendem FTP- und Web-Verkehr zeigen Internet-Verkehrsspuren aber eine TCP-Verbindungsverteilung, die *heavy tails* hat.

Die gemessene Selbstähnlichkeit auf dem Packet-Layer lässt sich also nicht mit dem $M/G/\infty$-Queuing-Model als eine Aggregation von TCP-Sitzungen erklären. Die Heavy-Tails-Eigenschaft der TCP-Ankünfte rührt von der komplexen *within-session*-Struktur von Web-Sitzungen her, also der Verteilung der TCP-Sitzungen innerhalb einer Sitzung.

Die Frage nach passenden Beschreibungsmodellen in diesem Gebiet ist noch lange nicht hinreichend geklärt. Es gibt auch neue Modelle die multifraktale Ansätze behandeln. Weitere Details finden sich in [9].

Literatur

1. D. R. Cox. Longrange dependence: A review. In: H. A. David, H. T. David (Hrsg.). *Statistics: An Appraisal*, S. 5574. Iowa State University Press, 1984.
2. A. Feldmann, A. C. Gilbert, W. Willinger, T. G. Kurtz. The changing nature of network traffic: Scaling phenomena. *ACM Computer Communication Review*, 28(2):5-29, 1998.
3. L. Fahrmeir, R. Künstler, I. Pigeot, G. Tutz. *Statistik - Der Weg zur Datenanalyse.* Springer-Verlag, Berlin, 1998.

4. V. Frost, B. Melamed. Traffic modelling for telecommunication networks. *IEEE Communications Magazine*, 32(3):70-81, 1994.
5. W. E. Leland, M. S. Taqqu, W. Willinger, D .V. Wilson. On the self-similar nature of Ethernet traffic. *IEEE/ACM Transactions on Networking* 2(1):1-15, 1994.
6. V. Paxson. Growth trends in wide-area TCP connections. *IEEE Network Magazine*, 8(4):8-17, 1994.
7. V. Paxson, S. Floyd. Wide area traffic: The failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226-244, 1995.
8. W. Willinger, V. Paxson. Where the mathematics meets the Internet. *Notices of the AMS*, 45(8):961-970, 1998.
9. W. Willinger, M. S. Taqqu, A. Erramilli. A bibliographical guide to self-similar traffic and performance modelling for modern high-speed networks. In: F. P. Kelly, S. Zachary, I. Zieldins (Hrsg.). *Stochastic Networks: Theory and Applications*, Band 4 der *Royal Statistical Lecture Note Series*, S. 339-366. Clarendon Press, Oxford, 1996.

Online-Algorithmen und verzögerte TCP-Quittungen

Michael Kuhn

Technische Universität München
kuhnm@in.tum.de

Zusammenfassung. Diese Arbeit gibt einen Einblick darin, wie sich sog. Online-Algorithmen dazu verwenden lassen, den Zeitpunkt des Versendens von TCP-Bestätigungen im laufenden Betrieb zu bestimmen. Das TCP-Protokoll erfordert die Bestätigung jedes versendeten Daten-Segmentes, welche allerdings nicht sofort erfolgen muss. Es werden unterschiedliche Kostenmodelle sowie unterschiedliche algorithmische Ansätze betrachtet und die Qualität der gelieferten Lösungen mittels kompetitiver Analyse miteinander verglichen.

1 Das TCP-Protokoll

1.1 Überblick

Das TCP-Protokoll, erstmals definiert in [4, 6], ist ein verbindungsorientiertes Protokoll der sogenannten Transport-Schicht des *Internet-Protocol-Stack* (vgl. [8, 7]). Es stellt den darüberliegenden Anwendungen einen zuverlässigen Bytestrom über einen meist unzuverlässigen, heterogenen Netzverbund (z.B. das Internet) zur Verfügung. Da die darunterliegende Verbindungsschicht (IP) keinerlei Garantie gibt, dass zu übertragende Datenpakete auch wirklich beim Empfänger ankommen, muss sich TCP mittels eines *timeout* darum kümmern, dass die Daten eventuell erneut übertragen werden, bis der Empänger das Eintreffen der Daten bestätigt hat. Da es sich um ein verbindungsorientiertes Protokoll handelt, muss vor der Übertragung von Daten zwischen zwei Endsystemen als erstes explizit eine Verbindung aufgebaut werden. Dies geschieht mittels des sogenannten *Three-Way-Handshake*. Nach Übertragung der Daten wird die Verbindung wieder abgebaut.

1.2 Der Segment-Header

TCP unterteilt die von der Anwedung gelieferten Daten in Segmente, die verschickt werden. Jedes byte der Nutzdaten ist mit einer 32-bit Folgenummer durchnummeriert. Segmente bestehen aus einem mindestens 20-byte großen Header, gefolgt von den Nutzdaten. Die Größe der Segmente wird unter anderem von der *maximalen Transfereinheit (MTU)* des Netzes beschränkt. Die Felder im TCP-Segment-Header haben folgende Zwecke:

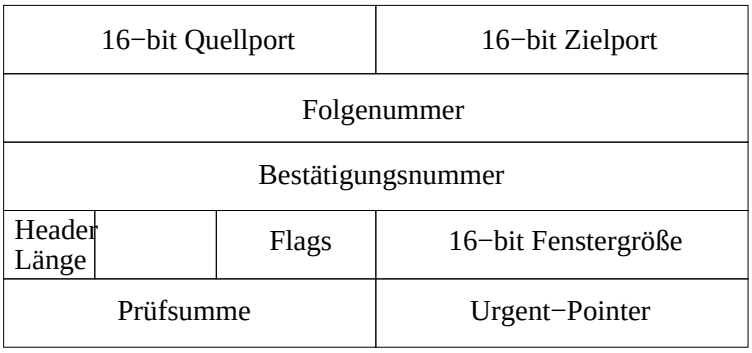


Abb. 1. TCP-Segment-Header.

- Die 16-bit-Felder „Quell- und Zielport“kennzeichnen die Endpunkte der Verbindung und bilden zusammen mit den Sende- und Empfänger-IP-Adressen sogenannte *sockets*, die eine Verbindung eindeutig identifizieren,
- „Folgenummer“ist die Nummer des ersten bytes der Nutzdaten,
- „Bestätigungsnummer“ist die Nummer des nächsten erwarteten Bytes vom Kommunikationspartner,
- „Header-Länge“gibt die Länge des TCP-Headers an, da zusätzlich zu den 20 byte noch optionale Teile im Header folgen können,
- TCP verwendet Schiebefenstertechnik für die Verwaltung des Empfangspuffers; die „Fenstergröße“gibt an, wieviel Bytes im Empfangsfenster noch frei sind,
- unter den „Flags“befindet sich unter anderem das *ack-flag*, welches kennzeichnet, dass dieses Segment auch eine gültige Bestätigung für die Datenbytes bis „Folgenummer-1“ist,
- „Prüfsumme“über den TCP-Header,
- „Urgent-Pointer“zur Übertragung dringender Daten.

1.3 Bestätigungen

Jedes versendete Datensegment muss vom Empfänger bestätigt werden, indem der Empfänger ein Segment mit gesetztem *ack-flag* zurück an den Sender schickt. Die Bestätigungsnummer zeigt dabei an, bis zum wievielten Byte der Empfänger die Nachricht erhalten hat. Treffen beim Empfänger mehrere Segmente vom gleichen Sender ein, so kann er deren Empfang zusammen mit nur einer Bestätigung bestätigen. Über die Bestätigungsnummer gibt er dabei an, bis zu welcher Byte-Nummer er durchgehend alle bytes vom Sender erhalten hat.

Bestätigungen können entweder nur aus einem TCP-Header bestehen, oder bei gleichzeitig ausgehenden Nutzdaten in deren TCP-Header mitreisen (*piggyback*, vgl. [7]), da jeder TCP-Header das Feld für die Bestätigungsnummer beinhaltet (dieses aber nur ausgewertet wird, falls das *ack-flag* gesetzt ist).

Um gezielt nicht jedes Segment für sich bestätigen zu müssen, kann der Empfänger auch absichtlich die Bestätigung für ein gerade eingetroffenes Segment um eine Zeit t verzögern. Nach Ablauf von t sendet der Empfänger dann eine Bestätigung und kann alle bis zu diesem Zeitpunkt noch zusätzlich eingetroffenen Segmente vom gleichen Sender mitbestätigen.

Gängige TCP-Implementierungen machen davon Gebrauch. So verwenden zum Beispiel BSD-Derivate einen 200ms *heartbeat-timer*. Das heißt, dass alle 200ms ein *timer* abläuft und alle bis dahin noch nicht bestätigten Segmente auf einmal bestätigt werden. Andere Implementierungen starten bei einem eintreffenden Segment einen 50ms *timer*, nach dessen Ablauf alle bis dahin noch eingetroffenen Segmente bestätigt werden.

Diese Ansätze verwenden jedoch alle statische *timer*. Ziel soll es jetzt sein, diese dynamisch zu bestimmen, d.h. je nachdem wieviel Segmente eintreffen und wann, den Zeitpunkt für das Versenden der Bestätigung entsprechend zu bestimmen.

Allerdings muss darauf geachtet werden, dass sich verzögerte Bestätigungen direkt auf das TCP-Verbindungsmanagement auswirken können. Erhält der Sender eines Segmentes nämlich nicht innerhalb des *retransmission-timeout* eine Bestätigung, so versendet er das Segment erneut. Dieses *timeout*-Intervall wird jedoch ständig berichtigt, indem gemessen wird, wie lange es dauert, bis ein gesendetes Segment bestätigt worden ist. Werden nun die Bestätigungen zu sehr verzögert, so wächst dieser *timeout*-Wert entsprechend und Übertragungsfehler werden unnötig spät erkannt. Da sich die TCP-Überlastungsüberwachung auch auf diesen *timer* abstützt kann, sie ebenso entsprechend negativ davon beeinflusst werden [8, 7, 3].

2 Online-Algorithmen zur dynamischen Berechnung der Verzögerung

Online-Algorithmen versuchen Probleme zu lösen, bei denen sie Eingaben erhalten, die ihnen vorher nicht bekannt sind. Sie müssen fortlaufend die Eingaben bearbeiten, ohne zu wissen wie die nächste Eingabe aussieht und entsprechend eine möglichst gute Lösung für das gegebene Problem finden (vgl. [5]).
Eines der bekanntesten Online-Probleme ist *Ski Rental*: Angenommen, man will Skifahren lernen. Nach jedem Tag trifft man erneut und nicht vorhersagbar die Entscheidung, ob man am nächsten Tag weiterhin skifahren möchte, oder es bleiben läßt. Jeden Morgen muss man nun neu entscheiden, ob man sich eine eigene Ausrüstung für x Euro kauft, oder eine für y Euro mietet ($x > y$). Wüsste man nun von vorne herein, wieviel Tage man tatsächlich fährt, wäre es kein Problem auszurechnen, ob es sich lohnt eine eigene Ausrüstung zu kaufen oder jeden Tag eine zu mieten. Da man sich aber jeden Tag neu entscheidet ob man überhaupt noch weiter skifahren will, ist dies nicht möglich.
Ganz ähnlich verhält es sich mit den verzögerten TCP-Bestätigungen. Bei jedem eintreffenden Segment, muss der Algorithmus neu entscheiden, ob er die Bestätigung verzögern soll, um eventuell noch eintreffende Segmente gleich mit-

bestätigen zu können, oder lieber gleich eine Bestätigung sendet. Würde man die Ankunftszeiten der Segmente vorher schon kennen, wäre es kein Problem eine optimale Lösung zu finden (vgl. [3] S. 253).

2.1 Bewertung von Online-Algorithmen

Um die Güte der Lösung eines Online-Algorithmus A zu bestimmen, vergleicht man dessen Lösung mit der eines optimalen Offline-Algorithmus Opt, dem die Eingabe vorher bekannt ist. Das sich ergebende Verhältnis zwischen der optimalen Lösung C_{Opt} und der des Online-Algorithmus C_A nennt man *kompetitives Verhältnis* (vgl. [5, 1]). Gilt

$$C_A \leq \alpha C_{\mathrm{Opt}} + c,$$

so sagen wir: A ist *α-kompetitiv*. Die Eingabe für den Algorithmus A liefert dabei ein Gegenspieler, der die Funktionsweise von A kennt, und die für diesen Algorithmus schlechtest mögliche Eingabe bestimmt. Daraus ergibt sich dann, wie schlecht das Ergebnis von A höchstens sein kann.

Für randomisierte Online-Algorithmen verwendet man dieses Verfahren analog. Allerdings mit dem Unterschied, dass man den Erwartungswert der vom Online-Algorithmus gelieferten Lösungen verwendet:

$$\mathbf{E}C_A \leq \alpha C_{\mathrm{Opt}} + c.$$

Die Eingabe für den randomisierten Algorithmus A liefert wiederum ein Gegenspieler, der genau weiß wie A funktioniert. Was ihm aber unbekannt ist, sind die zufälligen Entscheidungen, die A trifft. Der Gegenspieler hat jedoch die Möglichkeit, den Algorithmus A mit allen möglichen Kombinationen von Zufallsentscheidungen auf allen möglichen Eingaben intern zu simulieren, und wählt dann diejenige Eingabesequenz aus, für die sich erwartungsgemäß A am schlechtesten verhalten wird. Eine andere Möglichkeit für einen Gegenspieler bei randomisierten Algorithmen wäre einer, der auch über die Zufallsentscheidungen von A Bescheid weiß.

Kompetitive Analyse ist jedoch nicht unumstritten, da es eine klassische *worst-case*-Analysemethode ist und somit bekanntermaßen nur bedingt geeignet ist, Aussagen über das praktische Verhalten der Algorithmen zu treffen. Weiterhin geht man von dem sehr pessimistischen Ansatz aus, dass Online-Algorithmen überhaupt kein Wissen bezüglich der Zukunft haben können. So könnte man z.B. bei *Ski Rental* davon ausgehen, dass jemand der sich schon i-mal für Skifahren entschieden hat, mit großer Wahrscheinlichkeit am nächsten Tag wieder Skifahren gehen wird. Diese Möglichkeit wird jedoch bei der kompetitiven Analyse außer Acht gelassen. Trotzdem ist sie in der Informatik weit verbreitet und soll auch im Folgenden Anwendung finden.

2.2 Formalisierung

Gegeben ist eine Sequenz $\mathcal{A} = (a_1, \ldots, a_n)$ von Segmentankunftszeiten, die von den Algorithmen in k Partitionen $\sigma_1, \ldots, \sigma_k$ aufgeteilt werden soll, wobei am

Ende jeder solchen Partition alle in dieser Partition angekommenen Segmente mit einer Bestätigung zum Zeitpunkt t_i bestätigt werden ($t_k \geq a_n$).

Ziel ist nun, eine möglichst gute Partitionierung zu erreichen, die die Anzahl der gesendeten Bestätigungen minimiert, aber gleichzeitig die dadurch anfallende Latenz nicht zu groß werden lässt. Die Zielfunktion f_{lat} wägt dabei diese beiden Einflußgrößen gewichtet mit dem Faktor $\eta \in [0;1]$ gegeneinander ab

$$f_{\text{lat}} =_{\text{def}} \eta k + (1-\eta) \sum_{i=1}^{k} \text{lat}(\sigma_i, t_i).$$

Die Funktion f_{lat} gilt es nun zu minimieren, wobei die die Latenz berechnende Funktion $\text{lat}(\sigma_i, t_i)$ umkehrbar sein muss, und mit steigender Wartezeit und Anzahl der ankommenden Segmente wächst.

Im Weiteren werden für die Bewertung der Latenz zwei konkrete Modelle eingeführt, die zu den speziellen Zielfunktionen f_{sum} und f_{max} führen. Die zwei zu betrachtenden Online-Algorithmen $\text{GREEDY}_{\text{total}}$ und $\text{GREEDY}_{\text{new}}$ versuchen, diese zwei speziellen Zielfunktionen mit unterschiedlichen Ansätzen zu minimieren, wobei wir die Güte der jeweils berechneten Lösungen untersuchen werden.

2.3 Summarische Latenz

In der Zeilfunktion f_{sum} wird die Latenz jedes einzelnen Segmentes pro Partition aufsummiert und auf diese Weise die Gesamtlatenz berechnet, die dem System durch die verzögerten Bestätigungen zugeführt wird. Dafür setzen wir in f_{lat}

$$\text{lat}(\sigma_i, t_i) = \sum_{a_j \in \sigma_i} (t_i - a_j)$$

und erhalten daraus f_{sum}.

Um die Ergebnisse der Online-Algorithmen mittels kompetitiver Analyse bewerten zu können, wird zuerst ein Offline-Algorithmus benötigt, der die optimale Lösung bestimmt, sprich f_{sum} minimiert. Die Funktion f_{sum} kann so umgeformt werden, dass gilt

$$f_{\text{sum}} = \eta k + (1-\eta) \left(\sum_{i=1}^{k} |\sigma_i| t_i - \sum_{i=1}^{n} a_i \right).$$

Man sieht, dass die zweite Summe für die Berechnung des Algorithmus keine Rolle spielt und es somit reicht, wenn der Algorithmus nur den restlichen Teil der Zielfunktion minimiert. Nach Ablauf des Algorithmus wird vom Ergebnis einfach noch $\sum_{i=1}^{n} a_i$ abgezogen. Der Offline-Algorithmus berechnet die optimale Partitionierung der Sequenz $\mathcal{A}$, welche minimale Kosten verursacht, mittels dynamischer Programmierung.

Dabei wird das Feld $M[i,j]$ verwendet, welches die bisher berechnete minimale Kostenlösung für die Sequenz $(a_1, \ldots, a_i)$ enthält, wobei die vorletzte Bestätigung nach Segment a_{i-j} erfolgt und die letzte nach a_i. $M_{\text{min}}[i]$ speichert

```
Offline
1 Initialisiere M_min ← 0
2 Initialisiere M[1,1] ← 1 · η + (1 − η) · a_1
3 Initialisiere M_min[1] ← M[1,1]
4 Initialisiere M_pt[1] ← 1
5 For i ∈ [2, n]
6       M_min[i] ← ∞
7       For j ∈ [1, i]
8           M[i, j] ← 1 · η + (1 − η) · j · a_i + M_min[i − j]
9           If M[i, j] < M_min[i] then
10              M_min[i] ← M[i, j]
11              M_pt ← j
```

Abb. 2. Offline-Algortihmus für f_{sum}.

genau die minimalen Kosten bei i Segmenten und $M_{\text{pt}}[i]$ genau den Wert j (also die Stelle der vorletzten Bestätigung), so dass gilt $M[i,j] = M_{\min}[i]$. Daraus ergibt sich der Offline-Algorithmus aus Abbildung 2. Er hat eine Laufzeit von $O(n^2)$, da er über zwei verschachtelte Schleifen iteriert. Dabei baut er die endgültige Lösung sukzessive aus schon berechneten Teillösungen auf (Zeile 8).

Der Online-Algorithmus $\textsc{Greedy}_{\textbf{total}}$**.** Die Idee des Algorithmus ist es, die Kosten für sofortiges Bestätigen eines Segmentes und das Verzögern um eine Zeit $t_{\text{total,sum}}$ genau auszubalancieren und auf diese Weise $\mathcal{A}$ *greedy* zu partitionieren.

Sei a_j der Zeitpunkt des zuletzt eingetroffene Segmentes und σ die Menge der unbestätigten Segmente bis einschließlich a_j. Dann wartet der Algorithmus mit dem Versenden einer Bestätigung bis zum Zeitpunkt $a_j + t_{\text{total,sum}}$. Daraus ergeben sich extra Latenzkosten von $(1-\eta)|\sigma|t_{\text{total,sum}}$, da jedes Segment in σ zusätzlich um $t_{\text{total,sum}}$ verzögert wird. Außerdem sind evtl. schon Latenzkosten von $(1-\eta)\sum_{a_i \in \sigma}(a_j - a_i)$ vorhanden. Würde allerdings sofort eine Bestätigung gesendet werden, so ergeben sich Kosten von η für die Bestätigung, sowie ebenfalls $(1-\eta)\sum_{a_i \in \sigma}(a_j - a_i)$ Kosten für bis zu diesem Zeitpunkt schon vorhandene Latenz. Durch Gleichsetzen und Umformen erhält man

$$t_{\text{total,sum}} = \frac{\eta}{|\sigma|(1-\eta)}.$$

D.h., dass bei jedem neu eintreffenden Segment, $t_{\text{total,sum}}$ neu berechnet und ein Zeitzähler gestartet wird, der zum Zeitpunkt $a_j + t_{\text{total,sum}}$ abläuft und dann einen Alarm auslöst. Wird dieser ausgelöst bevor ein neues Segment eintrifft, werden alle bis dahin unbestätigten Segmente mit einer Bestätigung bestätigt. Andernfalls wird der Zeitzähler mit einer neu berechneten Zeitspanne entsprechend wieder gestartet. Falls man einen Lookahead $L = 1$ hätte, der Algorithmus also die Ankunftszeit des nächsten Segmentes kennen würde, so könnte er sofort eine Bestätigung senden, wenn $(a_{j+1} - a_j) > t_{\text{total,sum}}$.

Es gibt allerdings Fälle, in denen $\textsc{Greedy}_{\text{total}}$ ein denkbar schlechte Lösung liefert:

Lemma 1. [3] *Es existiert eine unendliche Familie von Sequenzen von Segmentankunftszeiten, auf der* $C_{\text{GREEDY}_{\text{total}}} = \Omega(\sqrt{n})C_{\text{Opt}}$ *für die Zielfunktion* f_{sum} *gilt.*

Der Gegenspieler lässt immer kurz vor Auslösen des Alarms ein neues Segment eintreffen, woraufhin der Zeitzähler mit einer neu berechneten Zeitspanne wieder gestartet wird und somit die Versendung der Bestätigung weiter verzögert wird. Es lässt sich eine solche Sequenz an Ankunftszeiten finden, die das Auslösen des Alarms immer um ein Stückchen weiter nach hinten verschiebt, was letztendlich dazu führt, dass erst nach dem allerletzten Segment eine Bestätigung erfolgt.

Der Online-Algorithmus $\text{GREEDY}_{\text{new}}$**.** Um diese ungünstige Entwicklung zu vermeiden, verhält sich der Algorithmus $\text{GREEDY}_{\text{new}}$ bei der Berechnung der Zeitspanne bis zum Auslösen des Alarms etwas geschickter. Genau wie $\text{GREEDY}_{\text{total}}$ wägt er die Kosten für eine verzögerte Bestätigung und das sofortige Versenden einer solchen gegeneinander ab. Jedoch werden bei ihm die schon angelaufenen Latenzkosten bei der Berechnung der Kosten für ein sofortiges Versenden der Bestätigung außer Acht gelassen. Daraus ergibt sich für die Zeitspanne $t_{\text{new,sum}}$ die Gleichung

$$t_{\text{new,sum}} = \frac{\eta}{|\sigma|(1-\eta)} - \frac{\sum_{a_i \in \sigma}(a_j - a_i)}{|\sigma|}.$$

Hier wird also im Gegensatz zu $t_{\text{total,sum}}$ die durchschnittliche Latenz der Segmente in σ noch abgezogen. Man kann nun zeigen [3], dass der Zeitpunkt der Auslösung des Alarms aus diesem Grund entweder fest bleibt, oder sich zeitlich nach vorne bewegt; und weiter sogar, dass die Zeitspanne bis zum Auslösen des Alarm so berechnet wird, dass die endgültige Gesamtlatenz einer Sequenz σ genau $\eta/(1-\eta)$ beträgt. Dieses Ergebnis gilt sogar unabhängig davon, welches Modell man für die Berechnung der Latenz verwendet.

Lemma 2. [3] *Für die generelle Zielfunktion* f_{lat} *gilt* $C_{\text{GREEDY}_{\text{new}}} \leq 2C_{\text{Opt}}$.

Beweis. Angenommen $\text{GREEDY}_{\text{new}}$ sendet k Bestätigungen, dann ergeben sich $k \cdot (1-\eta) \cdot \eta/(1-\eta) = k\eta$ Kosten für die Latenz und $k\eta$ Kosten für die k Bestätigungen. Dann ist $C_{\text{GREEDY}_{\text{new}}} = 2k\eta$. Weiter lässt sich zeigen, dass für die Kosten eines optimalen Algorithmus in diesem Zusammenhang gilt $C_{\text{Opt}} \geq k\eta$. Daraus folgt die Behauptung. □

Für die Zielfunktion f_{sum} ist $\text{GREEDY}_{\text{new}}$ sogar optimal, da sich eine untere Schranke für deterministische Online-Algortihmen im Zusammenhang mit f_{sum} finden lässt.

Theorem 3. [3] *Für jeden deterministischen Online-Algorithmus* A *mit Zielfunktion* f_{sum} *und konstantem Lookahead* L *gibt es eine unendliche Familie von Sequenzen von Segmentankunftszeiten mit* $C_A \geq 2C_{\text{Opt}} - c$ *mit beliebig kleinem* c.

2.4 Maximierende Latenz

Bei $f_{\max}$ wird die Gesamtlatenz einer Teilsequenz σ_i durch das Maximum der Segmentlatenzen in dieser Sequenz bestimmmt. Die Latenz insgesamt ergibt sich wiederum über die Summe der Latenzen der einzelnen Sequenzen. Wir setzen also in f_{lat} die Latenzfunktion als

$$\text{lat}(\sigma_i, t_i) = \max(t_i - a_j)$$

und erhalten dadurch $f_{\max}$. Natürlich gilt immer $\max_{a_j \in \sigma_i}(t_i - a_j) = t_i - a_{\text{first},i}$, wobei $a_{\text{first},i}$ die Ankunft des ersten Segmentes in σ_i bezeichnet.

Wie schon für f_{sum} werden auch hier die Algorithmen $\text{GREEDY}_{\text{total}}$ und $\text{GREEDY}_{\text{new}}$ entsprechend betrachtet. Um die Qualität Ihrer Lösungen beurteilen zu können, verwenden wir wieder den schon bekannten Offline-Algorithmus aus Abbildung 2) mit folgenden Anpassungen in Zeile 2 und 8:

2 `Initialisiere` $M[1,1] \leftarrow 1 \cdot \eta$
8 $M[i,j] \leftarrow 1 \cdot \eta + (1-\eta) \cdot (a_i - a_{i-j+1}) \cdot a_i + M_{\min}[i-j]$

Der Online-Algorithmus $\text{GREEDY}_{\text{total}}$. Der Algorithmus verhält sich bei der Optimierung der Zielfunktion $f_{\max}$ genauso wie vorher bei f_{sum}. D.h. er versucht wieder, die Kosten für das Verzögern einer Bestätigung um eine Zeit $t_{\text{total,max}}$ und die Kosten für ein sofortiges Versenden selbiger auszubalancieren.

Durch das Verzögern ergeben sich jetzt zusätzliche Latenzkosten von $(1-\eta) \cdot t_{\text{total,max}}$, wobei bis zu dem Zeitpunkt a_j des gerade eingetroffenen Segmentes schon Kosten von $(1-\eta)(a_j - a_{\text{first},i})$ entstanden sein können. Sofortiges Bestätigen würde Kosten von η verursachen. Dazu kommen ebenfalls noch die Kosten für bis zu diesem Zeitpunkt schon vorhandene Latenz von $(1-\eta)(a_j - a_{\text{first},i})$. Es ergibt sich

$$t_{\text{total,max}} = \frac{\eta}{(1-\eta)}.$$

Bei jedem neu eintreffenden Segment zum Zeitpunkt a_j wird also $t_{\text{total,max}}$ neu berechnet und ein Zeitzähler gestartet, der zum Zeitpunkt $a_j + t_{\text{total,max}}$ abläuft und einen Alarm auslöst. Trifft bis zum Auslösen des Alarms kein neues Segment mehr ein, so werden alle Segmente in σ_i zusammen mittels einer Bestätigung bestätigt. Im Gegensatz zum schlechten Abschneiden von $\text{GREEDY}_{\text{total}}$ im Zusammenhang mit der Zielfunktion f_{sum} stellt man hier jedoch fest, dass sich $\text{GREEDY}_{\text{total}}$ bzgl. der Funktion $f_{\max}$ wesentlich besser verhält.

Theorem 4. [3] $\text{GREEDY}_{\text{total}}$ *ist 2-kompetitiv für die Zielfunktion* $f_{\max}$ *mit Lookahead* $L = 0$.

Beweis. Um dies zu zeigen, muss als erstes gezeigt werden, dass eine optimale Lösung S für $f_{\max}$ dann und nur dann eine Bestätigung zum Zeitpunkt a_j sendet, wenn gilt $(a_{j+1} - a_j)(1-\eta) > \eta$ gilt. $\text{GREEDY}_{\text{total}}$ verhält sich nun genauso wie eine optimale Lösung, solange $a_{j+1} - a_j \leq \eta/(1-\eta)$ ist. Wenn nun $a_{j+1} - a_j > \eta/(1-\eta)$ gilt, dann würde S bei a_j eine Bestätigung senden. $\text{GREEDY}_{\text{total}}$

jedoch muss erst noch die Zeit $\eta/(1-\eta)$ auf das Auslösen seines Alarms warten, und sendet dann ebenfalls eine Bestätigung. Dadurch ergeben sich im Vergleich zu S zusätzliche Latenzkosten von $(1-\eta) \cdot \eta/(1-\eta) = \eta$ für $\text{GREEDY}_{\text{total}}$. Rechnet man nun Bestätigungskosten und Latenzkosten zusammen, so ergibt sich $C_{\text{GREEDY}_{\text{total}}} \leq 2C_{\text{Opt}}$. □

Corollary 5. $\text{GREEDY}_{\text{total}}$ *ist 1-kompetitiv mit Lookahead* $L = 1$.

Der Online-Algorithmus $\text{GREEDY}_{\text{new}}$**.** Analog zu $\text{GREEDY}_{\text{new}}$ bzgl. der Zielfunktion f_{sum} berechnet $\text{GREEDY}_{\text{new}}$ für $f_{\max}$ die Zeitspanne bis zum Auslösen des Alarms, indem wiederum die Kosten für sofortiges Bestätigen und die Kosten für verzögertes Bestätigen gegeneinander abgewogen werden. Wie vorher auch, werden jedoch beim sofortigen Bestätigen schon angefallene Latenzkosten nicht berücksichtigt. Daraus ergibt sich

$$t_{\text{new,max}} = \frac{\eta}{(1-\eta)} - (a_j - a_{\text{first},i}).$$

Wegen Lemma 2 gilt auch für $f_{\max}$ die 2-Kompetitivität von $\text{GREEDY}_{\text{new}}$. Außerdem kann für $\text{GREEDY}_{\text{new}}$ und die Zielfunktion $f_{\max}$ immer eine unendliche Familien von Sequenzen von Segmentankunftszeiten gefunden werden, so dass $C_{\text{GREEDY}_{\text{new}}} \geq 2C_{\text{Opt}} - c$ gilt [3]. Weiterhin gilt ganz allgemein und unabhängig vom gewählten Kostenmodell für die Latenz eine untere Schranke.

Theorem 6. [3] *Für jeden deterministischen Online-Algorithmus* A *mit Lookahead* $L = 0$ *und der allgemeinen Zielfunktion* f_{lat} *gibt es eine Familien von Sequenzen von Segmentankunftszeiten, auf der* $C_A \geq 2C_{\text{Opt}} - c$ *gilt.*

2.5 Ergebnisse

Nimmt man die Teilergebnisse der obigen Abschnitte zusammen, so ergibt sich folgendes Bild.

1. Ganz allgemein für f_{lat} kann kein deterministischer Online-Algorithmus mit Lookahead $L = 0$ besser als 2-kompetitiv sein.
2. $\text{GREEDY}_{\text{total}}$ und $\text{GREEDY}_{\text{new}}$ sind beide optimal für $f_{\max}$, und $\text{GREEDY}_{\text{new}}$ ist allgemein für f_{lat} optimal.
3. $\text{GREEDY}_{\text{total}}$ ist $\Omega(\sqrt{n})$-kompetitiv für die Zielfunktion f_{sum} und somit diesbezüglich nicht optimal.
4. Für Lookahead $L = 1$ findet man mit $\text{GREEDY}_{\text{total}}$ im Zusammenhang mit $f_{\max}$ jedoch einen Online-Algorithmus, der 1-kompetetiv ist, sprich genau so gute Ergebnisse liefert wie eine optimale Lösung.

	f_{sum}		$f_{\max}$	
	$L=0$	$L=1$	$L=0$	$L=1$
GREEDY$_{\text{total}}$	$\Omega(\sqrt{n})$	$\Omega(\sqrt{n})$	2	1
GREEDY$_{\text{new}}$	2	2	2	2
untere Schranke	2	2	2	1

Abb. 3. Die Kompetivität der Algorithmen im Überblick.

2.6 Simulationsergebnisse

Zu den beiden Algorithmen GREEDY$_{\text{total}}$ und GREEDY$_{\text{new}}$ haben auch praktische Messungen mit Beispielimplementierungen stattgefunden. Diese wurden mit den momentan verbreiteten Algorithmen verglichen, die mit statischen *timern* arbeiten (vgl. 1.3).
Die entsprechenden Grafiken und eine ausführlichere Auswertung findet sich in [3]. Die wichtigsten Resultate dabei waren folgende.

- Bei großem η und langen Eingabesequenzen sind die Online-Algorithmen GREEDY$_{\text{total}}$ und GREEDY$_{\text{new}}$ besser, da sie die „billige"Latenz besser ausnutzen können und somit mehr Segmente auf einmal bestätigen können.
- Bei sehr kleinem η („teurer"Latenz) verursachen Algorithmen mit statischen *timern* mehr Kosten als GREEDY$_{\text{total}}$ und GREEDY$_{\text{new}}$, die sich daran direkt anpassen.
- Bei sehr kleinen Eingabesequenzen und nicht zu kleinem η werden nur wenige Bestätigungen gebraucht. Somit unterscheiden sich alle Algorithmen im Wesentlichen durch die Kosten für die durch das Verzögern von Bestätigungen zusätzlich verursachte Latenz. Mit wachsendem η (billigerer Latenz) verzögern sich deshalb GREEDY$_{\text{total}}$ und GREEDY$_{\text{new}}$ unnötig lange bei $L=0$.

3 Ausblick: Randomisierte Algorithmen für verzögerte TCP-Quittungen

Der vorhergehende Abschnitt beschäftigte sich ausschließlich mit deterministischen Ansätzen, um zu bestimmen, wie lange mit dem Versenden von Bestätigungen gewartet werden soll. Auf der Grundlage des Kostenmodells und der dazugehörigen Zielfunktion f_{sum} lassen sich aber auch Algorithmen finden, die praktisch zufällig bestimmen, wann Bestätigungen versendet werden sollen. Diese randomisierten Algorithmen werden wieder mit dem Modell der kompetitiven Analyse untersucht. Hierbei treten sie gegen Gegenspieler an, die die Funktionsweise des Algorithmus genau kennen, nicht aber dessen Zufallsentscheidung (vgl. 2.1).

Die Frage ist nun, wie ein solcher randomisierter Algorithmus seine Entscheidungen treffen soll. Der erste Ansatz ist, die Latenz die der Algorithmus akzeptiert bis er eine Bestätigung absendet, einfach zufällig schwanken zu lassen.

Es hat sich aber gezeigt, dass solche Algorithmen keinesfalls besser sind als die deterministischen mit ihrer Kompetivität von 2.

Die entscheidende neue Idee ist, den Zufall und den Determinismus miteinander zu verknüpfen und somit einen Algorithmus zu erhalten, der bessere Ergebnisse liefert [2].

Gegeben sei eine Zufallsvariable z, die gemäß der Dichtefunktion $p(z) = e^z/(e-1)$ verteilt ist und Werte zwischen 0 und 1 annimmt für $0 \leq z \leq 1$. Weiter sei eine Menge von deterministischen Algorithmen A_z gegeben die folgendermaßen funktionieren:

1. Sei $P(t, t')$ die Menge der Segmente die zwischen dem Zeitpunkt t und t' eintreffen.
2. Die i-te Bestätigung erfolgte zum Zeitpunkt t_i mit $t_0 = 0$.
3. A_z sendet die nächste Bestätigung t_{i+1} genau dann, wenn es einen Zeitpunkt τ_{i+1} mit $t_i \leq \tau_{i+1} \leq i_{i+1}$ gibt, so dass $P(t_i, \tau_{i+1}) \cdot (t_{i+1} - \tau_{i+1}) = z$ gilt.

Anschaulich ausgedrückt bedeutet dies, dass man sich Latenzkosten von z hätte sparen können, wenn man schon zum Zeitpunkt τ_{i+1} eine Bestätigung gesendet hätte und nicht erst zum Zeitpunkt t_{i+1}. Trägt man in einem Koordinatensystem auf der x-Achse die Zeit ab und auf der y-Achse die Summe der eintreffenden Segmente, so beschreibt also $P(t_i, \tau_{i+1}) \cdot (t_{i+1} - \tau_{i+1})$ genau ein Rechteck der Fläche z.

Der randomisierte Online-Algorithmus A wählt also zuerst „zufällig“ein z aus und führt dann den dazu gehörigen Algorithmus A_z aus.

Theorem 7. [2] *Der gerade beschriebene randomisierte Online-Algorithmen A ist $e/(e-1)$-kompetitiv.*

Da $e/(e-1)$ ungefähr $1,58$ ist, liefert dieser randomisierte Online-Algorithmus also bessere Resultate, als die in 2.3 vorgestellten deterministischen Online-Algorithmen. Anzumerken bleibt jedoch in diesem Zusammenhang noch, dass der Vergleich der Ergebnisse der kompetitiven Analyse von deterministischen Algorithmen und randomisierten Algorithmen, unfair gegenüber den deterministischen ist. Diese werden ja immer im *worst-case* betrachtet, wobei bei den randomisierten Algorithmen der Erwartungswert der gelieferten Ergebnisse herangezogen wird.

Literatur

1. R. El Yaniv, A. Borodin. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.
2. A. R. Karlin, C. Kenyon, D. Randall. Dynamic TCP acknowledgement and other stories about e/(e-1). In: *Proceedings 33rd Annual ACM Symposium on Theory of Computing (STOC'2001)*, S. 502-509. ACM Press, 2001.
3. D. R. Dooly, S. A. Goldmann, S. D. Scott. On-line analysis of the TCP acknowledgement delay problem. *Journal of the ACM*, 48(2):243-273, March 2001.
4. J. Postel. Transmission Control Protocol. *IETF Request for Comments 793*, 1981.

5. S. Phillips, J. R. Westbrook. On-line algorithms: Competitive analysis and beyond. In: M. J. Atallah (Hrsg.). *Algorithms and Theory of Computation Handbook*, Kapitel 10. CRC Press, 1999.
6. R. Braden. Requirements for Internet hosts—communication layers. *IETF Request for Comments 1122*, 1989
7. W. R. Stevens. *TCP/IP Illustrated.* Band 1. Addison-Wesley, 1994.
8. A. S. Tanenbaum. *Computernetzwerke.* Prentice Hall, 1997.

Effizienter IP-Adressen-Lookup in Hochgeschwindigkeitsnetzen

Michael Mai

Technische Universität München
maim@in.tum.de

Zusammenfassung. Diese Arbeit befaßt sich mit dem Problem des effizienten *Address Lookup* für Router in Hochgeschwindigkeitsnetzen. Es werden drei Optimierungsmethoden vorgestellt, die einen Überblick über das Problem und seine Lösungen vermitteln sollen. Dazu wird auf Komprimierung der Routingtabelle und Optimierung der Zuordnung von Präfixstrings auf Speicher näher eingegangen. Zuletzt wird eine Möglichkeit zur Realisierung eines Suchalgorithmus auf einem reinen Hardwaresystem behandelt.

1 Einführung

Als Mitte der sechziger Jahre das ARPANET als erstes WAN von der ARPA im Auftrag des US-Verteidigungsministeriums entwickelt wurde, hätte niemand vermutet, daß daraus ein weltweites Netzwerk entstehen sollte. Angefangen bei Übertragungsraten von wenigen Kilobit pro Sekunde, erreichen wir heute Raten im Gigabitbereich. Damit diese Transferraten erreicht werden können, müssen nicht nur die Verbindungen extrem schnell sein, sondern auch die Vermittlungsstellen (Router) müssen diesen Anforderungen genügen.

Im folgenden soll eine detaillierte Beschreibung von einigen Ansatzmöglichkeiten zur Lösung des Effizienten Routings dargestellt werden.

In Kapitel 2 werden dazu die Grundlagen des Problems beschrieben, die zum weiteren Verständnis notwendig sind. Kapitel 3 erörtert die Lösung, wie sie in [1] veröffentlicht wurde. Kapitel 4 zeigt eine Möglichkeit auf, Baumstrukturen, wie sie in Address Lookups häufig verwendet werden, optimal auf hierarchisch angeordneten Speichern abzulegen. Dieses Kapitel nimmt Bezug auf die Arbeit von [2]. Kapitel 5 geht auf hardwarebasierte Lösungsansätze ein und stellt den Algorithmus aus [4] vor. Auf weitere Literatur über den Entwurf kleiner Routingtabellen [3] und über Leistungsmodellierung für schnelle IP-Adressen-Lookups [5] sei verwiesen.

2 IP-Adressierung und Kommunikation

Um das Problem richtig erfassen zu können, ist es vorher notwendig die Grundlagen der Adressierung und Kommunikation im Internet verstanden zu haben. Jeder Knoten (Host oder Router) im Internet besitzt eine eindeutige Adresse, unter

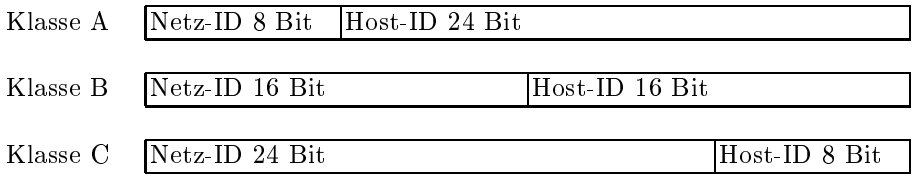


Abb. 1. Klassen von IP-Adressen.

welcher dieser erreichbar ist. Die Kommunikation erfolgt durch IP-Pakete, die im Nachrichtenkopf (Header) die Quell- und Zieladresse enthält. Diese Adressen sind im IPv4-Standard auf 32 Bit begrenzt. Theoretisch sind somit $2^{32} = 4294967296$ Knotenpunkte ansprechbar. Um die Lesbarkeit der IP-Adressen zu erhöhen werden sie oft in *dotted decimal notation* geschrieben. Dabei wird jedes Byte der Adresse als Dezimalzahl geschrieben und durch Punkte voneinander getrennt (z.B. 10.100.0.99 $\equiv$ 00001010 01100100 00000000 01100011).

Um eine strukturierte Verteilung ohne Redundanzen und eine einfachere Verwaltung zu gewährleisten, wurden die Adressen in Adressbereiche mit unterschiedlichen Einsatzgebieten aufgeteilt. Die IP-Adresse besitzt eine Netz-ID und eine Host-ID. Alle an einem Teilnetz (z.B. LAN) angeschlossenen Rechner besitzen immer eine gemeinsame Netz-ID und eine notwendigerweise verschiedene Host-ID. Die Menge der Adressen wurde wie in Abbildung 1 dargestellt in drei Klassen eingeteilt. Um die Klassen ohne zusätzlichen Aufwand voneinander unterscheiden zu können, sind die Netz-IDs eindeutig charakterisiert:

Klasse	Netz-ID	Adressbereich
A	0...	1.0.0.0 bis 127.255.255.255
B	10...	128.0.0.0 bis 191.255.255.255
C	110...	192.0.0.0 bis 223.255.255.255

Die übrigen Adressen sind Multicast- und noch unbelegte Adressen. Durch diese Strukturierung ergibt sich die maximale Anzahl Hosts pro Netzwerk. So dürfen in diesen Netzen jeweils höchstens $2^n - 2$ Hosts angeschlossen werden, wobei n die Länge der Host-ID in Bits bezeichnet. Durch das enorme Wachstum des Internets stieg der Bedarf an IP-Adressen und insbesondere aus dem Adressbereich der Klasse B. Also mußten die Adressbereiche angepaßt werden, um die Verschwendung unnütz vergebener Adressen zu verringern. Wichtig war hierbei aber die Strukturierung der Adressen des Internets weitest gehend zu übernehmen.

Die Lösung des Problems brachte 1993 die Einführung von *Classless Interdomain Routing* (CIDR). Dieses ist ein Zusatzprotokoll, welches es erlaubt ein Netz in mehrere Teilnetze zu unterteilen. Man erweiterte das Protokoll um eine Teilnetzmaske, die angibt welcher Teil der 32 Bits zur Netz-ID und welcher zur Host-ID gehört. In folgendem Beispiel wird ein Klasse-B-Netz in 64 Teilnetze mit jeweils 1024 Hosts zerlegt:

IP-Adresse:	Netz-ID	Teilnetz	Host-ID
Teilnetzmaske:	1111111111111111	111111	0000000000

Diese Lösung ist allerdings nicht auf Dauer befriedigend. Deshalb wird in den kommenden Jahren die Version 6 des IP-Protokolls eingeführt. Dieses ermöglicht eine 128-Bit-Adressierung.

3 Routingproblem bei Hochgeschwindigkeitsnetzen

Im Internet ist der Benutzer darauf angewiesen, auf andere Teilnetze Zugriff zu erlangen. Die Aufgabe, Teilnetze zu verbinden und Anfragen zu dem gewünschten Zielrechner weiterzuleiten, übernehmen Router. Router besitzen hierfür mehrere *interfaces* für eingehende, sowie ausgehende Pakete. Zusätzlich benötigt dieser eine *forwarding engine* und eine *routing table*. Erhält ein Router ein IP-Datenpaket, vergleicht er die Zieladresse aus dem Header des Pakets mit seiner internen Routingtabelle. Daraus ermittelt er, in welches Teilnetz er das Paket weiterleiten muß. Aufgrund des Anstiegs der Übertragungsgeschwindigkeit und der Bandbreite, soll die Latenzzeit im Router möglichst gering gehalten werden. Deshalb muß der *IP-Address Lookup* selbst bei Tabellen mit mehr als 40000 Einträgen sehr effizient sein.

Es gibt verschiedene Ansätze, um Routing zu optimieren. In dieser Arbeit sollen drei Methoden unterschiedlicher Optimierungsansätze vorgestellt werden.

4 Lösung mittels *Run Length Encoding* (RLE)

4.1 Vorüberlegungen

Crescenzi, Dardini und Grossi [1] gehen von aktuellen Meßdaten von Routern aus und versuchen den *IP-Address Lookup* aus dieser Sichtweise zu optimieren. Der entwickelte Algorithmus funktioniert für beliebige m-bit-Adressen. Das bietet den Vorteil, dass der Algorithmus auch nach der Einführung der IPv6-Adressen seine Gültigkeit behält. Die Berechnung erfolgt in zwei Phasen, der Expansions- und der Kompressionsphase. Die Expansionsphase leitet aus der ursprünglichen Routingtabelle T eine Tabelle T' ab, die alle Routinginformationen für jede mögliche IP-Adresse enthält. Das sind somit bei der aktuellen IPv4-Adressierung alle 2^{32} Adressen. Da diese zuviel Speicherplatz verbrauchen würden, muß die Tabelle stark komprimiert werden. Dies geschieht in einem zweiten Schritt, der Kompressionsphase. Die Daten werden in dieser Phase zweidimensional komprimiert. Der Algorithmus braucht insgesamt $O(2^{\frac{m}{2}} + |T|^2)$ Speicherplatz im schlechtesten Fall, d.h. die Komprimierung der Daten ist nicht möglich. In der Praxis taucht dieser Fall jedoch nicht auf, sodaß die Abschätzung zu pessimistisch erscheint. Tatsache ist aber, daß die Datenstruktur mehr Platz benötigt als eine einfache Routingtabelle.

Die Einträge in der Routingtabelle T sind als Tupel (p, h) abgelegt. Der Eintrag p bezeichnet das Präfix, h legt das Ausgangsinterface fest. Ein Beispiel für

Präfix	Ausgabeinterface
ϵ	A
10101010	B
10101010 111	C
10101010 011	D
10101010 110	C

Abb. 2. Beispiel einer Routingtabelle.

eine Routingtabelle ist in Abbildung 2 zu sehen. Hierbei steht das ϵ für das Standard-Ausgabeinterface. Dieses wird für alle Adressen benutzt, für die kein passendes Präfix in der Tabelle existiert. Wie man schon an diesem Beispiel erkennen kann, ließen sich der Eintrag 3 und 5 zu einem gemeinsamen Eintrag $(10101010\ 11, C)$ zusammenfassen. Ziel ist es, die Daten verlustfrei, aber möglichst komprimiert, zu speichern, sodaß man auf dieser Datenstruktur effizient suchen kann.

4.2 Expansionsphase

Die Expansionsphase erhält als Eingabe eine unsortierte Routingtabelle T (s.o.). Der erste Schritt in dieser Phase besteht darin, die Elemente der Tabelle absteigend zu sortieren. Wir erhalten die sortierte Sequenz der Einträge $T_1, T_2, \ldots, T_{|T|}$ durch folgende Definition

$$T_i < T_j \iff_{\text{def}} p_j \text{ ist Präfix von } p_i. \tag{1}$$

Aus (1) entstehen einzelne Gruppen von Präfixen. Diese werden dann lexikographisch sortiert.

Im nächsten Schritt nehmen wir uns jeden einzelnen Eintrag aus der Tabelle heraus und bilden seine Expansionsmenge $\text{Exp}(T_i)$. Diese ist für alle $i \in \{1, \ldots, |T|\}$ wie folgt definiert

$$\text{Exp}(T_i) =_{\text{def}} (p_i \cdot \Sigma^{m-|p_i|}) \times \{h_i\}. \tag{2}$$

Durch die Expansionsmenge (2) eines Tabelleneintrags bilden wir die Menge aller Adressen, die p_i als Präfix besitzen. Die gesamten Expansionsmengen sind allerdings noch nicht disjunkt, sodaß nun für ein und dieselbe Adresse mehrere Ausgabeinterfaces definiert wurden. Um diese Inkonsistenz wieder rückgängig zu machen, berechnen wir induktiv die Expansion T' von T gemäß

$$\begin{aligned} T'_1 &=_{\text{def}} \text{Exp}(T_1), \\ T'_i &=_{\text{def}} \text{Exp}(T_i) \setminus \bigcup_{j=1}^{i} T'_j \text{ für } i \geq 2\ , \\ T' &=_{\text{def}} \bigcup_{i=1}^{|T|} T'_i. \end{aligned} \tag{3}$$

Aufgrund des vorherigen Sortierens werden in (3) alle mehrfach auftauchenden Adreßeinträge eliminiert. Das Ergebnis der Expansionsphase ist eine Tabelle T', die für jede m-Bit-Adresse das Tupel (p, h) mit $|p| = m$ beinhaltet.

4.3 Kompressionsphase

Nun müssen die Daten komprimiert werden. Der Kompressionsfaktor hängt von dem Parameter k ab, wobei $1 \leq k \leq m$ gelten soll. Wir teilen die Bitstrings in Gruppen ein, die dieselben k Anfangsbits besitzen. Für einen String $x \in \Sigma^k$ bilden wir seine zugehörige Gruppe $T_{(x)}$

$$T_{(x)} =_{\text{def}} \{(y, h_{x \circ y}) | y \in \Sigma^{m-k} \wedge (x \circ y, h_{x \circ y}) \in T'\}. \tag{4}$$

Hierbei definiert $h_{x \circ y}$ das Ausgangsinterface für die Adresse, die sich aus der Konkatenation der Strings x und y ergibt. Klar ist, daß $|T_{(x)}| = 2^{m-k}$. Der Parameter k legt also die Anzahl der Gruppen fest. Mit (4) liefert uns Gleichung (5) einen Wert für α_k, der die Anzahl verschiedener Gruppen abzählt. Tauchen manche Gruppen öfters auf, lassen sie sich später auf eine abbilden. Somit gibt uns α_k einen ersten Wert an, wie gut die Komprimierung ist:

$$\alpha_k = |\{T_{(x)} | x \in \Sigma^k\}|. \tag{5}$$

In einem zweiten Schritt wollen wir die Folge der Tupel in $T_{(x)}$ mittels Lauflängencodierung (*run length encoding, RLE*) weiter komprimieren. Formal müssen wir dazu alle Tupel in $T_{(x)}$ aufsteigend sortieren, d.h. wir sortieren die Tupel $(y, h_{x \circ y})$ nach $y \in \Sigma^{m-k}$ lexikographisch aufsteigend. Danach numerieren wir die Tupel in der Reihenfolge durch, in der sie in $T_{(x)}$ auftreten:

$$T_{(x)} = \{(y_1, h_1), (y_2, h_2), \ldots, (y_{2^{m-k}}, h_{2^{m-k}})\}. \tag{6}$$

Jetzt codieren wir $T_{(x)}$ als Lauflängensequenz $s_{(x)}$ indem wir alle maximalen Läufe $(y_i, h_i), (y_{i+1}, h_{i+1}), \ldots, (y_{i+l}, h_{i+l})$ für die $h_i = h_{i+1} = \ldots = h_{i+l}$ und $0 \leq l \leq 2^{m-k} - 1$ gilt, durch ein neues Tupel $\langle h_i, l + 1 \rangle$. Das bedeutet, alle aufeinanderfolgenden Einträge, die dasselbe Ausgabeinterface besitzen, werden auf ein Tupel mit dem Wert des Interfaces und der Anzahl der ursprünglichen Einträge (Lauflänge, $l+1$) reduziert. Das Ergebnis ist meist eine kürzere Sequenz

$$s_{(x)} = \langle h_1, l_1 + 1 \rangle, \langle h_2, l_2 + 1 \rangle, \ldots. \tag{7}$$

Für eine Komprimierung aller Sequenzen wollen wir mit Hilfe der Funktion φ die einzelnen Lauflängen vereinheitlichen. Erst wenn alle Sequenzen die gleiche Unterteilung besitzen, können wir sie in einer geeigneten Datenstruktur ablegen. Die Funktion φ zerlegt die Lauflängen der zweiten Sequenz nach dem Vorbild der ersten. Sie ist für zwei Sequenzen $s = \langle a, f \rangle \cdot s_1$ und $t = \langle b, g \rangle \cdot t_1$ wie folgt definiert:

$$\varphi(s, t) =_{\text{def}} \begin{cases} t, & \text{falls } s_1 = t_1 = \epsilon, \\ \langle b, f \rangle \cdot \varphi(s_1, t_1), & \text{falls } f = g \text{ und } s_1, t_1 \neq \epsilon, \\ \langle b, f \rangle \cdot \varphi(s_1, \langle b, g - f \rangle \cdot t_1), & \text{falls } f < g \text{ und } s_1, t_1 \neq \epsilon, \\ \langle b, g \rangle \cdot \varphi(\langle a, f - g \rangle \cdot s_1, t_1), & \text{falls } f > g \text{ und } s_1, t_1 \neq \epsilon. \end{cases} \tag{8}$$

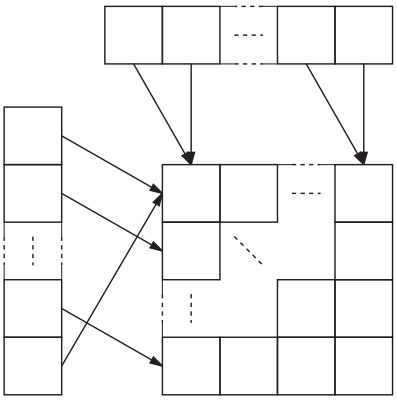


Abb. 3. Die Datenstruktur für den Algorithmus.

Wenden wir (8) nun paarweise auf alle Sequenzen an, erhalten wir als gemeinsame Lauflängencodierung

$$s'_q =_{\text{def}} \varphi(\varphi(\ldots\varphi(\varphi(\varphi(s_1, s_2), s_3), s_4)\ldots, s_{q-1}), s_q). \tag{9}$$

Die in (9) berechnete Sequenz s'_q müssen wir nun auf alle Sequenzen s_i mit $i < q$ anwenden. Wir bekommen

$$s'_i =_{\text{def}} \varphi(s'_q, s_i). \tag{10}$$

Aus den Schritten (6)-(10) können wir den zweiten Wert der Komprimierung ableiten. Dieser ist durch die Anzahl der Tupel in der Sequenz s'_q festgelegt, d.h. $\beta_k = |s'_q|$.

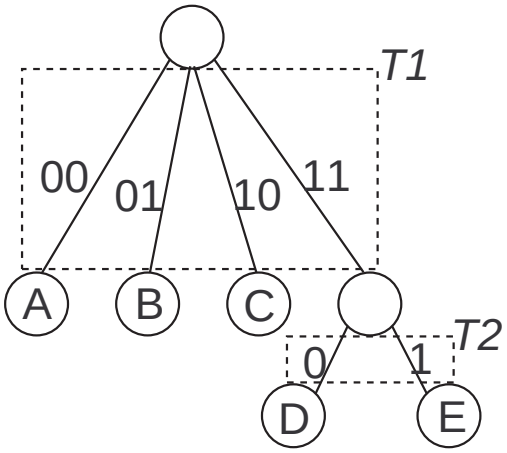
4.4 Die zugrundeliegende Datenstruktur

Für das Ablegen der Datenstruktur benötigen wir ein Array `row_index` der Größe 2^k und ein Array `col_index` der Größe 2^{m-k}. Diese Tabellen enthalten Zeiger auf eine Matrix `interface`, die das entsprechende Ausgabeinterface zurückliefert (siehe Abbildung 3). Die Rückgabe eines Interfaces kann in genau drei Speicherzugriffen erledigt werden:

$$h_x = \texttt{interface}[\texttt{row\_index}[x[1\ldots k]], \texttt{col\_index}[x[k+1\ldots m]]] \tag{11}$$

4.5 Die Wahl des Parameters *k*

Bisher wurde der Parameter k für Rechnungen gebraucht, aber noch nicht über seine Auswirkungen diskutiert. Der Parameter k bestimmt die Anzahl der Gruppen $T_{(x)}$ in (4). In der aktuellen IP-Version 4 sind die Adressen 32 bit lang. Alle Präfixe besitzen somit eine Länge zwischen 8 und 32 Bits. Aus Datenanalysen



Präfix	Interface	W 'keit
111	E	0,2
110	D	0,3
10	C	0,1
01	B	0,2
00	A	0,2

Abb. 4. Ein *generalized level-compressed trie* und seine Routingtabelle.

von Routern aus dem IPMA-Projekt[1] zeigt sich, daß die meisten gerouteten Pakete eine Präfixlänge von 16 oder 24 Bits haben. Dies entspricht der Klasseneinteilung von Netzen der Klasse B und C. Es empfiehlt sich k als Vielfaches von 8 zu wählen, da bit-Operationen schneller ausgeführt werden können. Die Autoren des Algorithmus empfehlen deshalb $k = 16$ zu verwenden. Damit sind die beiden Arrays `row_index` und `col_index` ausgeglichen.

5 Optimales Speicherdesign der Forwarding Table

5.1 Die Datenstruktur

Cheung und McCanne [2] beschäftigten sich mit dem Problem, wie man in einem System mit hierarchisch angeordneten Speichern durch günstiges Aufteilen der Routingtabelle die Suche beschleunigen könnte. Die Übergabe der Tabelle erfolgt als Baumstruktur. Sie versuchen nun Teilbäume, die während der Suche häufig durchlaufen werden, nach Möglichkeit in einem schnelleren Speicher abzulegen, als diejenigen, die seltener verwendet werden. Um dies zu realisieren, ist es notwendig sich zu den Präfixeinträgen in der Routingtabelle jeweils deren Häufigkeit bzw. Wahrscheinlichkeit zu merken. Als Ergebnis des Algorithmus erhalten wir einen *generalized level-compressed trie*, wie er in Abbildung 4 zu sehen ist, sowie die Zuordnung von Teilbäumen auf Speicherbereiche.

Um eine Routingtabelle in einen *generalized level-compressed trie* zu transformieren, sind einige Zwischenschritte erforderlich. Zuerst wird ein Binärbaum erzeugt. Dieser wird mittels der *controlled prefix expansion* genannten Technik, wie sie in [3] oder in [6] erklärt wird, in einen *complete binary tree* umgeformt. Ein derartiger Baum besitzt entweder zwei oder gar keine Kinder. In einem weiteren Schritt soll der nun ausgegebene Baum in den *generalized level-compressed*

[1] Internet Performance Measurement and Analysis, `http://www.merit.edu/~ipma`.

trie verwandelt werden. Hierzu werden alle maximalen, vollständigen Teilbäume der Höhe h durch einen Teilbaum mit exakt 2^h Kindern ersetzt. Damit wir die Zuordnung auf Speicherbereiche errechnen können, müssen wir eine Kostenfunktion aufstellen, die uns angibt, wie gut eine Zuordnung ist.

5.2 Die Kostenfunktion

Wir wollen jetzt eine Kostenfunktion C für ein Rechensystem R mit hierarchisch angeordneten Speichern der Größe S_i und den dazugehörigen Zugriffszeiten T_i aufstellen. Dabei soll $S_i \leq S_{i+1}$ und $T_i < T_{i+1}$ gelten. Außerdem soll j für das j-te Präfix in der Routingtabelle stehen, d.h. $1 \leq j \leq |T|$, und p_j soll die Wahrscheinlichkeit des Präfixes j sein. Um die Aufteilung der Teilbäume in Speicherbereiche in der Kostenfunktion C zu berücksichtigen, ist ein zusätzlicher Parameter pro vorhandenem Speicher $(a_j, b_j, c_j, \ldots)$ nötig. Dieser ist 1, wenn das Präfix j auf den jeweiligen Speicher zugreift, sonst besitzt dieser den Wert 0.

Für ein System mit zwei Speichern $R = \langle (S_1, T_1), (S_2, T_2) \rangle$ ergibt sich für unsere Kostenfunktion C

$$C = \sum_j p_j (a_j T_1 + b_j T_2).$$

Diese Funktion berechnet die Gesamtkosten durch die Summe der Kosten, die ein einzelnes Präfix durch die Summe seiner Zugriffszeiten auf die Speicher verursacht, unter Berücksichtigung der Wahrscheinlichkeit des Präfixes. Für die nachfolgenden Betrachtungen wollen wir die Koeffizienten der T_i als Gewicht w_i. Es gilt $w_1 = \sum_j p_j a_j$ und $w_2 = \sum_j p_j b_j$.

Der nachfolgende Algorithmus soll nun diese Kostenfunktion minimieren.

5.3 Der Algorithmus

Wir suchen für unser Rechnensystem R eine Berechnungsmöglichkeit, die uns als Ergebnis diejenige Speicherzuordnung liefert, bei der die Kostenfunktion minimal wird. Wir setzen dabei voraus, daß S_1 endliche Größe besitzt und daß die komplette Datenstruktur in unseren beiden Speichern untergebracht werden kann. D.h. wir können $S_2 = \infty$ annehmen. Ebenso gilt $T_1 < T_2$.

Wir definieren nun zwei rekursive *tree-packing*-Funktionen TP_1 und TP_2 mit den Parametern i, S_1 und $L_{h,i}$. Dabei sei i ein Knoten aus unserem *generalized level-compressed trie* und S_1 die Größe des noch zur Verfügung stehenden Platzes in dem schnelleren Speicher unseres Systems. Sei H_i die Gesamthöhe des Baumes ab dem Knoten i und h mit $1 \leq h \leq H_i$ die Höhe eines beliebigen Teilbaumes mit Wurzel i, dann bezeichnet $L_{h,i}$ die Menge aller Knoten auf dem h-ten Level unter der Wurzel i. Wollen wir einen Teilbaum im Speicher ablegen, so bleiben nur zwei Auswahlmöglichkeiten.

Die erste Möglichkeit ist, diesen Teil im schnellen Speicher abzulegen. Das hat aber zur Folge, daß sich S_1 um den benötigten Platz verringert, und daß weniger schneller Speicher für den Rest der Datenstruktur übrigbleibt.

Die andere Lösung ist, den Teilbaum im langsameren Speicher abzulegen, d.h. der Zugriff dauert länger, jedoch bleibt S_1 gleich. TP_1 wägt nun diese Möglichkeiten über die Gewichte w_i gegeneinander ab und minimiert so die Kosten:

$$\mathrm{TP}_1(i, S_1) =_{\mathrm{def}} \min_{1 \leq h \leq H_i} \min\{w_i T_1 + \mathrm{TP}_2(L_{h,i}, S_1 - 2^h), w_i T_2 + \mathrm{TP}_2(L_{h,i}, S_1)\}.$$

Damit die Funktionen TP_1 und TP_2 terminieren, definieren wir, daß alle Lösungen mit negativen Werten für S_1 unzulässig sind bzw. unendliche Kosten verursachen. Da S_1 nicht zwingend kleiner werden muß, ist noch ein weiterer Fall zu beachten. Für einen Knoten i wird in jedem Berechnungsschritt h erhöht. Dies hat zur Folge, daß der rekursive Aufruf der Funktion TP_2 irgendwann für $L_{h,i}$ die leere Menge übergeben bekommt. Für diesen Fall definieren wir, dass TP_2 den Wert 0 annimmt, da es keine Kosten verursachen soll, einen leeren Baum abzuspeichern.

Im Gegensatz zu TP_1 besitzt TP_2 eine Menge an Knoten als Parameter. Um alle Zuordnungsmöglichkeiten abzudecken, teilen wir die Menge L in zwei disjunkte Mengen L_1 und L_2 auf, d.h. es gilt $L_1 \cap L_2 = \emptyset$ und $L_1 \cup L_2 = L$. Hieraus folgt als Definition der Funktion TP_2:

$$\mathrm{TP}_2(L, S_1) =_{\mathrm{def}} \begin{cases} \mathrm{TP}_1(i, S_1), & \text{falls } L = \{i\}, \\ \min\limits_{0 \leq S \leq S_1} \mathrm{TP}_2(L_1, S) + \mathrm{TP}_2(L_2, S_1 - S), & \text{sonst.} \end{cases}$$

Klar ist, daß wir der Funktion TP_2 ebenfalls die Größe des freien Speichers S_1 übergeben müssen. Das Problem, welches sich nun aber ergibt, ist, wie wir im Falle des rekursiven Aufrufs den freien Speicher zwischen L_1 und L_2 aufteilen. Um das globale Minimum der Kostenfunktion zu erhalten, bleibt nur die Möglichkeit, alle Lösungen zu errechnen und davon das Minimum zu bilden.

5.4 Laufzeitanalyse des Algorithmus

Würde der Algorithmus in dieser Form implementiert, würde er exponentielle Laufzeit besitzen. Dies wäre nicht tragbar für einen Router, denn der Algorithmus muß nach jeder Veränderung der Routingtabelle erneut aufgerufen werden. Darum setzt man dynamische Programmierung ein. Hierzu wird eine Tabelle erstellt, die die Zwischenergebnisse abspeichert und so Teillösungen nur einmal berechnet. Ein Beispiel einer *dynamic programming table* (DPT) ist in Abbildung 5 zu sehen.

Damit kann die Laufzeit darauf beschränkt werden, die für die Erstellung der DPT benötigt wird. Dadurch ergibt sich eine pseudo-polynomielle Laufzeit von $O(HnS_1^2)$. Dabei sei H die Gesamthöhe des *complete binary trees*, der als Eingabe diente, n sei die Anzahl der Knoten in diesem Baum und S_1 der verfügbare Platz im ersten Speicher. In [2] wird die Laufzeit des Algorithmus durch den Lagrange-Approximationsalgorithmus noch weiter verbessert.

S_1	0	1	2
$TP_1(4,.)$			1,6
$TP_1(1,.)$	0,8	0,4	0,4
$TP_1(3,.)$	1,2	1,2	0,6
$TP_1(2,.)$	0,6	0,3	0,3
$TP_2(\{1,3\},.)$	2	1,6	1,4

Abb. 5. Beispiel für eine DPT.

6 Hardwarebasierte Lösungen des effizienten Routings

6.1 Eigenschaften von Hardwarelösungen

Zuletzt soll in diesem Überblick auf die Möglichkeit verwiesen werden, Algorithmen bzw. Suchstrukturen auf reinen Hardwaresystemen unterzubringen. Dies bringt den Vorteil, daß die Berechnung mittels Schaltern, Gattern und Multiplexern meist schneller ausgeführt werden können als auf einem softwarebasierendem System. Dieser Vorteil wird aber durch einen höheren Preis und durch die Unveränderlichkeit des Codes relativiert. Diese feste Berechnungsvorschrift kann dazu führen, daß hardwarebasierte Lösungen schnell veralten und ein komplett neues System erworben werden muß. Ein weiterer Nachteil ist die Verschwendung von Speicherplatz, der für den *worst case* reserviert werden muß, jedoch im Normalfall nicht benutzt wird. Insbesondere muß das Problem des *longest prefix matchings* für den *IP-Address Lookup* abgewandelt werden, da die Hardware nicht ohne weiteres mit variablen Längen der Präfixe umgehen kann.

6.2 Algorithmus für hardwarebasierten IP-Adressen-Lookup

Der Algorithmus, welcher in [4] vorgestellt wurde, benutzt als Repräsentation der Routingtabelle wiederum eine Baumstruktur. Damit variable Längen in unserem Baum vermieden werden, teilen wir die IP-Adresse in eine feste Anzahl von Teilstrings (drei bis fünf) auf. Einen solchen Teilstring nennen wir *Index*. Aufgrund der vorher erwähnten Einschränkung müssen wir alle Präfixe auf Längen gemäß der getroffenen Einteilung erweiteren. Dazu wenden wir wieder *controlled prefix expansion* an. Auf diese Weise erhalten wir einen Baum, der entweder keine oder alle Kinder besitzt. Wir benötigen zudem eine Markierung der Knoten. Entspricht der Knoten einem Eintrag in der Routingtabelle, so wird er als `valid` bezeichnet. Falls ein Knoten diese Bedingung nicht erfüllt, aber mindestens einer seiner Söhne `valid` ist, dann liegt der Knoten auf dem Pfad eines gültigen Präfixes. Er bekommt die Markierung `part`. Die dritte zulässige Markierung, `invalid`, erhalten alle Knoten die keine Söhne besitzen und selber keinen Eintrag in der Routingtabelle repräsentieren, d.h. es existiert kein passendes Präfix. Außerdem soll gelten, daß jeder Knoten nur eine Markierung tragen darf. Das bedeutet für Knoten, die sowohl `part` als auch `valid` sind, daß sie als `part` markiert werden und zusätzlich alle Söhne angehängt bekommen.

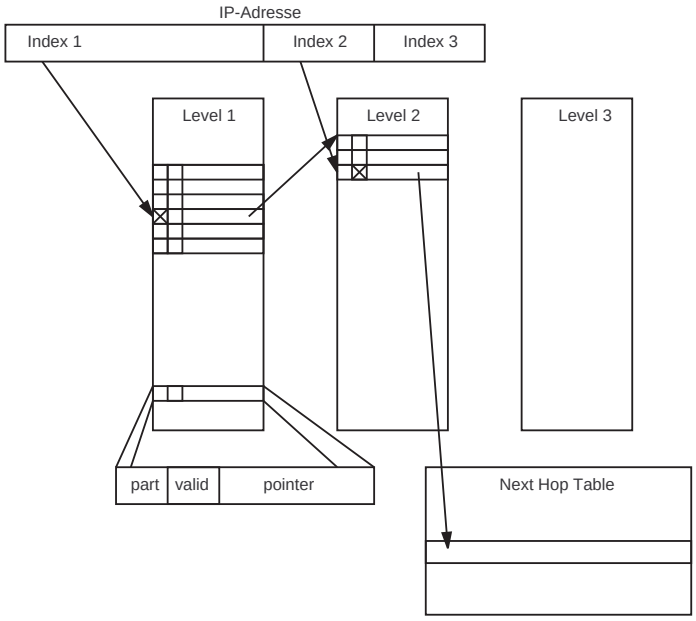


Abb. 6. Implementation der Datenstruktur (aus [4]).

Technisch realisiert wird dieser Baum als verlinkte Liste von Tabellen. Dazu besitzt das System für jeden Index eine eigene Tabelle. Zeigt ein Index, also ein Teil der IP-Adresse auf einen Knoten, der `invalid` ist, so wird das Paket an das Standard-Ausgabeinterface geliefert. Ist der Zielknoten `valid`, wird ein Pointer auf die Next-Hop-Tabelle zurückgeliefert und dort das Interface abgelesen. Falls der Knoten als `part` markiert wurde, so besteht der Eintrag in der Tabelle aus einem Zeiger auf den Anfang der weiterführenden Indextabelle. Auf diese Speicherstelle wird der nächste Indexstring als Offset aufaddiert, um den nächsten Knoten im Baum zu erreichen. Diese Funktionsweise des Algorithmus ist in Abbildung 3 zu erkennen. Der Nachteil bei dieser Methode besteht in der Verschwendung von Speicherplatz, denn für eine Tabelle muß so viel Speicher reserviert werden, wie im schlimmsten Fall auftreten kann. In der Regel wird jedoch weniger Platz benötigt. In [4] wird deshalb zusätzlich ein Speicheroptimierungverfahren vorgestellt, um die Speicherausnutzung zu verbessern.

Literatur

1. P. Crescenzi, L. Dardini, R. Grossi. IP address lookup made fast and simple. In: *Proceedings 7th Annual European Symposium on Algorithms (ESA'99)*, Band 1643 der *Lecture Notes in Computer Science*, S. 65-76. Springer-Verlag, 1999.
2. G. Cheung, S. McCanne. Optimal routing table design for IP address lookups under memory constraints. In: *Proceedings IEEE Conference on Computer Communication (INFOCOM'99)*, S. 1437-1444. IEEE Computer Society Press, 1999.

3. M. Degermark, A. Brodnik, S. Carlsson, S. Pink. Small forwarding tables for fast routing lookups. In: *Proceedings Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGGCOMM'97)*, S. 3-14. ACM Press, 1997.
4. A. Moestedt, P. Sjödin. IP address lookup in hardware for high-speed routing. Technischer Bericht, Computer and Network Architectures Laboratory, Swedish Institute of Computer Science, 1998.
5. G. J. Narlikar, F. Zane. Performance modeling for fast IP lookups. In: *Proceedings Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'2001)*, S. 1-12. ACM Press, 2001.
6. V. Srinivasan, G. Varghese. Faster IP lookups using controlled prefix expansion. In: *Proceedings Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'98)*, S. 1-10. ACM Press, 1998.

Verteilte Auswertung von Web-Queries

Fabian Kainzinger

Technische Universität München
kainzing@in.tum.de

Zusammenfassung. Bei der Auswertung von Anfragen an verteilte Datenquellen sind besondere Aspekte zu berücksichtigen, falls sich die Datenquellen im Internet befinden. Die sich ständig ändernde Topologie und das dynamische Auffinden weiterer relevanter Datenquellen müssen bei der Auswertung der Anfrage beachtet werden. In dieser Arbeit werden ein formaler Rahmen für die Modellierung der Verteilung, Kriterien für eine korrekte Antwort auf eine Anfrage und ein Algorithmus zur Berechnung einer Antwort auf eine Anfrage vorgestellt. Außerdem wird noch auf mögliche Optimierungen z.B. bei semistrukturierten Daten eingegangen.

1 Einführung

Bei klassischen verteilten Datenquellen bzw. Datenbanken ist die Verteilung der Daten über verschiedene logische Einheiten (meist unterschiedliche Rechner) einer zentralen Instanz bekannt. Diese ist für die Beantwortung von Anfragen an den verteilten Datenbestand zuständig und kann bei einer eingehenden Anfrage die zur Beantwortung nötigen Datenquellen lokalisieren und einen globalen Plan zur Berechnung der Antwort erstellen. Dieser Plan wird dann sukzessive abgearbeitet, wobei einzelne Rechner jeweils ihren Teil bearbeiten und Ergebnisse danach zusammengefasst werden.

Betrachtet man jedoch das Internet als Datenquelle, kann ein solches Modell nicht zum Tragen kommen. Die Struktur des Internet sieht keine zentrale Instanz vor, der die Verteilung aller zugänglichen Informationen bekannt ist. Die Erstellung eines Plans zur Bearbeitung einer Anfrage ist nicht von vornherein möglich. Da die Topologie des Internets sich andauernd ändert, müssen bei der Bearbeitung einer Anfrage neue, vorher unbekannte Datenquellen dynamisch berücksichtigt werden, d.h. in die Abarbeitung eingebunden werden, sobald man Hinweise auf sie erhalten hat.

Bei der Betrachtung von verteilten Informationen im Internet kann man Informationen nicht nur als Objekte in einer klassischen Datenbank betrachten. Bilder, Töne, Dokumente, Text auf Webseiten etc. sind Informationen. Zusätzlich dazu gibt es Verweise (Links) auf andere Informationsquellen (im WWW durch die Hyperlinks realisiert), die nicht die bei einer Anfrage gewünschten Informationen enthalten, sondern nur Hinweise auf weitere zu betrachtende Stellen. Beispiele dafür sind z.B. Bookmarklisten, Ergebnisse von Anfragen an Suchmaschinen oder auch in Dokumenten enthaltene Literaturhinweise.

Begibt man sich also im Internet auf die Suche nach einer Information, trifft man auf zwei verschiedene Dinge. Zum einen sind dies Informationen, die direkt zur Anfrage passen, zum anderen die oben erwähnten Verweise auf weitere Informationsquellen. Dementsprechend kann auch ein Server, der die Anfrage eines Clients bearbeitet, zwei unterschiedliche Arten von Antworten geben. Dies ist je nach den Umständen der Anfrage auch sinnvoll, da z.B. eine einzige Metainformation nützlicher sein kann als eine erschöpfende Aufzählung vieler Einzeldaten. Selbstverständlich sind auch Mischformen aus beidem möglich, d.h. z.T. direkte Informationsrückgabe und z.T. Rückgabe von Verweisen. Eine genauere Formulierung der Antwortarten folgt in Kapitel 2.

Diese unterschiedlichen möglichen Antworten auf eine Anfrage ziehen einige zu betrachtende Aspekte nach sich. Es stellt sich die Frage, welche Art von Antwort ein Server geben soll. Punkte wie die Auslastung der beteiligten Rechner, die Belastung des Netzwerks oder auch die Stellung des Clients (z.B. „Premium-Kunde“) sind hier wichtig. Darüberhinaus muss gerade bei vertraglichen Bindungen berücksichtigt werden, was eine vollständige und korrekte Antwort auf eine Anfrage darstellt, und vor allem, wie sich diese berechnen lässt. Trivialerweise ist jeweils der gesamte Wissensstand des Servers eine mögliche Antwort, dies dürfte aber nicht immer im Interesse des Clients liegen. Auf dem letzten Punkt liegt der Schwerpunkt dieser Arbeit, die Betrachtung folgt in Kapitel 3.

Auch die Effizienz und tatsächliche Durchführbarkeit eines gegebenen Verfahrens spielen eine Rolle. Sofern man über die verteilten Daten bestimmte Annahmen treffen kann (z.B. bezüglich der Struktur) lassen sich einige Betrachtungen zur Laufzeit und Optimierung anstellen. Dies folgt in Kapitel 4.

2 Formalismus

Zur Modellierung der Informationsverteilung benutzen wir Datalog mit einer Erweiterung, wie in [6] beschrieben. Anschließend zeigen wir anhand von DNS, wie konkrete verteilte Informationen damit modelliert werden können.

2.1 Datalog

Datalog ist eine Untermenge von Prolog und soll nur kurz vorgestellt werden. Weitere Informationen dazu finden sich in [2].

Die Informationen werden in Datalog durch Fakten und Regeln beschrieben, wobei hier keine genaue Unterscheidung getroffen wird. Wichtige Elemente sind Namen (z.B. `Hans, Maria, 129.187.176.20`) und Variablen (z.B. x, y). Wie in der relationalen Algebra der Datenbanken werden Informationen durch Relationen (auch Atome) formuliert. Will man z.B. ausdrücken, dass Maria die Mutter von Hans ist, formuliert man dies als `Mutter(Hans, Maria)`. Weitere Beispiele für Relationen sind `IP(Hans, 129.187.76.20)` oder `lieb(Hans)`.

Das zentrale Element in Datalog sind Regeln, die zur Herleitung neuer Informationen durch logisches Schließen verwendet werden. Ein intuitives Beispiel dazu ist

$$\texttt{Großmutter}(x, z) :- \texttt{Mutter}(x, y), \texttt{Mutter}(y, z).$$

Die Relation $\texttt{Großmutter}(x, z)$ gilt also, wenn $\texttt{Mutter}(x, y)$ und $\texttt{Mutter}(y, z)$ für bestimmte Atome gelten. Die allgemeine Syntax ist $A_0 :- A_1, \ldots, A_n$, wobei dies als Implikation von rechts nach links zu lesen ist. Wenn also $A_1, \ldots, A_n$ gelten, so gilt auch A_0. Für $n = 0$ spricht man von Fakten. Eine Anfrage lässt sich nun leicht mit Variablen formulieren, z.B.

$$\texttt{Großmutter}(\texttt{Claudia}, z).$$

Für die Verteilung der Informationen wird Datalog nun erweitert. Wenn eine Relation $R(t_1, \ldots, t_n)$ (Information) auf einem Rechner t vorhanden ist, lautet die Syntax

$$t\$R(t_1, \ldots, t_n).$$

Zum Beispiel besagt $\texttt{chief}\$\texttt{IP}(\texttt{Hans}, \texttt{129.187.176.20})$, dass die Information „Hans hat die IP 129.187.176.20“ auf dem Rechner `chief` vorhanden ist.

Die beiden erwähnten Antwortarten lassen sich nun genauer definieren: Eine Antwort mit direkter Übergabe der Informationen entspricht der Übergabe von Fakten, man nennt diese Antwort eine *extensionale* Antwort. Die andere Art (als *intensionale* Antwort bezeichnet) entspricht der Übergabe von Regeln.

2.2 Beispiel: DNS

Mit Hilfe des Formalismus lassen sich auch bereits bestehende verteilte Datenquellen im Internet bequem beschreiben. Als Beispiel soll hier das allgegenwärtige DNS dienen, von dem aber nur eine Grundfunktion dargestellt wird. Ausführliche Informationen zu DNS finden sich in u.a. in [1].

Mit Hilfe von DNS werden Informationen im Internet über einzelne Rechner dezentral verwaltet. Neben anderen Arten von Einträgen ist einer der meist benutzten die Zuordnung von IP-Adressen zu Rechnernamen. Die Rechnernamen sind dabei als Baum organisiert, ausgehend von einem Wurzel-Eintrag über „top-level-domains“ wie `.org` oder `.de` zu „sub-domains“ wie `www.fs.tum.de` oder `drehscheibe.in.tum.de`. Ähnlich funktioniert auch die Verwaltung der Daten. Da kurz nach der Entstehung des Internet eine zentrale Verwaltung aufgrund der vielen andauernden Änderungen und der großen Datenmengen nicht mehr möglich war, ist die Verwaltung auf verschiedene Name-Server aufgeteilt. Jeder Server ist für die Zuordnung in einem Teil des Baums zuständig, wobei er Teile seines Bereiches weiter an andere, hierarchisch niedrigere Name-Server delegieren kann.

Ein Name-Server kann auf zwei Arten antworten, wenn er einen Teil der Zuordnung nicht kennt: Entweder schickt er die Anfrage rekursiv an andere Server weiter, bis einer die Antwort liefern und er diese zurückgeben kann; oder er liefert ohne Weiterfragen die bestmögliche Antwort, was meist die Adresse

$r\$A(\texttt{www.fs.tum.de}, \texttt{129.187.176.84})$:–
$r\$A(\texttt{www.tum.de}, \texttt{129.187.102.10})$:–
$r\$\texttt{NS}(\texttt{129.187.10.25})$:–
$r\$\texttt{DNS}(x, y)$:– $A(x, y)$
$r\$\texttt{DNS}(x, y)$:– $\texttt{NS}(z),\ z\$\texttt{DNS}(x, y)$

Abb. 1. DNS-Regelmenge in erweitertem Datalog.

eines anderen Name-Servers ist. Dazu ist es nötig, dass jeder Server wenigstens die Rechner kennt, die die Wurzel des Baumes verwalten.

Sowohl die Informationen und auch das Verfahren zur Behandlung von Anfragen lassen sich im erweiterten Datalog modellieren. Als Beispiel soll das kleines Programm (Menge von Regeln) des Rechners r in Abbildung 1 dienen. Anfragen an r lauten z.B. $\texttt{DNS}(\texttt{www.in.tum.de}, x)$. Die ersten beiden Fakten modellieren einen Teil des Datenbestandes, wobei die Relation A als IP-Adresszuordnung zu verstehen ist. Das dritte Fakt kennzeichnet einen weiteren Name-Server. Von den beiden Regeln modelliert die erste eine direkte Antwort aus dem Datenbestand; wenn zu der durch die Anfrage bereits belegten Variable x in der Regel ein Fakt gefunden wird, das zu der Relation A passt, wird die Variable y in der Regel entsprechend besetzt. Als Antwort dient dann nur der Kopf der Regel. Die zweite Regel entspricht der Auswertung unter Zuhilfenahme eines weiteren Rechners z. Wenn auf z eine zur Anfrage passende DNS-Relation existiert, wird diese auch von r akzeptiert. Die oben erwähnten zwei Antwortarten bei Nichtkenntnis der gesuchten Zuordnung entsprechen zwei unterschiedlichen Anwendungen der Regel: Entweder wird nur ein anderer bekannter Name-Server eingesetzt und mit dieser Regel geantwortet, z.B. $r\$\texttt{DNS}(\texttt{www.in.tum.de}, y)$:– $\texttt{129.187.10.25}\$\texttt{DNS}(\texttt{www.in.tum.de}, y)$, oder der andere Name-Server wird gefragt, die Antwort eingesetzt und damit geantwortet. Dies ist dann für den Anfragenden transparent zu einer Antwort durch die erste Regel.

3 Auswertung von Anfragen

3.1 Eine „korrekte“ Antwort

Nach der formalen Modellierung der Informationsverteilung stellt sich die Frage, welche Antwort auf eine Anfrage als „korrekt“ angesehen werden kann. Die Korrektheit lässt sich zerlegen in:

- *Stichhaltigkeit (soundness):* Die Antwort darf den Client nicht zur Ableitung falscher Schlüsse führen. Diese klar einleuchtende Tatsache lässt sich auch leicht formal definieren.
- *Vollständigkeit (completeness):* Die Definition dieses Punktes gestaltet sich schon schwieriger. Eine triviale vollständige Antwort ist natürlich das komplette Programm des Servers, d.h. eine Mischung aus extensionaler und intensionaler Antwort. Jedoch ist der Client kaum an *allen* Informationen des Servers interessiert.

Eine mögliche Intuition für die Vollständigkeit ergibt sich, wenn man den weiteren Verlauf der Anfragebearbeitung beim Client betrachtet. Diesem sollte es möglich sein, die gesamte gewünschte Information zu berechnen (durch logische Schlüsse und Anfragen an andere Server), ohne dem Server noch einmal die selbe Anfrage stellen zu müssen. Nach obigem Beispiel darf also nach der Antwort auf $\texttt{Großmutter}(\texttt{Claudia}, z)$ diese Anfrage an den selben Server nicht mehr nötig sein, natürlich aber eine andere wie z.B. $\texttt{Großmutter}(\texttt{Theresa}, z)$. Bei einer Definition der Vollständigkeit muss also auf jeden Fall auch der Wissensstand des Clients berücksichtigt werden.

Vor der Definition noch einige Notationen, wie sie auch in der Literatur (z.B. [4]) verwendet werden: Wenn $\mathcal{R}$ eine Menge von Regeln ist, steht $\text{Mod}(\mathcal{R})$ für die Menge aller Modelle über $\mathcal{R}$. $\mathcal{R}_1 \models \mathcal{R}_2$ gibt an, dass jedes Modell von $\mathcal{R}_1$ auch ein Modell von $\mathcal{R}_2$ ist. Für die konkrete Instanz eines Atoms q steht $\text{inst}(q)$, d.h. $\text{inst}(q)$ bezeichnet die Menge aller Fakten, die sich aus q bilden lassen, wenn man die Variablen durch Konstanten substituiert. Für eine Menge $\mathcal{Q}$ gibt $\text{inst}(\mathcal{Q})$ die Vereinigungsmenge aller Instanzen der einzelnen Elemente von $\mathcal{Q}$ an. Außerdem soll $\mathcal{R}_\mathcal{Q}$ alle Regeln bezeichnen, deren Kopf zu der Anfrage $\mathcal{Q}$ passt, formal

$$\mathcal{R}_\mathcal{Q} =_{\text{def}} \{a :\!- a_1, \ldots, a_n | (a :\!- a_1, \ldots, a_n) \in \mathcal{R}, \text{inst}(a) \cap \text{inst}(\mathcal{Q}) \neq \emptyset\}.$$

Weiterhin verwenden wir $\mathcal{R}[\mathcal{F}]$ für die Instantiierung der Regeln aus $\mathcal{R}$ mit Fakten aus der Menge $\mathcal{F}$, also

$$\mathcal{R}[\mathcal{F}] =_{\text{def}} \{\theta(a :\!- a_1, \ldots, a_n) | (a :\!- a_1, \ldots, a_n) \in \mathcal{R}, \theta(a), \theta(a_1), \ldots, \theta(a_n) \in \mathcal{F}\}$$

wobei θ eine beliebige Substitution ist.

Beispiel 1. Bei dem DNS-Beispiel-Programm in Abbildung 1 und einer Anfrage $\mathcal{Q} = \{\texttt{DNS}(\texttt{www.fs.tum.de}, \ y)\}$ wäre

$$\mathcal{R}_\mathcal{Q} = \{r\$\texttt{DNS}(x, y) :\!- A(x, y), r\$\texttt{DNS}(x, y) :\!- \texttt{NS}(z), \ z\$\texttt{DNS}(x, y)\}.$$

Bei einer Regelmenge $\mathcal{R} = \{DNS(x, y) :\!- A(x, y)\}$ und den beiden ersten Fakten des DNS-Beispiels als Menge $\mathcal{F}$, ergibt sich

$$\begin{aligned}
&\mathcal{R}[\mathcal{F}] \\
&= \{\ \texttt{DNS}(\texttt{www.fs.tum.de}, \texttt{129.187.176.84}) \\
&\qquad\qquad :\!- A(\texttt{www.fs.tum.de}, \texttt{129.187.176.84}), \\
&\quad \texttt{DNS}(\texttt{www.in.tum.de}, \texttt{129.187.172.10}) \\
&\qquad\qquad :\!- A(\texttt{www.in.tum.de}, \texttt{129.187.172.10})\ \}.
\end{aligned}$$

Der Teil der Regel rechts von „:–“ wird meist weggelassen, da er durch Einsetzen eines Fakts wahr gemacht wurde und gemäß der Interpretation der Regel als Implikation nicht mehr nötig ist.

Unter der Annahme einiger weiterer Voraussetzungen wie z.B. sicheren Regeln (für Details siehe [6]) lässt sich nun folgende Definition angeben.

Definition 2. *Sei $\mathcal{P}$ ein Programm in obiger Notation mit minimalem Modell $\mathcal{F}$. Für eine Antwort $\mathcal{A}$ auf eine Anfrage $\mathcal{Q}$ gilt:*

- *$\mathcal{A}$ ist stichhaltig, wenn $\mathcal{F} \in Mod(\mathcal{A})$,*
- *$\mathcal{A}$ ist vollständig, wenn für jede Menge $\mathcal{R}$ von Regeln gilt*

$$\mathcal{R} \cup \mathcal{P}[\mathcal{F}]_{\mathcal{Q}} \models \mathcal{F} \Rightarrow \mathcal{R} \cup \mathcal{A} \models \mathcal{F}.$$

Diese Definition greift die oben gegebenen Intuitionen auf. Das vorkommende $\mathcal{P}[\mathcal{F}]_{\mathcal{Q}}$ kann als kanonische korrekte Antwort angesehen werden. Wie berechnet man jedoch eine der möglichen Antworten? Zuvor jedoch noch einige einfache, später benötigte Folgerungen aus der Definition.

Theorem 3. *1. Ist $\mathcal{A} \subseteq \mathcal{P}$, so ist $\mathcal{A}$ stichhaltig.*
2. Wenn $\mathcal{A}_1$ und $\mathcal{A}_2$ stichhaltig sind, dann auch ist $\mathcal{A}_1 \cup \mathcal{A}_2$.
3. Ist $\mathcal{P}_{\mathcal{Q}} \subseteq \mathcal{A}$, so ist $\mathcal{A}$ vollständig für $\mathcal{Q}$.
4. Wenn $\mathcal{A}_1$ vollständig ist für $\mathcal{Q}_1$ und $\mathcal{A}_2$ für $\mathcal{Q}_2$, dann ist $\mathcal{A}_1 \cup \mathcal{A}_2$ vollständig für $\mathcal{Q}_1 \cup \mathcal{Q}_2$.

Die ersten beiden Behauptungen sind trivialerweise richtig, für den Beweis der beiden anderen sei auf [6] verwiesen.

3.2 Berechnung einer Antwort

Der folgende aus [6] entnommene Algorithmus berechnet eine gemäß der obigen Definition korrekte Antwort auf eine Anfrage. Der Algorithmus ist nicht deterministisch. Viele bereits existierende Systeme (vgl. auch das DNS-Beispiel vorne), die verteilte Daten ohne zentrale Instanz vorhalten, lassen sich auf einen speziellen Lauf des Algorithmus zurückführen. Dieser stellt also eine sehr abstrakte Sicht auf die Anfrageauswertung dar.

Der Algorithmus läuft in einzelnen Schritten und verbessert dabei jeweils die Antwort. Er benutzt zwei Mengen, einen Cache für Klauseln RC_i und eine Menge von bereits selbst durchgeführten Anfragen an weitere Server QS_i. Er startet im Schritt 0 mit $RC_0 = \mathcal{P}_s$ (das eigene Programmstück des Rechners) und $QS_0 = \emptyset$.

Eine Notation wird für die Definition benötigt: QS_i *überdeckt* ein Atom $a_1 = t\$(\dots)$, wenn t gleich dem aktuellen Rechner entspricht oder ein Atom a in QS_i existiert, so dass $a_1 = \theta(a)$ für eine Substitution θ gilt. Von Überdecken spricht man also, wenn die Regeln über a_1 entweder im eigenen Programm des Rechners enthalten sind oder durch eine Anfrage an den anderen Rechner t bereits geholt wurden.

Im Schritt $i+1$ wird zufällig eine der folgenden Aktionen durchgeführt:

1. *Auswerten:*
   Der Algorithmus wählt zufällig eine beliebige Regel $r = a_0 :- a_1, \dots, a_n$ und ein Fakt $a :-$ aus RC_i, so dass QS_i das Atom a_1 überdeckt und $a = \theta(a_1)$ gilt. Dann setzt man $r' = \theta(a_0) :- \theta(a_1), \dots, \theta(a_n)$ und außerdem $QS_{i+1} \leftarrow QS_i$ und $RC_{i+1} \leftarrow RC_i \cup \{r'\}$.

2. *Kommunizieren:*
   Der Algorithmus wählt zufällig eine beliebige Regel $r = a_0 :- a_1, \ldots, a_n$ aus RC_i, so dass QS_i das Atom a_1 *nicht* überdeckt. Wenn s_1 der Rechner in a_1 ist, dann schickt man die Anfrage a_1 an s_1 und wartet auf die Antwort $\mathcal{A}_1$. Man setzt $QS_{i+1} \leftarrow QS_i \cup \{a_1\}$ und $RC_{i+1} \leftarrow RC_i \cup \mathcal{A}_1$.
3. *Stoppen:*
   Der Algorithmus stoppt und gibt die Antwort RC_i zurück.

Theorem 4. *Unter der Annahme, dass andere Rechner korrekte Antworten liefern, ist die Antwort RC_i des Algorithmus stichhaltig und auch korrekt für $\{q\} \cup QS_i$.*

Beweis. Wir beweisen das Theorem mittels Induktion über i.

Für den Induktionsanfang: Es gilt $RC_0 = \mathcal{P}_s$ und $QS_0 = \emptyset$; $\mathcal{P}_s$ ist nach Satz 1(1) und (3) trivialerweise korrekt für die Anfrage $\{q\}$.

Für den Induktionsschritt: Es sind 2 Fälle zu unterscheiden: Wenn RC_{i+1} durch einen *Auswerte*-Schritt erzeugt wurde, gilt $RC_{i+1} = RC_i \cup \{r\}$. Da r durch logisches Schließen aus den Regeln in RC_i gebildet wurde, ist RC_{i+1} nach Satz 1(1) weiterhin stichhaltig und trivialerweise auch vollständig. Wurde RC_{i+1} durch einen *Kommunizieren*-Schritt gebildet, gilt $RC_{i+1} = RC_i \cup \mathcal{A}_1$ und $QS_{i+1} = QS_i \cup \{q'\}$. Wenn $\mathcal{A}_1$ korrekt für q' ist, dann folgt mit Satz 1(2) und (4), dass auch RC_{i+1} stichhaltig und vollständig für QS_{i+1} ist. □

Der Algorithmus ist also korrekt bzgl. der oben angegebenen Definition für Korrektheit. Aufgrund des Nichtdeterminismus des Ablaufs bietet er eine gute Grundlage für den Nachweis der Korrektheit deterministischer Verfahren, sofern sich diese auf einen speziellen Lauf des Algorithmus zurückführen lassen.

4 Optimierungen

Die theoretische Korrektheit eines Algorithmus sagt noch nichts darüber aus, ob er sich auch praktisch durchführen lässt. Daher werden wir noch einige Betrachtungen zur Optimierung anstellen und auch ein Verfahren vorstellen, dass den Kommunikationsaufwand zwischen den verteilten Datenquellen reduziert.

4.1 Allgemeine Optimierungen

Die Definition der Korrektheit stellt sicher, dass nach einer von einem Server gegebenen Antwort $\mathcal{A}$ auf eine Anfrage $\mathcal{Q}$ dem Server vom Client nicht noch einmal die Anfrage $\mathcal{Q}$ gestellt werden muss, um das globale Ergebnis zu berechnen. Jedoch ist nicht sichergestellt, dass eine Anfrage an einen Server mehrmals von *unterschiedlichen* Servern gestellt wird. Dies lässt sich leicht an einem Beispiel verdeutlichen.

Angenommen auf dem Server s ist ein Programm bestehend aus den beiden Regeln $s\$R(x, y) :- s\$T(x, z), z\$Q(y)$ und $s\$T(s_1, s_2) :-$ vorhanden. Wenn s nun eine Anfrage $s\$R(u, v)$ erhält, kann er z.B. einen Auswerte-Schritt in der

ersten Regel durchführen, was dann zu $\{s\$R(x,y) :- s_2\$Q(y)\}$ führt. Nehmen wir weiterhin an, ein Kommuniziere-Schritt an s_2 liefere die Antwort $s_2\$Q(y)$:- $s_5\$T(y,x)$. Wenn s nun diese Antwort (zusammen mit weiteren Regeln) an den Client zurückgibt, weiß dieser nicht, dass die Regel $s_2\$Q(y) :- s_5\$T(y,x)$ für die Anfrage $s_2\$Q(y)$ bereits vollständig ist. (Wir gehen ja von vollständigen Antworten aller Server aus, also ist auch die an s gegebene Antwort vollständig.) Der Client wird also die Anfrage $Q(y)$ noch einmal an den Rechner s_2 stellen, was den Berechnungsaufwand bei diesem verdoppelt.

Eine einfache Lösung hierfür ist, die Menge bereits durchgeführter Anfragen nicht lokal, sondern global zu verwalten. Dafür wird der Algorithmus folgendermaßen modifiziert: Bei der Alternative 3 (Rückgabe der Berechnung) wird zusätzlich zu RC_i auch die Menge mit den bereits durchgeführten Anfragen $QS_i \cup \{q\}$ zurückgegeben. Der Client weiß also, welche Anfragen er nicht mehr stellen muss. Außerdem wird Alternative 2 so modifiziert, dass der Rechner (der in diesem Fall die Rolle eines Clients hat) ein Tupel $(\mathcal{A}_1, \mathcal{Q}_1)$ erwartet, und seinen eigenen Anfragen-Cache durch $QS_{i+1} = QS_i \cup \mathcal{Q}_1$ auf den aktuellen Stand bringt. Hierdurch wird der Aufwand für die Berechnungen reduziert - aber natürlich der Kommunikationsaufwand erhöht.

Eine weitere Optimierung bei der Auswertung einer Anfrage nach obigem Algorithmus bietet das „Magic Set Rewriting“[3]. Durch die in Datalog erfolgende „bottom-up“ Auswertung der Relationen werden bei bestimmten Anfragen viel zu umfangreiche Zwischenergebnisse erzeugt, von denen aber nur ein kleiner Teil benötigt wird. Dies wird an folgendem Beispiel deutlich. Wenn ein Programm wie

$$\texttt{Vorfahre}(x,y) \text{ :- } \texttt{Elternteil}(x,y)$$
$$\texttt{Vorfahre}(x,y) \text{ :- } \texttt{Elternteil}(x,z),\ \texttt{Vorfahre}(z,y)$$

gegeben ist und eine Anfrage $\texttt{Vorfahre}(\texttt{Hans}, y)$ gestellt wird, wird erst die gesamte Relation `Vorfahre` berechnet und dann die zur Anfrage passenden Ergebnisse selektiert. Diesen Aufwand kann man durch Einführen von *magic sets* reduzieren. Dabei wird durch ein künstliches Aufblähen der Regeln (spezifisch für jede Anfrage) die bereits durch die Anfrage vorhandene Information in die Regel mit eingebracht und dient dabei zur Reduzierung des Aufwands bei der weiteren Auswertung. Im Beispiel oben würde die zweite Regel um die Regeln

$$\texttt{magic}(\texttt{Hans}) \text{ :-}$$
$$\texttt{Vorfahre}(x,y) \text{ :- } \texttt{magic}(x),\ \texttt{Elternteil}(x,z),\ \texttt{Vorfahre}(z,y)$$

erweitern. Dieses Verfahren lässt sich leicht auch in den nicht deterministischen Algorithmus aus Kapitel 3.2 einbauen.

4.2 Optimierungen bei semistrukturierten Daten

Ein weiterer wichtiger Punkt bei der Betrachtung der Effizienz eines Verfahrens zur Auswertung von Anfragen an verteilte Datenbanken ist der für die Kommunikation zwischen den Rechnern nötige Aufwand. Bei der heutzutage verfügbaren enormen Rechnerleistung tritt der Kommunikationsaufwand nicht

mehr als verschwindend klein in den Hintergrund sondern muss auch betrachtet werden. Besonders bei verbindungorientierten Übertragungsprotokollen (wie z.B. das im Internet viel verwendete HTTP) sind die Kosten für einen Aufbau der Verbindung relativ hoch. Wenn die Auswertung einer Anfrage an eine verteilte Datenbank viele Kommunikationsschritte benötigt, kann dies eine bestimmende Größe für den Aufwand werden.

Im Folgenden werden die wesentlichen Aspekte eines in [5] vorgestellten Verfahrens dargelegt, bei dem die nötigen Kommunikationsschritte bei einer Anfrageauswertung auf semistrukturierten Daten konstant sind und die Menge der übertragenen Daten polynomiell von der Größe der Antwort und dem Grad der Verlinkung der verteilten Daten abhängt.

Semistrukturierte Daten lassen sich am einfachsten als Graphen mit Kantenbeschriftungen modellieren. Die Informationen befinden sich dabei in den Knoten, die Beschriftungen auf den Kanten sind Verweise zwischen den Informationen. Ein gutes Beispiel für semistrukturierte Daten sind Webseiten mit Hyperlinks. Die Verteilung der Daten auf unterschiedliche Rechner ist dabei durch zwei verschiedene Arten von Kanten gegeben; Kanten zwischen Knoten, die auf verschiedenen Rechnern liegen und Kanten, die innerhalb eines Rechners verlaufen. Ohne Einschränkung kann man annehmen, dass der Teilgraph eines jeden Rechners eine Wurzel hat, von der aus alle Knoten des Rechners erreichbar sind. Auch wenn der Graph auf einem Rechner meist eine baumartige Struktur hat, muss dies nicht immer gegeben sein.

Ab jetzt nehmen wir an, dass die für die Auswertung einer Suche relevanten Informationen durch die Kantenbeschriftungen gegeben sind; wir vernachlässigen also die Knoten, was aber keine Einschränkung darstellt. Demnach lässt sich eine Suche durch einen regulären Ausdruck angeben, der die Abfolge der Kantenbeschriftungen von der Wurzel bis zu einem gesuchten Knoten beschreibt. Ein intuitiv verständliches Beispiel dafür ist

$$* \Longrightarrow \textit{Lehrstuhl Mayr} \Longrightarrow * \Longrightarrow \textit{Technische Berichte},$$

welches von einem zentralen Knoten ausgehend zu allen Pfaden passt, die zuerst den Bezeichner „Lehrstuhl Mayr“ und dann den Bezeichner „Technische Berichte“ haben, mit beliebigen Bezeichnern davor und dazwischen. In einer SQL-ähnlichen Schreibweise würde man dies schreiben als

$$\textsf{select } t \\ \textsf{where } * \Longrightarrow \textit{Lehrstuhl Mayr} \Longrightarrow * \Longrightarrow \textit{Technische Berichte} \Longrightarrow t \textsf{ in } db.$$

Der reguläre Ausdruck zur Suche lässt sich leicht in einen Automaten umdeuten, der entsprechend den bereits passenden Bezeichnern beim Durchlaufen des Baums seinen Zustand ändert. Ein einfacher Algorithmus zur Bearbeitung einer Suche wird nun ausgehend von einer Wurzel die Kanten des Graphen ablaufen und dabei jeweils passend den Zustand des Automaten weiterschalten, d.h. sich die Position im regulären Ausdruck merken. Indem man jeden Knoten, den man in einem akzeptierenden Zustand des Automaten erreicht, einer Menge hinzufügt, kann man sukzessive eine Ergebnismenge aufbauen.

Der wesentliche Teil des Verfahrens sieht demnach grob wiedergegeben folgendermaßen aus. Dabei bezeichne s einen Zustand des Automaten, u einen Knoten und a einen Kantenbezeichner. Mit P wird eine Automatenweiterschaltung beschrieben. Die Funktion Visit wird von einem Startknoten aus gestartet.

```
function Visit(s, u)
    if (s, u) ∈ Visited then return Result[s, u]
    Visited ← Visited ∪ {(s, u)}
    if s ist akzeptierender Zustand des Automaten then Result[s, u] ← {u}
    forall u -a-> v do // Kante im Graphen
        forall s -P-> s' do // Automatenweiterschaltung
            if P(a) then Result[s, u] = Result[s, u] ∪ Visit(v, s')
    return Result[s, u]
```

Bei der Ausführung des Verfahrens wird die Suche quasi jedesmal von Rechner zu Rechner weitergegeben, wenn man auf eine Kante trifft, die Teilgraphen auf 2 Rechnern verbindet. Der Kommunikationsaufwand ist also proportional zu den vorhandenen Kanten zwischen den Rechnern. Dies kann sich wesentlich in der Ausführungszeit niederschlagen.

Eine wesentliche Verbesserung erhält man durch eine Modifizierung des globalen Ablaufs. Die Suche wird dazu nicht mehr von Rechner zu Rechner weitergereicht, sondern auf jedem Rechner wird lokal ein Teil (vor-)berechnet und dann schrittweise durch einige wenige Kommunikationsschritte mit dem Client das globale Resultat erzeugt.

Dazu ist es nötig, dass alle Rechner zwei Teilmengen ihrer Knoten vorberechnen; die eine beinhaltet alle Knoten, von denen Kanten zu anderen Rechnern führen (leicht zu bestimmen), die andere alle lokalen Knoten, die von Kanten anderer Rechner erreicht werden (mit etwas Aufwand vorzuberechnen). Nun wird auf jedem Rechner ein lokales Ergebnis berechnet, indem man von jedem möglichen Eingangsknoten aus in jedem möglichen Zustand des Automaten startet und die Berechnung dabei nicht den Rechner verlassen darf. Die obige Funktion *visit* wird dazu lediglich leicht modifiziert.

Um große, nicht zugängliche Teile des Graphen (d.h. nicht zur Query passende Knoten) auszuschließen, berechnet jeder Rechner einen Erreichbarkeitsgraph aus seiner lokalen, oben beschriebenen Lösung. Dieser Graph wird von allen Rechnern an den Client gesendet, der eine Koordinierungsaufgabe übernimmt und den globalen Erreichbarkeitsgraph berechnet. Dieser wird wiederum an die Clients gesendet, die damit den zugänglichen Teil ihrer lokalen Lösung (also das, was zur Suche passt) berechnen können und an den Client schicken. Dieser ermittelt nun schlussendlich die globale Lösung.

Es ist leicht zu sehen, dass bei dieser Art der Auswertung die Anzahl der Kommunikationsschritte konstant 4 ist. Ausserdem lässt sich zeigen, das die Menge der übertragenen Daten die Größe $O(n^2) + O(r)$ hat, wobei n die Anzahl der Kanten im Graphen ist, die verschiedene Rechner verbinden, und r die Größe der Antwort auf die Anfrage.

Literatur

1. P. Albitz, C. Liu. *DNS and BIND.* 2. Auflage. O'Reilly & Associates, 1997.
2. F. Belli. *Einführung in die logische Programmierung mit PROLOG.* 2. überarbeitete und erweiterte Auflage. BI-Wissenschaftsverlag, 1988.
3. C. Beeri, R. Ramakrishnan. On the power of magic. In: *Proceedings 6th ACM Symposium on Principles of Database Systems (PODS'87)*, S. 269-283. ACM Press, 1987.
4. Wolfgang Rautenberg. *Einführung in die mathematische Logik.* Vieweg-Verlag, 1996.
5. D. Suciu. Distributed query evaluation on semistructured data. *ACM Transactions on Database Systems*, 27(1):1-62, 2002.
6. T. Jim, D. Suciu. Dynamically distributed query evaluation. In: *Proceedings 20th ACM Symposium on Principles of Database Systems (PODS'2001)*, S. 28-39. ACM Press, 2001.

Automatische Datengenerierung aus HTML-Dokumenten

Matthias Hanitzsch

Technische Universität München
hanitzsc@in.tum.de

Zusammenfassung. Die Menge der im Internet verfügbaren Informationen wird immer größer. Damit steigt auch der Bedarf an Werkzeugen, um Daten aus diesen Informationen zu generieren. Wrapper sind Funktionen, die diese Aufgabe übernehmen. Sie durchsuchen Internetseiten und filtern relevante Daten aus diesen heraus. Interessant ist vor allem die maschinelle Erstellung solcher Wrapper. Dies soll an Hand von Beispielseiten, in denen die relevanten Daten markiert sind, erfolgen. Hier werden insgesamt 5 Arten von Wrappern vorgestellt. Dabei wird untersucht, wieviele Beispiele zum Lernen notwendig sind, wieviele Internetseiten sich damit in der Praxis verarbeiten lassen und wie das Laufzeitverhalten der einzelnen Algorithmen ist.

1 Einführung

Im Internet steht eine Vielzahl von Informationen zur Verfügung: Telefonverzeichnisse, Kataloge, Wettervorhersagen, Veranstaltungskalender und vieles mehr. Viele dieser Informationen liegen in HTML (Hyper Text Markup Language) vor. HTML ist eine Auszeichnungssprache, die es ermöglicht, die Struktur von Texten (Überschriften, Absätze, Tabellen usw.) zu beschreiben. Weiterhin lassen sich Bilder, Klänge und ähnliche Dinge in Seiten einbinden. Eines der wichtigsten Merkmale von HTML ist die Möglichkeit von Querverweisen zwischen Seiten. HTML-Seiten sind semistrukturiert und die maschinelle Datengenerierung aus diesen ist deshalb nicht immer einfach. Eine Möglichkeit der Gewinnung von Daten sind sogenannte Wrapper. Das sind Funktionen, die ein HTML-Dokument als Eingabe erhalten und relevante Daten aus diesem extrahieren, ohne jedoch, wie etwa beim Data Mining, nach Beziehungen zwischen Daten zu suchen oder Schlüsse daraus abzuleiten.

Solche Wrapper manuell zu erstellen, birgt aber verschiedene Probleme. Zum einen ist es notwendig, die innere Struktur des HTML-Dokuments genau zu kennen. Zum anderen kann sich die betreffende Internet-Seite recht schnell ändern. Schon kleinste Unterschiede im Layout können bewirken, dass der Wrapper seine Aufgabe nicht mehr erfüllen kann. Er muss also wieder angepasst werden. Das kann mitunter einen recht hohen Zeitaufwand für Entwicklung, Test und Pflege eines solchen Wrappers bedeuten und damit sind manuell erstellte Wrapper in vielen Fällen nicht sehr praktikabel.

Abb. 1. Eine fiktive Internetseite mit Ländern und ihrer Landesvorwahl.

Eine Alternative dazu ist das maschinelle Lernen von Wrappern. Dazu werden Techniken entwickelt, die es ermöglichen an Hand von Beispieldokumenten, in denen die relevanten Daten markiert sind, Wrapper automatisch zu erstellen. Kushmerick [6] beschreibt sechs Klassen von Wrappern, insbesondere die Algorithmen, um diese maschinell zu lernen. Wir wollen vier dieser Klassen untersuchen. Dabei interessiert uns vor allem, wieviele Internetseiten damit in der Praxis verarbeitet werden können, wie aufwendig die Markierung der Beispielseiten und der Vorgang des Lernens an sich ist.

In Abschnitt 2 wird zunächst das Problem der Wrappergenerierung untersucht. In Abschnitt 3 wird dann die Wrapper-Klasse LR zusammen mit den erforderlichen Techniken beschrieben. In den Abschnitten 4 bis 5 werden drei andere Klassen eingeführt, die im Wesentlichen eine Erweiterung der ersten sind. In Abschnitt 6 wird die Qualität der Klassen und der zum Lernen notwendige Aufwand untersucht. Ein etwas anderer Ansatz, der Verbesserungen bringt, wird in Abschnitt 7 beschrieben. Den Abschluss bildet Abschnitt 8, in dem die Erkenntnisse zusammengefasst werden.

2 Das Problem der Wrappergenerierung

Wir wollen zuerst klären, was es bedeutet einen Wrapper maschinell zu lernen. Um das Verständnis zu erleichtern, soll die fiktive Internetseite in Abb. 1 dienen. Die Seite stellt eine Liste von Ländern zusammen mit ihrer Landesvorwahl dar, daneben befindet sich der entsprechende HTML-Code.

Ausgehend von einem relationalem Datenmodell wird jeder Informationsquelle eine Menge von K Attributen zugeordnet. Jedes *Attribut* entspricht einer Spalte in diesem relationalen Modell. Im obigen Beispiel gibt es also $K = 2$ Attribute.

Ein *Tupel* ist ein Vektor $\langle a_1, \ldots, a_K \rangle$ von K Strings. Der String a_j ist dabei der Wert des $j$ten Attributs im Tupel. Die Menge aller Tupel einer Seite wird als *Inhalt* bezeichnet. Zur Repräsentation des Inhalts dienen sogenannte *Label*. Diese stellen den Inhalt in Form von Indizes dar. Für unser Beispiel erhalten wir also das folgendes Label

$$L_{CC} = \left\{ \begin{array}{c} \langle < 50, 55 >, < 63, 66 > \rangle, \\ \langle < 78, 83 >, < 91, 93 > \rangle, \\ \langle < 105, 111 >, < 119, 122 > \rangle, \\ \langle < 134, 139 >, < 147, 149 > \rangle \end{array} \right\}.$$

Die erste Zeile enthält die Indizes für *Congo* und 242.

Die allgemeine Form eines Labels ist

$$L = \left\{ \begin{array}{c} \langle < b_{1,1}, e_{1,1} >, \ldots, < b_{1,k}, e_{1,k} >, \ldots, < b_{1,K}, e_{1,K} > \rangle, \\ \vdots \\ \langle < b_{m,1}, e_{m,1} >, \ldots, < b_{m,k}, e_{m,k} >, \ldots, < b_{m,K}, e_{m,K} > \rangle, \\ \vdots \\ \langle < b_{|L|,1}, e_{|L|,1} >, \ldots, < b_{|L|,k}, e_{|L|,k} >, \ldots, < b_{|L|,K}, e_{|L|,K} > \rangle \end{array} \right\}.$$

Die Seite enthält $|L| > 0$ Tupel, jedes von diesen hat $K > 0$ Attribute. Die Zahlen $1 \leq k \leq K$ sind die Indizes der Attribute, und die Zahlen $1 \leq m \leq |L|$ sind die Indizes der Tupel. Jedes Paar $\langle b_{m.k}, e_{m.k} \rangle$ stellt den Start- und Endindex des $k$ten Attributs im Tupel m dar.

Ein Wrapper sei nun eine Funktion W, die zu einer Seite P das Label L berechnet: $W(P) = L$. Damit besteht das Problem der Wrappergenerierung darin, zu einer gegebenen Menge von Bespielseiten einen Wrapper W zu erzeugen, der alle diese Beispielseiten verarbeiten kann:

EINGABE: Menge $\mathcal{E} = \{\ldots, < P_n, L_n >, \ldots\}$ von Beispielen, wobei P_n eine Seite und L_n das entsprechende Label ist.

AUSGABE: Wrapper W $\in \mathcal{W}$, so dass $\mathrm{W}(P_n) = L_n$ für alle Beispiele $< P_n, L_n >$ aus der Beispielmenge $\mathcal{E}$. $\mathcal{W}$ bezeichnet dabei eine Wrapper-Klasse.

3 Die Wrapper-Klasse LR

Grundlegende Idee bei der Wrapper-Klasse LR (Left Right) ist, dass zu jedem Attribut k ein linker Begrenzer l_k und ein rechter Begrenzer r_k existiert. Die Prozedur Exec_{LR} zeigt die Arbeitsweise eines LR-Wrappers:

```
procedure Exec_LR (wrapper ⟨l_1, r_1, ..., l_K, r_K⟩, page P)
  m ← 0
  while there are more occurences of l_1 in P
    m ← m + 1
    for each ⟨l_k, r_k⟩ ∈ {⟨l_1, r_1⟩, ..., ⟨l_K, r_K⟩}
      scan in P to next occurence of l_k; save position as b_{m,k}
      scan in P to next occurence of r_k; save position as e_{m,k}
  return label {..., ⟨⟨b_{m,1}, e_{m,1}⟩, ..., ⟨b_{m,K}, e_{m,K}⟩⟩, ...}
```

Ein LR-Wrapper ist also ein Vektor $\langle l_1, r_1, \ldots, l_K, r_K \rangle$ von $2K$ Strings, den Begrenzern. Die Prozedur Exec_{LR} durchsucht die Seite P nach diesen Begrenzern, ermittelt so die Indizes der Attribute in den Tupeln und gibt schließlich das entsprechende Label zurück. Zur Generierung eines LR-Wrappers muss also ein passender Vektor von Begrenzern l_k,r_k gefunden werden.

Dazu wird aus der Menge der möglichen Kandidaten ein Begrenzer ausgewählt und auf seine Gültigkeit überprüft. Die Menge $\text{Cands}_l(k, \mathcal{E})$ ist dabei die Menge der Kandidaten für den Begrenzer l_k unter der Beispielmenge $\mathcal{E}$. Diese Menge besteht aus allen Suffixen des kürzesten Strings links des Attributs k in allen Beispielen aus $\mathcal{E}$. Entsprechendes gilt für die Menge $\text{Cands}_r(k, \mathcal{E})$. Sie besteht aus allen Präfixen des kürzesten Strings rechts des Attributs k in allen Beispielen aus $\mathcal{E}$.

Damit ein Kandidat u für einen Begrenzer r_k gültig ist, muss er die folgenden Bedingungen erfüllen:

1. *Bedingung* $\mathcal{C}_r^A$*:* Der String u darf kein Teilstring irgendeines Attributs k in irgendeinem Beispiel sein.
2. *Bedingung* $\mathcal{C}_r^B$*:* Der String u muss ein Präfix des Texts sein, der unmittelbar nach dem Attribut k in jedem Beispiel folgt.

Ist eine dieser Bedingungen verletzt, wird jeder Wrapper, der den Begrenzer $r_k = u$ enthält, zumindest bei einem der Beispiele scheitern. Wird die Bedingung $\mathcal{C}_r^A$ verletzt, dann ist das Attribut k zu kurz. Ist $\mathcal{C}_r^B$ verletzt, dann ist es zu lang.

Ein Kandidat u für einen Begrenzer l_k muss die folgenden Bedingungen erfüllen:

1. *Bedingung* $\mathcal{C}_l^A$*:* Der String u muss ein geeignetes Suffix des Texts sein, der unmittelbar vor jedem Attribut k in jedem Beispiel enthalten ist.
2. *Bedingung* $\mathcal{C}_l^B$*:* Für l_1 darf u kein Teilstring des Seitenrests (nach dem letzten Tupel) in irgendeinem Beispiel sein.

Enthält ein Wrapper den Begrenzer $l_k = u$, so wird er bei zumindest einem Beispiel scheitern. Bei Verletzung der Bedingung $\mathcal{C}_l^A$ wird zumindest einer der Startindizes $b_{m,k}$, die Exec_{LR} berechnet, falsch sein. Ist $\mathcal{C}_l^B$ nicht erfüllt, so wird Exec_{LR} versuchen, zu viele Attribute zu extrahieren.

Wir können jetzt die Prozedur Learn_{LR} zur Generierung eines LR-Wrappers angeben:

```
procedure Learn_LR (examples E)
   for each 1 ≤ k ≤ K
      for each u ∈ Cands_l(k, E):
         if Valid_l(u, k, E) then l_k ← u and terminate this loop
   for each 1 ≤ k ≤ K
      for each u ∈ Cands_r(k, E):
         if Valid_r(u, k, E) then r_k ← u and terminate this loop
 return LR wrapper ⟨l_1, r_1, ..., l_K, r_K⟩
```

Die Prozedur $\text{Valid}_l(u, k, \mathcal{E})$ überprüft hierbei die Gültigkeit der oben angebebenen Bedingungen für einen Begrenzer l_k, und die Prozedur $\text{Valid}_r(u, k, \mathcal{E})$ überprüft diese für einen Begrenzer r_k.

4 Die Wrapper-Klasse HLRT

Wie wir an der Prozedur Exec_{LR} sehen, durchsucht ein LR-Wrapper immer das gesamte Dokument nach Begrenzern. Das erfordert, dass für jedes Attribut k Begrenzer existieren, die im gesamten Dokument eindeutig sind. In unserem Beispiel aus Abb. 1 sind die Landesnamen fettgedruckt dargestellt. Würde jetzt noch mehr fettgedruckter Text, etwa eine Überschrift, hinzukommen, wäre es nicht mehr möglich, die Seite mit einem LR-Wrapper zu verarbeiten. Es gibt keinen l_1 Begrenzer, der zwischen fettgedruckten Landesnamen und anderem fettgedrucktem Text unterscheiden kann.

Die Klasse HLRT (Head Left Right Tail) benutzt zwei weitere Begrenzer, um den Bereich, innerhalb dessen nach Attributen gesucht werden soll, eingrenzen zu können. Der *head*-Begrenzer zeigt den Anfang des Suchbereichs und der *tail*-Begrenzer dessen Ende an. Ein HLRT-Wrapper ist also ein Vektor $\langle h, t, l_1, r_1, \ldots, l_K, r_K \rangle$ von $2K + 2$ Strings.

Die Arbeitsweise eines HLRT-Wrappers, nachfolgend dargestellt durch die Prozedur Exec_{HLRT}, ist ähnlich zu der eines Wrappers der Klasse LR. Hier wird aber zuerst nach dem Begrenzer h gesucht. Erst an dieser Stelle beginnt die Extraktion von Attributen. Abgebrochen wird, wenn der Begrenzer t vor dem nächsten l_1 gefunden ist.

```
procedure Exec_HLRT (wrapper ⟨h, t, l_1, r_1, ..., l_K, r_K⟩, page P)
    m ← 0
    scan in P to next occurence of h
    while the next l_1 in P is before the next t
      m ← m + 1
      for each ⟨l_k, r_k⟩ ∈ {⟨l_1, r_1⟩, ..., ⟨l_K, r_K⟩}
        scan in P to next occurence of l_k; save position as b_{m,k}
        scan in P to next occurence of r_k; save position as e_{m,k}
    return label {..., ⟨⟨b_{m,1}, e_{m,1}⟩, ..., ⟨b_{m,K}, e_{m,K}⟩⟩, ...}
```

Bei der Generierung eines LR-Wrappers konnten die Begrenzer unabhängig voneinander betrachtet werden. Nun beeinflussen sich aber die Begrenzer h, t und l_1. Deshalb ergeben sich in der Prozedur Learn_{HLRT} die drei ineinandergeschachtelten for-Loops. Beim Lernen der anderen Begrenzer kann allerdings auf Learn_{LR} zurückgegriffen werden.

```
procedure Learn_HLRT (examples E)
   ⟨·, r_1, ..., l_K, r_K⟩ ← Learn_LR(E)
   for each u_{l_1} ∈ Cands_l(1, E)
     for each u_h ∈ Cands_h(E)
       for each u_t ∈ Cands_t(E)
         if Valid_{l_1,h,t}(u_{l_1}, u_h, u_t, E) then
           l_1 ← u_{l_1}, h ← u_h, t ← u_t, and terminate all loops
   return HLRT wrapper ⟨h, t, l_1, r_1, ..., l_K, r_K⟩
```

5 Weitere Wrapper-Klassen

Der Vorteil der Klasse HLRT gegenüber LR ist, dass nur noch in einem bestimmten Bereich nach Attributen gesucht wurde. Es gibt noch zwei weitere Klassen von Wrappern, die einen ganz ähnlichen Ansatz verwenden.

Ein OCLR-Wrapper (Open Close Left Right) ist ein Vektor $\langle o, c, l_1, r_1, \ldots, l_K, r_K \rangle$ von $2K+2$ Strings, der wie HLRT zwei weitere Begrenzer benutzt. Mit dem *open*-Begrenzer wird hier der Beginn eines Tupels, und mit dem *close*-Begrenzer dessen Ende angezeigt. Die Prozeduren Exec_{OCLR} und Learn_{OCLR} funktionieren in gleicher Weise wie bei den anderen beiden Klassen.

Die Wrapper-Klasse HOCLRT (Head Open Close Left Right Tail) ist sozusagen die logische Fortsetzung der drei bisherigen Wrapper. Ein HOCLRT-Wrapper ist ein Vektor $\langle h, t, o, c, l_1, r_1, \ldots, l_K, r_K \rangle$ von $2K+4$ Strings und kombiniert die Funktionalität von HLRT und OCLR. Er benutzt *head*- und *tail*-Begrenzer, um den Suchbereich, sowie *open*- und *close*-Begrenzer, um die Tupel einzugrenzen.

6 Bewertung der Wrapper-Klassen

In diesem Abschnitt soll untersucht werden, wie gut die vorgestellten Wrapper-Klassen auf Internetseiten in der Praxis anwendbar sind. Da die Markierung von Beispielseiten einen nicht unerheblichen Arbeitsaufwand bedeuten kann, ist eine andere wichtige Frage, wieviele Beispielseiten im Durchschnitt nötig sind, um einen dieser Wrapper zu lernen. Schliesslich werden wir auch noch die Laufzeit der Algorithmen betrachten.

Um die eben genannten Fragen zu klären, wurden recht umfangreiche Tests durchgeführt. Dazu wurden 448 Internetseiten, die bei www.search.com gelistet waren, herangezogen. Von diesen wurden wiederum 30 Seiten per Zufall ausgewählt. Als nächstes wurden für jede dieser Seiten die Ergebnisse von 10 Anfragen gesammelt und die darin relevanten Daten von Hand markiert. Die Anzahl der Attribute K je Tupel reichte von 2 bis 18.

Nun wurde untersucht, wieviele der 30 Internetseiten von den einzelnen Wrapper-Klassen verarbeitet werden konnten. Dabei wurde folgendes Ergebnis erreicht: LR 16 (53%), HLRT 17 (57%), OCLR 16 (53%), HOCLRT 17 (57%). Insgesamt konnten 18 (60%) der 30 untersuchten Seiten mit den Wrapper-Klassen verarbeitet werden.

Die restlichen 40% der Seiten haben verschiedene Eigenschaften, die eine Verarbeitung unmöglich machten. So stellen zum Beispiel fehlende Attribute ein Problem dar, oder Attribute in unterschiedlicher Reihenfolge. Weiterhin erfordern die Wrapper, dass Attribute eindeutige Begrenzer haben. Auch das war nicht immer der Fall.

Als nächstes wurde die Frage geklärt, wieviele Beispielseiten zum Lernen eines Wrappers nötig sind. In diesem Zusammenhang soll das PAC-Modell (von *probably approximately correct*) vorgestellt werden, das im Bereich des maschinellen Lernens eine wichtige Rolle spielt. Mit diesem Modell wird es möglich,

eine obere Schranke für die notwendige Anzahl von Beispielseiten anzugeben. Wir gehen dabei von einem Zielwrapper $W_T \in \mathcal{W}$ aus. Dieser berechnet für alle Seiten aus der Beispielmenge das entsprechende Label. Ziel ist es nun, einen Wrapper $W \in \mathcal{W}$ zu finden, der W_T so nahe wie möglich kommt. Im besten Fall sollen W und W_T identisch sein.

Als Kriterium dafür dient die Funktion Error. Sie gibt die Wahrscheinlichkeit dafür an, dass der Wrapper W zu einer Seite P ein anderes Label L berechnet als der Zielwrapper W_T:

$$\text{Error}(W) = \text{Prob}[W(P) \neq W_T(P)].$$

Je näher Error(W) an 0 ist, desto näher ist der Wrapper W am Zielwrapper W_T. Wir geben nun einen Parameter $0 < \epsilon < 1$ an und wollen sicherstellen, dass Error(W) $< \epsilon$ gilt, egal wie klein ϵ gewählt wird. Natürlich können wir das nicht garantieren. Die Beispielseiten zum Lernen des Wrappers sind alle zufällig gewählt und können mitunter auch irreführend sein. Auch hier können wir nur mit einer bestimmten Wahrscheinlichkeit sagen, dass die Bedingung Error(W) $< \epsilon$ hält. Dazu wird ein weiterer Parameter $0 < \delta < 1$ eingeführt. Nun wird also verlangt, dass bei gegebenem ϵ und δ der Lernalgorithmus mit der Wahrscheinlichkeit $1 - \delta$ einen Wrapper erzeugt, so dass Error(W) $< \epsilon$ gilt.

Mit diesem Modell lässt sich herleiten, dass die Anzahl der Beispiele der folgenden Ungleichung genügen muss:

$$|\mathcal{E}| > \frac{1}{\epsilon}(\ln|\mathcal{W}| - \ln\delta).$$

Damit ergeben sich bei den einzelnen Wrapper-Klassen für die Anzahl der zum Lernen notwendigen Beispielseiten obere Schranken von einigen Tausend Seiten (abhängig von ϵ und δ). Das ist für die Praxis noch nicht sehr hilfreich und in [5] finden sich Ansätze zur weiteren Verbesserung des Modells. Dennoch ist das PAC-Modell ein wichtiges Hilfsmittel, um die Güte eines Lernalgorithmus zu beurteilen.

In Tests mit den 30 zufällig gewählten Internetseiten hat sich gezeigt, dass im Durchschnitt etwa 2-5 Beispielseiten notwendig sind, um einen Wrapper zu lernen. Das ist durchaus ein zufriedenstellendes Ergebnis. Wären 50, 100 oder mehr Beispiele zum Lernen nötig, dann hätten diese Wrapper kaum mehr praktische Bedeutung. So können aber schon mit wenigen Beispielseiten Wrapper erzeugt werden.

Natürlich ist auch die Laufzeit der Lernalgorithmen interessant. Dazu wurden folgenden Abschätzungen für deren Komplexität ermittelt:

LR :	$O(KM^2\|\mathcal{E}\|^2V^2)$
HLRT :	$O(KM^2\|\mathcal{E}\|^4V^6)$
OCLR :	$O(KM^4\|\mathcal{E}\|^2V^6)$
HOCLRT :	$O(KM^4\|\mathcal{E}\|^4V^{10})$

mit $V = \max_{1 \leq i \leq n} |P_i|$ maximale Länge der Beispielseite, $M = \sum_{i=1}^{n} |L_i|$ Gesamtzahl der Tupel in den Beispielen, jedes Tupel K Attribute.

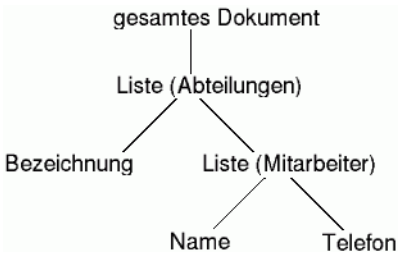


Abb. 2. Struktur einer Seite, die Abteilungen und ihre Mitarbeiter auflistet.

Die Wrapper können also in polynomieller Zeit gelernt werden. Die teilweise recht hohen Grade lassen relativ lange Ausführungszeiten erwarten. Auch die Parameter M und V sind normalerweise recht groß (V z.B. zwischen 899 und 57116). Deshalb verläuft das Lernen in einigen Fällen tatsächlich auch langsam (länger als 15 min). In anderen Fällen wurden aber auch Ausführungszeiten von unter einer Sekunde gemessen. Der Grund dafür ist, dass üblicherweise mehrere gültige Wrapper für eine Seite existieren. Wie schnell ein Wrapper gefunden wird, hängt auch davon ab, in welcher Reihenfolge mögliche Kandidaten für die Begrenzer getestet werden. Dabei ist es für sehr kurze oder sehr lange Kandidaten weniger wahrscheinlich, gültig zu sein. Hier ist es sinnvoll, die Kandidaten so zu ordnen, dass zuerst durchschnittlich lange Kandidaten getestet werden.

7 Ein anderer Ansatz: Stalker

Hier sollen knapp die grundlegenden Ideen von Stalker, einem weiteren Algorithmus zur Wrappergenerierung, beschrieben werden. Eine ausführliche Beschreibung findet sich in [8].

Internetseiten haben oft eine bestimmte hierarchische Struktur: Informationen sind in Form von Listen von Tupeln dargestellt. Diesen Umstand macht sich Stalker zu nutze und arbeitet intern mit einer baumartigen Darstellung der Struktur einer Seite. In dieser Baumstrukur befinden sich relevante Daten in den Blättern. Die inneren Knoten stellen Listen von k-Tupeln dar. Jedes Element eines solchen Tupels kann entweder ein Blatt oder wieder eine Liste sein. In Abb. 6 ist ein Beispiel eines solchen Baumes dargestellt.

Um nun Daten aus einem Dokument extrahieren zu können, benutzt Stalker die Baumstruktur der Seite und eine Menge von Regeln. Für jeden Knoten im Baum benötigt der Wrapper eine Regel, um ihn aus seinem Elternknoten extrahieren zu können. Stellt ein Knoten eine Liste dar, ist weiterhin eine Regel notwendig, mit der sich die Liste in ihre Tupel zerlegen lässt. Für einen gegebenen Strukturbaum und eine Menge von Regeln, besteht die Gewinnung von Daten darin, dem Pfad P von der Wurzel zum jeweiligen Knoten zu folgen und in jedem Schritt die entsprechenden Regeln anzuwenden. Diese Regeln haben die Form SkipTo(...) bzw. SkipUntil(...). Der Grundgedanke ist also, nach

Teilstrings (sogenannten *landmarks*) zu suchen, die eindeutig den Beginn eines Knotens innerhalb seines Elternknotens festlegen.

Zum Lernen der Regeln werden wie bei den anderen Wrappern Beispielseiten benutzt, in denen die relevanten Daten markiert sind. Angenommen wir haben die folgenden vier Beispiele und wollen Regeln lernen, um daraus die Vorwahlnummern zu extrahieren:

E1= Venice, Phone:1-800-555-1515,
E2= Palms, Phone:(818)-508-1570,
E3= LA, Phone:1-888-578-2293,
E4= Watts, Phone:(310) 798-0008.

Der Algorithmus Stalker geht dann so vor, dass er zuerst das kürzeste Beispiel (E2) betrachtet. Das letzte Zeichen vor der Vorwahl ist „(“und es gibt 2 Wildcards dafür: *Punctuation* und *Anything*. Somit erzeugt Stalker die 3 Kandidaten

$R1$ = SkipTo((),
$R2$ = SkipTo(*Punctuation*),
$R3$ = SkipTo(*Anything*).

Stalker benutzt nun eine Reihe von Kriterien, um aus verschiedenen Regeln eine auszuwählen. Hier wird die Regel $R1$ gewählt, da sie für die Beispiele E2 und E4 den Beginn der Vorwahl findet, jedoch für E1 und E3 nicht stoppt. Verbleiben also die Beispiele E1 und E3. Wieder wird nach gleichem Schema eine Regel gesucht. Am Ende liefert der Algorithmus die zwei Regeln

$R1$ = SkipTo((),
$R4$ = SkipTo(–).

Um die Vorwahl in den Beispielen zu finden wird nun entweder $R1$ oder $R4$ angewendet. Die Regeln für das Ende eines Attributs werden in gleicher Weise ermittelt.

Bei Tests hat sich gezeigt, dass Stalker mehr Internetseiten verarbeiten kann als die anderen vorgestellten Wrapper-Klassen. Stalker kann auch mit Seiten umgehen, in denen Attribute fehlen oder in unterschiedlicher Reihenfolge auftauchen. Wie in dem obigen Beispiel gezeigt, ist es auch möglich mehrere (durch *oder* verknüpfte) Regeln für einen Knoten zu haben. Dadurch entfällt die Forderung nach eindeutigen Begrenzern (hier *landmarks*) vor bzw. nach Attributen. Stalker wird damit um einiges flexibler und somit mächtiger.

8 Zusammenfassung

Mit der immer größer werdenden Menge von Informationen im Internet steigt sowohl das Interesse als auch die Notwendigkeit relevantes Material von irrelevantem zu trennen. Ein erster Ansatz waren manuell erstellte Wrapper, die dieses Filtern übernehmen. Da deren Erstellung aufwendig ist und sie bei fast jeder noch so kleinen Änderung der Informationsquelle angepasst werden müssen, wurde nach Wegen gesucht, den Prozess der Wrappergenerierung zu automatisieren.

Die hier vorgestellten Techniken ermöglichen es, mittels markierter Beispielseiten Wrapper automatisch zu generieren. Allerdings kann die Markierung von Beispielseiten durch den Benutzer einen nicht unerheblichen Aufwand bedeuten. Es müssen zuerst einmal genügend Beispiele gesammelt und die darin relevanten Daten markiert werden. Wieviele Beispiele notwendig sind, hängt von der Internetseite selbst und von dem zu verwendenden Wrapper ab. Ändert sich die Seite, muss der zugehörige Wrapper neu generiert werden. Das erfordert die erneute Markierung von Beispielen. Durch die Benutzung grafischer Systeme kann die Markierung zumindest erleichtert werden. [5] beschreibt Techniken, die diese Arbeit zum Teil automatisch erledigen. Hier werden Funktionen benutzt, die eine Zeichenkette als Zahl, Landesname oder ähnliches erkennen.

Das System RoadRunner [3] kommt ganz ohne die Markierung relevanter Daten in Beispielseiten aus. Zur Erzeugung eines Wrappers werden die Gemeinsamkeiten, Ähnlichkeiten und Unterschiede von je zwei Seiten des gleichen Typs untersucht. Unterschiedliche Teilstrings zwischen HTML-Tags bei sonst gleicher Struktur sind dann zum Beispiel ein Hinweis für zu extrahierende Daten. Besonders nützlich ist dieses System bei sehr datenintensiven Quellen. Die HTML-Seiten werden dann fast ausschließlich von Programmen erzeugt. Damit ist die Struktur der Dokumente gleich. Seiten unterscheiden sich dann (fast) nur in den relevanten Daten.

Lixto [2] benutzt den Parsebaum eines HTML-Dokuments und lernt interaktiv mittels einer grafischen Benutzeroberfläche Regeln, um relevante Teile des Dokuments in XML umzuformen. Das erleichtert zudem die Weiterverarbeitung der Daten.

Wichtig in der Praxis ist auch, dass die Menge der Informationen oft zu groß ist, um sie auf einer einzigen Seite übersichtlich darstellen zu können (z.B. ein Telefonverzeichnis). Die Daten sind dann über mehrere Seiten verteilt und der Benutzer kann innerhalb dieser navigieren. Sogenannte Web-Crawler verfolgen selbständig Links und sammeln so die Daten von mehreren Seiten. Auch Lixto bietet diese Möglichkeit [1].

Interessant ist außerdem die Frage nach der Gültigkeit von Wrappern. In [7] wird Rapture beschrieben. Dieser Algorithmus benutzt Länge der Daten, Anteil besonderer Zeichen wie . , : (, um festzustellen, inwiefern die erwarteten und die tatsächlich erhaltenen Daten übereinstimmen. Dieses System findet unter anderem dann Anwendung, wenn sich die Daten oft ändern, ihre Form aber gleich bleibt (zum Beispiel also bei Aktienkursen).

Bei den Lernalgorithmen der vorgestellten Wrapper-Klassen ist der Stellenwert aller Beispielseiten gleich. *Boosting* [4] ist eine Technik, die einen Lernalgorithmus wiederholt auf eine Beispielmenge anwendet. Dabei werden Beispielseiten jedoch unterschiedlich gewichtet, so daß schwierigere Seiten einen höheren Stellenwert erhalten. Dadurch können mit recht einfachen Lernalgorithmen verlässliche Wrapper gelernt werden.

Ein anderes Gebiet für Verbesserungen ist die Ausführungsgeschwindigkeit besonders der Wrapper aus Abschnitt 3 bis 5. Da im allgemeinem mehrere Wrap-

per für eine Seite existieren, kann versucht werden durch Entwicklung entsprechender Heuristiken möglichst schnell einen gültigen Wrapper zu finden.

Es ist auch wichtig darauf hinzuweisen, dass Wrapper in der Praxis auch mit Ausnahmesituationen konfrontiert werden. Nicht gefundene Seiten, lange Wartezeiten und vieles mehr sollten deshalb wenn möglich auch berücksichtigt werden.

Literatur

1. R. Baumgartner, S. Flesca, G. Gottlob. Declarative information extraction, web crawling and recursive wrapping with Lixto. In: *Proceedings 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'2001)*, Band 2173 der *Lecture Notes in Computer Science*, S. 21-41. Springer-Verlag, 2001.
2. R. Baumgartner, S. Flesca, G. Gottlob. Visual web information extraction with Lixto. In: *Proceedings 27th International Conference on Very Large Data Bases (VLDB'2001)*, S. 119-128. Morgan Kaufmann, 2001.
3. V. Crescenzi, G. Mecca, P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In: *Proceedings 27th International Conference on Very Large Data Bases (VLDB'2001)*, S. 109-118. Morgan Kaufmann, 2001.
4. D. Freitag, N. Kushmerick. Boosted wrapper induction. In: *Proceedings 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI'2000)*, S. 577-583. AAAI Press/The MIT Press, 2000.
5. N. Kushmerick. *Wrapper induction for information extraction.* Dissertation, Department of Computer Science & Engineering, University of Washington, Seattle, WA, 1997.
6. N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15-68, 1999.
7. N. Kushmerick. Wrapper verification. *World Wide Web*, 3(2):79-94, 2000.
8. I. Muslea, S. Minton, G. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93-114, 2001.